Івано-Франківський національний технічний університет нафти і газу
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

Копей Володимир Богданович

УДК 622.276.054:[004.89+004.94]

ДИСЕРТАЦІЯ

Науково-методологічні основи автоматизованого проектування обладнання штангової свердловинної насосної установки

05.05.12 – машини нафтової та газової промисловості

Частина 2
Додатки

Подається на здобуття наукового ступеня доктора технічних наук

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____В. Б. Копей

Науковий консультант Петрина Юрій Дмитрович, д-р техн. наук, професор

Івано-Франківськ – 2020

ЗМІСТ

с.

ДОДАТОК А

Список публікацій здобувача

Праці, в яких опубліковані основні наукові результати дисертації

1. Experimental study of the reinforcement of damaged steel pipe by composite bandage / Kopey B., Rozgonjuk V., **Kopey V.**, Maksymuk O., Scherbina N., Nayda A. // Wiertnictwo Nafta Gaz. 2004. r.21/1. P. 125-134.

2. Finite-element analysis of the tubing thread / Kopey B., **Kopey V.**, Bebnarz S., Savula S. // Wiertnictwo Nafta Gaz. 2006. r.23/2. P. 681-685.

3. Оптимізація товщини композитних бандажів при ремонті трубопроводів з дефектами / Копей Б. В., **Копей В. Б.**, Максимук О. В., Щербина Н. М., Найда А. М. // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2007. № 2(16). С. 101-107.

4. Копей Б. В., Зінченко Ю. С., **Копей В. Б.** Аналіз поломок насосних штанг в промислових умовах // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2008. № 2(18). С. 49-56.

5. **Копей В. Б.** Аналіз способів підвищення ресурсу муфтового різьбового з'єднання насосних штанг // Розвідка та розробка нафтових і газових родовищ. 2008. № 4(29). С. 66-72.

6. Копей Б. В., Кузьмін О. О., **Копей В. Б.** Механічні методи зняття відкладень парафіну та асфальто-смолистих речовин з поверхні свердловинного обладнання // Нафтогазова енергетика. 2008. № 3(8). С. 10-14.

7. Сучасні методи боротьби з корозією глибинного обладнання ШСНУ / Копей Б. В., Онищук О. О., Онищук С. Ю., **Копей В. Б.** // Нафтогазова енергетика. 2008. № 2(7). С. 13-16.

8. **Копей В. Б.** Застосування системи CAD/FEA для розрахунку і оптимізації різьбових з'єднань нафтогазового обладнання // Розвідка та розробка нафтових і газових родовищ. 2009. № 3(32). С. 43-49.

9. Копей Б. В., Михайлюк В. В., **Копей В. Б.** Моделювання різьб насосних штанг методом скінченних елементів // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2009. № 2(20). С. 61-67.

10. Копей Б. В., Кузьмін О. О., **Копей В. Б.** Розроблення з'єднань склопластикових порожнистих насосних штанг та визначення навантажень на них // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2010. № 1(23). С. 77-83.

11. **Копей В. Б.** Розробляння та аналіз параметричних скінченно-елементних моделей різьбових з'єднань в Abaqus® // Нафтогазова енергетика. 2010. № 1(12). С. 31-36.

12. **Копей В. Б.** Скінченно-елементний аналіз та оптимізація різьбових з'єднань // Вісник Національного технічного університету України "Київський політехнічний інститут". Серія машинобудування. 2010. № 58. С. 42-47.

13. **Копей В. Б.** Дослідження впливу геометричних параметрів муфтового різьбового з'єднання насосних штанг на напруження у впадинах різьби ніпеля // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2010. № 2(24). С. 81-85.

14. **Копей В. Б.**, Петрина Ю. Д. Принципи розробки бази знань з проблем надійності і довговічності різьбових з'єднань // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2010. № 4(26). С. 66-69.

15. **Копей В. Б.** Автоматизоване проектування з'єднання тіла склопластикової насосної штанги зі сталевою головкою // Комп'ютерно-інтегровані технології: освіта, наука, виробництво. 2011. №5. С. 142-147.

16. Використання явища резонансу для комплектування колони насосних штанг / Олійник А. П., Копей Б. В., Зінченко Ю. С., **Копей В. Б.** // Розвідка та розробка нафтових і газових родовищ. 2011. №1 (38). С. 69-75.

17. **Копей В. Б.**, Палійчук І. І. Застосування мови програмування Python для побудови баз знань з проблем надійності і довговічності штангових

свердловинних насосних установок // Нафтогазова енергетика. 2011. №2(15). С. 12-18.

18. **Копей В. Б.** Імітаційна модель свердловинної штангової насосної установки на основі абстрактних автоматів // Розвідка та розробка нафтових і газових родовищ. 2017. №3(64). С. 40-49.

19. **Копей В. Б.**, Копей Б. В., Кузьмін О. О. Принципи побудови моделі свердловинної штангової насосної установки для середовища Maplesoft MapleSim 7 // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2017. №2(43). С. 42-52.

20. **Копей В. Б.** Абстрактна модель інформаційної системи підтримки життєвого циклу виробу // Прикарпатський вісник НТШ. Число. 2017. №2(38). С. 71-96.

21. **Kopei V. B.**, Onysko O. R., Panchuk V. G. Computerized system based on FreeCAD for geometric simulation of the oil and gas equipment thread turning // IOP Conf. Ser.: Mater. Sci. Eng. 2019. 477:012032. DOI: 10.1088/1757-899X/477/1/012032. (*Scopus*)

22. **Kopei V. B.**, Onysko O. R., Panchuk V. G. Component-oriented acausal modeling of the dynamical systems in Python language on the example of the model of the sucker rod string // PeerJ Computer Science. 2019. 5:e227. DOI: 10.7717/peerj-cs.227. (*Scopus, Q1*)

23. **Копей В. Б.**, Онисько О. Р., Жигуц Ю. Ю. Обґрунтування застосування двоопорних різьбових з’єднань пустотілих насосних штанг // Науковий вісник Івано-Франківського національного технічного університету нафти і газу. 2019. №1(46). С. 7-15. DOI: 10.31471/1993-9965-2019-1(46)-7-15.

24. **Kopei V.**, Onysko O., Panchuk V. The application of the uncorrected tool with a negative rake angle for tapered thread turning // Ivanov V. et al. (eds) Advances in Design, Simulation and Manufacturing II. DSMIE 2019. Lecture Notes in Mechanical Engineering. Cham : Springer, 2020. P. 149-158. DOI: 10.1007/978-3-030-22365-6_15. (*Scopus*)

25. **Kopei V. B.**, Onysko O. R., Panchuk V. G. Principles of development of product lifecycle management system for threaded connections based on the Python programming language // J. Phys.: Conf. Ser. 2020. 1426:12033. DOI: 10.1088/1742-6596/1426/1/012033. (*Scopus*)

26. Onysko O. R., **Kopey V. B.**, Panchuk V. G. Theoretical investigation of the tapered thread joint surface contact pressure in the dependence on the profile and the geometric parameters of the threading turning tool // IOP Conf. Ser.: Mater. Sci. Eng. 2020. 749:012007. DOI: 10.1088/1757-899X/749/1/012007. (*Scopus*)

Праці, які засвідчують апробацію матеріалів дисертації

27. **Копей В. Б.** Аналіз способів підвищення ресурсу муфтового різьбового з'єднання насосних штанг // Анотації Міжнародної науково-практичної конференції молодих вчених "Техніка і прогресивні технології у нафтогазовій інженерії", м. Івано-Франківськ, 16-20 вересня 2008 р. Івано-Франківськ : Факел, 2008. С. 55. (*форма участі – очна*)

28. Скінчено-елементний аналіз насосних штанг з зарізьбовими канавками / Копей Б. В., **Копей В. Б.**, Петрина Ю. Д, Михайлюк В. В. // Анотації Міжнародної науково-технічної конференції "Нафтогазова енергетика: проблеми і перспективи". м. Івано-Франківськ, 20-23 жовтня 2009 р. Івано-Франківськ : ІФНТУНГ, 2009. С. 67. (*форма участі – очна*)

29. **Копей В. Б.**, Панчук А. Г. Дослідження залежності напружень в муфтовому різьбовому з'єднанні насосних штанг від характеристик матеріалів деталей з'єднання // Сучасні технології в промисловому виробництві : матеріали Всеукраїнської міжвузівської науково-технічної конференції, м. Суми, 19-23 квітня 2010 р. Ч. ІІ. Суми : Вид-во СумДУ, 2010. С. 122-123. (*форма участі – заочна*)

30. **Копей В. Б.** Скінченно-елементний аналіз та оптимізація різьбових з'єднань // Тези доповідей XI Міжнародної науково-технічної конференції "Прогресивна техніка і технологія - 2010", м. Київ, 18-21 травня 2010 р. Київ : НТУУ "КПІ", 2010. С. 99. (*форма участі – очна*)

31. **Копей В.** Скінченно-елементне моделювання та оптимізація параметрів муфтового різьбового з’єднання насосних штанг за критерієм втомної міцності // Комп'ютерні технології: наука і освіта : тези доповідей V Всеукраїнської науково-практичної конференції, м. Івано-Франківськ, 29.09-3.10.2010 р. Київ : Університет "Україна", 2010. С. 109-112. (*форма участі – очна*)

32. **Копей В. Б.**, Панчук А. Г. Оптимізація параметрів з’єднання тіла склопластикової насосної штанги зі сталевою головкою // Инновационные технологии в машиностроении : материалы Международной научно-практической конференции, г. Запорожье, 17-21 мая 2011 г. Том 2. Запорожье : Запорожская торгово-промышленная палата, 2011. С. 58-60. (*форма участі – заочна*)

33. **Копей В. Б.**, Венгрынюк Т. П. Моделирование дефектов труб в SolidWorks® // Надежность и безопасность магистрального трубопроводного транспорта : материалы VII Междунар. науч.-техн. конф., Новополоцк, 22 – 25 ноября 2011 г. / под общ. ред. д-ра техн. наук, проф. В. К. Липского. Новополоцк : Полоц. гос. ун-т, 2011. С. 250-251. (*форма участі – заочна*)

34. **Копей В. Б.**, Петрина Ю. Д., Венгрынюк Т. П. Конечно-элементное моделирование ремонта труб с дефектами стеклопластиковыми бандажами в SolidWorks® // Надежность и безопасность магистрального трубопроводного транспорта : материалы VII Междунар. науч.-техн. конф., Новополоцк, 22 – 25 ноября 2011 г. / под общ. ред. д-ра техн. наук, проф. В. К. Липского. Новополоцк : Полоц. гос. ун-т, 2011. С. 248-250. (*форма участі – заочна*)

35. **Копей В. Б.** Створення експертної системи з проблем надійності і довговічності різьбових з’єднань в редакторі онтологій Protégé-OWL 3.5 // Збірка наукових праць за матеріалами Міжнародної науково-практичної конференції "Наукові дослідження сучасності. Випуск 4", м. Київ, 30 травня 2012 р. Частина 1. Київ : НАІРІ, 2012. С. 99-101. (*форма участі – заочна*)

36. **Копей В. Б.** Створення експертної системи з проблем надійності і довговічності різьбових з’єднань мовою Python // Збірка наукових праць за матеріалами Міжнародної наукової конференції "Наука - XXI століття. Випуск 2", м. Київ, 27 червня 2012 р. Київ : НАІРІ, 2012. С. 117-122. (*форма участі – заочна*)

37. **Копей В. Б.** Скінченно-елементне моделювання та аналіз втомної міцності насосних штанг в зоні скруглення між піделеваторним буртом і тілом штанги // Тези доповідей Міжнародної науково-практичної конференції молодих учених та студентів "Техніка і прогресивні технології у нафтогазовій інженерії - 2012", м. Івано-Франківськ, 5-7 листопада 2012 р. Івано-Франківськ : ІФНТУНГ, 2012. С. 123-126. (*форма участі – очна*)

38. **Копей В. Б.** Обґрунтування доцільності збільшення довжини розвантажувальної канавки ніпеля насосної штанги // Технологічний аудит та резерви виробництва (Спецвипуск. Матеріали науково-практичної конференції "Наукові підсумки 2012р.", м. Харків, 2012 р.). 2012. № 6/2 (8). С. 7-8. (*форма участі – заочна*)

39. **Копей В. Б.** Моделювання клапана свердловинного штангового насоса методом обчислювальної гідродинаміки в Abaqus/CAE® // Матеріали Міжнародної науково-технічної конференції "Математичне моделювання прикладних задач математики, фізики, механіки ММАР-2013", м. Харків, 5 – 25 травня 2013 р. Харків : ХНАДУ, 2013. URL: http://files.khadi.kharkov.ua/images/Fizika.pdf (дата звернення: 29.02.2016). (*форма участі – заочна*)

40. **Kopey V.** Development of model of sucker-rod pumping system by using Maplesim™ software // International scientific and technical conference "Oil and Gas Power Engineering 2013" : abstracts, Ivano-Frankivsk, October 7-11, 2013. Ivano-Frankivsk : IFNTUOG, 2013. P. 116-118. (*форма участі – очна*)

41. **Копей В. Б.** Принципи побудови тривимірної параметричної моделі верстата-гойдалки в SolidWorks® 2012 // Всеукраїнський науково-практичний семінар "Графічна освіта у ВНЗ: стан та перспективи" : збірник тез доповідей, м. Івано-Франківськ, 19-20 вересня 2013 р. Івано-Франківськ : ІФНТУНГ, 2013. С. 87-89. (*форма участі – очна*)

42. **Копей В. Б.** Використання вільного програмного забезпечення для розробки системи скінченно-елементного аналізу різьбових з’єднань нафтогазового обладнання // Матеріали Міжнародної науково-технічної

конференції "Машини, обладнання і матеріали для нарощування вітчизняного видобутку та диверсифікації постачання нафти і газу" ІІМ – 2016, м. Івано-Франківськ, 16-20 травня 2016 р. Івано-Франківськ : ІФНТУНГ, 2016. С. 277-280. (*форма участі – очна*)

43. **Копей В. Б.** Моделювання гармонічного осьового навантажування пресового з'єднання склопластикової насосної штанги // Соціально-економічний розвиток в умовах глобалізації : матеріали XLIX Міжнародної науково-практичної конференції, м. Чернівці, 29-30 листопада 2016 р. Т. 1. Київ : Науково-видавничий центр "Лабораторія думки", 2016. С. 7-10. (*форма участі – заочна*)

44. **Копей В. Б.** Моделювання втомної міцності ніпеля склопластикової насосної штанги // Матеріали ІІ-ї Міжнародної науково-практичної конференції "Теоретичні та практичні аспекти розвитку науки", м. Київ, 29 – 30 листопада 2016 р. Ч. 2. Київ : МЦНД, 2016. С. 22-24. (*форма участі – заочна*)

45. **Копей В. Б.** Компонентно-орієнтоване моделювання кінематики механізмів мовою Python на прикладі механізму верстата-гойдалки // Тези доповідей Міжнародної науково-технічної конференції "Нафтогазова енергетика-2017", м. Івано-Франківськ, 15-19 травня 2017 р. Івано-Франківськ : ІФНТУНГ, 2017. С. 362-364. (*форма участі – очна*)

46. **Копей В. Б.**, Копей Б. В., Кузьмін О. О. Принципи побудови моделі свердловинної штангової насосної установки для середовища Maplesoft MapleSim 7 // Тези доповідей Міжнародної науково-технічної конференції "Нафтогазова енергетика-2017", м. Івано-Франківськ, 15-19 травня 2017 р. Івано-Франківськ : ІФНТУНГ, 2017. С. 364-365. (*форма участі – очна*)

47. **Копей В. Б.** Імітаційна модель свердловинної штангової насосної установки на основі абстрактних автоматів // Тези доповідей Міжнародної науково-технічної конференції "Нафтогазова енергетика-2017", м. Івано-Франківськ, 15-19 травня 2017 р. Івано-Франківськ : ІФНТУНГ, 2017. С. 365-366. (*форма участі – очна*)

48. **Копей В. Б.**, Воробець М. І. Система автоматизованого проектування металополімерних з'єднань на основі вільного програмного забезпечення // Машинобудування очима молодих: прогресивні ідеї – наука – виробництво (МОМ

– 2017) : матеріали тез доповідей XVII Міжнародної науково-практичної конференції, м. Чернігів, 01 – 03 листопада 2017 р. Чернігів : ЧНТУ, 2017. С. 31-33. (*форма участі – заочна*)

49. **Kopei V.**, Panchuk V., Onysko O. Component-oriented modelling of dynamical systems in Python language on the example of the model of the sucker rod string // International conference Innovative Ideas in Science 2017 : book of abstracts, Banja Luka, 02 - 03 November 2017. Banja Luka : University of Business Engineering and Management, 2017. P. 33. (*форма участі – очна*)

50. **Копей В. Б.** Статистичні моделі відмов колон насосних штанг // Матеріали II Міжнародної науково-технічної конференції "Машини, обладнання і матеріали для нарощування вітчизняного видобутку нафти і газу PGE – 2018", м. Івано-Франківськ, 24-27 квітня 2018 р. Івано-Франківськ : ІФНТУНГ, 2018. С. 310-313. (*форма участі – очна*)

51. **Копей В. Б.** Прогнозування частоти відмов колон насосних штанг за допомогою ансамблів дерев рішень // Комплексне забезпечення якості технологічних процесів та систем (КЗЯТПС – 2018) : матеріали тез доповідей VIII Міжнародної науково-практичної конференції, м. Чернігів, 10–12 травня 2018 р. : у 2-х т. / Чернігівський національний технологічний університет [та ін.]; відп. за вип.: Єрошенко Андрій Михайлович [та ін.]. Т. 2. Чернігів : ЧНТУ, 2018. С. 198-200. (*форма участі – заочна*)

52. **Kopei V.**, Onysko O., Panchuk V. Computerized system based on FreeCAD for geometric simulation of thread turning of oil and gas pipes // 6th International Conference of Applied Science, May 9-11, 2018, Banja Luka, Bosnia and Herzegovina : book of abstracts. Banja Luka : University of Banja Luka, 2018. P. 108. (*форма участі – очна*)

53. **Копей В. Б.** Алгоритм інтелектуальної системи на основі міждисциплінарних досліджень загальносистемних закономірностей // Комп’ютерне моделювання та оптимізація складних систем (КМОСС-2018) : матеріали IV Міжнародної науково-технічної конференції, м. Дніпро, 1-2 листопада 2018 р. / Міністерство освіти і науки України, Державний вищий

навчальний заклад "Український державний хіміко-технологічний університет". Дніпро : Баланс-клуб, 2018. С. 246-248. (*форма участі – заочна*)

54. **Копей В. Б.**, Онисько О. Р., Жигуц Ю. Ю. Обґрунтування застосування двоопорних муфтових різьбових з'єднань пустотілих насосних штанг // Матеріали доповідей VIII Міжнародної науково-технічної конференції "Прогресивні технології у машинобудуванні РТМЕ-2019", 4-8 лютого 2019 р., м. Івано-Франківськ-Яремче. Івано-Франківськ : ІФНТУНГ, 2019. С. 146-149. (*форма участі – очна*)

55. **Kopei V.**, Onysko O., Panchuk V. Principles of the product lifecycle management system development for threaded connections based on the Python programming language // International Conference of Applied Science ICAS 2019 : book of abstracts, May 9-11, 2019, Hunedoara, Romania. Timisoara : Editura EUROBIT, 2019. P. 42. (*форма участі – очна*)

56. **Kopei V. B.**, Kopei B. V. Harmonic axial loading analisys of the tubing threaded connection // Modern achievements of science and education : proceedings of XIV International scientific conference, September 26 - October 3, 2019, Netanya, Israel. Khmelnytskyi : KhNU, 2019. P. 29-37. (*форма участі – очна*)

Праці, які додатково відображають наукові результати дисертації

57. Комп'ютерна програма "Програма для побудови баз знань з проблем надійності і довговічності штангових свердловинних насосних установок" ("SuckerRodPumpingUnitKB") : свідоцтво про реєстрацію авторського права на твір № 42157 / **Копей Володимир Богданович**. Дата реєстрації 08.02.2012 // Каталог державної реєстрації. Вип.16/2012. С. 42.

58. Комп`ютерна програма "Модуль для компонентно-орієнтованого моделювання динамічних систем та приклади його застосування для побудови моделей колони насосних штанг" : свідоцтво про реєстрацію авторського права на твір № 73098 / **Копей Володимир Богданович**. Дата реєстрації 25.07.17 // Бюл. "Авторське право і суміжні права" № 46/2017. С. 154.

ДОДАТОК Б

Статистичні моделі відмов штангових колон

Код програм також доступний в GitHub (vkopey/RodStats: Statistical models of sucker rod strings failures. URL: http://github.com/vkopey/RodStats).

Таблиця Б.1 – Частина статистичних даних відмов колон ШН

N	Name	Year	Month	Pump	H	H_fail	D_fail	Type	H16	H19	H22	H25	Gas	Water	Product	Paraffin	Curv_begin	Curv_end	Stress
1	299	1999	12	95	815	416	25	4	0	0	0	816	222	98	4566	0			107
2	39	1999	12	70	1127	176	25	2	0	0	576	552	282	98	5027	0			94
3	514	1999	12	57	1241	0	0	5	0	232	656	248	450	97	2482	0	1030	1120	72
4	354	1999	12	57	1508	688	19	3	0	1144	208	152	181	75	2288	0			84
5	22 стр	1999	12	44	1552	752	22	8	0	480	864	168	324	74	1952	1			69
6	24 сп	1999	12	32	1080	400	22	4	0	304	720	16	514	54	564	1			37
7	30 сп	1999	12	44	1497	992	19	1	0	680	664	128	450	52	588	1			66
8	168 пд	1999	12	38	1680	416	22	1	0	0	1344	336				0			71
9	710	1999	12	57	1345	1089	22	1	0	0	768	560	279	86	2460	1			86
10	246	1999	12	44	1532	264	22	4	0	0	1344	176	101	90	3554	0			72
11	54 стр	1999	12	44	1644	430	22	8	0	240	1152	192	400	69	1757	1			74
12	51 стр	1999	12	44	1708	0	0	5	0	40	1136	472	246	72	2537	0			79
13	907	1999	12	44	1550	200	22	1	0	1032	488	16	208	72	2518	0			66
14	249	1999	12	57	1426	1320	19	1	0	200	896	288	217	85	4203	0	1080	1426	86
15	675	1999	12	57	1290	776	22	7	0	336	832	112				1	340	350	78
16	317	1999	12	95	1000	424	25	7	0	0	128	872	155	90	1210	0			127
17	235	1999	12	57	1097	656	22	8	0	312	688	88	250	97	3080	0			68
18	760	1999	12	38	1596	120	25	7	0	936	464	144		75	1396	0			60
19	60 стр	1999	12	44	1800	1280	19	2	0	480	896	488	256	63	1778	1			85
20	85 стр	1999	12	44	1732	880	22	8					468	71	631	1			
21	28	1999	12	44	1300	0	0	5					302	90	262	1	1000	1300	
22	238	1999	11	95	984	904	25	8	0	0	0	992	160	99	5482	1			127
23	820	1999	11	57	1454	720	22	4	0	0	1096	336	193	87	1750	1			90
24	244	1999	11	57	1497	480	22	2	0	48	1352	80	121	95	4970	1			90
25	824	1999	11	44	1473	1441	19	1	0	208	1104	40	223	95	2857	1			63
26	246	1999	11	44	1532	776	22	8	0	0	1352	160	101	90	3554	0			72
27	246	1999	11	44	1532	312	22	4	0	0	1352	160	101	90	3554	0			72
28	165 пд	1999	11	57	1347	568	22	1	0	0	880	392				0			82
29	40 стр	1999	11	44	1303	0	0	5	0	616	584	80				0			58
30	4 підб	1999	11	32	1838	640	19	1	0	696	440	688				0			66
31	79 підб	1999	11	32	1656	130	25	4	0	1008	448	224				0			57
32	210	1999	11	70	1192	672	19	2	0	464	208	520	217	92	6469	0			95
33	703	1999	11	44	1596	280	22	2	0	816	456	264	212	91	3655	0			69
34	1 ч	1999	11	32	1775	200	25	3	0	704	448	520	1333	74	42	0			60
35	51 стр	1999	11	44	1708	1304	22	2	0	40	1136	472	246	72	2537	0			79
36	921	1999	11	57	1501	440	22	2	0	232	832	408	269	90	9622	0			91
37	26 пд	1999	11	38	1600	696	22	2	0	712	752	272				1			69
38	916	1999	11	44	1667	360	22	2	0	608	640	376				0			74
39	141 пд	1999	11	44	1606	320	22	7	0	856	672	40	694	13	1236	0			68
40	34 пд	1999	11	32	1763	0	0	5								0			
41	176 пд	1999	11	29	1600	400	22	7	0	640	960	0				0			51
42	52 стр	1999	11	44	1676	1376	22	2	0	64	1408	152	207	65	2451	1			76
43	86 пд	1999	11	57	1335	280	25	1	0	88	672	520				0			83
44	326	1999	11	70	1264	1040	22	7	0	0	224	1032	222	94	4667	0	1185	1264	106

Лістинг Б.1 – polyfit3.py – програма для побудови статистичних моделей виду *f*(*h*)

```
#encoding: utf-8
from scipy.optimize import curve_fit
import numpy as np
import itertools

def score(y,ya): #r**2
    return np.corrcoef(y,ya)[0, 1]**2

def poly(x, A, P):
    """Значення полінома з коефіцієнтами A і степенями P"""
    comp=[a*x**p for a,p in zip(A,P)] # доданки полінома
    return sum(comp) # значення полінома a[0]+a[1]*x+a[2]*x**2

def pfit(x, y, P, Z):
    """Апрокс. поліномом з степенями P з врахуванням вектора занулення Z"""
    def polyz(x, *A): # поліном з коеф. A з їх зануленням вектором Z"""
        A=[a*z for a,z in zip(A,Z)] # коеф. полінома з врахув. занулення
        return poly(x, A, P)

    A, cov = curve_fit(polyz, x, y, p0=Z) # апроксимувати нелінійним
методом найменших квадратів
    s=score(y,poly(x, A, P)) # внутрішній критерій - наскільки добре p
описує точки x,y
    return A, s # коефіцієнти полінома і R**2

def crossValidation(x, y, P, Z):
    """Повертає узагальнений критерій"""
    # ділимо дані на групи
    A,s0=pfit(x,y,P,Z) # підгонка усього
    x1,y1=x[::-2],y[::-2] # непарні
    x2,y2=x[::2],y[::2] # парні
    A1,s01=pfit(x1,y1,P,Z) # підгонка групи 1
    s1=score(y2, poly(x2,A1,P))
    A2,s02=pfit(x2,y2,P,Z) # підгонка групи 2
    s2=score(y1, poly(x1,A2,P))
    s3=score(poly(x,A1,P), poly(x,A2,P))
    #return np.mean([s0,s3]) # або
    return np.mean([s0,s1,s2,s3])
    #return s1
    #return np.mean([s01,s1]) # найпростіший випадок

def combi(x,y,Np=4):
    """Комбінаторний алгоритм індуктивної самоорганізації моделі
```

```
    x,y - координати емпіричних точок
    Np - максимальна степінь полінома"""
    P=range(Np+1) # степені полінома [0,1,2,3,...]
    res=[] # результати
    # усі можливі комбінації
    for Z in itertools.product([0,1],repeat=Np+1):
        #Z - список для занулення коефіцієнтів (1,1,0,...)
        if any(Z[1:]): # окрім поліномів f=0 та f=const
            res.append([Z, crossValidation(x,y,P,Z)]) # узагальн. критерій
    res.sort(key=lambda x: x[1], reverse=True) # сорт. за спаданням score
    return res # відсортований список Z, score

def plot(x,y,Z,xmin=None,xmax=None):
    """Рисує дані і поліном Z"""
    Np=len(Z)-1
    P=range(Np+1)
    A,s=pfit(x, y, P, Z)
    print "A, R**2 = ", A,s
    import matplotlib.pyplot as plt
    plt.plot(x,y,'ro')
    if xmin==None: xmin=x.min()
    if xmax==None: xmax=x.max()
    x_=np.linspace(xmin,xmax,100)
    y_=poly(x_,A,P)
    plt.plot(x_,y_,'k-',linewidth=2)

    from scipy.integrate import cumtrapz, quad
    y_int = cumtrapz(y_, x_, initial=0) # знач. первісної шляхом інтегрув.
    print 'Площа під кривою=', quad(poly,xmin,xmax,args=(A,P))[0]
    plt.plot(x_,y_int,'k--',linewidth=2)
    plt.show()
    #print "integral, score=", score2(A,P)

# def score2(A,P):
#     """Додатковий критерій. Для підгонок функцій густини розподілу"""
#     from scipy.integrate import quad
#     s, err = quad(poly, 0, 1, args=(A,P)) # інтегрувати в межах
#     # s повинна бути рівна 1
#     return s, np.exp(-(s-1)**2) # повертає від 0 до 1

if __name__=='__main__': # приклад використання
    N=16 # кількість точок
    # координати емпіричних точок
    x = np.linspace(0, 10, N)
    y = np.array([0,2,1,4,5,7,6,7,8,7,9,9,8,9,9,9])
    Np=4 # степінь полінома
```

```
    res=combi(x,y,Np) # список моделей
    for i in res:
        print i
    plot(x,y,Z=res[0][0]) # рисуємо найкращий поліном
```

Лістинг Б.2 – forSclearn3.py – програма для прогнозування частоти аварій свердловини за її параметрами (регресія)

```
# -*- coding: utf-8 -*-
"""
Задача регресії - прогнозування частоти аварій свердловини за її
параметрами. Те саме що forSclearn2.py, але для пошуку оптимальних
параметрів використовували differential_evolution
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('123_2017_part.csv',sep=';') # прочитати з файлу csv
df=df[['Pump', 'Stress', 'Gas', 'Curv', 'Water', 'H', 'Product',
'Paraffin', 'Kv', 'H19', 'H22', 'H25', 'Month']]

df=df.dropna()
#print df
X=df.drop(['Kv'], axis=1)
y=df['Kv']

from sklearn.utils import shuffle
X, y = shuffle(X, y) # випадкове перемішування даних

from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

# дані для побудови моделі і екзамену
X_, x_test, y_, y_test =
train_test_split(X,y,test_size=0.2)#,random_state=0)

def f(x):
    #n_e, m_d = int(round(x[0])), int(round(x[1]))
    #model=RandomForestRegressor(n_estimators=n_e, max_depth=m_d)
```

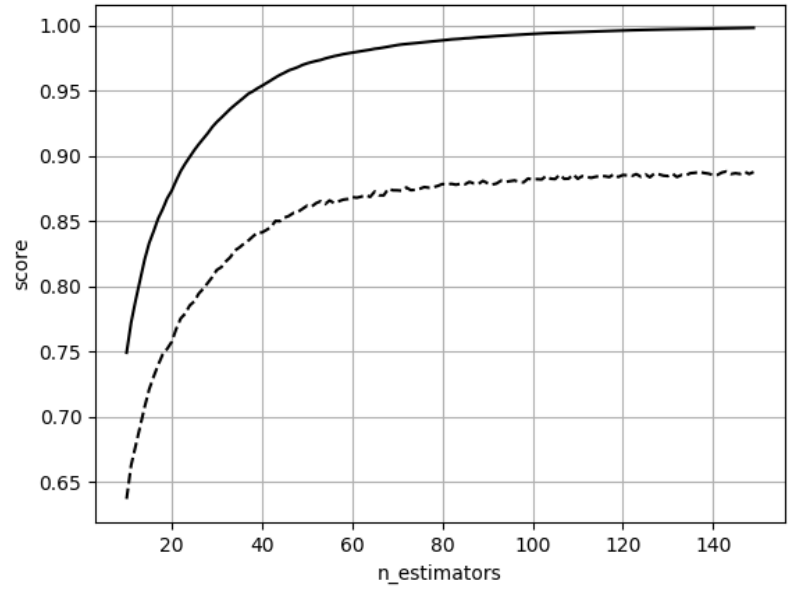
```
    n_e, l_r, m_d = int(round(x[0])), x[1], int(round(x[2]))
    model=GradientBoostingRegressor(n_estimators=n_e, learning_rate=l_r,
max_depth=m_d)
    s=cross_val_score(model, X_, y_, cv=7) # перехресна перевірка
    print s.mean()
    return -s.mean()

from scipy.optimize import differential_evolution
#bounds = [(50, 150), (3, 5)] # границі
bounds = [(50, 150), (0.01, 0.6), (3, 5)] # границі
# закоментуйте наступні 2 рядки, якщо параметри моделі уже відомі
#res = differential_evolution(f, bounds=bounds)
#print res
##
# найкраща модель
#model = RandomForestRegressor(n_estimators=133, max_depth=5)
model = GradientBoostingRegressor(n_estimators=120, learning_rate=0.15,
max_depth=4)
model.fit(X_, y_) # найкраща модель для даних для навчання
Y_test=model.predict(x_test) # екзамен на тестових даних (Мюллер с.279)
from sklearn.metrics import r2_score
print r2_score(y_test, Y_test) # або model.score(x_test, y_test)

plt.scatter(y_test, Y_test) # реальні і прогнозовані тестові дані
plt.xticks(np.arange(1,16))
#plt.yticks(np.arange(0,17))
#plt.axis('equal')
plt.grid()
plt.show()

imp=zip(model.feature_importances_, X.columns.values)
for score, name in sorted(imp,reverse=True):
    print score, name

## виконати ці рядки з консолі, а потім виконати програму N раз для
перехресної перевірки
#yt=dict.fromkeys(range(1,16))
#for i in yt: yt[i]=[]
##
for y,Y in zip(y_test, Y_test):
    yt[y].append(Y)

# зберігає у файл csv. Потім відкрити в Excel, замінити '.' на ',',
зберегти і передати в Statistica 10 для аналізу
df = pd.DataFrame() # об'єкт DataFrame
for i in yt: # кожний стопчик
```

```
    df[i] = pd.Series(yt[i]) # об'єкт Series
df.to_csv('yt_Yt.csv',sep=';',index=False,header=False,mode='w+') # увага!
режим додавання до файлу

# print np.mean(yt[1]), np.std(yt[1])
# plt.figure()
# plt.hist(yt[1]) # гістограма
# plt.show()
```

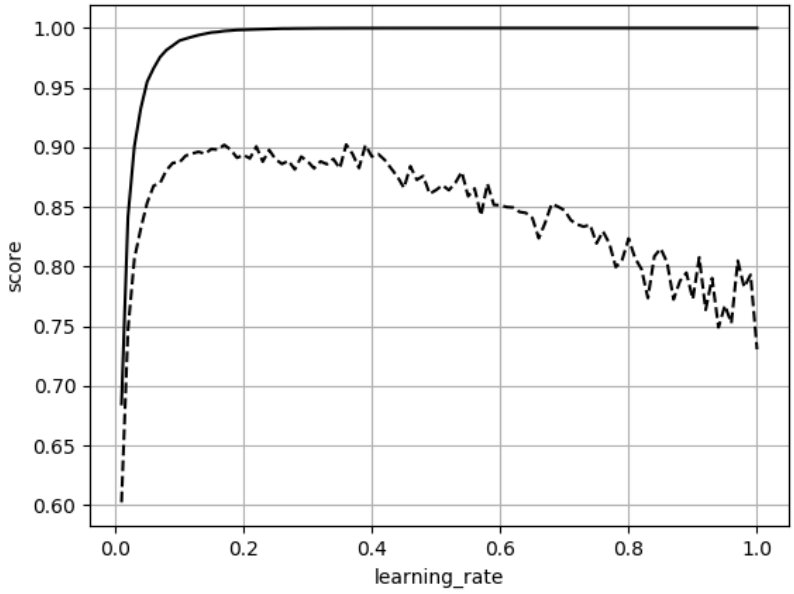
Лістинг Б.3 – forSclearn1.py – програма для прогнозування класу аварійності свердловини за її параметрами (класифікація)

```
# -*- coding: utf-8 -*-
"""
Задача класифікації - прогнозування класу аварійності свердловини за її
параметрами. Те саме що forSclearn0.py, але для пошуку оптимальних
параметрів використовували differential_evolution
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('123_2017_part.csv',sep=';') # прочитати з файлу csv
df=df[['Pump', 'Stress', 'Gas', 'Curv', 'Water', 'H', 'Product',
'Paraffin', 'H19', 'H22', 'H25', 'Month', 'Kv_Cat']]

df=df.dropna()
X=df.drop(['Kv_Cat'], axis=1)
y=df['Kv_Cat']

from sklearn.utils import shuffle
X, y = shuffle(X, y) # випадкове перемішування даних #, random_state=0
from sklearn.ensemble import
RandomForestClassifier,GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

# дані для побудови моделі і екзамену
X_, x_test, y_, y_test =
train_test_split(X,y,test_size=0.2)#,random_state=0)

def f(x):
```

```
    #n_e, m_d = int(round(x[0])), int(round(x[1]))
    #model=RandomForestClassifier(n_estimators=n_e, max_depth=m_d)
    n_e, l_r, m_d = int(round(x[0])), x[1], int(round(x[2]))
    model=GradientBoostingClassifier(n_estimators=n_e, learning_rate=l_r,
max_depth=m_d)
    s=cross_val_score(model, X_, y_, cv=7) # перехресна перевірка
    print s.mean()
    return -s.mean()

from scipy.optimize import differential_evolution
#bounds = [(50, 150), (3, 5)] # границі
bounds = [(50, 150), (0.01, 0.6), (3, 5)] # границі
# закоментуйте наступні 2 рядки, якщо параметри відомі
#res = differential_evolution(f, bounds=bounds)
#print res
##
#model = RandomForestClassifier(n_estimators=121, max_depth=5)
model=GradientBoostingClassifier(n_estimators=120,learning_rate=0.38,max_de
pth=5)
model.fit(X_,y_)

imp=zip(model.feature_importances_, X.columns.values)
for score, name in sorted(imp,reverse=True):
    print score, name

# звіт по класифікації (виконайте кілька раз і обчисліть середнє)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
Y_test=model.predict(x_test) # екзамен на тестових даних (Мюллер с.279)
print confusion_matrix(y_test, Y_test) # матриця помилок
print classification_report(y_test, Y_test) # повний звіт по класифікації

## крива точності-повноти
# будується для різних порогових значень імовірності
from sklearn.metrics import precision_recall_curve
y_scores=model.predict_proba(x_test)[:,1] # імовірності класу 1 тест. даних
precision, recall, thresholds = precision_recall_curve(y_test, y_scores)
plt.plot(precision, recall, 'k-')
for p,r,t in zip(precision[:-1], recall[:-1], thresholds)[::4]: # кожна
четверта імовірність
    plt.text(p,r,round(t,2))
plt.xlabel(u"Точність"), plt.ylabel(u"Повнота")
plt.show()
# середня точність класифікатора (площа під кривою точність-повнота)
from sklearn.metrics import average_precision_score
print average_precision_score(y_test, y_scores)
```



а



б

а) – для параметра *n_estimators*; б) – для параметра *learning_rate*

Рисунок Б.1 – Криві перевірки GradientBoostingRegressor

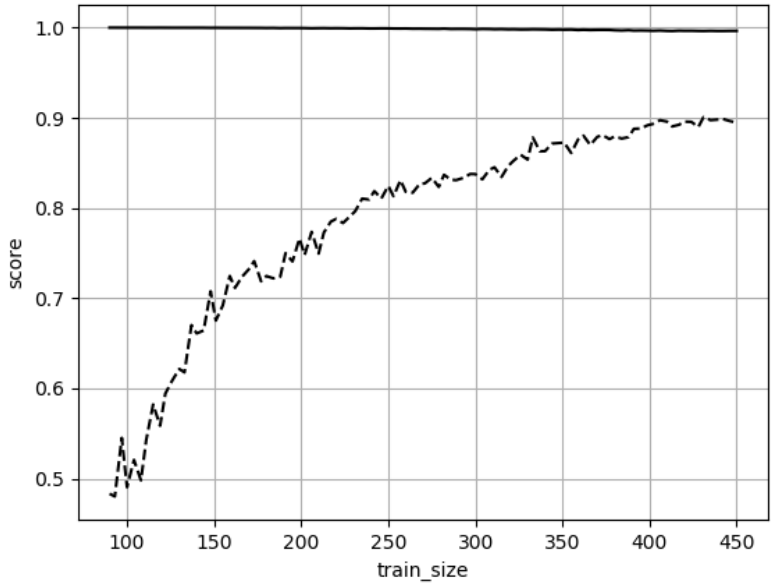
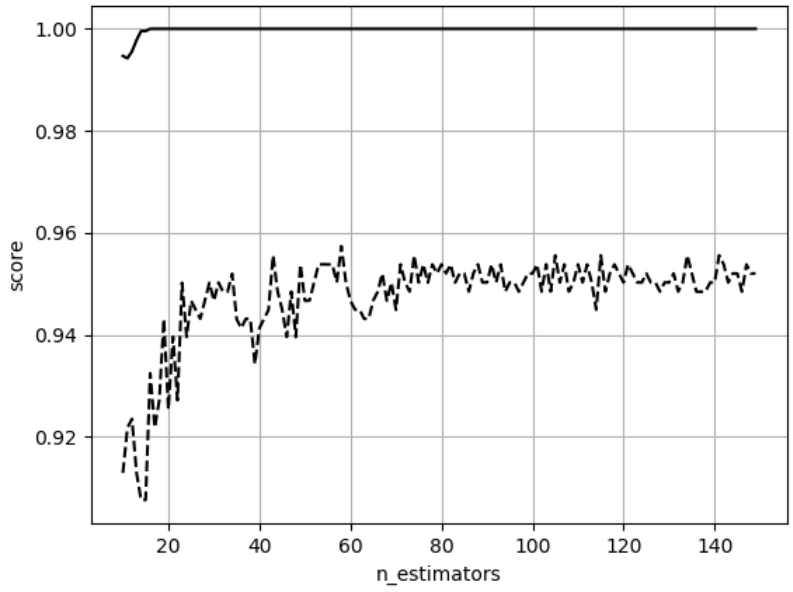
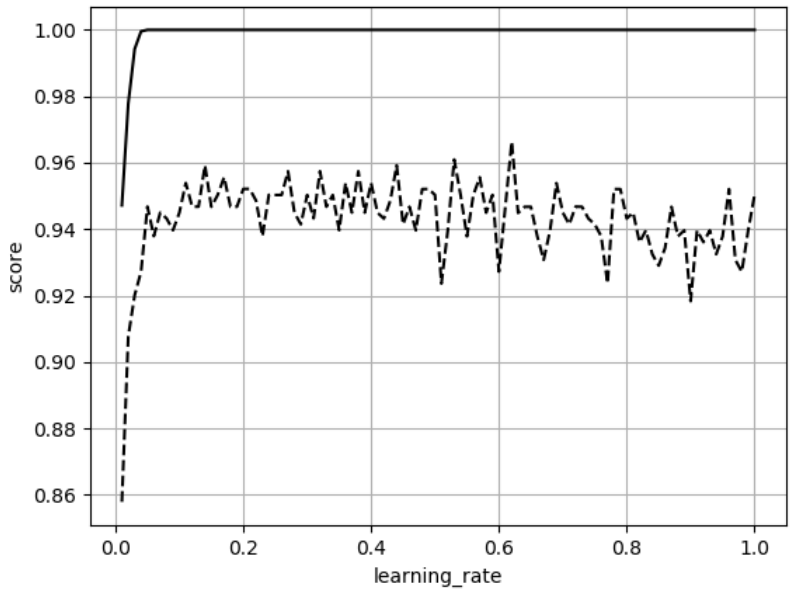


Рисунок Б.2 – Криві навчання GradientBoostingRegressor



а



б

а) – для параметра *n_estimators*; б) – для параметра *learning_rate*

Рисунок Б.3 – Криві перевірки GradientBoostingClassifier

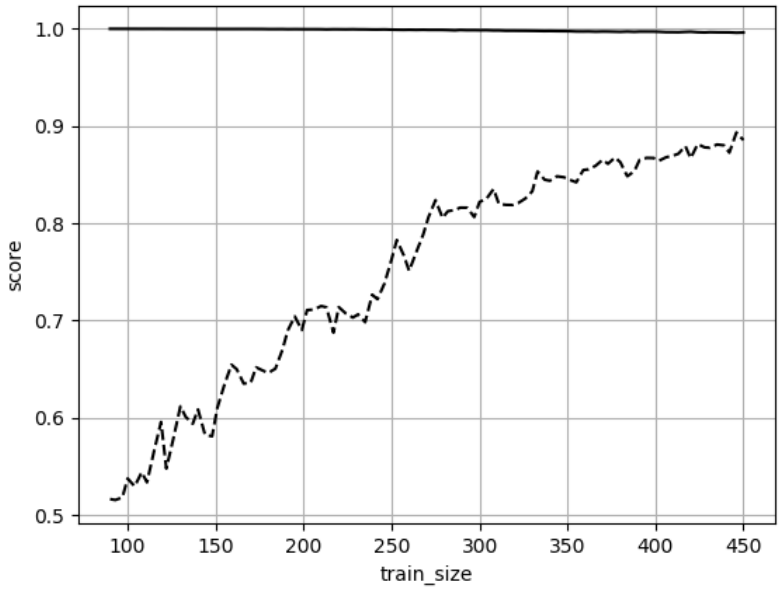
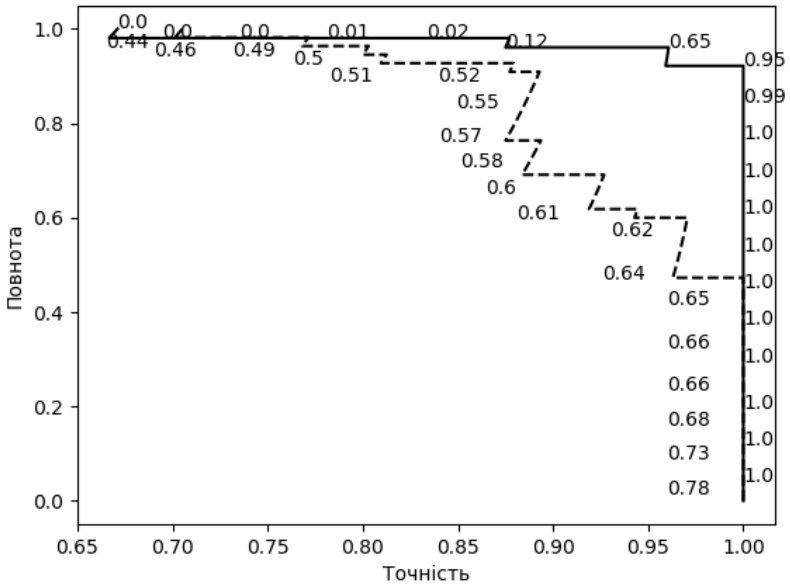


Рисунок Б.4 – Криві навчання GradientBoostingClassifier



(- -) – RandomForestClassifier; (—) – GradientBoostingClassifier

Рисунок Б.5 – Крива точність-повнота для моделей

ДОДАТОК В

Модель ШСНУ на основі абстрактних автоматів

Код програм також доступний в GitHub (vkopey/PU. URL: http://github.com/vkopey/PU).

Лістинг В.1 – PUmodel.py – модуль для підготовки параметрів моделей ШСНУ

```
# -*- coding: cp1251 -*-
import math
# Для моделювання порожнистої трубчатої штанги її замінюємо еквівалентною суцільною з такими ж: зовнішнім діаметром, пружністю, масою. Але тоді у неї буде інший модуль E і густина
def eqYoungsModulus(E,D,d):
    """Еквівалентний модуль Юнга суцільної штанги діаметром D
    з умови однаковості пружності k=E*A/L=Eeq*Aeq/L
    суцільної (діаметром D) і трубчатої штанги (зовн. діаметром D,
    внутрішнім d і модулем Юнга E)"""
    A=math.pi*(D**2.0-d**2.0)/4.0 # площа перетину труби
    Aeq=math.pi*(D**2.0)/4.0 # площа перетину суцільної штанги
    return E*A/Aeq

def eqDensity(Ro,D,d,Roo=0.0):
    """Еквівалентна густина суцільної штанги діаметром D
    з умови однаковості маси m=Ro*A*L+Roo*Ao*L=Roeq*Aeq*L
    суцільної (діаметром D) і трубчатої штанги (зовн. діаметром D,
    внутрішнім d, густиною тіла Ro і тіла отвору Roo)
    Якщо Roo=0 - порожнисті штанги наповнені повітрям"""
    A=math.pi*(D**2.0-d**2.0)/4.0 # площа перетину труби
    Ao=math.pi*(d**2.0)/4.0 # площа перетину отвору
    Aeq=math.pi*(D**2.0)/4.0 # площа перетину суцільної штанги
    return (Ro*A+Roo*Ao)/Aeq

class SuckerRod(object):
    """Клас насосних штанг або секції однотипних штанг. Одиниці СІ"""
    diametrs=[0.016,0.019,0.022,0.025] # діаметри тіла
    length=8.0 # довжина (за замовчуванням)
    materials={"сталь":{"модуль пружності":2.1e11,
                        "густина":8000.0}, # тут густина дещо завищена для врахування мас муфт і головок
               "склопластик":{"модуль пружності":0.5e11,
                              "густина":2100.0},
               "стальТр22Екв":{"модуль пружності":eqYoungsModulus(2.1e11,0.022,0.0085),

"густина":eqDensity(8000.0,0.022,0.0085,1000.0)},
```

```
            "склопластикТр22Екв":{"модуль
пружності":eqYoungsModulus(0.5e11,0.022,0.0085),

"густина":eqDensity(2100.0,0.022,0.0085,1000.0)}}
    #!!!якщо Roo=0, то порожнисті штанги є герметичними і наповнені
повітрям (але це недоцільно)
    alfa=0.0 # кут відхилення свердловини внизу секції від вертикалі (рад.)

    def __init__(self, PU, diametr, length=8.0, material="сталь",
alfa=0.0):
        """Конструктор"""
        self.PU=PU # свердловинна штангова насосна установка
        self.diametr=diametr # діаметр тіла штанги
        self.length=length # довжина штанги або секції
        self.material=material # матеріал
        self.alfa=alfa

    def area(self):
        """Площа поперечного перетину S=pi*r^2"""
        return math.pi*(self.diametr/2.0)**2

    def volume(self):
        """Об'єм, який займає штанга в рідині V=L*pi*r^2"""
        return self.length*math.pi*(self.diametr/2.0)**2

    def mass(self):
        """Маса m=V*ro"""
        return self.volume()*self.materials[self.material]["густина"]

    def fluidVolume(self):
        """Об'єм рідини в НКТ навколо штанги (секції)"""
        A=self.PU.well.sectionalArea()-self.area() # увага! тут площа НКТ,
а не плунжера
        return A*self.length

    def fluidMass(self):
        """Маса рідини в НКТ навколо штанги (секції)"""
        return self.PU.well.density*self.fluidVolume()

    def weight(self):
        """Вага в рідині"""
        fluidDensity=self.PU.well.density # густина рідини
        Farh=fluidDensity*9.81*self.volume() # сила Архімеда
        w=self.mass()*9.81 # вага в повітрі
        return w-Farh

    def stiffness(self):
        """Жорсткість k=E*S/L"""
        E=self.materials[self.material]["модуль пружності"]
        return E*self.area()/self.length
```

```
    def hydrodynamicRodResistance(self):
        """Гідродинамічний опір 'c' штанги (секції) за Пірвердяном
        Враховується під час ходу вниз. Сила опору F=c*v"""
        msh=self.PU.well.diametr/self.diametr
        Msh=1.0/((math.log(msh)*(msh**2+1)/(msh**2-1))-1) # коефіцієнт Мшт
        # v=2nS - швидкість штанг
        R=math.pi**2*self.PU.well.viscosity*self.PU.well.density*Msh
        return R*self.length

    def hydrodynamicRodResistance2(self):
        """Гідродинамічний опір 'c' штанги (секції) за Андреєвим
        Враховується під час ходу вниз. Сила опору F=c*v"""
        m=self.diametr/self.PU.well.diametr
        R=16.9*self.PU.well.viscosity*self.PU.well.density*m**5.49
        R=R*1000 # одиниці ?
        return R*self.length

    def materialRodResistance(self,sigmaA=None):
        """Еквівалентний коефіцієнт опору матеріалу, Н*с/м
        sigmaA - амплітуда напружень (Па)
        sigmaA=None - не залежить від ампл. напруж."""
        k=self.stiffness() # жорсткість стержня
        # визначаємо коефіцієнт поглинання матеріалу
        if self.material.startswith("сталь"):
            if sigmaA!=None: # якщо залежить від ампл. напруж.
                psi=6e-5*sigmaA/1000000.0
            else: # якщо не залежить від ампл. напруж.
                #psi<0.13 (метали в пружному діапазоні)
                #psi=0.25...0.5 (металічні структури)
                #psi=0.38...0.88 (металічні структури зі з'єднаннями)
                psi=0.01
        if self.material.startswith("склопластик"):
            if sigmaA!=None: # якщо залежить від ампл. напруж.
                psi=4e-4*sigmaA/1000000.0 # довідник "Вибрации в технике"
            else: # якщо не залежить від ампл. напруж.
                psi=0.08
        omega=self.PU.pumpingUnit.omega # кутова частота зовнішньої сили
        #m=self.mass() # маса стержня
        #omega=math.sqrt(k/m) # власна частота стержня
        #["Вибрации в технике" т6 с130 ф13,19]
        c=psi*k/(2*math.pi*omega) #k=E*S/L

#         #!уточнити врахування окремо опору з'єднань
#         nz=int(self.length/8) # кількість муфтових з'єднань на секції
#         kz=1000000.0 # жорсткість муфтового з'єднання ?
#         cz=0.2*kz/(2*math.pi*omega) # екв коеф опору муфтового з'єднання
(psi=0,006...0,4)
#         czs=1/(nz*(1/cz)) # екв коеф опору усіх з'єднань на секції
#         return 1/(1/c+1/czs) # екв коеф опору секції (матеріал+з'єднання)
        return 20.0*c # увага збільшено у 20.0 (для спрощ. моделі 50)
```

```
#################################################################
class SuckerRodString(object):
    """Клас колони насосних штанг"""
    items=[] # список штанг (секцій) [штанга1, штанга2,...](починаючи з
гирла)

    def __init__(self, PU, sections):
        """Конструктор.
        PU - ШСНУ
        sections=[(діаметр, довжина, матеріал, кут відхилення від
вертикалі(рад.),...]"""
        self.PU=PU
        for diametr, length, material, alfa in sections: # для кожної
секції
            # додати штангу (секцію однотипних штанг)
            self.items.append(SuckerRod(PU, diametr, length, material,
alfa))

    def length(self,start=0,end=None):
        """Довжина. start, end - індекси зрізу"""
        return sum([r.length for r in self.items[start:end]])

    def xList(self):
        """Список координат нижніх кінців секцій"""
        X=[]
        x=0.0 # координата
        for r in self.items:
            x=x+r.length # додати довжину даної секції
            X.append(x)
        return X

    def height(self):
        """Висота проекції колони штанг на вертикаль"""
        return sum([r.length*math.cos(r.alfa) for r in self.items])

    def volume(self,start=0,end=None):
        return sum([r.volume() for r in self.items[start:end]])

    def stiffness(self):
        """Сумарний коеф. жорсткості колони 1/k=1/k1+1/k2+..."""
        return 1/sum([1/r.stiffness() for r in self.items])

    def damping(self):
        """Сумарний коеф. демпфування колони 1/d=1/d1+1/d2+...
        [Вибрации в технике т6 с177]"""
        return 1/sum([1/r.materialRodResistance() for r in self.items])

    def mass(self,start=0,end=None):
        return sum([r.mass() for r in self.items[start:end]])

    def massList(self):
```

```
        """Список мас секцій"""
        return [r.mass() for r in self.items]

    def fluidMassList(self):
        """Список мас рідини біля вузлів"""
        return [r.fluidMass() for r in self.items]

    def weight(self,start=0,end=None):
        return sum([r.weight() for r in self.items[start:end]])

    def wellAngleList(self):
        """Список кутів свердловини внизу секцій"""
        return [x.alfa for x in self.items]

    def secAngleList(self):
        """Повертає список кутів секцій an_i
        за кутами свердловини внизу секції alfa_i.
             an0          an1          an2
        ------------O------------O------------O
                  alfa0        alfa1        alfa2
        Розраховує приблизно за умови рівності довжин секцій.
        """
        angles=[self.items[0].alfa] # кут верхньої секції=куту верхньої
ділянки свердловини
        aprev=self.items[0].alfa # кут першої ділянки свердловини (вверху).
Як правило 0.
        for r in self.items[1:]: # для кожної ділянки крім першої
            a=(aprev+r.alfa)/2 # розрахувати наближено кут
            angles.append(a) # додати в список
            aprev=r.alfa # запам'ятати як попередню секцію
        return angles

    def angleTop(self):
        """Кути між верхньою штангою і свердловиною"""
        AN=self.wellAngleList()
        ANS=self.secAngleList()
        return [ans-an for an,ans in zip(AN,ANS)]

    def angleBottom(self):
        """Кути між свердловиною і нижньою штангою
        Останнє значення 0"""
        AN=self.wellAngleList()
        ANS=self.secAngleList()
        return [an-ans for an,ans in zip(AN[:-1],ANS[1:])]+[0.0]

    def hydrodynamicRodResistanceList(self):
        """Список гідродинамічних опорів секцій"""
        return [r.hydrodynamicRodResistance() for r in self.items]

    def weightForceList(self):
        """Список осьових сил від ваги секцій
```

```
        (з врахуванням кута свердловини)"""
        return [r.weight()*math.cos(r.alfa) for r in self.items]

class Well(object):
    """Клас свердловини"""
    diameters=[60.3-2*5.0, 73.0-2*5.5, 73.0-2*7.0, 88.9-2*6.5, 114.3-2*7.0]
    diametr=diameters[4]/1000.0 #4 діаметр НКТ (однаковий на інтервалах)
    density=1000.0 #1000.0 густина суміші [Белов]
    viscosity=0.000002 #0.000001 # кінематична в'язкість (змінна по
висоті!!!)
    bulkModulus=2.2e9 # об'ємний модуль пружності

    def sectionalArea(self):
        """Площа поперечного перетину НКТ"""
        return math.pi*(self.diametr/2.0)**2

class Pump(object):
    """Клас свердловинного насоса"""
    diameters=[27, 32, 38, 44, 50, 57, 57, 63, 70, 95]
    diametr=38/1000.0 #38 діаметр плунжера
    valveDiametr=0.025 # діаметр отвору сідла клапана
    openValveResistance=0.00001 # опір відкритого клапана ???
    pistonGap=0.1e-3 #0.1e-3 зазор між плунжером і циліндром
    # далі v - швидкість плунжера, vv - швидкість в сідлі клапана  !!!

    def __init__(self,PU):
        self.PU=PU # ШСНУ

    def volumeFlowRate(self,v):
        """Теоретична подача насоса (об'ємна витрата) Q=A*v"""
        return self.pistonArea()*v #self.PU.pumpingUnit.velAvg()

    def pistonArea(self):
        """Площа поперечного перетину плунжера"""
        return math.pi*(self.diametr/2.0)**2 # !!! приблизно

    def valveDischargeCoefficient(self,Re):
        """Коефіцієнт витрати клапана. Залежить від числа Рейнольдса.
[Гіматудінов]"""
        if Re<=225:
            mu=0.0846*Re**0.2872
        elif Re<=30000:
            mu=0.4
        elif Re<=300000:
            mu=0.0085*Re**0.3764
        else: mu=1.0
        return mu

    def valveLossCoefficient(self,Re):
        """Коефіцієнт витрат клапана (місцевого опору)"""
        return 1.0/self.valveDischargeCoefficient(Re)**2.0
```

```
    def valveV(self,v):
        """Швидкість потоку в сідлі, v - швидкість плунжера"""
        return self.volumeFlowRate(v)/(math.pi*(self.valveDiametr/2.0)**2)

    def valveRe(self,vv):
        """Число Рейнольдса, vv - швидкість потоку в сідлі"""
        nu=self.PU.well.viscosity
        Re=abs(vv)*self.valveDiametr/nu
        return Re

    def dynamicPressure(self, vv):
        """Динамічний тиск, vv - швидкість потоку в сідлі"""
        ro=self.PU.well.density
        return vv*vv*ro/2

    def dPvalve(self,vv):
        """Перепад тисків на клапані [Гіматудінов]
        vv - швидкість потоку в сідлі"""
        Re=self.valveRe(vv)
        dP=self.valveLossCoefficient(Re)*self.dynamicPressure(vv)
        return dP

    def valveHydroResistForce(self,v):
        """Сила гідродинамічного опору клапана насоса.
        Діє під час ходу вниз і вверх. Направлена проти руху
        v - швидкість плунжера"""
        vv=self.valveV(v)
        return self.dPvalve(vv)*self.pistonArea() # повертає завжди додатне

    def pistonFrictionForce(self):
        """Сила тертя плунжера об стінки циліндра, Н.
        Формула Сердюка. Орієнтовно 1000 Н
        Діє під час ходу вниз і вверх. Направлена проти руху"""
        return 1.84*self.diametr/self.pistonGap-137 # для змащування водою

class PumpingUnit():
    """Клас верстата-гойдалки"""
    stroke=3.0 #2.0 # довжина ходу точки підвіски
    n=6.5 #10.0 # кількість гойдань (подвійних ходів) за хвилину
    fi=math.pi # початковий кут (почати з верхньої точки або з середини,
якщо колона стабілізована)
    # ланки механізму (довжина, маса, момент інерції відносно осі z, яка
проходить через центр мас)
    # Тип: СКД8-3-4000 (дезаксіальний)
    p1=dict(L=1.29,m=1982.0,Iz=635.0) # два кривошипи
    p2=dict(L=3.0,m=499.0,Iz=728.0) # два шатуна і траверса
    p3=dict(L=2.0,m=643.0,Iz=443.0) # заднє плече балансира
    p4=dict(L=2.29,m=911.0,Iz=1106.0) # переднє плече балансира
    p5=dict(L=0.789,m=4*750.0,Iz=0.0) # противаги (L - радіус центра мас)
Приймаємо, що маса зосереджена в точці (Iz=0.0)
```

```
    # Увага! Значення за замовчуванням p5 уточнити функцією balancing
    dx=1.345 # горизонтальна відстань між осями опори балансира і
тихохідного валу редуктора
    dy=3.012 # 3.0, 3.035 ? # вертикальна відстань між осями
(розраховується)
    r=0.84 # радіус кривошипа (залежить від ходу)
    # примітка до табл. на с.42 [Архіпов] ?

    def __init__(self):
        self.A=0.5*self.stroke # амплітуда
        self.ns=self.n/60.0 # частота, Гц
        self.omega=2*math.pi*self.ns # кутова частота, рад/с

    def velAvg(self):
        """Середня швидкість, м/с"""
        return 2*self.stroke*self.n/60.0

    def motionX(self, t):
        """Описує закон руху точки підвіски.
        Повертає координату X точки підвіски в момент часу t"""
        #!уточнити
        #v=self.A*self.omega*math.cos(self.omega*t+self.fi) # швидкість
        x=self.A*math.sin(self.omega*t+self.fi) # переміщення
        return x

class PU():
    """Клас свердловинної штангової насосної установки"""
    suckerRodString=None
    well=None
    pump=None
    outPres=100000.0 # 100000.0 тиск на гирлі в НКТ
    outTubePres=500000.0 # 500000.0 тиск в затрубному просторі
    heightDynamic = None # глибина динамічного рівня (якщо None, то за
замовчуванням)
    # зменшення heightDynamic зменшує тільки Fmax (див. Мищенко ф-ла 9.90)
    timeEnd=20.0 # кінець часу
    timeStep=0.1 # 0.01 крок часу

    def __init__(self):
        """Конструктор"""
        h=1500.0 # довжина колони
        sections=[(0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0),
                  (0.019, h/8.0, "сталь", 0.0)] # 0.2
        self.suckerRodString=SuckerRodString(PU=self,sections=sections) #
колона штанг
```

```
        self.well=Well() # свердловина
        self.pump=Pump(PU=self) # насос
        self.pumpingUnit=PumpingUnit() # верстат-гойдалка
        if self.heightDynamic==None:
            self.heightDynamic=self.suckerRodString.height()-100.0 # -100
орієнтовно
        r,m=self.balancing() # радіус і маса противаг
        self.pumpingUnit.p5['L']=r
        self.pumpingUnit.p5['m']=m

    def fluidVolume(self):
        """Об'єм стовпа рідини над плунжером V=H*S"""
        return self.suckerRodString.height()*self.pump.pistonArea()

    def fluidWeight(self): # не враховується інерція рідини
        """Вага рідини над плунжером w=V*ro*g.
        Діє на плунжер під час ходу вверх. Направлена вниз"""
        return self.fluidVolume()*self.well.density*9.81

    def outPresForce(self):
        """Навантаження від тиску на гирлі.
        Діє на плунжер під час ходу вверх. Направлене вниз"""
        return self.pump.pistonArea()*self.outPres

    def outTubePresForce(self):
        """Навантаження від тиску в затрубному просторі.
        Діє на плунжер під час ходу вверх. Направлене вверх"""
        return self.pump.pistonArea()*self.outTubePres

    def outTubeFluidWeight(self):
        """Вага рідини в затрубному просторі.
        Діє на плунжер під час ходу вверх. Направлена вверх"""
        v=self.pump.pistonArea()*(self.suckerRodString.height()-
self.heightDynamic) # обєм
        return v*self.well.density*9.81

    def tubePressureFrictLossForse(self,v):
        """Навантаження від втрати тиску від тертя потоку по довжині НКТ
        Діє на плунжер під час ходу вверх. Направлене вниз. Не для моделі
Maplesim"""
        v=self.pump.volumeFlowRate(v)/self.well.sectionalArea() # середня
швидкість в НКТ
        D=self.well.diametr
        L=self.suckerRodString.length()
        nu=self.well.viscosity
        ro=self.well.density
        Re=abs(v)*D/nu # число Рейнольдса
        fL=64/Re # для ламінарного
        fT=0.316/math.pow(Re, 1.0/4) # для турбулентного (формула Блазіуса)
        if Re<=2000: # якщо ламінарний режим
            f=fL
```

```
        elif Re>=4000: # якщо турбулентний режим
            f=fT
        else: # в іншому випадку
            f=fL+(fT-fL)*(Re-2000)/2000
        dP=(f*L/D)*v*v*ro/2 # втрати тиску
        return self.pump.pistonArea()*dP # завжди додатна

    def polishedRodForce(self):
        """Повертає кортеж (Fmin,Fmax) розрахований за наближеними
формулами
        Ці формули вірні, коли динамічний рівень = довжині спуску
        Тобто розраховується найбільш небезпечний випадок"""
        F1=self.suckerRodString.weight() # вага штанг в рідині
        F2=self.fluidWeight() # вага рідини
        k=(self.pumpingUnit.stroke*self.pumpingUnit.n**2)/1790
        Fmin=F1*(1-k) # формула Міллса
        Fmax=(F1+F2)*(1+k) # формула Кемлера
        return (Fmin,Fmax)

    def sigmaPr(self,d):
        """Приведене напруження [Круман]"""
        A=math.pi*(d/2.0)**2 # площа перетину верхньої штанги
        Fmin,Fmax=self.polishedRodForce()
        return (Fmax-0.5625*Fmin)/A

    def polishedRodForce2(self):
        """Повертає кортеж (Fmin,Fmax) розрахований за дуже наближеними
формулами"""
        F1=self.suckerRodString.weight() # вага штанг в рідині
        F2=self.fluidWeight() # вага рідини
        Fmin=F1
        Fmax=F1+F2
        return (Fmin,Fmax)

    def balancing(self,Pmin=None,Pmax=None):
        """Параметри орієнтовного урівноважування [Архіпов]. Для верстатів
типу СК і СКД. Повертає радіус і сумарну масу противаг"""
        if Pmin==None and Pmax==None: # якщо не задано
            Pmin,Pmax=self.polishedRodForce() # розрахувати за наближеними
формулами
        Mp=750.0 # маса 1 противаги
        A0=52000.0 # робота сил ваги неурівноважених частин верстата
        S=self.pumpingUnit.stroke
        K_list=[2,4,6,8] # список кількості противаг
        A=Pmin*S/2+Pmax*S/2 # робота сил, що урівноважуються
        R_list=[(A-A0)/(2*9.81*Mp*k) for k in K_list] # список радіусів
противаг
        #print "Rp=", R_list
        Rmax=self.pumpingUnit.p1['L']-0.2 # максимальний радіус противаги
?уточнити
        for i,r in enumerate(R_list):
```

```
            if r<Rmax: break # вибирають радіус з мінімальною кількістю
противаг
        return r, K_list[i]*Mp

    def info(self): # друкує інформацію
        s=""
        s+="вага штанг в рідині %d H\n"%self.suckerRodString.weight()
        s+="вага рідини %d H\n"%self.fluidWeight()
        s+="дуже наближено Fmin=%d Fmax=%d\n"%self.polishedRodForce2()
        s+="наближено Fmin=%d Fmax=%d\n"%self.polishedRodForce()
        s+="приведене напруження
%d\n"%self.sigmaPr(d=self.suckerRodString.items[0].diametr)
        s+="діаметр плунжера %f\n"%self.pump.diametr
        s+="довжина колони %d\n"%self.suckerRodString.length()
        s+="жорсткість колони %d Н/м\n"%self.suckerRodString.stiffness()
        s+="маса колони %d кг\n"%self.suckerRodString.mass()

n0=math.sqrt(self.suckerRodString.stiffness()/self.suckerRodString.mass())
# власна частота стержня
        s+="власна частота колони %f рад/с\n"%n0
        s+="динамічний рівень %d м\n"%self.heightDynamic
        s+="вага рідини в затр просторі %f H\n"%self.outTubeFluidWeight()
        s+="еквівалентний коефіцієнт опору матеріалу %f
\n"%self.suckerRodString.damping()
        s+="еквівалентний коефіцієнт опору матеріалу секції 1 %f
\n"%self.suckerRodString.items[0].materialRodResistance()
        # зі збільшенням довжини секції прямує до 0 (як і жорсткість)
        s+="жорсткість секції 1 %f
\n"%self.suckerRodString.items[0].stiffness()
        s+="маса секції 1 %f \n"%self.suckerRodString.items[0].mass()
        print s

if __name__ == '__main__':
    pu=PU()
    pu.info()
```

Лістинг В.2 – CAmodel.py – модель ШСНУ на основі абстрактних автоматів

```
# -*- coding: CP1251 -*-
import math, os
import PUmodel
import matplotlib.pyplot as plt
from matplotlib import rc
rc('font', family='Arial') # для підтримки Юнікоду

class Automaton(object):
    """Абстрактний автомат для моделювання руху вузла пружного стержня"""
    def __init__(self):
        """Конструктор автомата"""
        self.x=None # координата
```

```
        self.x0=None # початкова координата
        self.prevx=None # попередня координата
        self.bc=None # гранична умова (0 - нерухомий)
        self.f=0.0 # зовнішня сила
        self.fs=0.0 # вага секції знизу
        self.left=None # верхній сусід
        self.right=None # нижній сусід
        self.delta=None # початкова відстань до нижнього сусіда
(rSection.length)
        self.k=None # коефіцієнт жорсткості секції знизу
(rSection.stiffness())
        self.c=0.0 # коефіцієнт опору матеріалу секції знизу
        self.m=1.0 # маса секції знизу
        self.a=0.0 # прискорення
        self.v=0.0 # швидкість
        self.prevv=0.0 # попередня швидкість
        self.eps=0.0 # нев'язка
        self.dx=1.0 # крок наближення
        self.area=None # площа поперечного перетину секції знизу
(rSection.area())
        self.domain=None # об'єкт класу CA_model
        self.rSection=None # секція знизу
        self.alfa=0.0 # кут відхилення свердловини від вертикалі біля
автомата

    def rule(self):
        """Правило поведінки автомата"""

        if self.bc!=None: # якщо задана гранична умова
            self.x=self.bc # визначити x

        if self.domain.t!=0.0: # якщо час не 0
            self.v=self.velocity() # розрахувати швидкість
            self.a=self.acceleration() # розрахувати прискорення
        else: self.v=self.a=0.0

        if self.bc!=None: # якщо задана гранична умова
            return # вийти (не потрібно знаходити x)

        Fk=self.rightForce()-self.leftForce() # сили пружності
        Fv=self.rightForce2()-self.leftForce2() # сили демпфування
        Fa=self.dynamicForce() # сила інерції
        # зовнішні сили

F=self.fs+self.f+self.pistonForce()+self.frictionForce()+self.hydrodynamicR
esistanceForce()

        # рівновага усіх сил біля автомата: m*a+c*v+k*u=F
        # якщо сила>0, то вона направлена вниз
        # Напрямок осі x:
        # [0]---[1]---[2]--->+x,+v,+a,+f,нижній
```

```
        # обчислити нев'язку
        self.eps=Fa+Fv+Fk-F
        #print "eps=",self.eps

        # якщо нев'язка відрізняється від 0
        if self.eps>0:
            self.x=self.x-self.dx # збільшити x на малу величину
        if self.eps<0:
            self.x=self.x+self.dx # зменшити x на малу величину

    def velocity(self):
        return (self.x-self.prevx)/self.domain.dt # швидкість

    def acceleration(self):
        return (self.v-self.prevv)/self.domain.dt # прискорення

    def leftForce(self):
        """Повертає значення сили пружності секції зверху"""
        if self.left: # якщо є лівий сусід
            dLeftBegin=self.left.delta # початкова довжина пружини зверху
            dLeft=self.x-self.left.x # довжина пружини зверху
            fLeft=(dLeftBegin-dLeft)*self.left.k # сила пружності
        else:
            fLeft=0.0
        return fLeft

    def leftForce2(self):
        """Повертає значення сили демпфування секції зверху"""
        if self.left: # якщо є верхній сусід
            vrel=self.v-self.left.v # відносна швидкість
            fLeft=-self.left.rSection.materialRodResistance(None)*vrel
        else:
            fLeft=0.0
        return fLeft

    def rightForce(self):
        """Повертає значення сили пружності секції знизу"""
        if self.right: # якщо є нижній сусід
            dRightBegin=self.delta # початкова довжина пружини знизу
            dRight=self.right.x-self.x # довжина пружини знизу
            fRight=(dRightBegin-dRight)*self.k # сила пружності
        else:
            fRight=0.0
        return fRight

    def rightForce2(self):
        """Повертає значення сили демпфування секції знизу"""
        if self.right: # якщо є нижній сусід
            vrel=self.right.v-self.v # відносна швидкість
            fRight=-self.rSection.materialRodResistance(None)*vrel
        else:
```

```
            fRight=0.0
        return fRight

    def dynamicForce(self):
        """Сила інерції секції F=m*a"""
        return self.m*self.a

    def dynamicFluidForce(self):
        """Сила інерції рідини. Діє на плунжер"""

k=self.domain.PU.pump.pistonArea()/self.domain.PU.well.sectionalArea()
        a=self.a*k # прискорення рідини менше прискорення плунжера у k раз
        return -a*sum(self.domain.PU.suckerRodString.fluidMassList())

    def normalForce(self):
        """Сума нормальних сил у вузлі від ваги, верхнього і нижнього
натягу"""
        F1=self.f*math.tan(self.alfa) # нормальна сила від ваги (f - осьова
сила від ваги)
        F2=F3=0.0
        if self.left:
            F2=self.leftForce()*math.sin(self.an1) # нормальна сила від
верхнього натягу
        if self.right:
            F3=self.rightForce()*math.sin(self.an2) # нормальна сила від
нижнього натягу
        return F1+abs(F2+F3) # якщо нехтувати зазором між стінкою НКТ і
вузлом

    def frictionForce(self):
        """Сила тертя секції"""
        # сила тертя на інтервалі
        #Ft=C*(Psh*math.sin(a1)-F12*math.sin(a12-a1)) #
        #Ft=C*(Psh*math.sin(a1)+2*F12*math.tan(a2/2-a1/2)) # a12-a1=a2/2-
a1/2
        #Ft=C*(Psh*math.sin(a1)+F12*(a2-a1)) # формула Песляка

        v=self.v # швидкість
        Fn=self.normalForce() # нормальна сила # для перевірки напрямку
введіть тут 1000.0
        fc=0.16 #0.25 # коефіцієнт тертя ковзання (сталь-сталь зі
змащуванням)
        #F=-fc*abs(Fn)*math.tanh(v/0.01) # спрощена модель

        Fc=abs(Fn)*fc # сила тертя Кулона
        fmax=0.2 # коефіцієнт тертя ковзання (сталь-сталь з малим
змащуванням)
        Fmax=abs(Fn)*fmax # максимальна сила тертя ковзання
        vs=0.1 # коеф. швидкості ковзання Штрибека
        n=1.0 # степінь згасання (Decay exponent)
        # сумарна сила тертя
```

```
        F=-math.tanh(v/0.01)*(Fc+(Fmax-Fc)*math.exp(-(abs(v)/vs)**n))
        return F

    def hydrodynamicResistanceForce(self):
        """Сила гідродинамічного опору секції"""
        v=self.v # швидкість
        if v<=0: # якщо хід вверх

k=self.domain.PU.pump.pistonArea()/self.domain.PU.well.sectionalArea()
            vu=v-v*k # швидкість штанг відносно рідини
            F=-self.c*vu*10.0 #увага!!! збільшено
            # F=0.0 # якщо швидкості штанг і рідини рівні
        else: # якщо хід вниз
            F=-self.c*v*10.0  #увага!!! збільшено
        return F*math.tanh(abs(v)/0.01) # згладжуємо біля 0

    def pistonForce(self):
        """Сума зовнішніх сил на плунжері насоса"""
        v=self.v # швидкість
        F=0.0 # для усіх вузлів крім останнього
        if self.right!=None: return F # якщо не останній вузол (насос)

        if v<0: # якщо рух плунжера вгору
            # направлені вниз:
            F+=self.domain.PU.fluidWeight()
            F+=self.domain.PU.outPresForce()
            F+=abs(self.domain.PU.tubePressureFrictLossForse(v)) # тут abs
            F+=self.dynamicFluidForce()
            # направлені вверх:
            F+=-self.domain.PU.outTubePresForce()
            F+=-self.domain.PU.outTubeFluidWeight()
        if v!=0:
            # наступні сили діють під час РУХУ (!) вгору і вниз
            # і направлені проти руху плунжера
            F+=-math.copysign(1,
v)*abs(self.domain.PU.pump.valveHydroResistForce(v))
            F+=-math.copysign(1,
v)*self.domain.PU.pump.pistonFrictionForce() # модель тертя плунжера

        F=F*math.tanh(abs(v)/0.01) # ф-ція згладжування біля 0 (F=max, якщо
v=0.01)
        #або F=F*(1.0-math.exp(v/0.01))
        return F

    def rightStress(self):
        """Повертає значення напруження в пружині справа (внизу)"""
        if self.right:
            return self.rightForce()/self.area
        else:
            return 0.0
```

```
class CA_model(object):
    """Система автоматів (пружний стержень)"""
    def __init__(self,PU):
        """Створює систему автоматів за об'єктом класу PU (ШСНУ)
        Схема системи автоматів:
                секція1                      секція2
        O--------------------O-----------------------------------O
        m0=0      d0         m1                d1                m2
        c0=0      k0         c1                k1                c2
        f0        a0         f1                a1                f2
       alfa0=0   ans0       alfa1             ans1              alfa2
        Перший автомат (0) допоміжний.
        """

        self.PU=PU # ШСНУ
        rs=PU.suckerRodString.items # список секцій
        n=len(rs)+1 # кількість вузлів (автоматів)
        # Перший автомат (0) допоміжний.
        X=[0.0]+PU.suckerRodString.xList() # список координат вузлів
        M=[0.0]+PU.suckerRodString.massList() # маса вузлів
        Fs=[0.0]+PU.suckerRodString.weightForceList() # список сил від ваги
секцій
        C=[0.0]+PU.suckerRodString.hydrodynamicRodResistanceList() #
гідродинамічні опори секцій

        AN=[0.0]+PU.suckerRodString.wellAngleList()
        #ANS=PU.suckerRodString.secAngleList()
        AN1=[None]+pu.suckerRodString.angleTop()
        AN2=[0.0]+pu.suckerRodString.angleBottom()

        self.dt=self.PU.timeStep # крок часу
        self.T=self.timeList(PU.timeEnd, PU.timeStep) # список значень часу
[t0,t1,t2,...]
        self.t=self.T[0] # поточне значення часу (0.0)

        BCdict={} # словник граничних умов (див. self.BCList)
        # закон руху точки підвіски
        BCdict[0]=[PU.pumpingUnit.motionX(t) for t in self.T]
        self.BC=CA_model.BCList(BCdict, n, self.T, default=None) # список
історії граничних умов [[bc0,bc1,bc2,...],...]

        F=[0.0]*n # список інших зовнішніх сил (якщо потрібно)
        Fdict=dict(zip(range(n),F)) # словник сил (див. self.BCList)
        self.F=CA_model.BCList(Fdict, n, self.T, default=0.0) # список
історії зовнішніх сил [[f0,f1,f2,...],...]

        self.p=[Automaton() for i in range(n)] # список автоматів

        # для кожного автомата, крім останнього, визначити властивості
        for i in range(n-1):
            self.p[i].rSection=rs[i]
```

```
            self.p[i].k=rs[i].stiffness()
            self.p[i].delta=rs[i].length
            self.p[i].area=rs[i].area()

        # задаємо інші властивості усіх автоматів
        for i in range(n):
            self.p[i].bc=self.BC[0][i]
            self.p[i].f=self.F[0][i]
            self.p[i].fs=Fs[i]
            self.p[i].m=M[i]
            self.p[i].c=C[i]
            self.p[i].alfa=AN[i]
            self.p[i].x=X[i]
            self.p[i].x0=X[i]
            self.p[i].domain=self
            self.p[i].an1=AN1[i]
            self.p[i].an2=AN2[i]

        for i in range(n): # задаємо сусідів
            if i!=0: # якщо не перший
                self.p[i].left=self.p[i-1] # визначити верхнього сусіда
            if i!=len(self.p)-1: # якщо не останній
                self.p[i].right=self.p[i+1] # визначити нижнього сусіда

        # словник історій результатів
        # наприклад, список історії значень x [[x0,x1,x2,...],...]

self.history={'T':self.T,'X':[],'V':[],'A':[],'Fl':[],'Fr':[],'Sr':[]}

    def run(self):
        """Еволюція системи автоматів в конкретний момент часу. Одна з
можливих реалізацій. Повільна, але не потребує сторонніх алгоритмів."""
        eps=0.01*len(self.p) # похибка розрахунку
        e=eps+1 # сума нев'язок
        eprev=eps+1 # попередня сума нев'язок
        # початковий крок наближення (залежить від delta)
        dx=min([a.delta for a in self.p if a.delta!=None])/10.0 # !!!
Змінювати в межах 10-100
        niter=0 # кількість ітерацій (для оптимізації алгоритму)
        while e>eps: # поки сума нев'язок > eps
            for a in self.p: # для кожного автомата
                a.dx=dx # змінити крок наближення
                a.rule() # застосувати правило поведінки автомата
            e=sum([abs(a.eps) for a in self.p]) # сума нев'язок
            #print e
            if e>=eprev: # якщо сума нев'язок >= попередньої
                dx=dx*0.935 # зменшити крок !!! Змінювати в межах 0.1-0.99
                #print "dx=",dx
            eprev=e # запам'ятати суму нев'язок
            niter=niter+1
        #print niter
```

```
#         a=self.p[1]
#         print 'lf=',a.leftForce()
#         print 'rf=',a.rightForce()
#         print 'lf2=',a.leftForce2()
#         print 'lvrel=',a.v-a.left.v
#         print 'rf2=',a.rightForce2()
#         print 'rvrel=',a.right.v-a.v

    def runDynamic(self):
        """Динамічна еволюція системи автоматів"""
        # початкові значення координат, швидкостей і прискорень
        for a in self.p: # для кожного автомата
            a.prevx=a.x # попередня координата
            a.prevv=0.0 # попередня швидкість
        for i,t in enumerate(self.T): # для кожного значення часу
            self.t=t # поточне значення часу
            print "solve t=",t
            for a,f,bc in zip(self.p, self.F[i], self.BC[i]): # для кожного
автомата
                a.f=f # визначити силу в момент часу t
                a.bc=bc # визначити граничну умову в момент часу t
            self.run() # знайти урівноважений стан
            for a in self.p: # для кожного автомата
                a.prevx=a.x # запам'ятати поточні x та v
                a.prevv=a.v
            self.appendHistory()# запам'ятати історії
        self.writeHistory()

    def appendHistory(self):
        """Додає в списки історії список значень для поточного часу"""
        self.history['X'].append([a.x for a in self.p])
        self.history['V'].append([a.v for a in self.p])
        self.history['A'].append([a.a for a in self.p])
        self.history['Fl'].append([a.leftForce() for a in self.p])
        self.history['Fr'].append([a.rightForce() for a in self.p])
        self.history['Sr'].append([a.rightStress() for a in self.p])

    def draw(self):
        """Рисує положення автоматів псевдографікою (в текстовому
режимі)"""
        s=""
        prev=0
        for a in self.p:
            s+="-"*int(round(a.x-prev)-1)+"O"
            prev=a.x # координата попереднього автомата
        print s
        #time.sleep(1)

    def drawPlot(self, itemIndexes, key, x0scale=1, toFile=True):
        """Рисує залежність величини x[i] від t, де i - індекс автомата
```

```
        itemIndexes - список індексів автоматів
        key - ключ історії x
        x0scale - масштаб відстаней між першими точками кривих по осі X
(1...0):
                  1 - реальний, 0 - відстані рівні 0.
                  Крива для 0-го вузла не зміщується.
                  Використ. для покращення візуалізації переміщень вузлів,
                  якщо вони малі відносно відстаней між вузлами"""
        plt.figure()
        plt.gca().invert_yaxis() # обернути вісь y
        T=self.history['T'] # список значень часу
        x00=self.history[key][0][0] # x 0-го вузла для t=0
        for i in itemIndexes: # для кожного індексу
            x0i=self.history[key][0][i] # x i-го вузла для t=0
            x0=(1-x0scale)*(x0i-x00) # величина зміщення по осі X
            X=[x[i]-x0 for x in self.history[key]] # спис. істор. вузла "i"
            plt.plot(T, X, 'k-') # (T, X, 'r-', T, X,'ro')
        plt.grid(True)
        plt.xlabel(u't, c') # надпис осі x
        plt.ylabel(key) # надпис осі y
        if toFile:
            fileName='CAmodelPlot{0}{1}.png'.format(itemIndexes,key)
            plt.savefig(fileName)
        else:
            plt.show()

    def drawDynamometerCard(self, itemIndexes, keyX, keyY, signY=-1,
toFile=True, tstart=None, tend=None):
        """Рисує динамограму - залежність Y(X), наприклад,
сила(переміщення)"""
        plt.figure()
        plt.gca().invert_xaxis() # обернути вісь x
        plt.grid(True)
        plt.xlabel(u"S, м") #надпис осі x
        plt.ylabel(u"F, Н") #надпис осі y
        for i in itemIndexes: # для кожного вузла
            # індекси початкового і кінцевого часу в історії
            tstarti=self.history['T'].index(tstart) if tstart in self.T
else None
            tendi=self.history['T'].index(tend)+1 if tend in self.T else
None
            X=[x[i] for x in self.history[keyX][tstarti:tendi]] # список
історії X вузла "i"
            x0=self.history[keyX][0][i] # початкове значення
            X=[x-x0 for x in X] # відносно початкової координати
            Y=[signY*y[i] for y in self.history[keyY][tstarti:tendi]] #
список історії Y вузла "i"
            plt.plot(X, Y, 'k-')
        if toFile:
            fileName='CAmodelDynCard{0}{1}.png'.format(keyX,keyY)
            plt.savefig(fileName)
```

```
        else:
            plt.show()

    def writeHistory(self):
        """Зберігає історію в бінарний файл"""
        import pickle
        f = open('CAmodelHistory.pkl', 'wb')
        pickle.dump(self.history, f) # зберегти історію
        f.close()

    def readHistory(self):
        """Читає історію з бінарного файлу"""
        import pickle
        fileName='CAmodelHistory.pkl'
        if not os.path.isfile(fileName): # якщо файлу немає
            print "No file "+fileName
            return False # вийти, повернувши False
        f = open(fileName, 'rb')
        self.history=pickle.load(f) # завантажити історію
        f.close()
        print "Warning! Old results"
        return True

    def writeCSV(self,itemIndexes,key):
        """Записує значення історій у файл CSV.
        itemIndexes - список індексів автоматів
        key - ключ історії"""
        import csv
        csv_file=open("RodHistory_"+key+".csv", "wb")
        writer = csv.writer(csv_file,delimiter = ';')
        for time,data in zip(self.T,self.history[key]):
            X=[data[i] for i in itemIndexes] # лише дані за вказаними
індексами
            writer.writerow([time]+X) # записати рядок
        csv_file.close()

    def info(self):
        """Виводить значення усіх атрибутів автоматів"""
        attrs=["rSection","k","delta","area","bc","x","prevx","v","prevv",\
               "a","m","alfa","an1","an2","c","eps","f"]
        for attr in attrs:
            print attr+"=", [a.__getattribute__(attr) for a in self.p]
        print "leftForce=", [a.leftForce() for a in self.p]
        print "rightForce=", [a.rightForce() for a in self.p]


    @staticmethod
    def timeList(timeEnd,timeStep):
        """Повертає список значень часу"""
        nTimeSteps=int(timeEnd/timeStep)
        return [x*timeStep for x in range(nTimeSteps+1)]
```

```
    @staticmethod
    def BCList(BC, n, timeList, default=None):
        """
        Повертає список граничних умов (або сил) для автоматів 1,2,... у
вигляді:
        [[bc0,bc1,...],...],
        де перший список відповідає першому значенню часу.

        BC - словник граничних умов, де ключі - індекси КА, а значеннями
мужуть бути:
        1.Дійсні (постійне значення): 0.0
        2.Словники пар час-значення: {0.0:1.0,0.05:2.0}
        3.Списки історії значень:
[1.0,2.0,3.0,4.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0]
             або: [math.sin(x) for x in range(nt)]
        n - кількість автоматів
        timeList - список значень часу
        default - значення списків за замовчуванням"""

        nt=len(timeList) # кількість значень часу
        res=[] # результат
        for i in range(nt):
            res.append([default]*n) #  заповнюємо res різними(!) списками

        for k in BC: # для кожного заданого вузла
            if type(BC[k]).__name__=='dict': # якщо задана для конкретних
інтервалів часу
                i=0
                prev=None # попередній
                for t in res: # для кожного значення часу
                    if BC[k].has_key(timeList[i]):
                        t[k]=BC[k][timeList[i]]
                        prev=t[k]
                    else:
                        t[k]=prev
                    i=i+1
            elif type(BC[k]).__name__=='list': # якщо задана для кожного
інтервалу часу
                i=0
                for t in res: # для кожного значення часу
                    t[k]=BC[k][i]
                    i=i+1
            else: # якщо константа
                for t in res: # для кожного значення часу
                    t[k]=BC[k]
        #print res
        return res

    def resultsCAmodel(self):
        """Виводить результати автоматної моделі"""
```

```
        if self.history['X']==[]: # якщо немає результатів
            oldHistExist=self.readHistory() # прочитати старі результати
            if not oldHistExist: return

        tend=self.T[-1]
        T=1.0/self.PU.pumpingUnit.ns # період
        tstart=tend-T # тільки останній період
        # знайти найближче значення до tstart у списку self.T
        dt=[abs(t-tstart) for t in self.T]
        tstart=self.T[dt.index(min(dt))]
        self.drawDynamometerCard([0,1,2,3,4,5,6,7], "X", "Fr",
tstart=tstart, tend=tend)
        self.drawPlot([0,1,2,3,4,5,6,7,8],'X', x0scale=0.001)
        self.drawPlot([0,1,2,3,4,5,6,7],'Sr')

if __name__ == '__main__':
    pu=PUmodel.PU()
    model=CA_model(pu)
    #model.info()
    #model.run() # статика
    #model.info()

    model.runDynamic() # динаміка
    #model.info()
    model.resultsCAmodel()
```

ДОДАТОК Г

Модель ШСНУ мовою Modelica

Код також доступний в GitHub (vkopey/PU. URL: http://github.com/vkopey/PU).

Лістинг Г.1 – Main.mo - Modelica-код без анотацій

```
model Main
   import Modelica.Constants.inf;
   import Pi=Modelica.Constants.pi;
   import pi=Modelica.Constants.pi;

   inner parameter Modelica.SIunits.Density rhoFluid = FP1.rhoFluid;
   inner parameter Modelica.SIunits.BulkModulus ElFluid = FP1.ElFluid;
   inner parameter Modelica.SIunits.KinematicViscosity nuFluid =
FP1.nuFluid;
   public inner Maplesoft.Multibody.World
world(gravityDir=Maplesoft.Multibody.Selectors.UnitVector.negY,
gravityAcc=9.81);
   public Modelica.Blocks.Sources.Sine S1(amplitude=1.5,
freqHz=.108333333333, phase=3.14159265359, offset=0, startTime=200) ;
   public Modelica.Mechanics.Translational.Sources.Position
P1(useSupport=false, exact=true, f_crit=50) ;
   public Modelica.Mechanics.Translational.Components.Fixed F3(s0=0) ;
   public Maplesoft.Hydraulics.Basic.HydraulicCylinder
HC1(A=0.113411494795e-2) ;
   public Maplesoft.Hydraulics.Basic.CheckValve CV2(Ropen=178.945936903,
Gclosed=0.10e-11, Startclosed=false) ;
   public Maplesoft.Hydraulics.Basic.CheckValve CV1(Ropen=178.945936903,
Gclosed=0.10e-11, Startclosed=true) ;
   public Maplesoft.Hydraulics.Basic.AtmosphericPressure AP1(P=.0) ;
   public Maplesoft.Hydraulics.Basic.CircularPipe CP1(D=.1003, L=1500.0,
epsilon=0.15e-4, ReL=2000, ReT=4000, rho=rhoFluid, nu=nuFluid) ;
   public Maplesoft.Hydraulics.Basic.AtmosphericPressure AP2(P=100000.0) ;
   public inner Maplesoft.Hydraulics.FluidProperties FP1(rhoFluid=1000.0,
ElFluid=800000000.0, nuFluid=0.2e-5) ;
   public MapleSimStandaloneSubsystem n0 ;
   public Maplesoft.Hydraulics.Basic.FluidInertia FI1(A=0.761764747263e-2,
L=1500.0, rho=rhoFluid) ;
   public Maplesoft.Mechanical.Basic.TransFriction TF1(fs=702.75, fc=562.2,
d=0, vs=.1, n=1, v0=0.1e-1) ;
   public MapleSimStandaloneSubsystem n1 ;
   public MapleSimStandaloneSubsystem n2 ;
   public MapleSimStandaloneSubsystem n3 ;
   public MapleSimStandaloneSubsystem n4 ;
```

```
   public MapleSimStandaloneSubsystem n5 ;
   public MapleSimStandaloneSubsystem n6 ;
   public MapleSimStandaloneSubsystem n7 ;
   public Maplesoft.Hydraulics.Basic.FixedPressure FP2(P=0.13234e8) ;
equation
   connect(S1.y, P1.s_ref) ;
   connect(CV2.portA, AP1.portA) ;
   connect(CV2.portB, CV1.portA) ;
   connect(F3.flange, HC1.flange_b) ;
   connect(HC1.portA, CV2.portB) ;
   connect(P1.flange, n0.b1) ;
   connect(CP1.portB, FI1.portA) ;
   connect(FI1.portB, AP2.portA) ;
   connect(HC1.flange_a, TF1.flange_b) ;
   connect(TF1.flange_a, F3.flange) ;
   connect(n1.b1, n0.a1) ;
   connect(n2.b1, n1.a1) ;
   connect(n3.b1, n2.a1) ;
   connect(n3.a1, n4.b1) ;
   connect(n4.a1, n5.b1) ;
   connect(n5.a1, n6.b1) ;
   connect(n6.a1, n7.b1) ;
   connect(n7.a1, HC1.flange_a) ;
   connect(FP2.portB, CP1.portA) ;
   connect(CV1.portB, FP2.portA) ;
end Main;
model VerticalPipe
    extends Maplesoft.Icons.CustomComponent;
    parameter Real z = 0 "z";
    parameter Real rho = 1000 "rho";
    parameter Real g = 9.81 "g";
    Real Pin;
    Real Qin;
    Real dP;
    Real Pout;
    Real Qout;
    Maplesoft.Hydraulics.Interfaces.Port_a portA;
    Maplesoft.Hydraulics.Interfaces.Port_a portB;
equation
    Qin = Qout;
    dP = rho * g * z;
    Pin - Pout = dP;
    portA.p = Pin;
    portA.q = Qin;
    portB.p = Pout;
    portB.q = -Qout;
end VerticalPipe;
model MapleSimStandaloneSubsystem
   import Modelica.Constants.inf;
   import Pi=Modelica.Constants.pi;
   import pi=Modelica.Constants.pi;
```

```
   public Modelica.Mechanics.Translational.Components.Mass
M1(m=425.29310548, L=0) ;
   public Maplesoft.Sensors.SpeedSensor SS1(dimension="Velocity",
fromUnit="m/s", toUnit="m/s", scale=1.0, shift=.0) ;
   public Modelica.Mechanics.Translational.Sources.Force
F2(useSupport=false) ;
   public Modelica.Blocks.Math.BooleanToReal BTR2(realTrue=1.0,
realFalse=.0) ;
   public Modelica.Blocks.Math.Gain G2(k=46.9936711616) ;
   public Modelica.Blocks.Logical.LessThreshold LT1(threshold=0) ;
   public Modelica.Mechanics.Translational.Sources.ConstantForce
CF1(useSupport=false, f_constant=-3650.60969416) ;
   public Modelica.Blocks.Math.Product P3 ;
   public Modelica.Mechanics.Translational.Interfaces.Flange_b b1 ;
   public Modelica.Mechanics.Translational.Interfaces.Flange_a a1 ;
   public Maplesoft.Mechanical.Basic.TransFriction TF1(fs=0., fc=0.,
d=46.9936711616, vs=.1, n=1, v0=0.1e-1) ;
   public Modelica.Mechanics.Translational.Components.Fixed F1(s0=0) ;
   public Modelica.Blocks.Math.Gain G3(k=100) ;
   public Modelica.Blocks.Math.Tanh T1 ;
   public Modelica.Blocks.Math.Product P1 ;
   protected Modelica.Blocks.Interfaces.RealOutput RO_1 ;
   public Maplesoft.Sensors.ForceSensor FS1(dimension="Force",
fromUnit="N", toUnit="N", scale=1.0, shift=.0) ;
   public Maplesoft.Sensors.ForceSensor FS2(dimension="Force",
fromUnit="N", toUnit="N", scale=1.0, shift=.0) ;
   public Modelica.Blocks.Math.Gain G1(k=.0) ;
   public Modelica.Blocks.Math.Gain G4(k=.0) ;
   public Modelica.Blocks.Math.Add A1(k1=1, k2=1) ;
   public Modelica.Blocks.Math.Abs abs1_1 ;
   public Modelica.Blocks.Math.Add A2(k1=1, k2=-1) ;
   public Modelica.Mechanics.Translational.Components.SpringDamper
SD1(s_nominal=0.10e-3, c=317552.185425, d=14849.8907517, s_rel0=0) ;
   //public outer Maplesoft.Hydraulics.FluidProperties FP1;
equation
   connect(M1.flange_a, SS1.F) ;
   connect(CF1.flange, M1.flange_a) ;
   connect(LT1.y, BTR2.u) ;
   connect(BTR2.y, G2.u) ;
   connect(F2.flange, M1.flange_a) ;
   connect(F1.flange, TF1.flange_a) ;
   connect(G3.y, T1.u) ;
   connect(P1.y, F2.f) ;
   connect(TF1.flange_b, M1.flange_a) ;
   connect(SS1.RO, LT1.u) ;
   connect(P3.u2, G2.y) ;
   connect(G3.u, RO_1) ;
   connect(SS1.RO, RO_1) ;
   connect(RO_1, P3.u1) ;
   connect(P1.u1, T1.y) ;
```

```
   connect(P3.y, A2.u1) ;
   connect(P1.u2, A2.y) ;
   connect(A2.u2, abs1_1.y) ;
   connect(A1.y, abs1_1.u) ;
   connect(G4.y, A1.u2) ;
   connect(G1.y, A1.u1) ;
   connect(FS1.RO, G1.u) ;
   connect(FS1.F1, b1) ;
   connect(FS2.F1, M1.flange_a) ;
   connect(FS2.RO, G4.u) ;
   connect(FS2.F2, a1) ;
   connect(SD1.flange_b, FS1.F2) ;
   connect(M1.flange_b, SD1.flange_a) ;
end MapleSimStandaloneSubsystem;
```

Лістинг Г.2 – MAPLESIMmodel.py – модель ШСНУ для Maplesim 7

```
# -*- coding: cp1251 -*-
import math, os
import matplotlib.pyplot as plt
from matplotlib import rc
rc('font', family='Arial') # для підтримки Юнікоду
import PUmodel, maplepy

# Бажано задавати SD1_s_rel0. Визначити його можна шляхом моделювання
непрацюючої установки (задавши S1_T0>0).

class Maplesim_model():
    """Модель ШСНУ у Maplesim
    Потребує файлу моделі PUmodel.msim"""
    pu=None # об'єкт класу PUmodel.PU
    ms=None # об'єкт класу maplepy.MapleInterface4Maplesim

    def prepareParams(self):
        """Підготовлює параметри Maplesim моделі"""
        pu=self.pu # ШСНУ
        rs=pu.suckerRodString.items # список секцій
        n=len(rs) # кількість секцій

        M=pu.suckerRodString.massList() # маса секцій
        M2=pu.suckerRodString.fluidMassList() # маса рідини біля секцій
        C2=pu.suckerRodString.hydrodynamicRodResistanceList()
        W=pu.suckerRodString.weightForceList()
        C=[r.stiffness() for r in rs]
        D=[r.materialRodResistance() for r in rs]
        v=pu.pumpingUnit.velAvg()

        AN=pu.suckerRodString.wellAngleList() #список кутів alfa (довж. n)
```

```
        ANS=pu.suckerRodString.secAngleList() # кути відх. секцій від верт.
        AN1=pu.suckerRodString.angleTop()
        AN2=pu.suckerRodString.angleBottom()

#         for p in 'M M2 C2 W C D v AN ANS AN1 AN2'.split():
#             print p, eval(p) # надрукувати

        # підготувати параметри для моделі Maplesim
        modelParams={'S1_amplitude':pu.pumpingUnit.A, # параметри верстата
                     'S1_freqHz':pu.pumpingUnit.ns,
                     'S1_phase':pu.pumpingUnit.fi,
                     'HC1_A':pu.pump.pistonArea(), # парам. гідравл. част.
                     'CV1_Ropen':pu.pump.valveLossCoefficient(v),
                     'CV2_Ropen':pu.pump.valveLossCoefficient(v),
                     'CP1_D':pu.well.diametr, # ?
                     'CP1_L':pu.suckerRodString.length(),
                     #Замість VerticalPipe1 можна використовувати FP2_P
                     #'VerticalPipe1_z':pu.suckerRodString.height(),
                     'AP1_P':0.0, # тиск на вході
                     'AP2_P':pu.outPres, # тиск на гирлі
                     # гідростатичний тиск, який діє на плунжер під час
ходу вверх
                     # враховується також гідростат. тиск в затр. просторі,
який діє з другої сторони плунжера
                     'FP2_P':pu.fluidWeight()/pu.pump.pistonArea()-
pu.outTubeFluidWeight()/pu.pump.pistonArea()-pu.outTubePres,
                     'rhoFluid':pu.well.density,
                     'nuFluid':pu.well.viscosity,
                     # середня площа перетину труби
                     'FI1_A':
(sum(M2)/pu.well.density)/pu.suckerRodString.length(),
                     # ввести FI1_L=0, якщо не моделюється інерція рідини
                     'FI1_L':pu.suckerRodString.length(),
                     'TF1_fs':1.25*pu.pump.pistonFrictionForce(),
                     'TF1_fc':pu.pump.pistonFrictionForce()
                     }
        #W[0]+=15000.0 # !!! сила тертя в ущільненні
        for i in range(n): # для кожної секції ni (i=0, 1, 2...)
            modelParams['n'+str(i)+'_M1_m']=M[i]
            # !!! увага тут і нижче коефіцієнти демпфування збільшено
            modelParams['n'+str(i)+'_G2_k']=10*C2[i] # узгодити з TF1_d!!!
гасить вібрації під час ходу вниз
            modelParams['n'+str(i)+'_CF1_f_constant']=-W[i] # увага знак
            modelParams['n'+str(i)+'_TF1_fs']=0.2*W[i]*math.tan(AN[i]) #
сила тертя від ваги max
            modelParams['n'+str(i)+'_TF1_fc']=0.16*W[i]*math.tan(AN[i]) #
сила тертя від ваги
            modelParams['n'+str(i)+'_TF1_d']=10*C2[i] # коеф. в'язкого
тертя (узгодити з G2_k!!!) гасить вібрації
            modelParams['n'+str(i)+'_G1_k']=0.16*math.sin(AN1[i]) # множник
сили тертя від верхнього натягу
```

```
            modelParams['n'+str(i)+'_G4_k']=0.16*math.sin(AN2[i]) # множник
сили тертя від нижнього натягу
            modelParams['n'+str(i)+'_SD1_c']=C[i]
            modelParams['n'+str(i)+'_SD1_d']=D[i] # повинен бути більшим,
оскільки існує ще конструкційне демпфування

#         for p in sorted(modelParams.keys()):
#             print p, modelParams[p]

        self.modelParams=modelParams

    def createMaplesimModel(self, execute=False, resFileName=None):
        """Створює Maplesim модель. Виконує її, якщо execute=True"""
        ms=maplepy.MapleInterface4Maplesim()
        ms.path=os.getcwd().replace('\\', '/')
        ms.filenameMaplesim="PUmodel.msim"
        ms.paramDictMaplesim=self.modelParams # {} якщо не передавати
нічого
        #print ms.getCode()
        if execute: ms.execute=True
        if resFileName: ms.resultCSVfile=resFileName
        ms.runMaple()
        self.ms=ms

    def drawDynamometerCard(self, S0=1.0):
        """Створює файл-рисунок з динамограмою
        Першою пробою в PUmodel.msim має бути проба Length і Force
        на місці динамометра (верхня штанга)
        S0 - площа поперечного перетину верхньої штанги (якщо S0=1,
виводить силу)"""
        results=self.ms.readCSVfile(5) # список результатів

        # рисує тільки останній період
        T=1.0/self.modelParams['S1_freqHz'] # період
        tend=results[-1][0] # кінцевий час
        tstart=tend-T # початок останнього періоду

        f=[] # сила або напруження
        s=[] # переміщення
        f1=[]
        s1=[]
        for res in results: # для кожного часу
            if res[0]>=tstart: # починати з часу tstart
                f.append(res[1]/S0) # !!!напруження
                s.append(res[2]) # переміщення
                f1.append(res[3])
                s1.append(res[4])
        plt.plot(s, f, 'k-')
        plt.plot(s1, f1, 'b-')
        plt.grid(True)
        plt.xlabel(u"S, м") #надпис осі x
```

```
        if S0==1.0:
            plt.ylabel(u"F, H") #надпис осі y
        else:
            plt.ylabel("s, Pa") #надпис осі y
        plt.savefig("MaplesimDynCard.png")
        return min(f),max(f),min(s1),max(s1),min(f1),max(f1)

    def drawDynamometerCardMulti(self,
filenames=[],S0L=[],freqL=[],styleL=[]):
        """Створює файл-рисунок з динамограмою для багатьох файлів-
результатів
        див. документацію drawDynamometerCard"""
        ms=maplepy.MapleInterface4Maplesim()
        ms.path=os.getcwd().replace('\\', '/')
        for name,S0,freq,style in zip(filenames,S0L,freqL,styleL):
            ms.resultCSVfile=name
            results=ms.readCSVfile(3) # список результатів

            # рисує тільки останній період
            T=1.0/freq # період
            tend=results[-1][0] # кінцевий час
            tstart=tend-T # початок останнього періоду

            f=[] # сила або напруження
            s=[] # переміщення

            delta=0
            #if name=='13.csv': delta=0 # !!!поправка практичної
динамограми, якщо потрібно

            for res in results: # для кожного часу
                if res[0]>=tstart: # починати з часу tstart
                    f.append(res[1]/S0-delta) # !!!напруження
                    s.append(res[2]) # переміщення
            plt.plot(s, f, style)
        plt.grid(True)
        plt.xlabel(u"S, м") #надпис осі x
        plt.ylabel(u"F, H") #надпис осі y
        plt.savefig("MaplesimDynCard.png")

    def optimize(self):
        """Розраховує модель для різних значень її параметрів"""
        parList=[i/10.0 for i in [1,2,3,4,5,6,7,8,9,10]] # список значень
параметра
        fl=open("ampl-freq.csv", "w")
        self.pu=PUmodel.PU()
        for par in parList: # для кожного значення
            self.pu.pumpingUnit.ns=par # змінити значення параметра
            self.prepareParams()
            # створює файли результатів x_MaplesimResults.csv
```

```
            #self.createMaplesimModel(True,
str(par)+'_MaplesimResults.csv')
            self.createMaplesimModel(True)
            fl.write("%f;%f;%f;%f;%f;%f\n"%self.drawDynamometerCard())
            fl.flush()
        fl.close()

class Maplesim_model2(Maplesim_model): # успадковує Maplesim_model
    """Спрощена модель ШСНУ у Maplesim
    Потребує файлу моделі PUmodel.msim"""

    def prepareParams(self):
        """Підготовлює параметри Maplesim моделі"""
        pu=self.pu # ШСНУ

        M=pu.suckerRodString.mass() # маса колони
        W=pu.suckerRodString.weight() # вага колони
        WF=pu.fluidWeight() # вага рідини
        C=pu.suckerRodString.stiffness() # жорсткість колони
        D=pu.suckerRodString.damping()# еквівалентний коефіцієнт опору
        #D=50*D # увага, збільшено в 50! (або прийняти psi=0.5)
        #D=psi*math.sqrt(C*M)/(2*math.pi) # ?

        # підготувати параметри для моделі Maplesim
        modelParams={'S1_amplitude':pu.pumpingUnit.A, # параметри верстата-
гойдалки
                     'S1_freqHz':pu.pumpingUnit.ns,
                     'S1_phase':pu.pumpingUnit.fi,
                     'SD1_c':C, # параметри гідравлічної частини
                     'SD1_d':D,
                     'M1_m':M,
                     'CF1_f_constant':-W,
                     'G1_k':-WF
                     }

        for p in sorted(modelParams.keys()):
            print p, modelParams[p]

        self.modelParams=modelParams

class Maplesim_model3(Maplesim_model): # успадковує Maplesim_model
    """Модель верстата-гойдалки у Maplesim
    Потребує файлу моделі PUmodel.msim"""

    def prepareParams(self):
        """Підготовлює парметри Maplesim моделі"""
        pu=self.pu # ШСНУ
        # підготувати параметри для моделі Maplesim
        modelParams={'p1_L':pu.pumpingUnit.p1['L'], # параметри ланок
                     'p1_m':pu.pumpingUnit.p1['m'],
                     'p1_Iz':pu.pumpingUnit.p1['Iz'],
```

```
                    'p2_L':pu.pumpingUnit.p2['L'],
                    'p2_m':pu.pumpingUnit.p2['m'],
                    'p2_Iz':pu.pumpingUnit.p2['Iz'],
                    'p3_L':pu.pumpingUnit.p3['L'],
                    'p3_m':pu.pumpingUnit.p3['m'],
                    'p3_Iz':pu.pumpingUnit.p3['Iz'],
                    'p4_L':pu.pumpingUnit.p4['L'],
                    'p4_m':pu.pumpingUnit.p4['m'],
                    'p4_Iz':pu.pumpingUnit.p4['Iz'],
                    'p5_L':pu.pumpingUnit.p5['L'], # радіус противаги
                    'p5_m':pu.pumpingUnit.p5['m'], # маса противаги
                    'p5_Iz':pu.pumpingUnit.p5['Iz'],
                    'dx':pu.pumpingUnit.dx, # відстані між осями
                    'dy':pu.pumpingUnit.dy,
                    'r_k':pu.pumpingUnit.r, # радіус кривошипа
                    'const1_k':-pu.pumpingUnit.omega # кутова швидкість
кривошипів, рад/с
                    # знак "-" для від'ємного дезаксіалу (Архіпов с.15)
                    # тоді хід вверх довший
                    # ще бажано FF1_teta=pi/2
                    }

        for p in sorted(modelParams.keys()):
            print p, modelParams[p]

        self.modelParams=modelParams


if __name__ == '__main__':
    model=Maplesim_model() # !!! визначити вірний клас
    #model.optimize()

    model.pu=PUmodel.PU()
    print "Fmin,Fmax=", model.pu.polishedRodForce()
    #print "Rp=", model.pu.balancing()
    model.prepareParams()

    model.createMaplesimModel(execute=1)
    #model.drawDynamometerCard(S0=math.pi*0.019**2/4) #-
math.pi*0.0085**2/4)
    #model.drawDynamometerCard()
    # іншим способом побудови динамограм є збереження maplesim plots у .csv
    #model.drawDynamometerCardMulti(['1.csv','2.csv','3.csv'],[326.0e-
6,326.0e-6,326.0e-6],[0.166666,0.166666,0.166666],['k-','k--','k:'])

    #f=model.modelParams['S1_freqHz']
    model.drawDynamometerCardMulti(['testAPI.csv'],[1.0],[0.10833],['k-'])

#model.drawDynamometerCardMulti(['2_.csv','1_.csv','2.csv','1.csv'],[1.0,1.
0,1.0,1.0],[0.10833,0.10833,f,f],['k--','k--','k-','k-'])
```

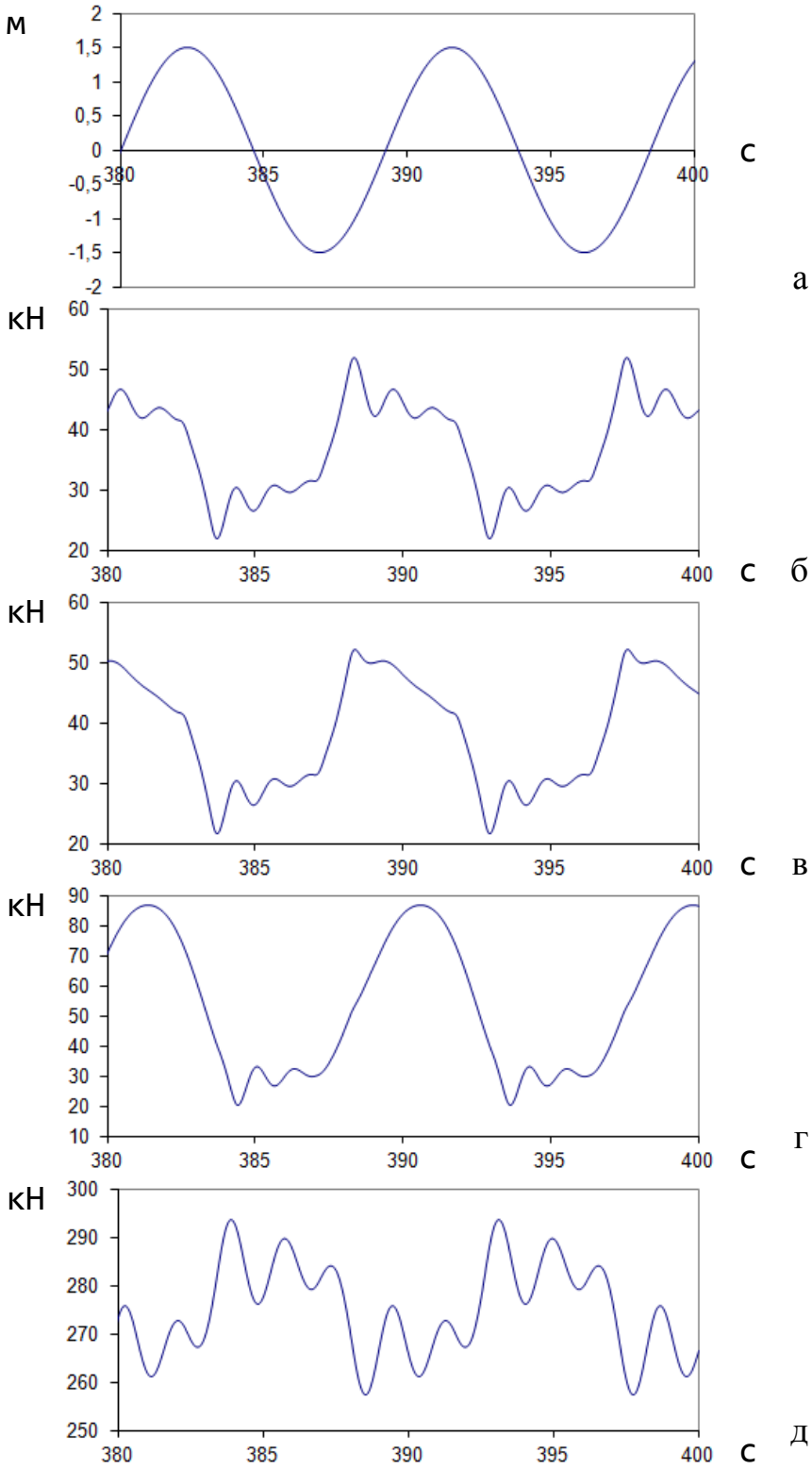
Лістинг Г.3 – maplepy.py – Python-інтерфейс до Maplesim

```
# -*- coding: CP1251 -*-
class MapleInterface(object):
    """Інтерфейс до Maple"""
    mapleExePath=r"c:\Program Files\Maple 18\bin.win\cmaple.exe"
    codeTemplate="" # шаблон коду Maple
    def getCode(self):
        """Повертає код Maple"""
        return ""
    def runMaple(self):
        """Виконує код Maple"""
        import subprocess,tempfile,os
        tf=tempfile.NamedTemporaryFile(delete=False, suffix='.mpl') #
тимчасовий файл .mpl
        tf.write(self.getCode()) # записати код
        tf.close()
        subprocess.Popen(self.mapleExePath+' '+tf.name).communicate() #
виконати код Maple
        os.unlink(tf.name) # видалити тимчасовий файл

class MapleInterface4Maplesim(MapleInterface):
    """Інтерфейс до Maplesim"""
    path="c:/9" # шлях до файлів моделі та результатів (розділювач тільки
/)
    filenameMaplesim="testAPI.msim" # файл моделі Maplesim
    # приклад доступу до параметру SD1_c підсистеми n: n_SD1_c
    paramDictMaplesim={"SD1_c":1, "SD1_d":0.1} # словник параметрів моделі
Maplesim
    execute=False # чи виконувати відразу симуляцію
    resultCSVfile="testAPI.csv" # файл результатів CSV
    # шаблони коду Maple
    codeTemplate1=r"""A :=MapleSim:-
LinkModel('filename'="{filenameMaplesim}"):
A:-SetParameters([{setParams}]):
"""
    codeTemplate2=r"""simData:=A:-Simulate(output = datapoint):
ExportMatrix("{resultCSVfile}", simData, target=delimited, delimiter=";"):
"""
    def paramString(self,paramDictMaplesim):
        """Повертає рядок параметрів MapleSim"""
        paramList=[k+" = "+str(v) for k,v in paramDictMaplesim.iteritems()]
# перетворити в список
        return ", ".join(paramList) # об'єднати в рядок

    def getCode(self):
        """Повертає код Maple"""
        setParams=self.paramString(self.paramDictMaplesim)
        filenameMaplesim=self.path+'/'+self.filenameMaplesim
        if self.execute: # якщо відразу виконувати симуляцію
            allCodeTemplate=self.codeTemplate1+self.codeTemplate2
```

```
            resultCSVfile=self.path+'/'+self.resultCSVfile
            code=allCodeTemplate.format(filenameMaplesim=filenameMaplesim,
setParams=setParams, resultCSVfile=resultCSVfile)
        else:

code=self.codeTemplate1.format(filenameMaplesim=filenameMaplesim,
setParams=setParams)
        return code

    def readCSVfile(self,nc=2):
        """Читає файл результатів CSV та повертає список з результатами
        у вигляді [[t1,x1,y1,...],[t2,x2,y2,...],...]
        nc - кількість змінних (мінімум 2)"""
        import csv
        resultCSVfile=self.path+'/'+self.resultCSVfile
        csv_file=open(resultCSVfile, "rb")
        reader=csv.reader(csv_file,delimiter = ';')
        resultList=[]
        for row in reader: # для кожного рядка
            oneResult=[] # список результатів з одного рядка
            for i in range(nc): # для кожної змінної
                oneResult.append(float(row[i])) # добавити значення змінної
            resultList.append(oneResult) # добавити список результатів з
одного рядка
        csv_file.close()
        return resultList

# class MapleInterface4Maplesim2(MapleInterface4Maplesim):
#     """Інтерфейс до Maplesim. Для моделей з підсистемами"""
#     paramDictMaplesim={'SD1_c':2} # параметри моделі
#     paramDictMaplesim2={'n':{'SD1_d':0.2}} # параметри підсистем моделі
#     CodeTemplate3="""A:-SetSubsystemName("{subSystemName}"):
# A:-SetParameters([{setParams}]):
# """
#
#     def getSubCode(self):
#         """Повертає код Maple для вводу параметрів підсистем"""
#         codeList=[]
#         for sname in self.paramDictMaplesim2:
#             setParams=self.paramString(self.paramDictMaplesim2[sname])
#
subCode=self.CodeTemplate3.format(subSystemName=sname,setParams=setParams)
#             codeList.append(subCode)
#         return '\n'.join(codeList)
#
#     def getCode(self):
#         """Повертає код Maple"""
#         setParams=self.paramString(self.paramDictMaplesim)
#         filenameMaplesim=self.path+'/'+self.filenameMaplesim
#         if self.execute: # якщо відразу виконувати симуляцію
```

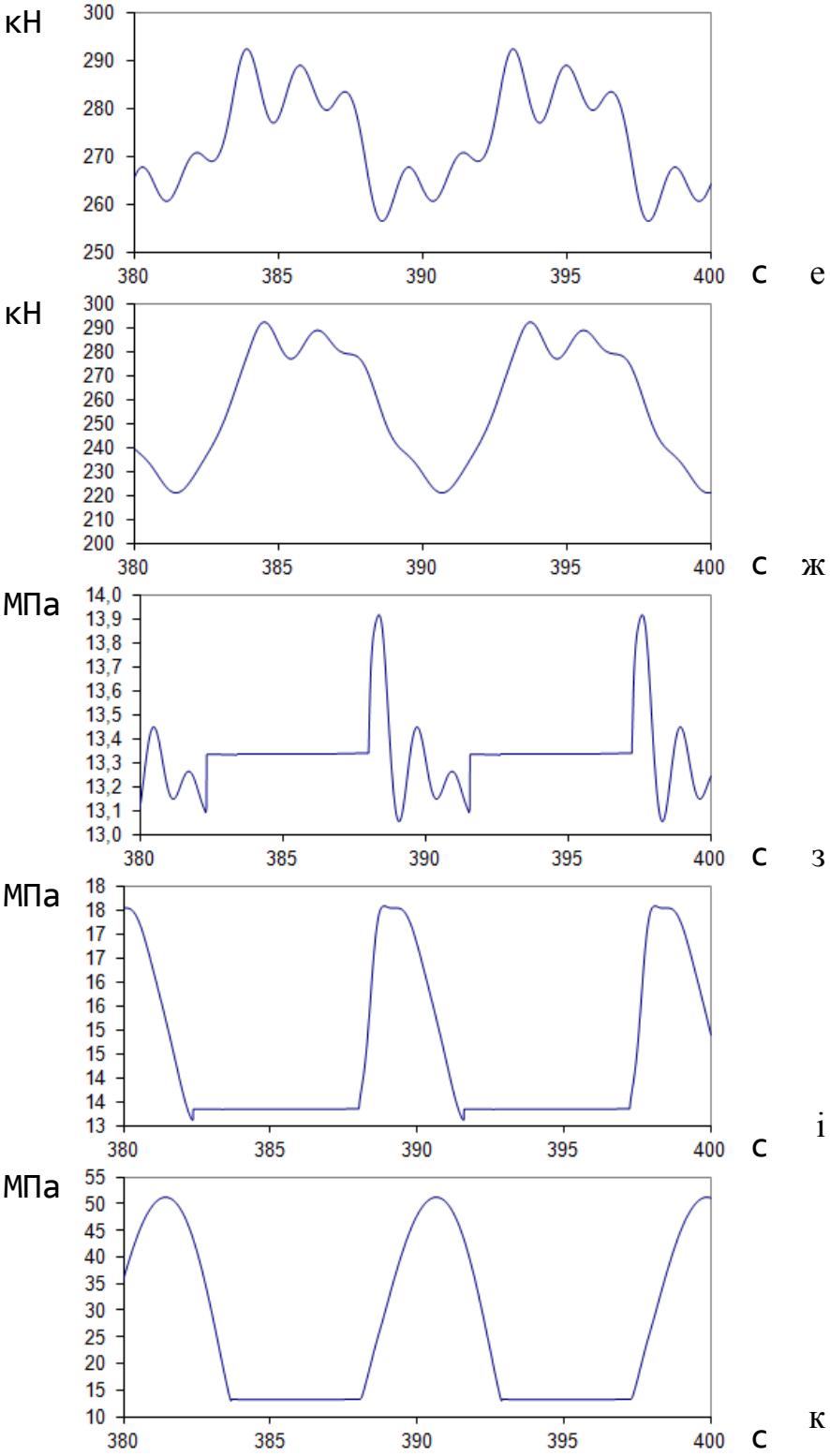
```
#
allCodeTemplate=self.codeTemplate1+self.getSubCode()+self.codeTemplate2
#             resultCSVfile=self.path+'/'+self.resultCSVfile
#
code=allCodeTemplate.format(filenameMaplesim=filenameMaplesim,
setParams=setParams, resultCSVfile=resultCSVfile)
#         else:
#             allCodeTemplate=self.codeTemplate1+self.getSubCode()
#
code=allCodeTemplate.format(filenameMaplesim=filenameMaplesim,
setParams=setParams)
#         return code

if __name__ == '__main__':
    ms=MapleInterface4Maplesim()
    print ms.getCode()
    #ms.runMaple()
    #print ms.readCSVfile(3)
```

ДОДАТОК Д
Графіки циклічних навантажень на колони ШН і НКТ



а – переміщення полірованого штока; б, в, г – сила у верхній частині колони ШН; д – сила у верхній частині колони НКТ; г – *d*=10 мм; в – *d*=20 мм

Рисунок Д.1, аркуш 1– Залежності параметрів від часу симуляції в умовах відсутності (б, д, з) та наявності СПУ на довжині 500 м, що зменшили діаметр отвору НКТ до значення *d*



е, ж – сила у верхній частині колони НКТ; з, і, к – тиск у нижній частині колони НКТ; ж, к – *d*=10 мм; е, і – *d*=20 мм

Рисунок Д.1, аркуш 2

ДОДАТОК Е

Пакет для моделювання кулькового клапана методами обчислювальної гідродинаміки

Код також доступний в GitHub (vkopey/BallValveAbaqus: Abaqus scripts for modelling of ball valve. URL: http://github.com/vkopey/BallValveAbaqus). Вміст пакету:

- `pickleIPC.py` – бібліотека для взаємодії між процесами;
- `server.py` – сервер для паралельних задач;
- `script.py` – сценарій AbaqusCAE.

Лістинг Е.1 – `pickleIPC.py`

```
# -*- coding: CP1251 -*-
"""Бібліотека для IPC. Містить функції, які дозволяють процесам
обмінюватись об'єктами python через сокети та тимчасові файли"""
import os, pickle, tempfile

def writeSocket(_socket, data):
    """Відсилає об'єкт python (data) через сокет"""
    f = _socket.makefile('wb') #,buffer_size # створити файл
    pickle.dump(data, f, pickle.HIGHEST_PROTOCOL) # законсервувати дані
    f.close() # закрити файл

def readSocket(_socket):
    """Повертає об'єкт python отриманий з сокета"""
    f = _socket.makefile('rb') #,buffer_size # створити файл
    data = pickle.load(f) # розконсервувати дані з файлу
    f.close() # закрити файл
    return data

def writeTempFile(data, tmpFileName="data4AbaqusScript.tmp"):
    """Записує об'єкт (data) у тимчасовому файлі в тимчасовій папці"""
    name=os.path.join(tempfile.gettempdir(),tmpFileName)
    f=open(name, "wb") # відкрити бінарний файл для запису
    pickle.dump(data,f) # законсервувати дані у файлі
    f.close() # закрити файл

def readTempFile(tmpFileName="data4AbaqusScript.tmp"):
    """Повертає об'єкт шляхом читання тимчасового файлу у тимчасовій папці"""
    name=os.path.join(tempfile.gettempdir(),tmpFileName)
    f=open(name, "rb") # відкрити бінарний файл для читання
    data=pickle.load(f) # розконсервувати дані з файлу
    f.close() # закрити файл
    return data
```

Лістинг Е.2 – server.py

```
# -*- coding: CP1251 -*-
import csv,pickleIPC,subprocess,datetime

csv_file=open("results.csv", "wb") # CSV файл результатів
writer = csv.writer(csv_file,delimiter = ';') # об'єкт для запису у файл
for h in [0.04, 0.05]: # для значень h у списку
    pickleIPC.writeTempFile(h) # записати дані в тимчасовий файл
    print datetime.datetime.now().isoformat() # час початку
    print "Abaqus CAE started. Please wait"
    # виконує скрипт в Abaqus та чекає завершення
    AbaqusPath=r"d:\SIMULIA\Abaqus\6.12-3\code\bin\abq6123.exe"
    subprocess.Popen(AbaqusPath+' cae noGUI=script.py').communicate()
    print datetime.datetime.now().isoformat() # час завершення
    print "Abaqus CAE finished"
    data=pickleIPC.readTempFile() # прочитати дані, які повернув скрипт
    writer.writerow(data) # записати у файл CSV
    csv_file.flush() # очистити буфер
csv_file.close() # закрити файл CSV
```

Лістинг Е.3 – script.py

```
# -*- coding: CP1251 -*-
from part import *
from material import *
from section import *
from optimization import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *

def set_values(part,feature,par):
    '''Присвоює значення параметрам. Приклад:
    par={'h1':0.0002,'h2':0.00004}
    set_values(part='Al',feature='Shell planar-1',par=par)'''
    p=model.parts[part] # деталь
    f=p.features[feature] # елемент
    s=model.ConstrainedSketch(name='__edit__', objectToCopy=f.sketch) #
тимчасовий ескіз
    p.projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s,
upToFeature=f) # спроектувати
    for k,v in par.iteritems(): # для всіх параметрів
        s.parameters[k].setValues(expression=str(v)) # установити значення
```

```
    f.setValues(sketch=s) # установити ескіз
    del s # знищити
    p.regenerate() # регенерувати деталь

def readODB_set2(set,step,var,pos=NODAL):
    '''Читає результати з останнього фрейму кроку на заданій множині
    set - множина, step - крок, var - змінна:
    ('S','Pressure'), ('U','Magnitude'), ...
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set2(set='Cont',step='Step-1',var=('S','Mises'))
    '''
    if pos==NODAL:
        s=odb.rootAssembly.nodeSets[set.upper()] # множина вузлів
    if pos==INTEGRATION_POINT:
        s=odb.rootAssembly.elementSets[set.upper()] # множина елементів
    fo=odb.steps[step].frames[-
1].fieldOutputs[var[0]].getSubset(region=s,position=pos) # дані
    #openOdb(r'C:/Temp/Model-1.odb').steps['Step-
1'].frames[4].fieldOutputs['CPRESS'].getSubset(position=NODAL,
region=openOdb(r'C:/Temp/Model-
1.odb').rootAssembly.nodeSets['CONT']).values[0].data
    res=[] # список результатів
    for v in fo.values: # для кожного вузла/елемента
        # додати до списку результатів
        if var[1]=='Pressure': res.append(v.press)
        if var[0]=='U' and var[1]=='Magnitude': res.append(v.magnitude)
        if var[1]=='U1': res.append(v.data.tolist()[0])
        if var[1]=='U2': res.append(v.data.tolist()[1])
        if var[0]=='PRESSURE': res.append(v.data)
    return res # повертає список значень

import pickleIPC,shutil,os
os.chdir('C:/Abaqus') # визначити робочий каталог
openMdb(pathName='C:/Abaqus/testCFD.cae') # відкрити модель
model=mdb.models['Model-1'] # створити об'єкт моделі
h=pickleIPC.readTempFile() # прочитати дані, які передав сервер
# надати значення параметру 'h' та перебудувати геометрію
#!'Solid revolve-1'
set_values(part='Part-1',feature='Solid extrude-1',par={'h':h})
model.parts['Part-1'].generateMesh() # генерувати сітку елементів
mdb.jobs['Model-1'].submit() # виконати задачу
mdb.jobs['Model-1'].waitForCompletion() # чекати завершення
# зберегти файл результатів .odb під унікальною назвою
shutil.copyfile(r'C:/Abaqus/Model-1.odb',
r'C:/Abaqus/results_'+str(h)+'.odb')
odb=openOdb(r'C:/Abaqus/Model-1.odb') # відкрити базу даних результатів
results=readODB_set2(set='bot',step='Step-1',var=('PRESSURE',')) #
отримати результати
pickleIPC.writeTempFile([h, sum(results)/len(results)]) # записати дані для
передачі їх серверу
odb.close() # закрити базу даних результатів
```

ДОДАТОК Є

pycodyn – пакет для компонентно-орієнтованого акаузального моделювання мовою Python

Код також доступний в GitHub (vkopey/pycodyn: Component-oriented acausal modeling of the dynamical systems in Python language. URL: http://github.com/vkopey/pycodyn). Вміст пакету:

- Pycodyn.mo – моделі штангової колони (мова Modelica, для порівняння);
- pycodyn.py – компоненти та розв’язувач (метод Ейлера);
- pycodynDAE.py – компоненти та розв’язувач (DAE);
- main1.py – модель вільних коливань колони (метод Ейлера);
- trapComponents.py – компоненти (метод трапецій);
- main1T.py – модель вільних коливань колони (метод трапецій);
- main1DAE.py – модель вільних коливань колони (DAE);
- main1Sym.py – модель вільних коливань колони (аналітична);
- main2s.py – односекційна модель процесу відкачування (метод Ейлера);
- main2.py – двосекційна модель процесу відкачування (метод Ейлера);
- main2V.py – двосекційна модель обриву колони (метод Ейлера, події);
- main2sDAE.py – односекційна модель процесу відкачування (DAE);
- main2DAE.py – двосекційна модель процесу відкачування (DAE);
- eliminate.py – скорочення системи рівнянь для SymPy.

Для установлення просто помістіть усі модулі в одну папку. Вимоги:

- Python 3.7 (рекомендується) або Python 2.7;
- numpy-1.16.4+mkl;
- scipy-1.2.2 (необов'язково);
- mpmath-1.1.0;
- sympy-1.4;
- Assimulo-2.9 (для pycodynDAE);
- matplotlib-2.2.4 (необов'язково, для графіків);

– OpenModelica 1.12 (для Pycodyn.mo).

Приклад використання (Python):

```
d:\WinPython32_37\python-3.7.4\python.exe main1.py
```

Приклад використання (OpenModelica):

```
loadModel(Modelica)
loadFile("Pycodyn.mo")
simulate(Pycodyn.Oscillator, stopTime=10)
plot(mass1.s)
simulate(Pycodyn.Pumping, stopTime=100)
plotParametric(motion1.flange.s, spring1.flange_a.f)
```

Лістинг Є.1 – Pycodyn.mo

```
within;
package Pycodyn
extends Modelica.Icons.Package;

connector Flange // клас-connector
  Real s; // змінна (переміщення в точці з'єднання рівні)
  flow Real f; // змінна (сума сил в точці з'єднання рівна нулю)
end Flange;

model Fixed // клас-model
  parameter Real s0=0; // параметр зі значенням за замовчуванням (постійний у часі)
  Flange flange; // об'єкт класу Flange
equation // рівняння моделі
  flange.s = s0;
end Fixed;

partial model Transl // клас-model
  Flange flange_a; // об'єкт класу Flange
  Flange flange_b; // об'єкт класу Flange
end Transl;

model Mass // клас-model
  extends Transl; // успадкування класу Transl
  parameter Real m(min=0, start=1); // параметр
  Real s; // змінна
  Real v(start=0); // змінна з початковою умовою
  Real a(start=0); // змінна з початковою умовою
equation // рівняння моделі
  v = der(s);
  a = der(v);
  m*a = flange_a.f + flange_b.f;
  flange_a.s = s;
  flange_b.s = s;
```

```
34 end Mass;
35
36 model SpringDamper // клас-model
37   extends Transl; // успадкування класу Transl
38   parameter Real c(final min=0, start=1); // параметр
39   parameter Real d(final min=0, start=1); // параметр
40   Real s_rel(start=0); // змінна
41   Real v_rel(start=0); // змінна
42   Real f; // змінна
43 equation // рівняння моделі
44   f = c*s_rel+d*v_rel;
45   s_rel = flange_b.s - flange_a.s;
46   v_rel = der(s_rel);
47   flange_b.f = f;
48   flange_a.f = -f;
49 end SpringDamper;
50
51 model Motion // клас-model
52   parameter Real A=2.1/2;
53   parameter Real n=6.4/60;
54   Flange flange; // об'єкт класу Flange
55 equation // рівняння моделі
56   flange.s = A*sin(6.283185307179586*n*time);
57 end Motion;
58
59 model Force // клас-model
60   Real f(start=0);
61   Flange flange; // об'єкт класу Flange
62 equation // рівняння моделі
63   flange.f = f;
64 end Force;
65
66 model Oscillator // клас-model
67   Mass mass1(s(start=-1), v(start=0), m=3961.0); // об'єкт з початковими
   умовами
68   SpringDamper spring1(c=44650.0, d=2120.7); // об'єкт
69   Fixed fixed1(s0=0); // об'єкт
70 equation // додаткові рівняння
71   // створює систему рівнянь (див. коментарі класу Flange)
72   connect(fixed1.flange, spring1.flange_a);
73   connect(spring1.flange_b, mass1.flange_a);
74 end Oscillator;
75
76 model Pumping // клас-model
77   Mass mass1(s(start=-0.9), v, m=3961.0); // об'єкт з початковими
   умовами
78   SpringDamper spring1(c=44650.0, d=2120.7); // об'єкт
79   Motion motion1(A=2.1/2, n=6.4/60);
80   Force force1;
81 equation // додаткові рівняння
82   connect(motion1.flange, spring1.flange_a);
```

```
83   connect(spring1.flange_b, mass1.flange_a);
84   connect(mass1.flange_b, force1.flange);
85 algorithm
86   if mass1.v <= 0 then
87     force1.f:=34687.0;
88   else
89     force1.f:=34687.0+18499.0*tanh(abs(mass1.v )/0.01);
90   end if;
91 end Pumping;
92
93 end Pycodyn;
```

Лістинг Є.2 – pycodyn.py

```
# -*- coding: utf-8 -*-
from sympy import *
import math, time
import matplotlib.pyplot as plt

def (d,name): # повертає значення за назвою символу
    for k in d:
        if repr(k)==name: return d[k]

dt=0.1 # крок часу
#dt=Symbol('dt') # тільки для отримання рівнянь в символьній формі

class Translational1D(object):
    """Базовий клас механічних поступальних 1D компонентів"""
    def __init__(self, name, args):
        self.name=name # назва компонента
        for k,v in args.items(): # для кожної пари ключ-значення
            if k in ['name','self']: continue # окрім name і self
            if v==None: # якщо значення є None
                # створити символьну змінну з назвою name+'_'+k
                self.__dict__[k]=Symbol(name+'_'+k)
            elif type(v) in [float,Float]: # якщо значення є дійсним числом
                self.__dict__[k]=Number(v) # створити константу
        self.eqs=[] # список рівнянь
        self.pins=[] # список фланців

    def pinEqs(self,pindex,pins):
        eqs=[] # список рівнянь фланця
        f=Number(0) # сума сил на фланцях інших компонентів
        for pin in pins: # для кожного фланця інших компонентів
            # додати рівняння, що описують рівність на фланці:
            eqs.append(Eq(self.pins[pindex]['x'], pin['x'])) # переміщень
            eqs.append(Eq(self.pins[pindex]['xp'], pin['xp'])) # переміщень
в час t-dt
            f+=pin['f'] # додати до суми сил
        eqs.append(Eq(self.pins[pindex]['f'], -f)) # рівність нулю суми сил
на фланці
```

```
        return eqs

class Mass(Translational1D):
    """Маса, зосереджена у точці, що рухається поступально"""
    def __init__(self, name, m=1.0, x=None, xp=None, v=None, vp=None,
a=None, f1=None, f2=None):
        Translational1D.__init__(self, name, locals()) # виклик
конструктора базового класу
        # система рівнянь
        self.eqs=[Eq(self.m*self.a, self.f1+self.f2),
                  Eq(self.a, (self.v-self.vp)/dt),
                  Eq(self.v, (self.x-self.xp)/dt)]
        self.pins=[dict(x=self.x, xp=self.xp, f=self.f1),
                   dict(x=self.x, xp=self.xp, f=self.f2)] # два фланця

class SpringDamper(Translational1D):
    """Поступальні 1D пружина і демпфер, з'єднані паралельно"""
    def __init__(self, name, c=1.0, d=0.1, x1=None, x2=None, x1p=None,
x2p=None, vrel=None, f1=None, f2=None):
        Translational1D.__init__(self, name, locals())
        # система рівнянь
        self.eqs=[Eq(self.c*(self.x2-self.x1)+self.d*self.vrel, self.f2),
                  Eq(-self.f2, self.f1),
                  Eq(self.vrel, (self.x2-self.x2p)/dt-(self.x1-
self.x1p)/dt)]

        self.pins=[dict(x=self.x1, xp=self.x1p, f=self.f1),
                   dict(x=self.x2, xp=self.x2p, f=self.f2)] # два фланця

class Force(Translational1D):
    """1D сила, точка прикладення якої рухається поступально"""
    def __init__(self,name,f=None,x=None,xp=None):
        Translational1D.__init__(self, name, locals())
        self.pins=[dict(x=self.x, xp=self.xp, f=-self.f)] # один фланець

class System(object):
    """Система компонентів, з'єднаних фланцями"""
    def __init__(self, els, eqs):
        self.els=els # список компонентів
        self.elsd=dict([(e.name,e) for e in els]) # те саме, але словник
        self.eqs=[] # список рівнянь системи
        for e in self.els: # для кожного компонента
            self.eqs+=e.eqs # з'єднати з рівняннями компонента
        self.eqs=self.eqs+eqs # з'єднати з додатковими рівняннями

    def solveN(self, eqs): # розв'язує систему алг. рівнянь за доп. scipy
        import scipy.optimize
        eqs0=[]
        vrs=set()
        for e in eqs:
            eq=e.lhs-e.rhs
```

```
            eqs0.append(eq)
            for a in eq.atoms():
                if a.is_Symbol:
                    vrs.add(a)
        vrs=list(vrs)
        f=lambdify([vrs], eqs0, 'numpy')
        goals=[0.0 for i in vrs]
        sol=scipy.optimize.root(f, goals, method='lm')
        d=dict(zip(vrs,sol.x))
        return d

    def solve(self, ics): # розв’язує систему алг. рівнянь в момент часу t
        eqs=[e.subs(ics) for e in self.eqs] # підстановка початкових умов
        eqs=[e for e in eqs if e not in (True,False)] # відкинути усі
вироджені рівняння
        # розв’язати систему за допомогою:
        #sol=nsolve(eqs) # SymPy (повільно) #або solve
        sol=self.solveN(eqs) # SciPy (швидше)
        sol.update(ics) # оновити словник словником поч. умов ics
        return sol

    def solv(self, preState): # розв’язує шляхом підстановки у вираз sympy
        state=preState.copy()
        for k in self.ceqs: # поточні рівняння
            state[k]=self.ceqs[k].subs(preState).evalf()
            #assert type(state[k]) in [float,Float]
        return state

    def solvN(self, preState): # те саме, але за доп. lambda-функції
        # використовуйте Python 3.7 для швидшого виконання
        state=preState.copy()
        ls=dict([(repr(a),state[a]) for a in self.vrsp]) # словник аргум.
        res=self.ceqsf(**ls) # виклик lambda-функції
        for a,v in zip([i[0] for i in self.ceqsi], res):
            state[a]=v # оновити state
        return state

    def createCurEqs(self, fnBC): # створити поточні 'швидкі рівняння'
        eqs=Tuple(*self.eqs)
        vrs={i for i in eqs.atoms(Symbol) if repr(i)[-1]!='p'} # змінні без
'p'
        vrsbc=set(fnBC.vrs)
        vrs=vrs-vrsbc # невідомі змінні на поточному кроці
        self.ceqs=solve(eqs,vrs) # поточні вирази
        self.vrsp={i for i in eqs.atoms(Symbol) if repr(i)[-1]=='p'} #
змінні з 'p'
        self.vrsp.update(vrsbc) # відомі змінні на поточному кроці
        self.ceqsi=self.ceqs.items() # впорядковані вирази
        self.ceqsf=lambdify(self.vrsp,[i[1] for i in self.ceqsi],'numpy') #
поточна lambda-функція
```

```
    def solveDyn(self, state, timeEnd, fnBC):
        # розв’язує динамічну задачу
        t=0.0 # змінна часу
        T=[] # список значень часу
        Res=[] # список результатів

        self.createCurEqs(fnBC)
        ics={} # для self.solve()
        start = time.time()
        while t<timeEnd:
            for k in state: # попередні значення "xp=x"...
                if repr(k)[-1]=='p':
                    state[k]=byName(state,repr(k)[:-1])
                    ics[k]=byName(state,repr(k)[:-1]) # для self.solve()

            state.update(fnBC(state, t)) # оновити граничні умови
            #state=self.solv(state) # шляхом виразу sympy (повільно)
            state=self.solvN(state) # шляхом виразу numpy (швидше)

            # ics.update(fnBC(state, t)) # оновити граничні умови
            # state=self.solve(ics) # шляхом scipy.optimize.root (повільно,
для частих подій)

            print(t)
            T.append(t)
            Res.append(state) # зберегти результати
            t+=dt # збільшити значення часу

            self.event(state) # обробник подій
        end = time.time()
        print('simulation time',end-start)
        return T,Res

    def event(self, state): # обробник подій
        pass
```

Лістинг Є.3 – pycodynDAE.py

```
# -*- coding: utf-8 -*-
import numpy as np
from sympy import *
t=Symbol('t')

class Translational1D(object):
    """ Базовий клас механічних поступальних 1D компонентів """
    def __init__(self, name, args):
        self.name=name # назва компонента
        for k,v in args.items(): # для кожної пари ключ-значення
            if k in ['name','self']: continue # окрім name і self
            if v==None: # якщо значення є None
                # створити символьну змінну з назвою name+'_'+k
```

```
                self.__dict__[k]=Symbol(name+'_'+k)
            elif type(v) in [float,Float]: # якщо значення є дійсним числом
                self.__dict__[k]=Number(v) # створити константу
        self.eqs=[] # список рівнянь
        self.pins=[] # список фланців

    def pinEqs(self,pindex,pins):
        eqs=[] # список рівнянь фланця
        f=Number(0) # сума сил на фланцях інших компонентів
        for pin in pins: # для кожного фланця інших компонентів
            # додати рівняння, що описують рівність на фланці:
            eqs.append(Eq(self.pins[pindex]['x'], pin['x'])) # переміщень
            f+=pin['f'] # додати до суми сил
        eqs.append(Eq(self.pins[pindex]['f'], -f)) # рівність нулю суми сил
на фланці
        return eqs

class Mass(Translational1D):
    """Маса, зосереджена у точці, що рухається поступально"""
    def __init__(self, name, m=1.0, x=None, v=None, a=None, f1=None,
f2=None, Dx=None, Dv=None):
        Translational1D.__init__(self, name, locals()) # виклик
конструктора базового класу
        # система рівнянь
        self.eqs=[Eq(self.m*self.a, self.f1+self.f2),
                  Eq(self.a, self.Dv),
                  Eq(self.v, self.Dx)]
        self.pins=[dict(x=self.x, f=self.f1),
                   dict(x=self.x, f=self.f2)] # два фланця

class SpringDamper(Translational1D):
    """Поступальні 1D пружина і демпфер, з’єднані паралельно"""
    def __init__(self, name, c=1.0, d=0.1, x1=None, x2=None, vrel=None,
f1=None, f2=None, Dx1=None, Dx2=None):
        Translational1D.__init__(self, name, locals())
        # система рівнянь
        self.eqs=[Eq(self.c*(self.x2-self.x1)+self.d*self.vrel, self.f2),
                  Eq(-self.f2, self.f1),
                  Eq(self.vrel, self.Dx2-self.Dx1)]

        self.pins=[dict(x=self.x1, f=self.f1),
                   dict(x=self.x2, f=self.f2)] # два фланця

class Force(Translational1D):
    """1D сила, точка прикладення якої рухається поступально """
    def __init__(self,name,f=None, x=None):
        Translational1D.__init__(self, name, locals())
        self.pins=[dict(x=self.x, f=-self.f)] # один фланець

class System(object):
    """Система компонентів, з’єднаних фланцями"""
```

```
    def __init__(self, els, eqs):
        self.els=els # список компонентів
        self.elsd=dict([(e.name,e) for e in els]) # те саме, але словник
        self.eqs=[] # список рівнянь системи
        for e in self.els: # для кожного компонента
            self.eqs+=e.eqs # з’єднати з рівняннями компонента
        self.eqs=self.eqs+eqs # з’єднати з додатковими рівняннями
        self.eqs=Tuple(*self.eqs)

    def residualArgs(self,eq):
        "Повертає впорядковані аргументи для нев’язок (функцій і похідних
eq)"
        ss=eq.atoms(Symbol) # множина символів рівняння
        ss.discard(t) # без t
        dss=dict([(i.name,i) for i in ss]) # словник назва:символ
        y=set();yd=set() # функція; похідна
        for a in ss:
            if 'D' in a.name: yd.add(a)
            else: y.add(a)
        y_=[];yd_=[] # пари функція; похідна
        for a in yd:
            b_name=a.name.replace('D','') # знайти пару
            if dss.get(b_name): # якщо пара
                yd_.append(a)
                y_.append(dss[b_name])
        yyd=set(y_+yd_)
        y=y_+list(y-yyd)
        yd=yd_+list(yd-yyd)
        return y,yd

    def residual(self,t,y,yd): # нев’язки для Assimulo
        yyd=np.concatenate([[t],y,yd])[:self.nv]
        r=self.lambdfun(*yyd)
        return np.array(r)

    def solveDAE(self, eq, state, stopTime=10.0):
        """Розв’язує динамічну задачу за доп. Assimulo (ODASSL, IDA)
        state - словник з початковим станом"""
        y,yd=self.residualArgs(eq)
        self.y=y
        self.yd=yd
        self.nv=len(y+yd)+1 # кількість аргументів для lambdfun (з t)
        eq0=[e.rhs-e.lhs for e in eq]
        self.lambdfun=lambdify([t]+y+yd,eq0,'numpy')
        #import inspect
        #print(inspect.getsource(self.lambdfun))

        y0=[state[i] for i in y] # початкові умови
        yd0=[state[i] for i in yd]
        # забезпечує однакову довжину y0, yd0 (важливо для ODASSL):
        dn=len(y0)-len(yd0)
```

```
        if dn>0: yd0+=[0.0]*dn
        else: y0+=[0.0]*abs(dn)

        from assimulo.problem import Overdetermined_Problem,
Implicit_Problem
        from assimulo.solvers import ODASSL,IDA

        # model = Overdetermined_Problem(self.residual, y0=y0, yd0=yd0)
        # sim = ODASSL(model)

        model = Implicit_Problem(self.residual, y0=y0, yd0=yd0)
        model.algvar = [1]*len(yd)+[0]*dn #[1,1,1,0,0,0,0,0]
        sim = IDA(model)
        sim.suppress_alg = True
        print(sim.get_options())

        T, Y, Yd = sim.simulate(stopTime)
        #sim.plot()
        return T, Y, Yd

    def solve(self,eq,ics): # для статичних задач
        eq=eq.subs(ics)
        return solve(eq) # розв’язувач sympy

def prnt(eq): # друк рівнянь
    print('\nEquations=')
    for i in eq: print(i)
```

Лістинг Є.4 – main1.py

```
1   # -*- coding: utf-8 -*-
2   from pycodyn import *
3
4   # створити компоненти:
5   s1=SpringDamper(name='s1', c=44650.0, d=2120.0)
6   m1=Mass(name='m1',m=3961.0)
7   peqs=s1.pinEqs(1,[m1.pins[0]]) # список додаткових рівнянь
8   s=System(els=[s1,m1], eqs=peqs) # система
9
10  # розв’язує статичну задачу - колона розтягнута на 1 м
11  ics={m1.x:-1.0, m1.v:0.0, m1.a:0.0, s1.x1:0.0, s1.x1p:0.0,
    m1.vp:0.0}
12  d=s.solve(ics)
13
14  def fnBC(d, t):
15      """граничні умови в час t для компонентів fnBC.vrs"""
16      val = 0.0, 0.0
17      return dict(zip(fnBC.vrs, val))
18  fnBC.vrs = s.elsd['s1'].x1, s.elsd['m1'].f2
19
20  #розв’язує динамічну задачу - вільні коливання колони
```

```
21  T,R=s.solveDyn(d, timeEnd=10, fnBC=fnBC)
22  plt.plot(T, [d[m1.x] for d in R])
23  plt.xlabel('t, s'); plt.ylabel('m1.x, m')
24  plt.show()
```

Лістинг Є.5 – trapComponents.py

```
# -*- coding: utf-8 -*-
from pycodyn import *

class Mass(Translational1D):
    def __init__(self,name,m=1.0,x=None,xp=None,v=None,vp=None,
a=None,ap=None,f1=None,f2=None):
        Translational1D.__init__(self, name, locals()) # виклик
конструктора базового класу

        self.eqs=[Eq(self.m*self.a, self.f1+self.f2),
                  Eq((self.a+self.ap)/2, (self.v-self.vp)/dt),
                  Eq((self.v+self.vp)/2, (self.x-self.xp)/dt)] # система
рівнянь
        self.pins=[dict(x=self.x, xp=self.xp, f=self.f1),
                   dict(x=self.x, xp=self.xp, f=self.f2)] # два фланця

class SpringDamper(Translational1D):
    def
__init__(self,name,c=1.0,d=0.1,x1=None,x2=None,x1p=None,x2p=None,vrel=None,
f1=None,f2=None,v1=None,v2=None,v1p=None,v2p=None):
        Translational1D.__init__(self, name, locals())

        self.eqs=[Eq(self.c*(self.x2-self.x1)+self.d*self.vrel, self.f2),
                  Eq(-self.f2, self.f1),
                  Eq((self.v1+self.v1p)/2, (self.x1-self.x1p)/dt),
                  Eq((self.v2+self.v2p)/2, (self.x2-self.x2p)/dt),
                  Eq(self.vrel, self.v2-self.v1)] # система рівнянь

        self.pins=[dict(x=self.x1, xp=self.x1p, f=self.f1),
                   dict(x=self.x2, xp=self.x2p, f=self.f2)] # два фланця
```

Лістинг Є.6 – main1T.py

```
# -*- coding: utf-8 -*-
from pycodyn import *
from trapComponents import SpringDamper,Mass

# створити компоненти:
s1=SpringDamper(name='s1', c=44650.0, d=2120.0)
m1=Mass(name='m1', m=3961.0)
peqs=s1.pinEqs(1,[m1.pins[0]]) # список додаткових рівнянь
s=System(els=[s1,m1], eqs=peqs) # система
# розв’язує статичну задачу - колона розтягнута на 1 м
```

```
ics={m1.x:-1.0, m1.v:0.0, m1.a:0.0, s1.x1:0.0, s1.x1p:0.0, m1.vp:0.0,
m1.ap:0.0, s1.v1:0.0, s1.v1p:0.0, s1.v2:0.0, s1.v2p:0.0}
d=s.solve(ics)

def fnBC(d, t):
    """граничні умови в час t для компонентів fnBC.vrs"""
    val = 0.0, 0.0
    return dict(zip(fnBC.vrs, val))
fnBC.vrs = s.elsd['s1'].x1, s.elsd['m1'].f2

# розв’язує динамічну задачу - вільні коливання колони
T,R=s.solveDyn(d, timeEnd=10, fnBC=fnBC)
plt.plot(T, [d[m1.x] for d in R])
plt.xlabel('t, s'); plt.ylabel('m1.x, m')
plt.show()
```

Лістинг Є.7 – main1DAE.py

```
1  # -*- coding: utf-8 -*-
2  from pycodynDAE import *
3
4  # створити компоненти:
5  s1=SpringDamper(name='s1', c=44650.0, d=2120.0)
6  m1=Mass(name='m1',m=3961.0)
7  peqs=s1.pinEqs(1,[m1.pins[0]]) # список додаткових рівнянь
8  s=System(els=[s1,m1], eqs=peqs) # система
9  prnt(s.eqs)
10
11 bc={s1.x1:0.0, s1.Dx1:0.0}
12 eq=s.eqs.subs(bc) # постійні граничні умови
13 prnt(eq)
14
15 # статика - колона розтягнута на 1 м
16 ics={m1.x:-1.0, m1.v:0.0, m1.a:0.0, s1.Dx2:0.0}
17 state=s.solve(eq,ics)
18 state.update(ics)
19
20 # динаміка - вільні коливання колони
21 eq=eq.subs({m1.f2:0.0}) # додаткові граничні умови
22 T,Y,Yd=s.solveDAE(eq, state, 10.0)
23
24 import matplotlib.pyplot as plt
25 index=s.y.index(m1.x)
26 plt.plot(T, [v[index] for v in Y])
27 plt.show()
```

Лістинг Є.8 – main1Sym.py

```
1  # -*- coding: utf-8 -*-
2  from pycodynDAE import *
```

```
3
4  # створити компоненти:
5  s1=SpringDamper(name='s1', c=44650.0, d=2120.0)
6  m1=Mass(name='m1', m=3961.0)
7  peqs=s1.pinEqs(1,[m1.pins[0]]) # список додаткових рівнянь
8  s=System(els=[s1,m1], eqs=peqs) # система
9  prnt(s.eqs)
10 eq=s.eqs.subs({m1.f2:0.0, s1.x1:0.0, s1.Dx1:0.0}) # граничні умови
11 prnt(eq)
12
13 # усунути змінні вручну:
14 eq=eq.subs({s1.f2:-s1.f1, m1.f1:s1.f1, s1.x2:m1.x, s1.Dx2:m1.Dx,
   s1.vrel:m1.v, m1.a:m1.Dv})
15 eq=eq.subs(s1.f1, solve(eq[3], s1.f1)[0])
16 eq=Tuple(*set(eq)-{True})
17
18 # або автоматично (спрощений алгоритм):
19 # from eliminate import *
20 # eq=eliminate(eq, keep={m1.Dv,m1.Dx,m1.v,m1.x}, maxEqLen=2)
21 prnt(eq)
22
23 # підстановка функцій і похідних
24 eq=eq.subs({m1.x:Function('m1_x')(t), m1.v:Function('m1_v')(t)})
25 eq=eq.subs({m1.Dx:Derivative('m1_x(t)', t), m1.Dv:Derivative('m1_v(t)',
   t)})
26 prnt(eq)
27 eq=dsolve(eq, ics={Function('m1_x')(0):-1.0, Function('m1_v')(0):0.0}) #
   розв’язати ODE аналітично
28 print(eq)
29 plot(eq[1].rhs, xlim=(0,10), ylim=(-1,1))
```

Лістинг Є.9 – main2s.py

```
# encoding: utf-8
from pycodyn import *

fs=-34687.0 # вага снкції
fr=-18499.0 # вага рідини над плунжером
# компоненти:
s1=SpringDamper(name='s1', c=44650.0, d=2120.7)
m1=Mass(name='m1', m=3961.0)
f1=Force(name='f1')#, f=fs+fr

# додаткові рівняння моделі колони, утворені шляхом з’єднання фланців
компонентів
peqs=s1.pinEqs(1,[m1.pins[0]])
peqs+=m1.pinEqs(1,[f1.pins[0]])
s=System(els=[s1,m1,f1], eqs=peqs) # система

# статична задача - колона під дією максимальних статичних навантажень
```

```
ics={m1.v:0.0, m1.a:0.0, s1.x1:0.0, s1.x1p:0.0, f1.f:fs+fr}
d=s.solve(ics)
print(d[m1.x])

def motion(t):
    """описує гармонічний рух верхньої точки колони і повертає її
переміщення в час t"""
    A=2.1/2 # амплітуда
    n=6.4/60 # частота
    return A*math.sin(2*math.pi*n*t) # переміщення

def force(v):
    """повертає значення сили на плунжері насоса F, в залежності від його
швидкості v"""
    F=fs # вага секції
    if v>0: # якщо рух вверх
        F+=fr # збільшити силу на значення ваги рідини
    return F*math.tanh(abs(v)/0.01) # згладжування біля точки v=0

def fnBC(d, t):
    """граничні умови у час t для компонентів fnBC.vrs"""
    val = motion(t), force(d[m1.v])
    return dict(zip(fnBC.vrs, val))
fnBC.vrs = s.elsd['s1'].x1, s.elsd['f1'].f

# розв’язати динамічну задачу - верхня точка має гармонічний рух
T,R=s.solveDyn(d, timeEnd=2*60/6.4, fnBC=fnBC)
R=[r for t,r in zip(T,R) if t>60/6.4] # тільки для останнього періоду
plt.plot([d[s1.x1] for d in R], [d[s1.f1]/1000 for d in R]) # гирлова
динамограма
plt.plot([d[m1.x] for d in R], [(-d[m1.f2]+fs)/1000 for d in R]) #
плунжерна динамограма
plt.xlabel('x, m'); plt.ylabel('f, kN')
plt.show()
```

Лістинг Є.10 – main2.py

```
1  # encoding: utf-8
2  from pycodyn import *
3
4  fs=(-18494.0, -16193.0) # вага секцій
5  fr=-18499.0 # вага рідини над плунжером
6  # компоненти:
7  s1=SpringDamper(name='s1', c=114926.0, d=5458.0)
8  m1=Mass(name='m1',m=2112.0)
9  f1=Force(name='f1', f=fs[0])
10 s2=SpringDamper(name='s2', c=73021.0, d=3468.0)
11 m2=Mass(name='m2', m=1850.0)
12 f2=Force(name='f2') #, f=fs[1]+fr
13 # додаткові рівняння моделі колони, утворені шляхом з’єднання фланців
   компонентів
```

```
peqs=s1.pinEqs(1,[m1.pins[0]])
peqs+=m1.pinEqs(1,[s2.pins[0],f1.pins[0]])
peqs+=s2.pinEqs(1,[m2.pins[0]])
peqs+=m2.pinEqs(1,[f2.pins[0]])
s=System(els=[s1,m1,s2,m2,f1,f2], eqs=peqs) # система

# статична задача - колона під дією максимальних статичних навантажень
ics={m1.v:0.0, m1.a:0.0, m2.v:0.0, m2.a:0.0, s1.x1:0.0, s1.x1p:0.0,
f2.f:fs[1]+fr}
d=s.solve(ics)
print(d[m2.x])

def motion(t):
    """описує гармонічний рух верхньої точки колони і повертає її
переміщення в час t"""
    A=2.1/2 # амплітуда
    n=6.4/60 # частота
    return A*math.sin(2*math.pi*n*t) # переміщення

def force(v):
    """повертає значення сили на плунжері насоса F, в залежності від
його швидкості v"""
    F=fs[1] # вага другої секції
    if v>0: # якщо рух вверх
        F+=fr # збільшити силу на значення ваги рідини
    return F*math.tanh(abs(v)/0.01) # згладжування біля точки v=0

def fnBC(d, t):
    """граничні умови у час t для компонентів fnBC.vrs"""
    val = motion(t), force(d[m2.v])
    return dict(zip(fnBC.vrs, val))
fnBC.vrs = s.elsd['s1'].x1, s.elsd['f2'].f

# розв’язати динамічну задачу - верхня точка має гармонічний рух
T,R=s.solveDyn(d, timeEnd=2*60/6.4, fnBC=fnBC)
R=[r for t,r in zip(T,R) if t>60/6.4] # тільки останній період
plt.plot([d[s1.x1] for d in R], [d[s1.f1]/1000 for d in R]) # гирлова
динамограма
plt.plot([d[m2.x] for d in R], [(-d[m2.f2]+fs[1])/1000 for d in R]) #
плунжерна динамограма
plt.xlabel('x, m'); plt.ylabel('f, kN')
plt.show()
```

Лістинг Є.11 – main2V.py

```
# encoding: utf-8
from pycodyn import *

def event(self, state): # обробник подій
    # симуляція обриву другої секції, коли сила>56000
```

```
6       if state[s1.f1]>56000:
7           self.fnBC=fnBC2
8           peqs=s1.pinEqs(1,[m1.pins[0]])
9           peqs+=m1.pinEqs(1,[f1.pins[0]])
10          self.__init__(els=[s1,m1,f1], eqs=peqs) # зміна системи
11          self.createCurEqs(fnBC2) # нові поточні рівняння
12  System.event=event
13
14  fs=(-18494.0, -16193.0) # вага секцій
15  fr=-18499.0 # вага рідини
16  # компоненти:
17  s1=SpringDamper(name='s1', c=114926.0, d=5458.0)
18  m1=Mass(name='m1',m=2112.0)
19  f1=Force(name='f1', f=fs[0])
20  s2=SpringDamper(name='s2', c=73021.0, d=3468.0)
21  m2=Mass(name='m2', m=1850.0)
22  f2=Force(name='f2')
23  # додаткові рівняння:
24  peqs=s1.pinEqs(1,[m1.pins[0]])
25  peqs+=m1.pinEqs(1,[s2.pins[0],f1.pins[0]])
26  peqs+=s2.pinEqs(1,[m2.pins[0]])
27  peqs+=m2.pinEqs(1,[f2.pins[0]])
28  s=System(els=[s1,m1,s2,m2,f1,f2], eqs=peqs) # система
29
30  # статична задача - колона під дією максимальних статичних навантажень
31  ics={m1.v:0.0, m1.a:0.0, m2.v:0.0, m2.a:0.0, s1.x1:0.0, s1.x1p:0.0,
    f2.f:fs[1]+fr}
32  d=s.solve(ics)
33  print(d[m2.x])
34
35  def motion(t):
36      """описує гармонічний рух верхньої точки колони і повертає її
    переміщення в час t"""
37      A=2.1/2 # амплітуда
38      n=6.4/60 # частота
39      return A*math.sin(2*math.pi*n*t) # переміщення
40
41  def force(v):
42      """повертає значення сили на плунжері насоса F, в залежності від
    його швидкості v"""
43      F=fs[1] # вага другої секції
44      if v>0: # якщо рух вверх
45          F+=fr # збільшити силу на значення ваги рідини
46      return F*math.tanh(abs(v)/0.01) # згладжування біля точки v=0
47
48  def fnBC(d, t):
49      """граничні умови у час t для компонентів fnBC.vrs"""
50      val = motion(t), force(d[m2.v])
51      return dict(zip(fnBC.vrs, val))
52  fnBC.vrs = s.elsd['s1'].x1, s.elsd['f2'].f
53
```

```
54 def fnBC2(d, t):
55     """граничні умови 2 у час t для компонентів fnBC2.vrs"""
56     val = (motion(t), )
57     return dict(zip(fnBC.vrs, val))
58 fnBC2.vrs = (s.elsd['s1'].x1, )
59
60 # розв’язати динамічну задачу - верхня точка має гармонічний рух
61 T,R=s.solveDyn(d, timeEnd=2*60/6.4+10, fnBC=fnBC)
62 plt.plot([d[s1.x1] for d in R], [d[s1.f1]/1000 for d in R]) # гирлова
   динамограма
63 plt.xlabel('x, m'); plt.ylabel('f, kN')
64 plt.show()
```

Лістинг Є.12 – main2sDAE.py

```
1  # -*- coding: utf-8 -*-
2  from pycodynDAE import *
3
4  fs=-34687.0 # вага секцій
5  fr=-18499.0 # вага рідини
6  # компоненти:
7  s1=SpringDamper(name='s1', c=44650.0, d=2120.7)
8  m1=Mass(name='m1', m=3961.0)
9  peqs=s1.pinEqs(1,[m1.pins[0]]) # список додаткових рівнянь
10 s=System(els=[s1,m1], eqs=peqs) # система
11 prnt(s.eqs)
12
13 # статична задача - колона під дією максимальних статичних навантажень
14 ics={s1.x1:0.0, s1.Dx1:0.0, m1.f2:fs+fr, m1.v:0.0, m1.a:0.0, s1.Dx2:0.0}
15 state=s.solve(s.eqs,ics)
16 state.update(ics)
17
18 # динамічна задача - верхня точка має гармонічний рух
19 A=2.1/2 # амплітуда
20 n=6.4/60 # частота
21 eq=s.eqs.subs({s1.x1: A*sin(2*pi*n*t), s1.Dx1: A*sin(2*pi*n*t).diff(t),
   m1.f2: Piecewise((fs, m1.v<0), (fs+fr*tanh(abs(m1.v)/0.01), m1.v>=0))})
22 # або додати рівняння:
23 #eq=s.eqs+Tuple(Eq(s1.x1, A*sin(2*pi*n*t)), Eq(m1.f2, Piecewise((fs,
   m1.v<0), (fs+fr*tanh(abs(m1.v)/0.01), m1.v>=0))) )
24 T,Y,Yd=s.solveDAE(eq, state, 20.0)
25
26 import matplotlib.pyplot as plt
27 index=s.y.index(s1.f1)
28 Y=[y for t,y in zip(T,Y) if t>60/6.4] # тільки останній період
29 T=[t for t in T if t>60/6.4]
30 plt.plot(A*np.sin(2*np.pi*n*np.array(T)), [v[index]/1000 for v in Y])
31 plt.show()
```

Лістинг Є.13 – main2DAE.py

```
# -*- coding: utf-8 -*-
from pycodynDAE import *

fs=(-18494.0, -16193.0) # вага секцій
fr=-18499.0 # вага рідини
# компоненти:
s1=SpringDamper(name='s1', c=114926.0, d=5458.0)
m1=Mass(name='m1', m=2112.0)
f1=Force(name='f1', f=fs[0])
s2=SpringDamper(name='s2', c=73021.0, d=3468.0)
m2=Mass(name='m2', m=1850.0)
f2=Force(name='f2') #, f=fs[1]+fr
# додаткові рівняння
peqs=s1.pinEqs(1,[m1.pins[0]])
peqs+=m1.pinEqs(1,[s2.pins[0],f1.pins[0]])
peqs+=s2.pinEqs(1,[m2.pins[0]])
peqs+=m2.pinEqs(1,[f2.pins[0]])
s=System(els=[s1,m1,s2,m2,f1,f2], eqs=peqs) # система
prnt(s.eqs)

# статична задача - колона під дією максимальних статичних навантажень
ics={s1.x1:0.0, s1.Dx1:0.0, f2.f:fs[1]+fr, m1.v:0.0, m1.a:0.0,
   s1.Dx2:0.0, s2.Dx1:0.0, s2.Dx2:0.0, m2.v:0.0, m2.a:0.0}
state=s.solve(s.eqs,ics)
state.update(ics)

# динамічна задача - верхня точка має гармонічний рух
A=2.1/2 # амплітуда
n=6.4/60 # частота
eq=s.eqs.subs({s1.x1: A*sin(2*pi*n*t), s1.Dx1: A*sin(2*pi*n*t).diff(t),
   m2.f2: Piecewise((fs[1], m2.v<0), (fs[1]+fr*tanh(abs(m2.v)/0.01),
   m2.v>=0))})
# або додати рівняння:
#eq=s.eqs+Tuple(Eq(s1.x1, A*sin(2*pi*n*t)), Eq(m1.f2, Piecewise((fs,
   m1.v<0), (fs+fr*tanh(abs(m1.v)/0.01), m1.v>=0))) )

T,Y,Yd=s.solveDAE(eq, state, 2*60/6.4)

import matplotlib.pyplot as plt
index=s.y.index(s1.f1)
Y=[y for t,y in zip(T,Y) if t>60/6.4] # тільки останній період
T=[t for t in T if t>60/6.4]
plt.plot(A*np.sin(2*np.pi*n*np.array(T)), [v[index]/1000 for v in Y])
plt.show()
```

Лістинг Є.14 – eliminate.py

```
from sympy import *
def eliminate(eq, keep, maxEqLen=2):
```

```
    """Усуває змінні з рівнянь eq з метою їхнього спрощення
    keep - намагатись зберегти ці змінні
    maxEqLen - максимальна кількість рівнянь після усунення"""
    while len(eq)>maxEqLen:
        e=solve(eq, eq.free_symbols-keep, exclude=keep) # для усунення
        e=sorted(e.items(), key=lambda x:len(x[1].atoms())) # спочатку
найкоротші вирази
        eq=eq.subs(e[0][0], e[0][1]) # усунути
        eq=Tuple(*set(eq)-set([True])) # без дублікатів і True
    return eq
```

ДОДАТОК Ж

Компонентно-орієнтована модель верстата-качалки для симуляції кінематики

Код програми також доступний в GitHub (vkopey/MMSKDM: Methods of modeling and simulation of kinematics and dynamics of machines. URL: http://github.com/vkopey/MMSKDM).

Лістинг Ж.1 – mehanizm_geom7.py

```
#encoding: utf-8
# визначення положення, швидкості і прискорення ланок за кутом повороту a кривошипа

from __future__ import division
import matplotlib.pyplot as plt
from scipy.optimize import root # функція для розв'язування системи рівнянь
from math import pi,sin,cos,tan,degrees,atan

class Frame:
    "Компонент описує ланку механізму"
    def __init__(self,x1,y1,x2,y2,L):
        "x1,y1,x2,y2,L - координати точок і довжина ланки"
        self.x1,self.y1,self.x2,self.y2,self.L=x1,y1,x2,y2,L
    def eqs(self): # система рівнянь компонента
        eqs=[]
        eqs+= [(self.x2-self.x1)**2 + (self.y2-self.y1)**2 - self.L**2] # незмінна відстань між точками
        return eqs
    def plot(self): # рисує компонент
        plt.plot([self.x1,self.x2],[self.y1,self.y2],'ko-')

class Connector:
    "Компонент описує шарнірне з'єднання двох ланок"
    def __init__(self,e1,e2):
        "e1,e2 - дві ланки, які з'єднуються точками 2 і 1 відповідно"
        self.e1,self.e2=e1,e2
    def eqs(self): # система рівнянь компонента
        eqs=[]
        eqs+= [self.e1.x2-self.e2.x1] # e1.x2=e2.x1
        eqs+= [self.e1.y2-self.e2.y1] # e1.y2=e2.y1
        return eqs
    def plot(self): # рисує компонент
        plt.plot([self.e1.x2],[self.e1.y2],'ro')
```

```
class Connector2:
    "Компонент описує нерухоме з'єднання двох ланок"
    def __init__(self,e1,e2):
        "e1,e2 - дві ланки, які з'єднуються точками 2 і 1 відповідно"
        self.e1,self.e2=e1,e2
    def eqs(self): # система рівнянь компонента
        eqs=[]
        eqs+= [self.e1.x2-self.e2.x1] # e1.x2=e2.x1
        eqs+= [self.e1.y2-self.e2.y1] # e1.y2=e2.y1
        # однаковий кут повороту ланок: tan(a1)=tan(a2)
        eqs+= [(self.e1.y1-self.e1.y2)/(self.e1.x1-self.e1.x2)-(self.e2.y2-
self.e2.y1)/(self.e2.x2-self.e2.x1)]
        return eqs
    def plot(self): # рисує компонент
        plt.plot([self.e1.x2],[self.e1.y2],'yo')

class System:
    "Система компонентів"
    def __init__(self,e):
        "e - список компонентів"
        self.e=e
    def eqs(self): # система усіх рівнянь, які описують поведінку системи
        eqs=[]
        for ei in self.e: # для кожного компонента
            eqs+= ei.eqs() # додати в список рівнянь усі рівняння
компонента
        return eqs
    def plot(self): # рисує усі компоненти системи
        for ei in self.e:
            ei.plot()

def f(X, s): # векторна функція повертає значення лівих частин рівнянь
    exec rootstr+"=X" # невідомі (X - вектор початкових наближень)
    eqs=s.eqs()
    return eqs # якщо усі елементи eqs близькі до 0, то X - шукані корені

t,a=0,0 # початкові значення часу і кута повороту кривошипа
T,X=[],[] # списки значень часу і невідомих
d = type('', (), dict(xa=0, ya=0, xb=-1.345, yb=3.01195, L0=0.81371,
L1=3.0, L2=2.0, L3=2.29))() # відомі постійні параметри механізму
fr0=Frame(x1=0.0,y1=0.0, x2=0.81371,y2=0, L=d.L0) # кривошип
fr1=Frame(x1=0.81371,y1=0, x2=0.65,y2=3, L=3.0) # шатун
fr2=Frame(x1=0.65,y1=3, x2=-1.345,y2=3.01195, L=2.0) # заднє плече
балансира
fr3=Frame(x1=-1.345,y1=3.01195, x2=-3.6,y2=3, L=2.29) # переднє плече
балансира
con0=Connector(fr0,fr1) # шарнір між кривошипом і шатуном
con1=Connector(fr1,fr2) # шарнір між шатуном і балансиром
con2=Connector2(fr2,fr3) # нерухоме з'єднання плечей балансира
s=System([fr0, fr1, fr2, fr3, con0, con1, con2]) # механізм верстата-
гойдалки
```

```
rootstr="s.e[1].x1, s.e[1].y1, s.e[1].x2, s.e[1].y2, s.e[2].x1, s.e[2].y1,
s.e[3].x2, s.e[3].y2"
# рядок rootstr потрібен щоб назви коренів записувались в програмі тільки
один раз

while a<2*pi: # поки кут < 360 градусів
    s.e[0].x2=s.e[0].L*cos(a)+s.e[0].x1 # координата точки кривошипа
dx/L=cos(a)
    s.e[0].y2=s.e[0].L*sin(a)+s.e[0].y1 # координата точки кривошипа
dy/L=sin(a)
    # увага! потрібно запобігати тому щоб початкові наближення коренів =0
    roots=[x+0.001 for x in eval(rootstr)] # наприклад шляхом додавання
0.001
    sol = root(f, roots, args=(s,), method='lm') # розв'язати систему
рівнянь
    exec rootstr+"=sol.x" # корені
    #print eval(rootstr)
    b=atan((s.e[3].y2-s.e[3].y1)/(s.e[3].x2-s.e[3].x1)) # кут повороту
балансира atan(dy/dx)
    x,y=s.e[0].x1+s.e[3].x1-s.e[3].L, -s.e[3].L*b # координати точки
підвісу
    plt.plot([x],[y],'bo')
    plt.text(x+0.1, y, t)
    plt.text(s.e[0].x2, s.e[0].y2, t)
    s.plot() # нарисувати положення ланок механізму
    T.append(t)
    X.append(y)
    t+=1 # збільшити час на крок
    a+=pi/16 #15.5 # збільшити кут на крок
plt.show()

import scipy
dt=T[1]-T[0]
V=scipy.gradient(X, dt) # швидкість (central differences)
A=scipy.gradient(V, dt) # прискорення (central differences)
plt.plot(T, X, 'k-', lw=2)
plt.plot(T, V, 'k--', lw=2)
plt.plot(T, A, 'r', lw=2)
plt.show()
```

ДОДАТОК З

Python-класи для перебудови параметричної моделі верстата-качалки у SOLIDWORKS

Код програми та інші компоненти моделі доступні в GitHub (vkopey/PumpingUnitModel: Parametric model of pumping unit in SOLIDWORKS. URL: http://github.com/vkopey/PumpingUnitModel).

Лістинг З.1 – PumpingUnit.py

```
# -*- coding: CP1251 -*-
import os
import codecs
import subprocess
from math import *

class SWmodel(object):
    """Клас моделі SolidWorks"""
    d={} # словник параметрів
    fileName=None # ім'я файлу моделі
    fileType=".SLDPRT" # розширення файлу моделі (".SLDPRT", ".SLDASM")

    def create(self):
        """Розраховує параметри моделі"""
        pass

    def rebuildModel(self):
        """Перебудовує файл рівнянь та модель SolidWorks"""
        self.write_dict_to_SW_equations()
        self.rebuildAndSaveModel()

    def rebuildAndSaveModel(self):
        """Перебудовує та зберігає SolidWorks модель шляхом виконання VBS
скрипта"""

        vbs=r"""'Скрипт VBS для перебудови моделі SolidWorks
Dim swApp 'SldWorks.SldWorks
Dim Part 'SldWorks.ModelDoc2
Set swApp = CreateObject("SldWorks.Application")
Set Part = swApp.OpenDoc("{fullFileNameExt}", {docType})
Set Part = swApp.ActivateDoc("{fileNameExt}")
Part.EditRebuild
Part.SaveSilent
Set Part = Nothing
swApp.CloseDoc "{fileName}"
```

```
Set swApp=Nothing
        """
        fileName=self.fileName
        fileNameExt=self.fileName+self.fileType
        fullFileNameExt=os.path.join(os.getcwd(), fileNameExt)
        if self.fileType==".SLDPRT":
            docType="1" # якщо деталь
        else:
            docType="2" # якщо збірка
        vbs=vbs.format(fullFileNameExt=fullFileNameExt,
fileNameExt=fileNameExt, fileName=fileName, docType=docType)
        scriptFileName=os.path.join(os.getcwd(), "RebuildSWmodelTemp.vbs")
        f=open(scriptFileName, 'w')
        f.write(vbs)
        f.close()
        # виконує процес та чекає його завершення
        subprocess.Popen(r'c:\Windows\system32\wscript.exe
'+scriptFileName).wait()

    def read_dict_from_SW_equations(self):
        """Додає елементи в словник self.d
        з текстового файлу (utf-8-sig) рівнянь SolidWorks"""
        filename=self.fileName+".txt" # файл рівнянь SolidWorks
        if not os.path.exists(filename): return # якщо файлу не існує,
вийти
        f=codecs.open(filename,'r','utf-8-sig') # відкрити файл для читання
        for line in f.readlines(): # для усіх рядків у списку
            if '=' in line: # якщо в рядку є символ "="
                pair=line.split('=') # розділити рядок
                pair=pair[0].strip()[1:-1], pair[1].strip() # видалити
пробіли і лапки

                if pair[1].isdigit(): # якщо всі символи цифри
                    val=int(pair[1])
                else: # інакше
                    try: # якщо дійсне число
                        val=float(pair[1])
                    except ValueError: # якщо рядок
                        val=pair[1]

                self.d[pair[0].encode('CP1251')]=val # записати в словник
        f.close() # закрити файл

    def write_dict_to_SW_equations(self):
        """Записує значення елементів словника self.d
        в текстовий файл (utf-8-sig) рівнянь SolidWorks"""
        d={} # словник з unicode ключами
        for k in self.d: # перетворюємо ключі в unicode
            d[k.decode('CP1251')]=self.d[k]

        filename=self.fileName+".txt" # файл рівнянь SolidWorks
```

```
        if not os.path.exists(filename): return # якщо файлу не існує,
вийти
        f=codecs.open(filename,'r','utf-8-sig') # відкрити файл для читання
        oldlines=f.readlines() # старий список рядків
        newlines=[] # новий список рядків
        for line in oldlines: # для усіх рядків у списку
            if '=' in line: # якщо в рядку є символ "="
                pair=line.split('=') # розділити рядок
                pair=pair[0].strip()[1:-1],pair[1].strip() # видалити
пробіли і лапки
                if pair[0] in d.keys(): # якщо ліва частина від "=" є серед
ключів словника
                    line='"'+pair[0]+'" = '+str(d[pair[0]])+"\n" #
формувати новий рядок
            newlines.append(line) # добавити рядок в новий список рядків
        f.close() # закрити файл

        f=codecs.open(filename,'w','utf-8-sig') # відкрити файл для запису
        f.writelines(newlines) # записати список нових рядків
        f.close() # закрити файл

    def assignParam(self, profil, name, nameList):
        """Присвоює значення заданим елементам словника параметрів.
        profil - конструктор відповідного профілю (див. класи профілів),
        name - назва профілю,
        nameList - список назв елементів SolidWorks.
        """
        p=profil()
        p.create(name=name)
        self.d=p.setSWParam(self.d, nameList)

class SWmodelPRT(SWmodel):
    """Клас моделі деталі SolidWorks"""
    fileType=".SLDPRT"

class SWmodelASM(SWmodel):
    """Клас моделі зборки SolidWorks"""
    fileType=".SLDASM"

class PumpingUnit(SWmodelASM):
    """Клас верстата-гойдалки"""
    fileName="Верстат"
    d={
    "Тип" : "СКД8-3-4000",
    "Максимальне допустиме навантаження" : 80.0,
    "Список довжин ходу полірованого штока" : [1200.0, 1600.0, 2000.0,
2500.0, 3000.0],
    "Довжина ходу полірованого штока" : 2000.0,
    "Список кількості гойдань" : [5.0,12.0],
    "Кількість гойдань" : 5.0,
    "Максимальний крутний момент" : 40.0,
```

```
    "Довжина переднього плеча балансира" : 2290.0,
    "Довжина заднього плеча балансира" : 2000.0,
    "Довжина шатуна" : 3000.0,
    "Найбільший радіус кривошипа" : 1290.0,
    "Радіус кривошипа" : None,
    "Горизонтальна відстань між осями опори балансира і тихохідного валу
редуктора" : 1345.0,
    "Вертикальна відстань між осями опори балансира і тихохідного валу
редуктора" : None,
    "Довжина" : 6900.0,
    "Ширина" : 2250.0,
    "Висота" : 4910.0,
    "Маса" : 11780.0,
    "Система урівноважування" : "кривошипна",
    "Вага кривошипних противаг" : 750.0,
    "Максимальна кількість кривошипних противаг" : 6,
    "Номінальна потужність електродвигуна" : 18.5,
    "Редуктор" : "Ц2НШ-750 Б"}

    def create(self):
        """Розраховує параметри моделі"""
        sList=self.d["Список довжин ходу полірованого штока"]
        s0=self.d["Довжина ходу полірованого штока"]
        # розрахунок вертикальної відстані між осями опори балансира і
тихохідного валу редуктора
        smax=max(sList) # найбільша довжина ходу
        rmax=self.d["Найбільший радіус кривошипа"]
        l=self.d["Довжина шатуна"]
        k=self.d["Довжина заднього плеча балансира"]
        k1=self.d["Довжина переднього плеча балансира"]
        l1=self.d["Горизонтальна відстань між осями опори балансира і
тихохідного валу редуктора"]
        h=smax*k/k1 # вертикальний хід опори траверси
        b=sqrt(k**2-(h/2)**2) # відстань від опори балансира до вертикалі h
        H=sqrt((l-rmax)**2-(b-l1)**2)+h/2
        self.d["Вертикальна відстань між осями опори балансира і
тихохідного валу редуктора"]=H

        # розрахунок радіуса кривошипа
        h=s0*k/k1 # вертикальний хід опори траверси
        b=sqrt(k**2-(h/2)**2) # відстань від опори балансира до вертикалі h
        r=l-sqrt((H-h/2)**2+(b-l1)**2)
        self.d["Радіус кривошипа"]=r

        # створення та розрахунок деталей
        self.GolovBalansir=GolovBalansir()
        self.GolovBalansir.create(paramPU=self.d)
        self.Balansir=Balansir()
        self.Balansir.create(paramPU=self.d,
paramGolovBalansir=self.GolovBalansir.d)
```

```
        self.GolovBalansir.d["Висота між опорами@Профіль
головки"]=self.Balansir.d["Висота@Двотавр профіль"]
        self.Shatun=Shatun()
        self.Shatun.create(paramPU=self.d)
        self.Krivoshyp=Krivoshyp()
        self.Krivoshyp.create(paramPU=self.d)
        self.Traversa=Traversa()
        self.Traversa.create(paramPU=self.d)
        self.Val=Val()
        self.Val.create(paramPU=self.d, paramShatun=self.Shatun.d,
paramTraversa=self.Traversa.d)
        self.Reduktor=Reduktor()
        self.Reduktor.create()
        self.Stiyka=Stiyka()
        self.Stiyka.create(paramPU=self.d, paramReduktor=self.Reduktor.d,
paramVal=self.Val.d, paramKrivoshyp=self.Krivoshyp.d)
        self.Rama=Rama()
        self.Rama.create(paramPU=self.d, paramStiyka=self.Stiyka.d,
paramReduktor=self.Reduktor.d)
        self.Protyvaga=Protyvaga()
        self.Protyvaga.create(paramKrivoshyp=self.Krivoshyp.d)
        # зборки:
        self.BalansirVZbori=BalansirVZbori()
        self.Krivoshypy=Krivoshypy()
        self.TraversaVZbori=TraversaVZbori()
        self.Karkas=Karkas()

    def rebuildModel(self):
        self.GolovBalansir.rebuildModel()
        self.Balansir.rebuildModel()
        self.Shatun.rebuildModel()
        self.Krivoshyp.rebuildModel()
        self.Traversa.rebuildModel()
        self.Val.rebuildModel()
        self.Reduktor.rebuildModel()
        self.Stiyka.rebuildModel()
        self.Rama.rebuildModel()
        self.Protyvaga.rebuildModel()
        # зборки:
        self.BalansirVZbori.rebuildModel()
        self.Krivoshypy.rebuildModel()
        self.TraversaVZbori.rebuildModel()
        self.Karkas.rebuildModel()

class Profil(object):
    """Клас профілю"""
    fileName=None
    d={} # словник параметрів

    def create(self, name):
        self.getParamFromCSV(name)
```

```
    def setSWParam(self,d,nameList):
        """Повертає словник параметрів для SolidWorks.
        d - початковий словник параметрів SolidWorks,
        nameList - список назв елементів SolidWorks.
        Приклад:
        setSWParam({"Висота@Профіль1":0, "Висота@Профіль2":0},
            ["Профіль1","Профіль2"])
        """
        for k in d: # для усіх ключів словника параметрів для SolidWorks
            if k.count('@')==1: # якщо в ключі є тільки 1 символ '@'
                pair=k.split('@') # розділити на дві частини
                if pair[1] in nameList: # якщо назва елементу в списку
                    d[k]=self.d[pair[0]] # змінити значення
        return d

    def getParamFromCSV(self, name):
        """З файлу CSV записує в словник параметрів параметри профілю з
назвою name"""
        import csv
        csv_file=open(self.fileName+".csv", "rb")
        reader=csv.DictReader(csv_file, delimiter = ';')
        for row in reader:
            if row["Назва"]==name:
                self.d=row
        csv_file.close()

        # перетворює значення словника в дійсні числа
        for k in self.d:
            if k!="Назва": # окрім назви
                if "," in self.d[k]: # якщо є символ ","
                    self.d[k]=float(self.d[k].replace(",", "."))
                else:
                    self.d[k]=float(self.d[k])

class DvotavrProfil(Profil):
    """Клас двотавра"""
    fileName="Двотаври ГОСТ 8239-89"

class ShvelerProfil(Profil):
    """Клас швелера"""
    fileName="Швелери серія У ДСТУ 3436-96"

class KutnykProfil(Profil):
    """Клас кутника"""
    fileName="Кутник ДСТУ 2251-93"

class Balansir(SWmodelPRT):
    """Клас балансира"""
    fileName="Балансир"
    #словник параметрів
```

```
    d={
    "Двотавр номер профілю" : None}

    def create(self, paramPU, paramGolovBalansir):
        self.read_dict_from_SW_equations() # читати параметри з файлу
рівнянь

        # профіль двотавра
        self.assignParam(DvotavrProfil, self.d["Двотавр номер профілю"],
["Двотавр профіль"])

        # довжина плеч
        lpp=paramPU["Довжина переднього плеча балансира"]
        lzp=paramPU["Довжина заднього плеча балансира"]
        # довжина головки балансира
        lgb=paramGolovBalansir["Ширина@Профіль головки"]-
paramGolovBalansir["Глибина опори@Профіль головки"]
        self.d["Довжина переднього плеча балансира"]=lpp-lgb
        self.d["Довжина@Двотавр"]=lpp-lgb+self.d["Координата@Профіль опори
балансира"]+self.d["Ширина@Профіль опори балансира"]/2
        self.d["Координата@Профіль опори балансира"]=lzp-
self.d["Ширина@Профіль опори балансира"]/2+self.d["Ширина@Профіль
опори"]/2+self.d["Координата@Профіль опори"]
        # координати ребер жорсткості
        self.d["Координата@Ребро профіль"]=100.0
        self.d["Координата@Масив ребер1"]=(lzp-100)/2
        self.d["Координата@Масив ребер2"]=(lzp+100)
        self.d["Координата@Масив ребер3"]=lzp+lpp/2

class GolovBalansir(SWmodelPRT):
    """Клас головки балансира"""
    fileName="Головка балансира"
    #словник параметрів
    d={
    "Двотавр номер профілю" : None,
    "Радіус@Профіль головки" : 2290.0,
    "Висота між опорами@Профіль головки" : 640.0,
    "Ширина@Профіль головки" : 1000.0,
    "Глибина опори@Профіль головки" : 300.0,
    "Зовнішній радіус@Двотавр профіль" : 6.0,
    "Внутрішній радіус@Двотавр профіль" : 14.0,
    "Товщина середини основи@Двотавр профіль" : 12.3,
    "Товщина стінки@Двотавр профіль" : 7.5,
    "Ширина@Двотавр профіль" : 145.0,
    "Висота@Двотавр профіль" : 360.0,
    "Висота@Перемістити грань1" : 70.0,
    "Висота@Перемістити грань2" : 70.0,
    "Висота осі опори@Профіль головки" : 120.0,
    "Кут@Профіль головки" : 75.0}

    def create(self, paramPU):
```

```
        # профіль двотавра
        self.assignParam(DvotavrProfil, self.d["Двотавр номер профілю"],
["Двотавр профіль"])

        r=paramPU["Довжина переднього плеча балансира"]
        self.d["Радіус@Профіль головки"]=r
        s=max(paramPU["Список довжин ходу полірованого штока"]) # найбільша
        self.d["Кут@Профіль головки"]=degrees(s/r)

class Shatun(SWmodelPRT):
    """Клас шатуна"""
    fileName="Шатун"
    #словник параметрів
    d={"Довжина@Тіло" : 3000.0,
       "Ширина@Опора" : 70.0}
    def create(self, paramPU=None):
        dsh=paramPU["Довжина шатуна"]
        self.d["Довжина@Тіло"]=dsh

class Krivoshyp(SWmodelPRT):
    """Клас кривошипа"""
    fileName="Кривошип"
    #словник параметрів
    d={"Довжина@Ескіз" : 2000.0,
       "Радіус@Ескіз" : 1290.0,
       "Ширина@Профіль" : 150.0,
       "Висота@Ескіз" : 400.0,
       "Координата отвору@Ескіз" : 300.0}

    def create(self, paramPU=None):
        r=paramPU["Радіус кривошипа"]
        rmax=paramPU["Найбільший радіус кривошипа"]
        self.d["Радіус@Ескіз"]=r
        self.d["Довжина@Ескіз"]=rmax+600.0

class Traversa(SWmodelPRT):
    """Клас траверси"""
    fileName="Траверса"
    #словник параметрів
    d={"Зовнішній радіус@Швелер профіль" : 6.0,
       "Внутрішній радіус@Швелер профіль" : 15.0,
       "Товщина середини основи@Швелер профіль" : 13.5,
       "Товщина стінки@Швелер профіль" : 8.0,
       "Кут основи@Швелер профіль" : 5.73,
       "Ширина@Швелер профіль" : 115.0,
       "Висота@Швелер профіль" : 400.0,
       "Півдовжина@Швелер" : 1125.0,
       "Товщина@Профіль ребра" : 10.0,
       "Координата@Профіль ребра" : 100.0,
       "Координата@Масив ребер" : 562.5,
       "Кількість@Масив ребер" : 2,
```

```
        "Діаметр@Ескіз отвору під шатун" : 50.0,
        "Координата@Ескіз отвору під шатун" : 100.0,
        "Висота@Ескіз опори" : 240.0,
        "Діаметр отвору@Ескіз опори" : 100.0,
        "Висота осі@Ескіз опори" : 120.0,
        "Товщина@Опора" : 30.0,
        "Координата@Опора" : 170.0,
        "Товщина@Профіль основи опори" : 10.0,
        "Півдовжина@Основа опори": 240.0}

    def create(self, paramPU=None):
        # профіль швелера
        self.assignParam(ShvelerProfil, self.d["Швелер номер профілю"],
["Швелер профіль"])

        ln=paramPU["Ширина"]
        self.d["Півдовжина@Швелер"]=ln/2
        self.d["Координата@Масив ребер"]=self.d["Півдовжина@Швелер"]/2

class Val(SWmodelPRT):
    """Клас валу"""
    fileName="Вал"
    #словник параметрів
    d={"Довжина@Вал" : 2400.0}
    def create(self, paramPU, paramShatun, paramTraversa):
        l1=paramShatun["Ширина@Опора"]
        ln=paramTraversa["Півдовжина@Швелер"]
        l2=paramTraversa["Координата@Ескіз отвору під шатун"]
        self.d["Довжина@Вал"]=2*(ln-l2-l1/2)

class Reduktor(SWmodelPRT):
    """Клас редуктора"""
    fileName="Редуктор"
    #словник параметрів
    d={"Висота осі@Профіль отвору" : 350.0,
       "Координата осі@Профіль отвору" : 300.0,
       "Довжина@Профіль корпуса" : 1100.0}

class Stiyka(SWmodelPRT):
    """Клас стійки"""
    fileName="Стійка"
    #словник параметрів
    d={"Ширина@Ескіз основи" : 2000.0,
       "Довжина@Ескіз основи" : 2000.0,
       "Висота@Основа" : 4500.0,
       "Висота@Секція1" : 500.0,
       "Висота@Секція2" : 1500.0,
       "Висота@Секція3" : 2500.0,
       "Висота@Секція4" : 3500.0,
       "Ширина@Ескіз опори балансира" : 400.0,
       "Висота@Ескіз опори балансира" : 400.0,
```

```
       "Висота осі@Ескіз опори балансира" : 200.0,
       "Товщина@Опора балансира" : 100.0,
       "Координата@Опора балансира" : 170.0,
       "Товщина@Основа опори" : 20.0,
       "Кут" : 8}

    def create(self, paramPU, paramReduktor, paramVal, paramKrivoshyp):
        s=paramVal["Довжина@Вал"]-2*paramKrivoshyp["Ширина@Профіль"]-100
        self.d["Ширина@Ескіз основи"]=s
        self.d["Довжина@Ескіз основи"]=s
        h=paramPU["Вертикальна відстань між осями опори балансира і
тихохідного валу редуктора"]
        hr=paramReduktor["Висота осі@Профіль отвору"]
        self.d["Висота@Основа"]=h+hr-self.d["Висота осі@Ескіз опори
балансира"]-self.d["Товщина@Основа опори"]
        hsect1=self.d["Висота@Основа"]/9
        self.d["Висота@Секція1"]=hsect1
        hsect=(self.d["Висота@Основа"]-hsect1)/4
        self.d["Висота@Секція2"]=hsect1+hsect
        self.d["Висота@Секція3"]=hsect1+2*hsect
        self.d["Висота@Секція4"]=hsect1+3*hsect

class Rama(SWmodelPRT):
    """Клас рами"""
    fileName="Рама"
    #словник параметрів
    d={"Довжина@Ескіз" : 5000.0,
       "Ширина@Ескіз" : 2000.0,
       "L1@Ескіз" : 200.0,
       "L2@Ескіз" : 2000.0,
       "L3@Ескіз" : 400.0,
       "L4@Ескіз" : 1000.0,
       "L5@Ескіз" : 300.0}

    def create(self, paramPU, paramStiyka, paramReduktor):
        l0=paramPU["Горизонтальна відстань між осями опори балансира і
тихохідного валу редуктора"]
        s=paramStiyka["Ширина@Ескіз основи"]
        l2=paramStiyka["Довжина@Ескіз основи"]
        lr=paramReduktor["Довжина@Профіль корпуса"]
        ko=paramReduktor["Координата осі@Профіль отвору"]

        self.d["Ширина@Ескіз"]=s
        self.d["L2@Ескіз"]=l2
        self.d["L3@Ескіз"]=l0-l2/2-ko
        self.d["L4@Ескіз"]=lr-100

self.d["Довжина@Ескіз"]=self.d["L1@Ескіз"]+self.d["L2@Ескіз"]+self.d["L3@Ес
кіз"]+self.d["L4@Ескіз"]+self.d["L5@Ескіз"]+700

class Protyvaga(SWmodelPRT):
```

```
    """Клас противаги"""
    fileName="Противага"
    #словник параметрів
    d={"Радіус@Эскиз1" : 1625.0,
       "Півширина кривошипа@Эскиз1" : 200.0,
       "Ширина@Эскиз1" : 800.0,
       "Товщина@Бобышка-Вытянуть1" : 150.0}

    def create(self, paramKrivoshyp):
        self.d["Радіус@Эскиз1"]=paramKrivoshyp["Довжина@Ескіз"]-
paramKrivoshyp["Координата отвору@Ескіз"]
        self.d["Півширина
кривошипа@Эскиз1"]=paramKrivoshyp["Висота@Ескіз"]/2
        self.d["Товщина@Бобышка-
Вытянуть1"]=paramKrivoshyp["Ширина@Профіль"]
        # ширина залежить від маси

class BalansirVZbori(SWmodelASM):
    """Клас балансира в зборі"""
    fileName="Балансир в зборі"

class Krivoshypy(SWmodelASM):
    """Клас кривошипів з валом в зборі"""
    fileName="Кривошипи"

class TraversaVZbori(SWmodelASM):
    """Клас траверси з шатунами в зборі"""
    fileName="Траверса з шатунами"

class Karkas(SWmodelASM):
    """Клас каркасу в зборі"""
    fileName="Каркас"

# pu=PumpingUnit()
# pu.create()
# pu.rebuildModel()

# k=KutnykProfil()
# k.create("150x150x12")
# print k.d["Зовнішній радіус"]
```

ДОДАТОК И
Моделі РЗ для Abaqus/CAE

Доступні також в GitHub (vkopey/ThreadsAbaqus: Abaqus/CAE python scripts for modelling threaded connections of oil and gas equipment. URL: http://github.com/vkopey/ThreadsAbaqus).

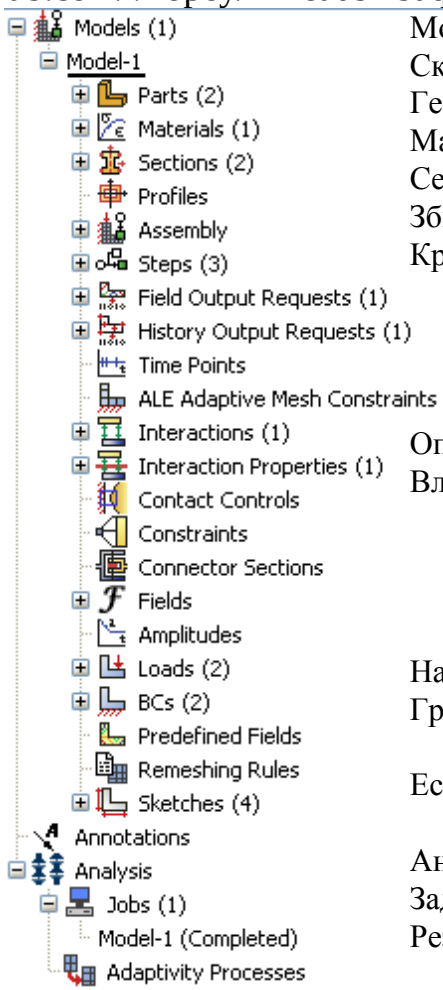


Рисунок И.1 – Дерево побудови моделі муфтового РЗ НКТ в Abaqus 6.8

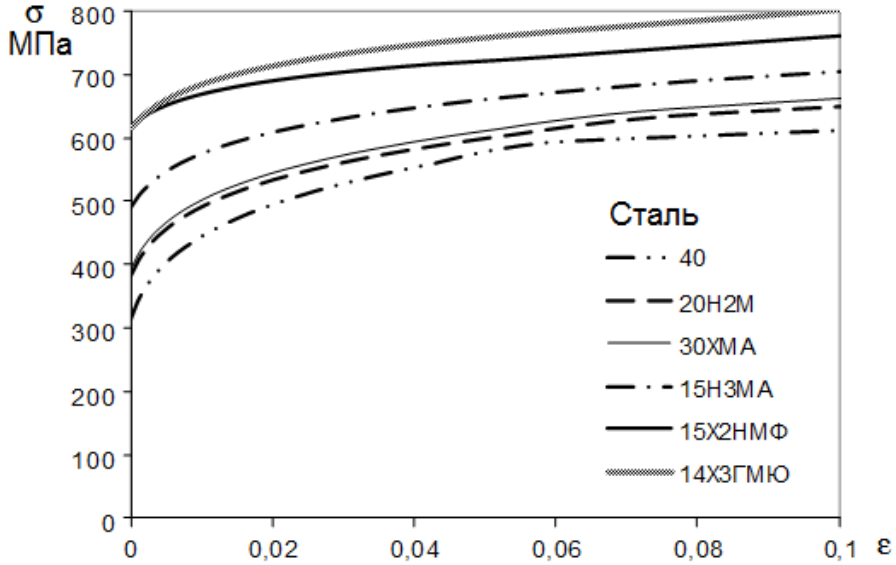
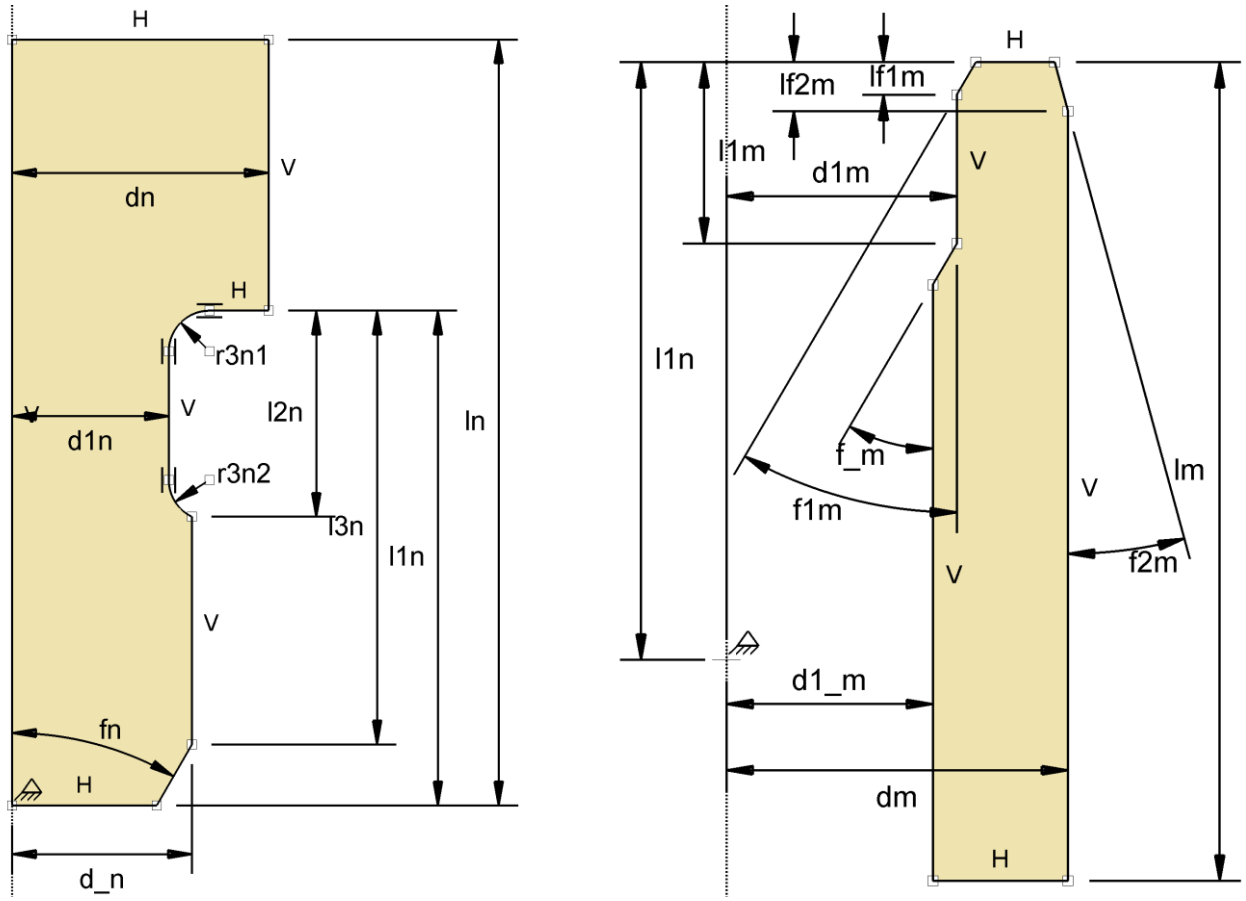
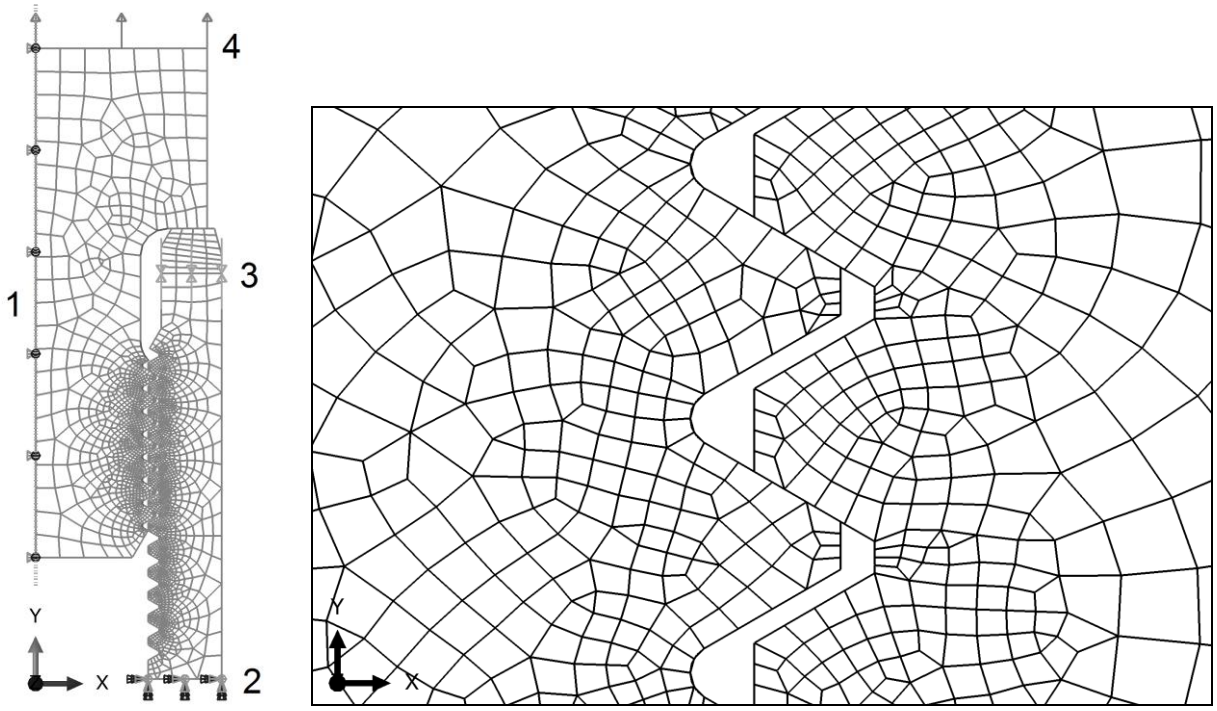


Рисунок И.2 – Моделі істинних діаграм деформування сталей для ШН і муфт



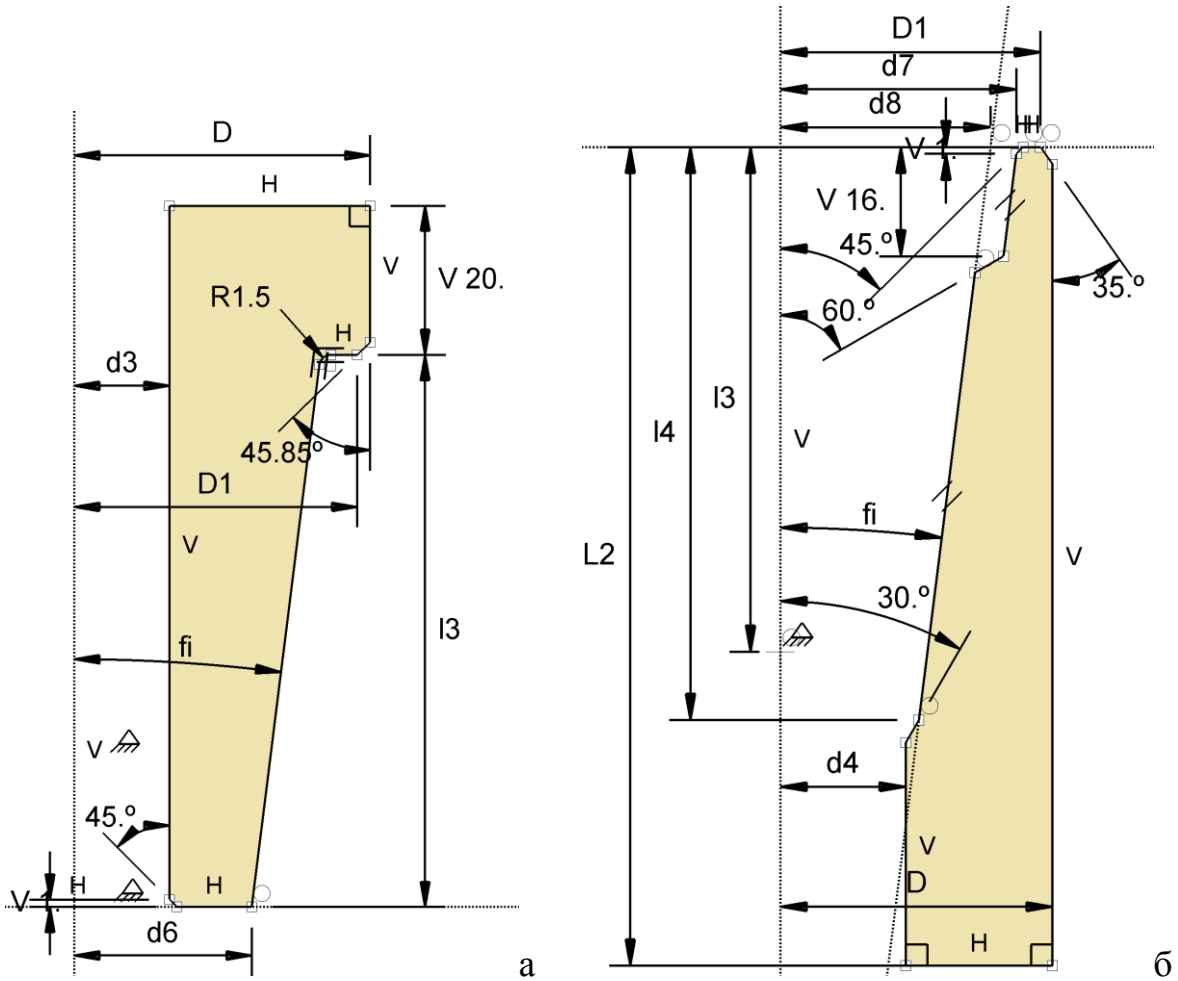
а – заготовки ніпеля, б – заготовки муфти, в – для "вирізання" профілю різьби ніпеля, г – для "вирізання" профілю різьби муфти

Рисунок И.3 – Ескізи для побудови деталей РЗ ШН



Граничні умови: 1 – (переміщення Ux=0); 2 – (Ux=0, Uy=0); 3- Δ; 4 – тиск

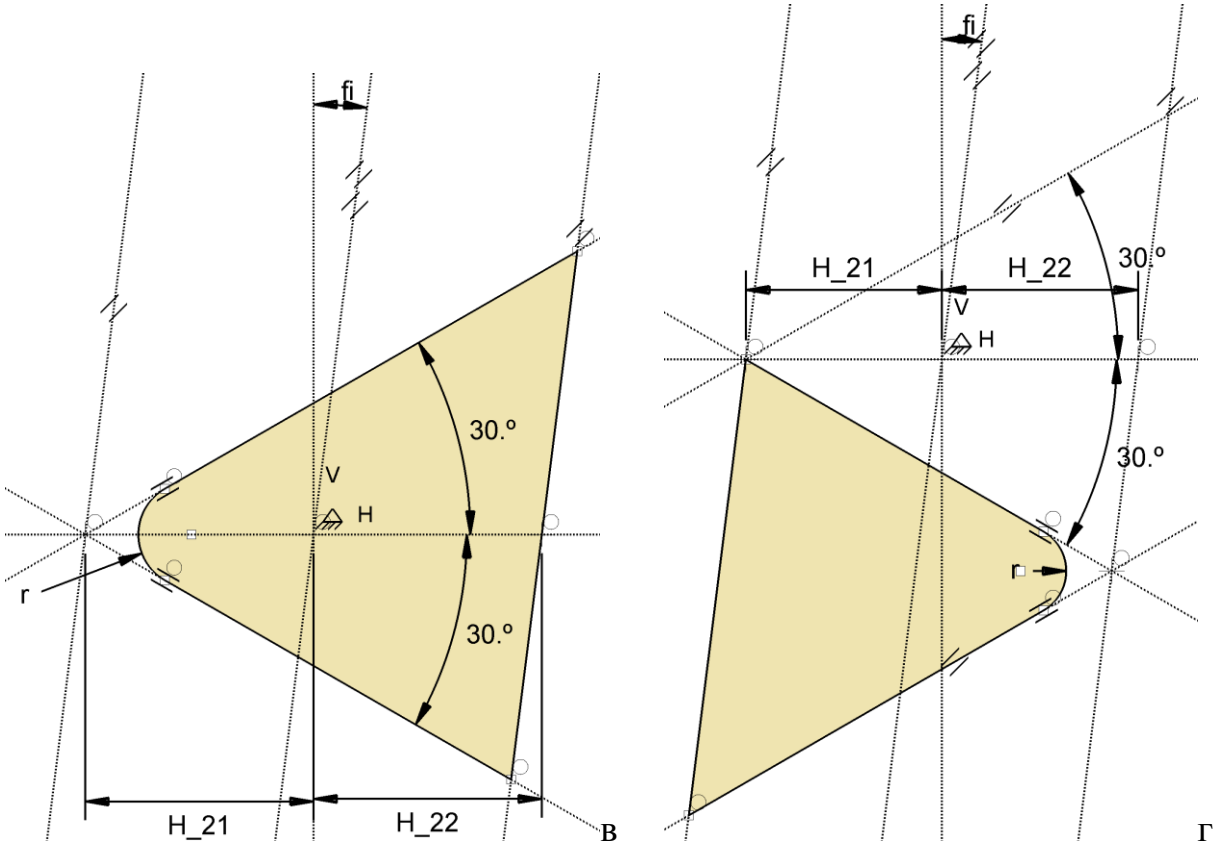
Рисунок И.4 – Осесиметрична СЕМ муфтового РЗ ШН діаметром 19 мм



а б

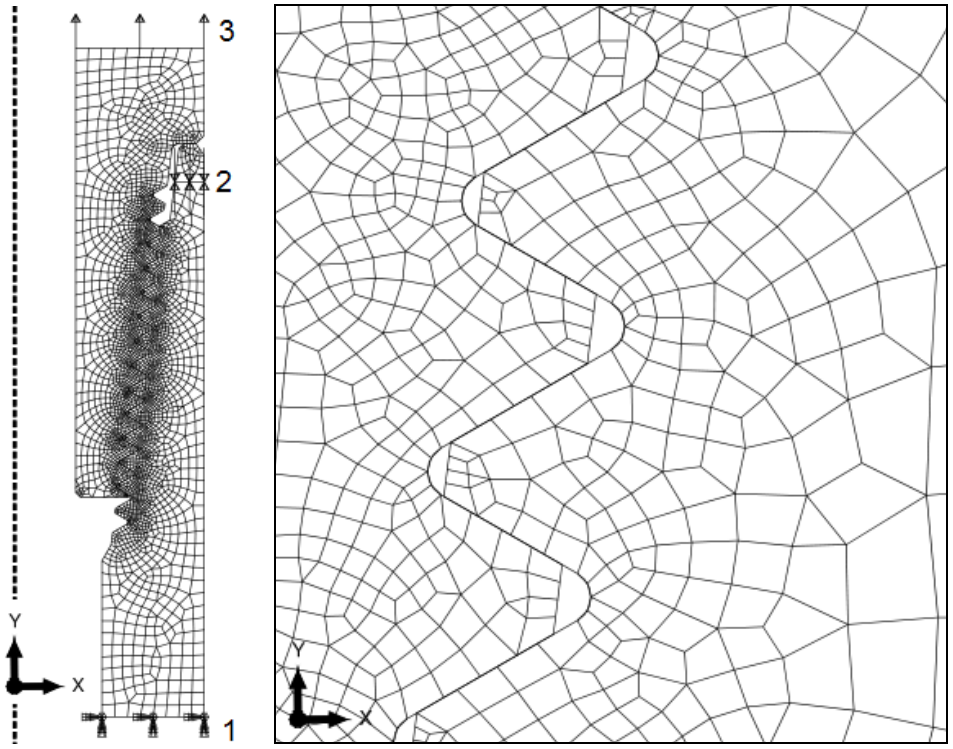
а – заготовки ніпеля, б – заготовки муфти

Рисунок И.5, аркуш 1 – Ескізи для побудови деталей замкового РЗ



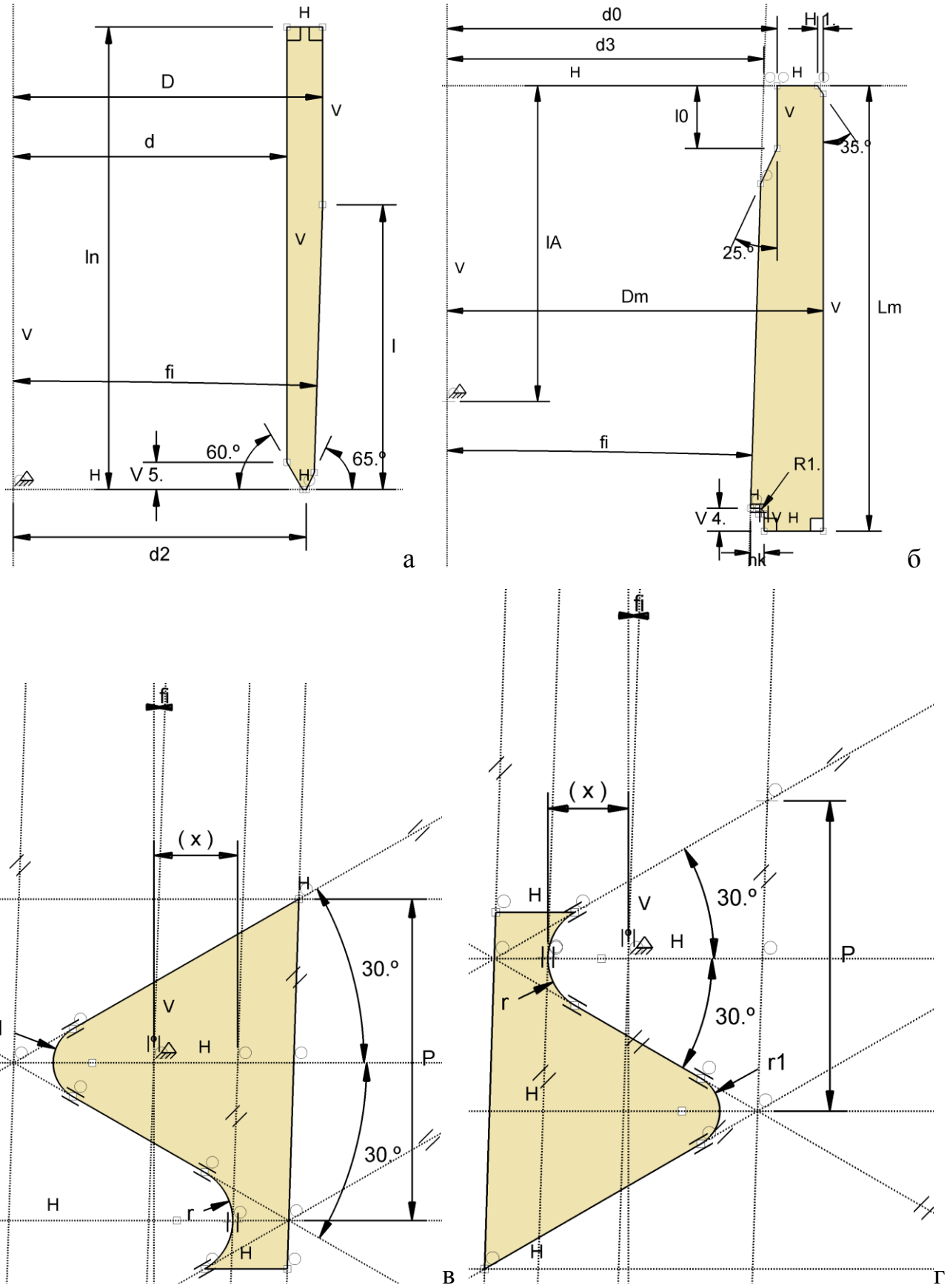
в – для "вирізання" профілю різьби ніпеля, г – для "вирізання"профілю різьби муфти

Рисунок И.5, аркуш 2



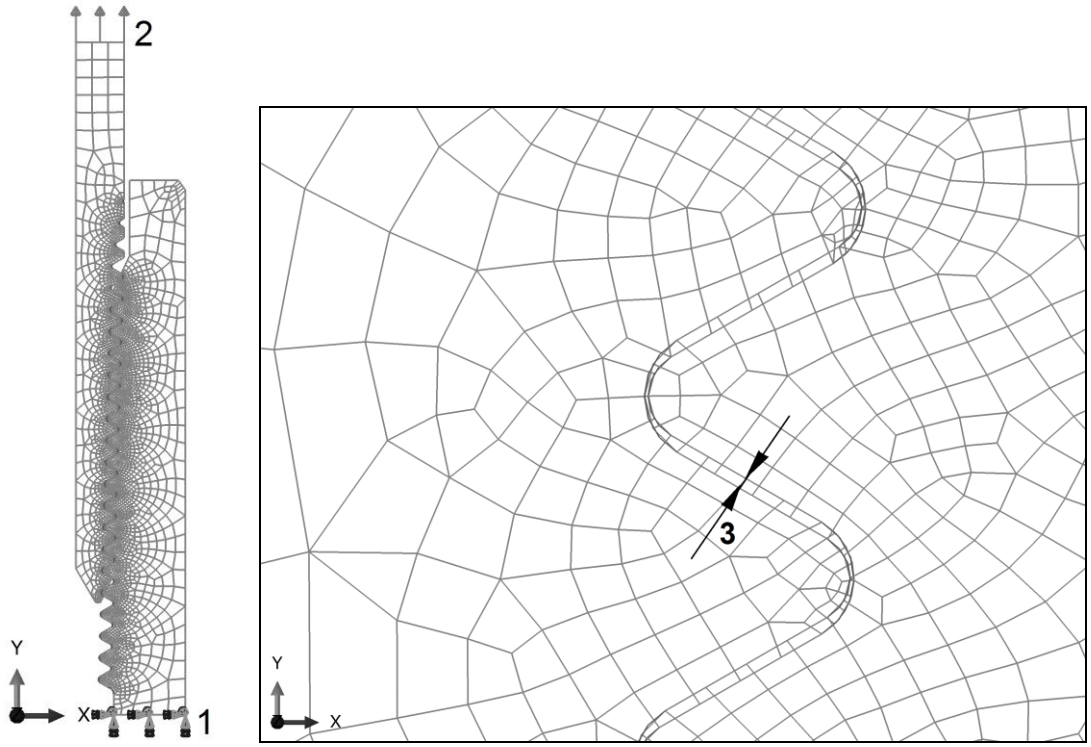
Граничні умови: 1 – (переміщення Ux=0, Uy=0); 2 – Δ; 3 – тиск

Рисунок И.6 – Осесиметрична СЕМ замкового РЗ



а – заготовки ніпеля, б – заготовки муфти, в – для "вирізання"профілю різьби ніпеля, г – для "вирізання"профілю різьби муфти

Рисунок И.7 – Ескізи для побудови деталей РЗ НКТ



Граничні умови: 1 – (переміщення Ux=0, Uy=0); 2 – тиск, 3 – Interference

Рисунок И.8 – Осесиметрична СЕМ муфтового РЗ НКТ

Макроси для побудови СЕМ РЗ в Abaqus

Вміст пакету:

- tools.py – компоненти для побудови моделі;
- gost13877_96.py – модель муфтового РЗ ШН (ГОСТ 13877-96);
- gost5286_75.py – модель замкового РЗ (ГОСТ 5286-75);
- gost633_80.py – модель муфтового РЗ НКТ (ГОСТ 633-80);
- main.py – розрахунок муфтового РЗ ШН;
- main2.py – розрахунок замкового РЗ;
- main3.py – розрахунок РЗ НКТ.

Лістинг И.1 – tools.py

```
# -*- coding: cp1251 -*-
from math import *
from part import *
from material import *
```

```
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
import regionToolset
from visualization import *
from connectorBehavior import *
import subprocess

if _my_thread_model==2: openMdb(pathName='C:/Temp/zamok.cae')
if _my_thread_model==3: openMdb(pathName='C:/Temp/nkt.cae')
model=mdb.models['Model-1']

class Dim:
    "Клас описує поняття розміру"
    n=0.0 # номінальний розмір
    ei=0.0 # нижнє відхилення
    es=0.0 # верхнє відхилення
    v=0.0 # дійсне значення
    def __init__(self,*x):
        "конструктор, x-кортеж"
        self.n=x[0][0]
        self.ei=x[0][1]
        self.es=x[0][2]
    def min(self):
        "повертає мінімальний розмір"
        return self.n+self.ei
    def max(self):
        "повертає максимальний розмір"
        return self.n+self.es
class Line(object):
    '''Лінія (відрізок або пряма)
    p1,p2 - перша і друга точки
    len - довжина
    angle - кут до 0X в градусах
    '''
    def __init__(self,p1,p2=None,len=None,angle=None):
        self.p1=p1
        x1,y1=self.p1[0],self.p1[1]
        if len==None and p2==None:
            len=1.0
        if angle!=None:
            angle=radians(angle)
        if p2==None:
            self.p2=(x1+len*cos(angle),y1+len*sin(angle))
        else:
            self.p2=p2
```

```
        x2,y2=self.p2[0],self.p2[1]
        #коефіцієнти рівняння Ax+By+C=0
        self.A=y1-y2
        self.B=x2-x1
        self.C=x1*y2-x2*y1
    def points(self):
        '''повертає x1,y1,x2,y2'''
        return self.p1[0],self.p1[1],self.p2[0],self.p2[1]
    def angleOX(self):
        '''кут нахилу до ox в радіанах'''
        return atan(-self.A/self.B)
    def len(self):
        '''довжина'''
        x1,y1,x2,y2=self.points()
        return sqrt((y2-y1)**2+(x2-x1)**2)
    def x(self,y):
        '''координата x точки лінії за координатою y'''
        x1,y1,x2,y2=self.points()
        return (y-y1)*(x2-x1)/(y2-y1)+x1#x
    def y(self,x):
        '''координата y точки лінії за координатою x'''
        x1,y1,x2,y2=self.points()
        return (x-x1)*(y2-y1)/(x2-x1)+y1#y
    def mpoint(self):
        '''середня точка'''
        x1,y1,x2,y2=self.points()
        return ((x2-x1)/2+x1,(y2-y1)/2+y1)
    def dist(self,point):
        '''відстань від лінії до точки point'''
        x,y=point[0],point[1]
        return abs((self.A*x+self.B*y+self.C)/sqrt(self.A**2+self.B**2))
    def cros_point(self,line):
        '''точка перетину з лінією line'''
        x=(self.B*line.C-line.B*self.C)/(self.A*line.B-line.A*self.B)
        y=(self.C*line.A-line.C*self.A)/(self.A*line.B-line.A*self.B)
        return (x,y)
    def drawAbaqus(self,sketch):
        '''рисувати в Abaqus'''
        g=sketch.Line(point1=self.p1, point2=self.p2)
        return g

class N_angle(object):
    '''N-кутник зі скругленнями
    L - список ліній (по порядку)
    R - список радіусів скруглень (по порядку).
    Наприклад, R[0] - скруглення між L[0] і L[1]
    Приклад:
    L1=Line(p1=(1.0,1.0),len=2.0,angle=170)
    cp=Line(p1=L1.p1,angle=90+30).cros_point(Line(p1=L1.p2,angle=90-30))
    L2=Line(p1=L1.p1,p2=cp)
    L3=Line(p1=L1.p2,p2=cp)
```

```
    T1=N_angle([L1,L2,L3],[0.05,0.05,0.05])
    '''
    def __init__(self,L,R):
        self.N=len(L) # кількість ліній (кутів)
        L.append(L[0]) # добавити в список ліній першу: [L1,L2,L3,L1]
        self.V=range(self.N) # список вершин
        for i in range(self.N):
            self.V[i]=self.Vert(L[i],L[i+1],R[i])
    def Vert(self,La,Lb,R):
        '''повертає вершину в форматі:
        (точка, перша лінія, друга лінія, радіус скруглення).
        Якщо перша точка першої лінії співпадає з першою
        або другою точкою другої лінії, то вона є вершиною'''
        if La.p1==Lb.p1 or La.p1==Lb.p2:
            V=(La.p1,La,Lb,R)
        else: V=(Lb.p2,La,Lb,R)
        return V
    def drawAbaqus(self,sketch):
        '''рисувати в Abaqus'''
        g=range(self.N)
        for i in range(self.N): # рисувати лінії
            g[i]=self.V[i][1].drawAbaqus(sketch)
        g.append(g[0]) # додати в список ліній першу: [g1,g2,g3,g1]
        for i in range(self.N): # рисувати скруглення
            if self.V[i][3]!=0: # якщо радіус скруглення не 0
                sketch.FilletByRadius(curve1=g[i], curve2=g[i+1],
nearPoint1=self.V[i][1].mpoint(), nearPoint2=self.V[i][2].mpoint(),
radius=self.V[i][3])

class Material:
    '''Клас описує поняття матеріалу
    В Abaqus задається істинна діаграма деформування (див.Stress and strain
measures)
    E - модуль пружності, Па
    mu - коефіцієнт Пуассона
    st - границя текучості, Па
    et - деформація для st
    sb - істинна границя міцності, Па (sv - умовна границя)
    eb - істинна деформація, яка відповідає границі міцності
    delta - відносне видовження
    psi - відносне звуження
    '''
    def __init__(self,E,mu,st,sv,delta,psi):
        '''конструктор'''
        self.E=E # модуль пружності
        self.mu=mu # коефіцієнт Пуассона
        self.st=st # границя текучості
        self.et=st/E # деформація для st
        self.delta=delta/100.0 # відносне видовження після розриву
        self.psi=psi/100.0 # відносне звуження після розриву
        k=0.4 # коефіцієнт(eb=(0.1...0.4,0.2...0.8)delta)
```

```
        self.sv=sv # границя міцності
        self.sb=sv*(1+k*self.delta) # істинна границя міцності
        self.eb=log(1+k*self.delta) # істинна деформація, яка відповідає
границі міцності
        # істинне напруження і деформація в момент руйнування
        self.sk=0.8*self.sv/(1-self.psi) # 0.8-коефіцієнт руйнуючого
навантаження
        #self.sk=self.sv*(1+1.35*self.psi)
        self.ek=log(1/(1-self.psi))
    def bilinear(self):
        '''Повертає словник елестичних і пластичних властивостей'''
        return {'el':((self.E,self.mu),),
                'pl':((self.st,0.0), # білінійна залежність
                (self.sb,self.eb))} # або (self.sk,self.ek)
    def e(self,s,n):
        '''Степенева залежність e(s)
        n - степінь
        '''
        return self.et*(s/self.st)**n
    def power(self,k):
        '''Повертає словник елестичних і пластичних властивостей
        k - кількість ліній для апроксимації пластичної ділянки (2,4,8...)
        '''
        # n визначається з умови проходження через точку (eb+et,sb),
n=6...10
        n=log((self.eb+self.et)/self.et)/log(self.sb/self.st)
        ds=self.sb-self.st
        # степенева залежність
        k_=float(k)
        s_e=[(self.st+i/k_*ds,self.e(self.st+i/k_*ds,n)-self.et) for i in
range(0,k+1)]
        #s_e=[(self.st+i*ds,self.e(self.st+i*ds,n)-self.et) for i in
[0.0,0.25,0.5,0.75,1.0]]
        s_e.append((self.sk,self.ek)) # додати точку руйнування
        return {'el':((self.E,self.mu),),
                'pl':tuple(s_e)}

matlib={
'40':Material(E=210000.0e+6,mu=0.28,st=314.0e+6,sv=559.0e+6,delta=16.0,psi=
45.0),
'40fesafe':Material(E=200000.0e+6,mu=0.33,st=314.0e+6,sv=559.0e+6,delta=16.
0,psi=45.0),
'20Н2М':Material(E=210000.0e+6,mu=0.28,st=382.0e+6,sv=588.0e+6,delta=21.0,p
si=56.0),
'30ХМА':Material(E=210000.0e+6,mu=0.28,st=392.0e+6,sv=598.0e+6,delta=20.0,p
si=62.0),
'15Н3МА':Material(E=210000.0e+6,mu=0.28,st=490.0e+6,sv=637.0e+6,delta=22.0,
psi=60.0),
'15Х2НМФ':Material(E=210000.0e+6,mu=0.28,st=617.0e+6,sv=686.0e+6,delta=16.0
,psi=63.0),
```

```
'15Х2ГМФ':Material(E=210000.0e+6,mu=0.28,st=617.0e+6,sv=686.0e+6,delta=16.0
,psi=63.0),
'14Х3ГМЮ':Material(E=210000.0e+6,mu=0.28,st=617.0e+6,sv=725.0e+6,delta=16.0
,psi=63.0),
}
steel20={'el':((210000000000.0, 0.28), ),
               'pl':((620000000.0, 0.0),
                     (640000000.0, 0.02),
                     (800000000.0, 0.04),
                     (860000000.0, 0.08),
                     (864000000.0, 0.11))}
steel45={'el':((210000000000.0, 0.28), ),
               'pl':((620000000.0, 0.0),
                     (640000000.0, 0.02),
                     (800000000.0, 0.04),
                     (860000000.0, 0.08),
                     (864000000.0, 0.11))}

def delCutExtrude():
    "Знищує усі елементи, назва яких починається з 'Cut extrude' і
'Partition face-1'"
    if model.parts['Part-2'].features.has_key('Partition face-1'):
        del model.parts['Part-2'].features['Partition face-1']  # знищує
Partition face
    for f in model.parts['Part-1'].features.values():
        if f.name[:11]=='Cut extrude': model.parts['Part-
1'].deleteFeatures((f.name,))
    for f in model.parts['Part-2'].features.values():
        if f.name[:11]=='Cut extrude': model.parts['Part-
2'].deleteFeatures((f.name,))
def set_values(sketch,p):
    '''
    Присвоює значення параметрам ескізу.
    Приклад:
    par={'aint':0,'aext':0,'Rint':0,'Rext':dn.v,'len':l1n.v+20}
    set_values(sketch='nipple',p=par)
    '''
    s=model.sketches[sketch]
    for k,v in p.iteritems():
        s.parameters[k].setValues(expression=str(v))
def set_values2(sketch,base,p):
    '''
    Присвоює значення параметрам ескізу.
    Приклад:
    par={'aint':0,'aext':0,'Rint':0,'Rext':dn.v,'len':l1n.v+20}
    set_values2(sketch='nipple',base='Quad_h',p=par)
    '''
    s=model.ConstrainedSketch(name=sketch, objectToCopy=mdb.models['Model-
1'].sketches[base])
    for k,v in p.iteritems():
        s.parameters[k].setValues(expression=str(v))
```

```
def part_builder(part,sketch='Sketch-1',vector=(0.0,0.0),oper='shell'):
    '''Додає до деталі виріз або поверхню задану ескізом
    Приклад:
    part_builder(p, 'groove', (0,0), 'cut')
    '''
    s=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
    part.projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s)
    s.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
    s.retrieveSketch(sketch=model.sketches[sketch])
    s.move(objectList=s.geometry.values(),vector=vector)
    if oper=='shell': part.Shell(sketch=s)
    if oper=='cut': part.Cut(sketch=s)
    del s
def createPart(n,s):
    '''Створює деталь
    n - ім'я
    s - ескіз'''
    model.Part(dimensionality=AXISYMMETRIC, name=n, type=DEFORMABLE_BODY)
    model.parts[n].BaseShell(sketch=model.sketches[s])
def createPart3D(frompart,sketch,part):
    '''Створює 3D деталь на основі осесиметричної деталі
    Створюється ескіз sketch як проекція осесиметричної деталі'''
    s=model.ConstrainedSketch(name=sketch,sheetSize=200.)
    tmp=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
    #s.sketchOptions.setValues(viewStyle=AXISYM)

model.parts[frompart].projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=tmp)
    ln=tmp.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
    tmp.FixedConstraint(entity=ln)
    tmp.assignCenterline(line=ln)
    s.retrieveSketch(sketch=tmp)
    #s.sketchOptions.setValues(constructionGeometry=ON)
    p=model.Part(dimensionality=THREE_D, name=part, type=DEFORMABLE_BODY)
    p.BaseSolidRevolve(angle=360.0, flipRevolveDirection=OFF, sketch=s)
def createCut(Part,Sketch,Begin,P,Fi,Len,X,Y,dx,dy):
    '''Створює частину профілю різьби
    Part - деталь (рядок)
    Sketch - ескіз (рядок)
    Begin - початок різьби (ціле)
    P - крок різьби (дійсне)
    Fi - кут конуса конічної різьби (градуси)
    Len - довжина різьби (дійсне)
    X,Y - початкові координати центра профілю
    dx - радіальний напрямок подачі (+1 - вправо, -1 - вліво)
    dy - осьовий напрямок подачі (+1 - вверх, -1 - вниз)
    '''
    # можна це зробити також за допомогою LinearInstancePattern
    i=Begin # номер витка (0-перший)
    while i*P<=Len: # довжина різьби
        s=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
```

```
model.parts[Part].projectReferencesOntoSketch(filter=COPLANAR_EDGES,
sketch=s)
        s.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
        s.retrieveSketch(sketch=model.sketches[Sketch])
        s.delete(objectList=(s.vertices.findAt((0.0,0.0),), ))

s.move(objectList=s.geometry.values(),vector=(X+dx*P*tan(Fi*pi/180)*i,Y+dy*
P*i))
        model.parts[Part].Cut(sketch=s)
        del s
        i=i+1
    return i-1
def createPartition(part,offset):
    '''Ділить поверхню деталі лінією (0.0,offset,0.0),
(2000.0,offset,0.0)'''
    model.parts[part].PartitionFaceByShortestPath(faces=
        model.parts[part].faces[0], point1=(0.0,offset,0.0),
point2=(2000.0,offset,0.0))
def createPartition3D(part,offset):
    '''Ділить об'єм деталі площиною зміщеною від XZPLANE на відстань
offset'''
    p = model.parts[part]
    p.DatumPlaneByPrincipalPlane(principalPlane=XZPLANE, offset=offset)
    dp = p.datums.values()[-1]
    p.PartitionCellByDatumPlane(datumPlane=dp, cells=p.cells[0])
def createMaterial(n,et,pt,kinematic=False):
    '''Створює матеріал
    n - ім'я
    et - пружні характеристики
    pt - пластичні характеристики
    kinematic - модель зміцнення (True - кінематичне, False - ізотропне)
    '''
    m=model.Material(name=n)
    m.Elastic(table=et)
    if kinematic:
        m.Plastic(hardening=KINEMATIC, table=pt)# pt - тільки дві точки !
    else:
        m.Plastic(table=pt)

def createSectionAssign(n,m,p):
    '''Створює і присвоює секції деталі
    n - ім'я
    m - матеріал
    p - деталь
    '''
    model.HomogeneousSolidSection(material=m, name=n, thickness=None)
    model.parts[p].SectionAssignment(region=Region(
        faces=model.parts[p].faces), sectionName=n)
def createSectionAssign3D(n,m,p):
    '''Створює і присвоює секції деталі
```

```
    n - ім'я
    m - матеріал
    p - деталь
    '''
    if model.parts[p].sectionAssignments:
        del model.parts[p].sectionAssignments[0]
    model.HomogeneousSolidSection(material=m, name=n, thickness=None)

model.parts[p].SectionAssignment(region=Region(cells=model.parts[p].cells),
sectionName=n)

def createAssemblyInstance(n,p):
    '''Створює елемент збірки
    n - ім'я
    p - деталь
    '''
    model.rootAssembly.Instance(dependent=OFF, name=n, part=model.parts[p])
def createAssemblyInstance3D(n,p):
    '''Створює елемент збірки
    n - ім'я
    p - деталь
    '''
    #model.rootAssembly.DatumCsysByDefault(CARTESIAN)
    model.rootAssembly.Instance(dependent=OFF, name=n, part=model.parts[p])
def createStep(n,pr):
    '''Створює крок
    n - ім'я
    pr - попередній крок
    '''
    model.StaticStep(name=n, previous=pr)
def createContactSet(n,i,ep):
    '''Створює набір для контакту
    n - ім'я
    i - елемент зборки
    ep - кортеж точок кромок не для контакту
    '''
    model.rootAssembly.regenerate()
    ae=model.rootAssembly.instances[i].edges
    e=ae.findAt(*ep) # *ep - розпакування кортежу
    p=[x.pointOn for x in ae if x not in e]
    model.rootAssembly.Set(name=n,edges=ae.findAt(*p))
def createContactSet3D(n,i,ep):
    '''Створює набір для контакту
    n - ім'я
    i - елемент збірки
    ep - кортеж точок кромок не для контакту
    '''
    model.rootAssembly.regenerate()
    ae=model.rootAssembly.instances[i].faces
    e=ae.findAt(*ep) # *ep - розпакування кортежу
    p=[x.pointOn for x in ae if x not in e]
```

```
    model.rootAssembly.Set(name=n,faces=ae.findAt(*p))
def createContactProperty():
    '''Створює властивості контакту'''
    model.ContactProperty('IntProp-1')
    model.interactionProperties['IntProp-1'].TangentialBehavior(
        dependencies=0, directionality=ISOTROPIC,
elasticSlipStiffness=None,
        formulation=PENALTY, fraction=0.005, maximumElasticSlip=FRACTION,
        pressureDependency=OFF, shearStressLimit=None,
slipRateDependency=OFF,
        table=((0.05, ), ), temperatureDependency=OFF)
    model.interactionProperties['IntProp-1'].NormalBehavior(
        allowSeparation=ON,
constraintEnforcementMethod=DEFAULT,pressureOverclosure=HARD)
def createContact():
    '''Створює контакт'''
    sm=model.rootAssembly.sets['Master']
    ss=model.rootAssembly.sets['Slave']
    model.SurfaceToSurfaceContactStd(adjustMethod=NONE,
        clearanceRegion=None, createStepName='Step-1', datumAxis=None,
enforcement=
        SURFACE_TO_SURFACE, initialClearance=OMIT,
interactionProperty='IntProp-1',
        interferenceType=SHRINK_FIT, master=Region(side1Edges=sm.edges),
name='Int-1',
        slave=Region(side1Edges=ss.edges), sliding=SMALL,
surfaceSmoothing=NONE,
        thickness=ON)
def createContact3D():
    '''Створює контакт'''
    sm=model.rootAssembly.sets['Master']
    ss=model.rootAssembly.sets['Slave']
    model.SurfaceToSurfaceContactStd(adjustMethod=NONE,
        clearanceRegion=None, createStepName='Step-1', datumAxis=None,
        initialClearance=OMIT, interactionProperty='IntProp-1',
master=Region(
        side1Faces=sm.faces), name='Int-1', slave=Region(
        side1Faces=ss.faces), sliding=FINITE, smooth=0.2)
def createBCSet(n,i,ep):
    '''Створює набір для граничної умови
    n - ім'я
    i - елемент зборки
    ep - кортеж точок кромок для граничної умови
    '''

s=model.rootAssembly.Set(edges=model.rootAssembly.instances[i].edges.findAt
(ep), name=n)
def createBCSet3D(n,i,ep):
    '''Створює набір для граничної умови
    n - ім'я
    i - елемент зборки
```

```
    ep - кортеж точок кромок для граничної умови
    '''

s=model.rootAssembly.Set(faces=model.rootAssembly.instances[i].faces.findAt
(ep), name=n)

def createBC_Pressure(step):
    '''Створює тиск. Приклад:
    createBC_Pressure([('Step-1',-1.0),('Step-2',-
276.0e+6*d0.v**2/dn.v**2)])'''
    s=model.rootAssembly.sets['Pressure']
    model.Pressure(amplitude=UNSET, createStepName=step[0][0],
        distributionType=UNIFORM, field='', magnitude=step[0][1],
name='Pressure',
        region=Region(side1Edges=s.edges))
    for x in step:
        model.loads['Pressure'].setValuesInStep(magnitude=x[1],
stepName=x[0])
def createBC_Axis():
    '''Створює граничні умови на осі (для осесиметричних моделей)'''
    s=model.rootAssembly.sets['Axis']
    model.DisplacementBC(amplitude=UNSET, createStepName='Step-1',
    distributionType=UNIFORM, fieldName='', fixed=OFF, localCsys=None,
name=
    'Axis', region=Region(edges=s.edges), u1=0.0, u2=UNSET, ur3=0.0)
def createBC_Encastre():
    '''Створює закріплення'''
    s=model.rootAssembly.sets['Encastre']
    model.EncastreBC(createStepName='Step-1', name='Encastre',
region=Region(edges=s.edges))
def createBC_BoltLoad(part,point,value):
    '''Створює BoltLoad. Приклад:
    createBC_BoltLoad('Part-2-1',em3,-0.1)'''
    model.BoltLoad(boltMethod=ADJUST_LENGTH, createStepName='Step-1',
datumAxis=
        model.rootAssembly.instances[part].datums[1],
        magnitude=value, name='BoltLoad', region=Region(
        side1Edges=model.rootAssembly.instances[part].edges.findAt((point,
))))
def createBC_Pressure3D(step):
    '''Створює тиск. Приклад:
    createBC_Pressure([('Step-1',-1.0),('Step-2',-
276.0e+6*d0.v**2/dn.v**2)])'''
    s=model.rootAssembly.sets['Pressure']
    model.Pressure(amplitude=UNSET, createStepName=step[0][0],
        distributionType=UNIFORM, field='', magnitude=step[0][1],
name='Pressure',
        region=Region(side1Faces=s.faces))
    for x in step:
        model.loads['Pressure'].setValuesInStep(magnitude=x[1],
stepName=x[0])
```

```
def createBC_Encastre3D():
    '''Створює закріплення'''
    s=model.rootAssembly.sets['Encastre']
    model.EncastreBC(createStepName='Step-1', name='Encastre',
region=Region(faces=s.faces))
def createBC_BoltLoad3D(part,point,value):
    '''Створює BoltLoad. Приклад:
    createBC_BoltLoad('Part-2-1',em3,-0.1)'''
    model.BoltLoad(boltMethod=ADJUST_LENGTH, createStepName='Step-1',
datumAxis=
        model.rootAssembly.instances[part].datums[1],
        magnitude=value, name='BoltLoad', region=Region(
        side1Faces=model.rootAssembly.instances[part].faces.findAt((point,
))))
def createMesh():
    '''Створює сітку'''
    model.rootAssembly.seedPartInstance(deviationFactor=0.1,
        regions=(model.rootAssembly.instances['Part-1-1'],
        model.rootAssembly.instances['Part-2-1']), size=2.6)
    sm=model.rootAssembly.sets['Master']
    ss=model.rootAssembly.sets['Slave']
    model.rootAssembly.seedEdgeByNumber(edges=sm.edges, number=4)
    model.rootAssembly.seedEdgeByNumber(edges=ss.edges, number=4)
    model.rootAssembly.generateMesh(regions=(
        model.rootAssembly.instances['Part-1-1'],
        model.rootAssembly.instances['Part-2-1']))
def createMesh3D():
    '''Створює сітку'''
    model.rootAssembly.setMeshControls(elemShape=TET, regions=
        model.rootAssembly.instances['Part-3-1'].cells+\
        model.rootAssembly.instances['Part-4-1'].cells, technique=FREE)
    model.rootAssembly.seedPartInstance(deviationFactor=0.1,
        regions=(model.rootAssembly.instances['Part-3-1'],
        model.rootAssembly.instances['Part-4-1']), size=2.8)
    model.rootAssembly.generateMesh(regions=(
        model.rootAssembly.instances['Part-3-1'],
        model.rootAssembly.instances['Part-4-1']))
def createEdgesSet(n, i, p, exclude=False):
    '''Створює Set з ребер (ребро визначається довільною точкою на ньому)
    exclude=True - з усіх ребер крім заданих (тільки для OdbSet!)
    createEdgesSet(n='Set-6',i='Part-1-1',p=((enr1, ),(en1, )))'''
    if exclude==True:
        edges=model.rootAssembly.instances[i].edges[:]
        xEdges=model.rootAssembly.instances[i].edges.findAt(*p)
        model.rootAssembly.Set(edges=edges, xEdges=xEdges, name=n)
    else:
        edges=model.rootAssembly.instances[i].edges.findAt(*p)
        model.rootAssembly.Set(edges=edges, name=n)
def createVerticesSet(n, i, p):
    '''Створює Set з вершин
    createVerticesSet(n='Set-7',i='Part-1-1',p=(((0,0,0), ),))'''
```

```
model.rootAssembly.Set(vertices=model.rootAssembly.instances[i].vertices.fi
ndAt(*p), name=n)
def createSet(n, r):
    '''Створює Set з Region
    r = regionToolset.Region(edges=e,vertices=v,xEdges=xe,xVertices=xv)
    createSet(n='Set-8', r=r)'''
    model.rootAssembly.Set(region=r, name=n)
def delItems():
    '''Знищує елементи'''
    if model.steps.has_key('Step-2'): del model.steps['Step-2']
    if model.steps.has_key('Step-1'): del model.steps['Step-1']
def createJobSubmit():
    '''Створює задачу і виконує її'''
    myJob = mdb.Job(name=model.name, model=model.name)
    myJob.submit()
    # Чекати поки задача не буде розв'язана
    myJob.waitForCompletion()
def createObdNodeSet(coords,name='MYSET',prt='Part-1-
1',item_type='vertex'):
    '''Створює ObdNodeSet за заданим координатами ребром або вершиною'''
    if item_type=='edge':

_item=model.rootAssembly.instances[prt].edges.findAt(coordinates=coords)
        #або getClosest()
    if item_type=='vertex':

_item=model.rootAssembly.instances[prt].vertices.findAt(coordinates=coords)
    _nodes=_item.getNodes() #model.rootAssembly.sets['Set-6'].nodes
    _nodeLabels=[]
    for x in _nodes:
        _nodeLabels.append(x.label)

_nset=myOdb.rootAssembly.NodeSetFromNodeLabels(name=name,nodeLabels=((prt.u
pper(), _nodeLabels),))
def readODB_path(path,step,var,intersections=False):
    '''Читає результати з останнього фрейму кроку на заданому шляху з точок
    path - шлях (список точок, >=1)
    step - крок
    var - змінна:
    (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
    (('S', INTEGRATION_POINT, ((INVARIANT, 'Pressure'), )), )
    (('S', INTEGRATION_POINT, ((COMPONENT, 'S11'), )), )
    (('U', NODAL, ((INVARIANT, 'Magnitude'), )), )
    (('U', NODAL, ((COMPONENT, 'U1'), )), )
    (('CPRESS', ELEMENT_NODAL), )
    'D' #коефіцієнт запасу втомної міцності
    intersections=True - додає проміжні точки
    Приклад: readODB_path(path=((1.97212e+001,  3.65000e+001, 0.0),
),step='Step-1',var=var)
    '''
```

```
    pth=session.Path(name='Path-tmp', type=POINT_LIST, expression=path)
#шлях
    if var=='D':
        frame1 = session.scratchOdbs['Model-1.odb'].steps['Session
Step'].frames[0]
        session.viewports['Viewport: 1'].odbDisplay.setFrame(frame=frame1)
        dat=session.XYDataFromPath(name='XYData-1', path=pth,
includeIntersections=intersections, shape=UNDEFORMED, labelType=SEQ_ID)
    else:
        st=myOdb.steps[step].number-1 # індекс кроку
        fr=len(myOdb.steps[step].frames)-1 # індекс останнього фрейму
        #session.viewports['Viewport: 1'].odbDisplay.setFrame(step=st,
frame=fr)
        #session.viewports['Viewport:
1'].odbDisplay.setPrimaryVariable(variableLabel='S',
outputPosition=INTEGRATION_POINT, refinement=(INVARIANT, 'Tresca'))
        dat=session.XYDataFromPath(name='XYData-1', path=pth,
includeIntersections=intersections,shape=UNDEFORMED,
labelType=SEQ_ID,step=st,frame=fr,variable=var) # дані
    res=[] # список результатів
    for x in dat.data:
        res.append(x[1]) # додати до списку результатів
    # видалити тимчасові дані
    del session.paths['Path-tmp']
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set(set,step,var,pos=NODAL):
    '''Читає результати з останнього фрейму кроку на заданій множині
    set - множина
    step - крок
    var - змінна:
    (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
    (('CPRESS', ELEMENT_NODAL), )
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set(set='Cont',step='Step-1',var=var)
    '''
    if pos==NODAL:
        dat=session.xyDataListFromField(odb=myOdb, outputPosition=NODAL,
variable=var, nodeSets=(set.upper(),)) # дані
    if pos==INTEGRATION_POINT:
        dat=session.xyDataListFromField(odb=myOdb,
outputPosition=INTEGRATION_POINT, variable=var, elementSets=(set.upper(),))
#дані
    res=[] # список результатів
    for x in dat: # для всіх вузлів
        n=0
        for k in myOdb.steps.keys(): # для всіх кроків
            n=n+len(myOdb.steps[k].frames) # сумарна кількість фреймів до k
кроку включно
```

```
            if k==step: res.append(x.data[n-1][1]) # додати до списку
результатів
            # data - це ((час,значення),(час,значення)...)
    # видалити тимчасові дані
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set_(set,var):
    '''Повертає список результатів в вузлах заданої множини
    (для змінних fe-safe)
    set - множина
    var - змінна:
    (('LOGLife-Repeats', ELEMENT_NODAL), )
    (('FOS@Life=Infinite', ELEMENT_NODAL), )
    (('%%Failure@Life=5E6-Repeats', ELEMENT_NODAL), )
    Приклад: readODB_set_(set='Set-1',var=var)
    '''
    # отримати дані
    dat=session.xyDataListFromField(odb=myOdb, outputPosition=NODAL,
variable=var, nodeSets=(set.upper(),)) # дані

    res=[] # список результатів
    for x in dat: # для всіх вузлів
        # x.data - це ((час,значення),(час,значення)...)
        res.append(x.data) # дані

    # видалити тимчасові дані
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set2(set,step,var,pos=NODAL):
    '''Читає результати з останнього фрейму кроку на заданій множині
    (менш універсальна альтернатива readODB_set())
    set - множина
    step - крок
    var - змінна:
    ('S','Mises')
    ('S','Pressure')
    ('U','Magnitude')
    ('U','U1')
    ('CPRESS','')
    ('D','') # коефіцієнт запасу втомної міцності
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set2(set='Cont',step='Step-1',var=('S','Mises'))
    '''
    if pos==NODAL:
        s=myOdb.rootAssembly.nodeSets[set.upper()] # множина вузлів
    if pos==INTEGRATION_POINT:
        s=myOdb.rootAssembly.elementSets[set.upper()] # множина елементів
```

```
    if var[0]=='D':
        fo=session.scratchOdbs['Model-1.odb'].steps['Session
Step'].frames[-1].fieldOutputs['D'].getSubset(region=s,position=pos) # дані
    else:
        fo=myOdb.steps[step].frames[-
1].fieldOutputs[var[0]].getSubset(region=s,position=pos) # дані
        #openOdb(r'C:/Temp/Model-1.odb').steps['Step-
1'].frames[4].fieldOutputs['CPRESS'].getSubset(position=NODAL,
region=openOdb(r'C:/Temp/Model-
1.odb').rootAssembly.nodeSets['CONT']).values[0].data
    res=[] # список результатів
    for v in fo.values: # для кожного вузла/елемента
        if var[1]=='Mises': res.append(v.mises) # додати до списку
результатів
        if var[1]=='Pressure': res.append(v.press)
        if var[0]=='U' and var[1]=='Magnitude': res.append(v.magnitude)
        if var[1]=='U1': res.append(v.data.tolist()[0])
        if var[1]=='U2': res.append(v.data.tolist()[1])
        if var[0]=='CPRESS': res.append(v.data)
        if var[0]=='D': res.append(v.data)
    return res # повертає список значень

def runFeSafe(input_odb,input_stlx,output_odb):
    '''Виконує аналіз втомної міцності у fe-safe
    input_odb - назва вхідного файлу результатів Abaqus (без розширення
.odb)
    input_stlx - назва файлу моделі fe-safe (без розширення .stlx)
    output_odb - назва вихідного файлу результатів Abaqus (без розширення
.odb)
    Якщо з FEA моделі імпортуються напруження і деформації, то перед
створенням файлу sltx в налаштуваннях Analysis Options необхідно вибрати
Read strains from FE Models.
    '''
    s=r'd:\Program Files\Safe_Technology\fe-safe\version.6.2\exe\fe-
safe_cl.exe -s j=c:\1\{iodb}.odb b=c:\1\{istlx}.stlx o=c:\1\{oodb}.odb'
    s=s.format(iodb=input_odb, istlx=input_stlx, oodb=output_odb)
    # виконує обчислення в fe-safe та чекає завершення
    subprocess.Popen(s).communicate()

def writeLDFfile(filename,lst):
    '''Змінює вміст файлу визначення навантаження LDF fe-safe
    filename - ім'я файлу
    lst - список рядків-даних
    '''
    f=open(filename, "w")
    s="""
# .ldf file created by fe-safe compliant product 6.2-01[mswin]

INIT
transitions=Yes
END
```

```
# Block number 1
BLOCK n=1, scale=1
lh=0 1 , ds=1, scale=1
lh=0 {x} , ds=2, scale=1
END

""" # шаблон файлу
    s=s.format(x=lst[0]) # вміст файлу з шаблону
    f.write(s)
    f.close()

def SF_field(s1='Step-1',s2='Step-2',Sn=207000000,m=1):
    '''Розраховує поле коефіцієнта запасу втомної міцності за критерієм
Сайнса'''

    S3_2 = myOdb.steps[s2].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MAX_PRINCIPAL)
    S3_1 = myOdb.steps[s1].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MAX_PRINCIPAL)
    Sm3=(S3_2+S3_1)/2
    Sa3=(S3_2-S3_1)/2

    S2_2 = myOdb.steps[s2].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MID_PRINCIPAL)
    S2_1 = myOdb.steps[s1].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MID_PRINCIPAL)
    Sm2=(S2_2+S2_1)/2
    Sa2=(S2_2-S2_1)/2

    S1_2 = myOdb.steps[s2].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MIN_PRINCIPAL)
    S1_1 = myOdb.steps[s1].frames[-
1].fieldOutputs['S'].getScalarField(invariant=MIN_PRINCIPAL)
    Sm1=(S1_2+S1_1)/2
    Sa1=(S1_2-S1_1)/2

    D =(Sn-m*(Sm1+Sm2+Sm3)/3)/power(((power((Sa1-Sa2), 2)+power((Sa2-Sa3),
2)+power((Sa3-Sa1), 2))/2), 0.5)

    scratchOdb = session.ScratchOdb(odb=myOdb)
    sessionStep = scratchOdb.Step(name='Session Step',description='Step for
Viewer non-persistent fields',domain=TIME,timePeriod=1.0)
    sessionFrame = sessionStep.Frame(frameId=0, frameValue=0.0,
description='Session Frame')
    sessionField = sessionFrame.FieldOutput(name='D', description='D',
field=D)
def saveDB(name):
    import shutil,os,shelve
    if not os.path.exists('MyDB'):
        os.mkdir('MyDB')
```

```
    shutil.copyfile(model.name + '.odb', 'MyDB/'+name+'.odb') # копіювати
файл

    my_db = shelve.open("MyDB/mydb") # відкрити файл полиці
    my_db[name]=[mat1,mat2,d,d_n] # записати у полицю під ключем name
    my_db.close() # закрити файл полиці
def readDB(name):
    import shelve
    my_db = shelve.open("MyDB/mydb") # відкрити файл полиці
    print my_db.keys() # вивести список усіх ключів
    if my_db.has_key(name): # якщо є ключ name
        print my_db[name]
    my_db.close() # закрити файл полиці
```

Лістинг И.2 – gost13877_96.py

```
# -*- coding: cp1251 -*-
if _my_thread_model!=1: from tools import *

rod19={'d_n':(27, -0.48, -0.376),#
    'd2_n':(25.35, -0.204, -0.047),#
    'd1_n':(24.25, 0, -0.415),
    'r_n':(0.28, 0, 0.08),
    'dn':(38.1, -0.25, 0.13),
    'd1n':(23.24, -0.13, 0.13),
    'l1n':(36.5, 0, 1.6),
    'l2n':(15, 0.2, 1),
    'l3n':(32, 0, 1.5),
    'l4n':(48, -1, 1.5),
    'r3n':(3, 0, 0.8),
    'd_m':(27, 0, 0.27),
    'd2_m':(25.35, 0, 0.202),
    'd1_m':(24.25, 0, 0.54),
    'dm':(41.3, -0.25, 0.13),
    'd1m':(27.43, 0, 0.25),
    'lm':(102, -1, 1),
    'd0':(19.1,-0.41,0.2),
    'p_n':(2.54,0,0),
    'p_m':(2.54,0,0)}
rod22={'d_n':(27, -0.48, -0.376),#
    'd2_n':(25.35, -0.204, -0.047),#
    'd1_n':(24.25, 0, -0.415),
    'r_n':(0.28, 0, 0.08),
    'dn':(38.1, -0.25, 0.13),
    'd1n':(23.24, -0.13, 0.13),
    'l1n':(36.5, 0, 1.6),
    'l2n':(15, 0.2, 1),
    'l3n':(32, 0, 1.5),
    'l4n':(48, -1, 1.5),
    'r3n':(3, 0, 0.8),
```

```
    'd_m':(27, 0, 0.27),
    'd2_m':(25.35, 0, 0.202),
    'd1_m':(24.25, 0, 0.54),
    'dm':(41.3, -0.25, 0.13),
    'd1m':(27.43, 0, 0.25),
    'lm':(102, -1, 1),
    'd0':(19.1,-0.41,0.2),
    'p_n':(2.54,0,0),
    'p_m':(2.54,0,0)}

rod={19:rod19,22:rod22} # словник типорозмірів штанг
diameter=19 # діаметр штанги
d={} # словник усіх розмірів моделі
for x in rod[diameter].iterkeys():
    d[x]=Dim(rod[diameter][x]) # копіюємо ключі, а значення перетворюємо в
розміри Dim

l_=0 # скорочення муфти при згвинчуванні (0, якщо задано Bolt Load)
#=====================параметри ніпеля штанги==================
d['d_n'] = d['d_n'].min() / 2 # зовнішній діаметр різьби/2
d['d2_n'] = d['d2_n'].min() / 2 # середній діаметр різьби/2
d['d1_n'] = d['d1_n'].min() / 2 # внутрішній діаметр різьби/2!ei*
d['r_n'] = d['r_n'].min() # радіус западин різьби
d['p_n'] = d['p_n'].min() # крок різьби
d['dn'] = d['dn'].min() / 2 # діаметр бурта/2
d['d1n'] = d['d1n'].min() / 2 # діаметр зарізьбової канавки/2
d['l1n'] = d['l1n'].min()+my_iter2 # довжина ніпеля
d['l2n'] = d['l2n'].min()+my_iter2 # довжина зарізьбової канавки
d['l3n'] = d['l3n'].min()+my_iter2 # довжина ніпеля без фаски на різьбі
d['l4n'] = d['l4n'].min()+my_iter2 # довжина ніпеля з буртом
d['r3n'] = d['r3n'].min() # радіус скруглень зарізьбової канавки
d['d0']=d['d0'].min()/2 # діаметр тіла/2
#=====================параметри муфти===========================
d['d_m'] = d['d_m'].max() / 2 # зовнішній діаметр різьби/2!es*
d['d2_m'] = d['d2_m'].max() / 2 # середній діаметр різьби/2
d['d1_m'] = d['d1_m'].max() / 2 # внутрішній діаметр різьби/2
d['p_m'] = d['p_m'].min() # крок різьби
d['dm'] = d['dm'].min() / 2 # зовнішній діаметр/2
d['d1m'] = d['d1m'].max() / 2 # внутрішній діаметр опорної поверхні/2
d['lm'] = d['lm'].min() / 2 +my_iter2 # довжина муфти/2
#================допоміжні параметри=======================
d['dn_']=d['d2_n']+0.25*d['p_n']/tan(30*pi/180) # зовнішній діаметр вершин
трикутника профілю ніпеля
d['ln_']=d['l1n']-d['l2n'] # z-координата першої западини ніпеля (довжина
різьби ніпеля)
d['dm_']=d['d2_m']-0.25*d['p_m']/tan(30*pi/180) # внутрішній діаметр вершин
трикутника профілю муфти
d['lm_']=d['lm']-11.1 # довжина різьби муфти
d['l2m_']=d['ln_']+ceil((d['l2n']-11.1)/d['p_m'])*d['p_m']-3*d['p_m']/2-
(d['d2_m']-d['d2_n'])*tan(30*pi/180) # z-координата першої западини муфти
#ceil((d['l2n']-11.1)/d['p_m'])*d['p_m'] - перші неробочі витки муфти
```

```
#-3*d['p_m']/2-(d['d2_m']-d['d2_n'])*tan(30*pi/180) - зміщення профілю
муфти
#================точки характерних кромок моделі=======================
en1=(d['dn']/2, d['l1n']+20, 0.0) # верхній торець штанги
en2=(0.0, (d['l1n']+20)/2, 0.0) # вісь ніпеля
en3=(d['d1_n']/2,0.0,0.0) # нижній торець штанги
en4=(d['dn'],d['l1n']+10,0.0) # зовнішній циліндр бурта
enr1=(d['d2_n']-0.25*d['p_n']/tan(30*pi/180)+d['r_n']/sin(30*pi/180)-
d['r_n'],d['ln_'],0.0) # центр першої западини ніпеля
em1=((d['dm']+d['d_m'])/2, d['l1n']-d['lm']+l_, 0.0) # нижній торець муфти
(зміщення +l_)
em2=(d['dm'],d['l1n']/2,0.0) # зовнішній циліндр муфти
em3=((d['dm']+d['d1m'])/2,d['l1n']-5,0.) # центр Partition face-1 (для Bolt
Load)
em4=((d['dm']+d['d1m'])/2,d['l1n'],0.) # верхній торець муфти
nn=8 # кількість западин ніпеля для дослідження
#mat1=matlib['40'].power(8) # матеріал 1
#mat2=matlib['40'].power(8) # матеріал 2
mat1=matlib['40fesafe'].bilinear() # матеріал 1
mat2=matlib['40fesafe'].bilinear() # матеріал 2
bolt_load=-my_iter1 #-0.1
load1=-1*d['d0']**2/d['dn']**2
load2=-170.0e+6*d['d0']**2/d['dn']**2

def createSketch1():
    "Створює ескіз профілю різьби ніпеля"
    s=model.ConstrainedSketch(name='Sketch-1', sheetSize=200.0)
    # Геометрія
    g1=s.ConstructionLine(angle=0.0, point1=(0.0, 0.0))
    s.HorizontalConstraint(entity=g1)
    s.FixedConstraint(entity=g1)
    g2=s.Line(point1=(0.0, -15.0), point2=(0.0, 15.0))
    s.VerticalConstraint(entity=g2)
    g3=s.Line(point1=(0.0, 15.0), point2=(-20.0, 5.0))
    g4=s.Line(point1=(0.0, -15.0), point2=(-20.0, -5.0))
    g5=s.ArcByStartEndTangent(entity=g3, point1=(-20.0, 5.0), point2=(-
20.0, -5.0))
    s.TangentConstraint(entity1=g4, entity2=g5)
    s.CoincidentConstraint(entity1=g5.getVertices()[2], entity2=g1)
    # Розміри і параметри
    d1=s.VerticalDimension(vertex1=g2.getVertices()[0],
vertex2=g2.getVertices()[1],textPoint=(0.0, 0.0))
    s.Parameter(name='p_n', path='dimensions[0]')
    d2=s.AngularDimension(line1=g1, line2=g3, textPoint=(10.0, 10.0))
    s.Parameter(name='alf1', path='dimensions[1]')
    d3=s.AngularDimension(line1=g1, line2=g4, textPoint=(10.0, -10.0))
    s.Parameter(name='alf2', path='dimensions[2]')
    d4=s.RadialDimension(curve=g5, textPoint=(0.0, 0.0))
    s.Parameter(name='r_n', path='dimensions[3]')
def createSketch2():
    "Створює ескіз профілю різьби муфти"
```

```
    s=model.ConstrainedSketch(name='Sketch-2', sheetSize=200.0)
    # Геометрія
    g1=s.ConstructionLine(angle=0.0,point1=(0.0, 0.0))
    s.HorizontalConstraint(entity=g1)
    s.FixedConstraint(entity=g1)
    g2=s.Line(point1=(0.0, 0.0), point2=(0.0, 20.0))
    s.VerticalConstraint(entity=g2)
    s.PerpendicularConstraint(entity1=g1, entity2=g2)
    s.CoincidentConstraint(entity1=g2.getVertices()[0], entity2=g1)
    g3=s.Line(point1=(0.0, 0.0), point2=(0.0, -20.0))
    s.VerticalConstraint(entity=g3)
    s.ParallelConstraint(entity1=g2, entity2=g3)
    s.EqualLengthConstraint(entity1=g2, entity2=g3)
    g4=s.Line(point1=(0.0, -20.0), point2=(25.0, -5.0))
    g5=s.Line(point1=(25.0, -5.0), point2=(25.0, 5.0))
    s.VerticalConstraint(entity=g5)
    g6=s.Line(point1=(25.0, 5.0), point2=(0.0, 20.0))
    # Розміри і параметри
    d1=s.VerticalDimension(vertex1=g2.getVertices()[1],
vertex2=g3.getVertices()[1],textPoint=(0.0, 0.0))
    s.Parameter(name='p_m', path='dimensions[0]')
    d2=s.AngularDimension(line1=g1, line2=g6, textPoint=(-10.0, 10.0))
    s.Parameter(name='alf1', path='dimensions[1]')
    d3=s.AngularDimension(line1=g1, line2=g4, textPoint=(-10.0, -10.0))
    s.Parameter(name='alf2', path='dimensions[2]')
    d4=s.DistanceDimension(entity1=g2, entity2=g5, textPoint=(0.0, 0.0))
    s.Parameter(name='h', path='dimensions[3]')
def createSketch3():
    "Створює ескіз заготовки ніпеля"
    s=model.ConstrainedSketch(name='Sketch-3', sheetSize=200.0)
    # Геометрія
    g1=s.Line(point1=(0.0, 0.0), point2=(0.0, 50.0))
    s.VerticalConstraint(entity=g1)
    s.FixedConstraint(entity=g1.getVertices()[0])
    g2=s.Line(point1=(0.0, 50.0), point2=(20.0, 50.0))
    s.HorizontalConstraint(entity=g2)
    g3=s.Line(point1=(20.0, 50.0), point2=(20.0, 40.0))
    s.VerticalConstraint(entity=g3)
    g4=s.Line(point1=(20.0, 40.0), point2=(10.0, 40.0))
    s.HorizontalConstraint(entity=g4)
    g5=s.ArcByStartEndTangent(entity=g4, point1=(10.0, 40.0), point2=(5.0,
35.0))
    g6=s.Line(point1=(5.0, 35.0), point2=(5.0, 30.0))
    s.VerticalConstraint(entity=g6)
    s.TangentConstraint(entity1=g5, entity2=g6)
    g7=s.ArcByStartEndTangent(entity=g6, point1=(5.0, 30.0), point2=(10.0,
25.0))
    g8=s.Line(point1=(10.0, 25.0), point2=(10.0, 5.0))
    s.VerticalConstraint(entity=g8)
    g9=s.Line(point1=(10.0, 5.0), point2=(5.0, 0.0))
    g10=s.Line(point1=(5.0, 0.0), point2=(0.0, 0.0))
```

```
    s.HorizontalConstraint(entity=g10)
    # Розміри і параметри
    d1=s.DistanceDimension(entity1=g2, entity2=g10, textPoint=(0.0, 0.0))
    s.Parameter(name='ln', path='dimensions[0]')
    d2=s.DistanceDimension(entity1=g1, entity2=g3, textPoint=(0.0, 0.0))
    s.Parameter(name='dn', path='dimensions[1]')
    d3=s.DistanceDimension(entity1=g1, entity2=g8, textPoint=(0.0, 0.0))
    s.Parameter(name='d_n', path='dimensions[2]')
    d4=s.DistanceDimension(entity1=g1, entity2=g6, textPoint=(0.0, 0.0))
    s.Parameter(name='d1n', path='dimensions[3]')
    d5=s.DistanceDimension(entity1=g4, entity2=g10, textPoint=(0.0, 0.0))
    s.Parameter(name='l1n', path='dimensions[4]')
    d6=s.DistanceDimension(entity1=g4, entity2=g8.getVertices()[0],
textPoint=(0.0, 0.0))
    s.Parameter(name='l2n', path='dimensions[5]')
    d7=s.DistanceDimension(entity1=g4, entity2=g8.getVertices()[1],
textPoint=(0.0, 0.0))
    s.Parameter(name='l3n', path='dimensions[6]')
    d8=s.AngularDimension(line1=g9, line2=g1, textPoint=(10.0, 10.0))
    s.Parameter(name='fn', path='dimensions[7]')
    d9=s.RadialDimension(curve=g5, textPoint=(0.0, 0.0))
    s.Parameter(name='r3n1', path='dimensions[8]')
    d10=s.RadialDimension(curve=g7, textPoint=(0.0, 0.0))
    s.Parameter(name='r3n2', path='dimensions[9]')
    s.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
def createSketch4():
    "Створює ескіз заготовки муфти"
    s=model.ConstrainedSketch(name='Sketch-4', sheetSize=200.0)
    # Геометрія
    v1=s.Spot(point=(0.0, 0.0))
    s.FixedConstraint(entity=v1)
    g1=s.Line(point1=(10.0, -10.0), point2=(10.0, 20.0))
    s.VerticalConstraint(entity=g1)
    g2=s.Line(point1=(10.0, 20.0), point2=(15.0, 25.0))
    g3=s.Line(point1=(15.0, 25.0), point2=(15.0, 35.0))
    s.VerticalConstraint(entity=g3)
    g4=s.Line(point1=(15.0, 35.0), point2=(20.0, 40.0))
    g5=s.Line(point1=(20.0, 40.0), point2=(25.0, 40.0))
    s.HorizontalConstraint(entity=g5)
    g6=s.Line(point1=(25.0, 40.0), point2=(30.0, 35.0))
    g7=s.Line(point1=(30.0, 35.0), point2=(30.0, -10.0))
    s.VerticalConstraint(entity=g7)
    g8=s.Line(point1=(30.0, -10.0), point2=(10.0, -10.0))
    s.HorizontalConstraint(entity=g8)
    # Розміри і параметри
    d1=s.DistanceDimension(entity1=g5, entity2=v1, textPoint=(0.0, 0.0))
    s.Parameter(name='l1n', path='dimensions[0]')
    d2=s.DistanceDimension(entity1=g5, entity2=g2.getVertices()[1],
textPoint=(0.0, 0.0))
    s.Parameter(name='l1m', path='dimensions[1]')
```

```
    d3=s.DistanceDimension(entity1=g5, entity2=g3.getVertices()[1],
textPoint=(0.0, 0.0))
    s.Parameter(name='lf1m', path='dimensions[2]')
    d4=s.DistanceDimension(entity1=g5, entity2=g6.getVertices()[1],
textPoint=(0.0, 0.0))
    s.Parameter(name='lf2m', path='dimensions[3]')
    d5=s.DistanceDimension(entity1=g5, entity2=g8, textPoint=(0.0, 0.0))
    s.Parameter(name='lm', path='dimensions[4]')
    d6=s.DistanceDimension(entity1=g1, entity2=v1, textPoint=(0.0, 0.0))
    s.Parameter(name='d1_m', path='dimensions[5]')
    d7=s.DistanceDimension(entity1=g3, entity2=v1, textPoint=(0.0, 0.0))
    s.Parameter(name='d1m', path='dimensions[6]')
    d8=s.DistanceDimension(entity1=g7, entity2=v1, textPoint=(0.0, 0.0))
    s.Parameter(name='dm', path='dimensions[7]')
    d9=s.AngularDimension(line1=g1, line2=g2, textPoint=(10.0, 10.0))
    s.Parameter(name='f_m', path='dimensions[8]')
    d10=s.AngularDimension(line1=g3, line2=g4, textPoint=(10.0, 10.0))
    s.Parameter(name='f1m', path='dimensions[9]')
    d11=s.AngularDimension(line1=g6, line2=g7, textPoint=(50.0, -50.0))
    s.Parameter(name='f2m', path='dimensions[10]')
    s.ConstructionLine(point1=(0.0, -100.0), point2=(0.0, 100.0))
def create_nipple2():
    par={'aint':0,'aext':0,'Rint':0,'Rext':d['dn'],'len':d['l1n']+20}
    set_values2(sketch='nipple',base='Quad_h',p=par)
    p=model.Part(dimensionality=AXISYMMETRIC, name='Part-1',
type=DEFORMABLE_BODY)
    p.BaseShell(sketch=model.sketches['nipple'])
    del model.sketches['nipple']


par={'aint':0,'aext':0,'Rint':d['d1n'],'Rext':d['dn'],'len':d['l2n'],'Ra':d
['r3n'],'Rb':d['r3n']}
    set_values2(sketch='groove',base='Quad_h_fillet_int',p=par)
    part_builder(p, 'groove', (0,d['l1n']-d['l2n']), 'cut')
    del model.sketches['groove']

    par={'aint':0,'aext':0,'Rint':d['d_n'],'Rext':d['dn'],'len':d['l1n']-
d['l2n']/2}
    set_values2(sketch='cut_nipple',base='Quad_h',p=par)
    part_builder(p, 'cut_nipple', (0,0), 'cut')
    del model.sketches['cut_nipple']

    par={'aa':30,'ab':90,'h':d['l1n']-d['l3n']}
    set_values2(sketch='chamfer',base='Triangle_v_int',p=par)
    part_builder(p, 'chamfer', (d['d_n'],0), 'cut')
    del model.sketches['chamfer']

    par={'aa':60,'ab':60,'len':d['p_n'],'R':d['r_n']}
    set_values2(sketch='thread',base='Triangle_v_fillet_int',p=par)
    i=0 # номер витка (0-перший)
    while i*d['p_n']<=d['ln_']: # довжина різьби
```

```
        part_builder(p, 'thread', (d['dn_'],d['ln_']-d['p_n']*i), 'cut')
        i+=1
    del model.sketches['thread']

def create_coupling2():
    par={'aint':0,'aext':0,'Rint':d['d1_m'],'Rext':d['dm'],'len':d['l1n']-
11.1}
    set_values2(sketch='coupling',base='Quad_h',p=par)
    p=model.Part(dimensionality=AXISYMMETRIC, name='Part-2',
type=DEFORMABLE_BODY)
    p.BaseShell(sketch=model.sketches['coupling'])
    del model.sketches['coupling']

    par={'aint':0,'aext':0,'Rint':d['d1_m'],'Rext':d['dm'],'len':d['lm']-
d['l1n']}
    set_values2(sketch='coupling_bot',base='Quad_h',p=par)
    part_builder(p, 'coupling_bot', (0,-(d['lm']-d['l1n'])), 'shell')
    del model.sketches['coupling_bot']

    par={'aint':0,'aext':0,'Rint':d['d1m'],'Rext':d['dm'],'len':11.1}
    set_values2(sketch='coupling_top',base='Quad_h',p=par)
    part_builder(p, 'coupling_top', (0,d['l1n']-11.1), 'shell')
    del model.sketches['coupling_top']

    #par={'aa':90,'ab':30,'t':d['d1m']-d['d1_m']}
    s=model.ConstrainedSketch(name='Sketch-1', sheetSize=200.0)
    s.Line(point1=(0.0, d['l1n']-11.1), point2=(d['d1m'], d['l1n']-11.1))
    s.Line(point1=(0.0, d['l1n']-11.1), point2=(0.0, (d['l1n']-11.1)-
d['d1m']/tan(radians(30))))
    s.Line(point1=(d['d1m'], d['l1n']-11.1), point2=(0.0, (d['l1n']-11.1)-
d['d1m']/tan(radians(30))))
    # створити клас
    #set_values2(sketch='chamfer',base='Sketch-1',p=par)
    part_builder(p, 'Sketch-1', (0,0), 'cut')
    del model.sketches['Sketch-1']

    par={'aa':90,'ab':30,'h':2}
    set_values2(sketch='chamfer',base='Triangle_v_ext',p=par)
    part_builder(p, 'chamfer', (d['d1m'],d['l1n']-2), 'cut')
    del model.sketches['chamfer']

    par={'aa':90,'ab':15,'h':3}
    set_values2(sketch='chamfer',base='Triangle_v_int',p=par)
    part_builder(p, 'chamfer', (d['dm'],d['l1n']-3), 'cut')
    del model.sketches['chamfer']

    par={'aa':120,'ab':120,'h':d['p_m'],'t':2.2-0.275}
    set_values2(sketch='thread',base='Quad_v',p=par)
    i=0
    while i*d['p_m']<=d['lm_']-3*d['p_m']/2: # довжина різьби-3*d['p_m']/2
        part_builder(p, 'thread', (d['dm_'],d['l2m_']-d['p_m']*i), 'cut')
```

```
        i+=1
    del model.sketches['thread']

def create_nipple3():
    s=model.ConstrainedSketch(name='nipple', sheetSize=200.0)
    L1=Line(p1=(0,0),len=d['l1n']+20,angle=90)
    L2=Line(p1=L1.p1,len=d['dn'],angle=0)
    L3=Line(p1=L2.p2,len=d['l1n']+20,angle=90)
    L4=Line(p1=L3.p2,p2=L1.p2)
    N_angle([L1,L2,L3,L4], [0,0,0,0]).drawAbaqus(s)
    del L1,L2,L3,L4
    s.ConstructionLine(point1=(0.0, 0.0), angle=90.0)
    p=model.Part(dimensionality=AXISYMMETRIC, name='Part-1',
type=DEFORMABLE_BODY)
    p.BaseShell(sketch=model.sketches['nipple'])
    del s

    s=model.ConstrainedSketch(name='groove', sheetSize=200.0)
    L1=Line(p1=(d['dn'],d['l1n']-d['l2n']),len=d['l2n'],angle=90)
    L2=Line(p1=L1.p2,len=d['dn']-d['d1n'],angle=180)
    L3=Line(p1=L2.p2,len=d['l2n'],angle=360-90)
    L4=Line(p1=L3.p2,p2=L1.p1)
    N_angle([L1,L2,L3,L4], [0,d['r3n'],d['r3n'],0]).drawAbaqus(s)
    del L1,L2,L3,L4
    part_builder(p, 'groove', (0,0), 'cut')
    del s

    s=model.ConstrainedSketch(name='cut_nipple', sheetSize=200.0)
    L1=Line(p1=(d['d_n'],0),len=d['l1n']-d['l2n']/2,angle=90)
    L2=Line(p1=L1.p2,len=d['dn']-d['d_n'],angle=0)
    L3=Line(p1=L2.p2,len=d['l1n']-d['l2n']/2,angle=360-90)
    L4=Line(p1=L3.p2,p2=L1.p1)
    N_angle([L1,L2,L3,L4], [0,0,0,0]).drawAbaqus(s)
    del L1,L2,L3,L4
    part_builder(p, 'cut_nipple', (0,0), 'cut')
    del s

    s=model.ConstrainedSketch(name='chamfer', sheetSize=200.0)
    L1=Line(p1=(d['d_n'],0),len=d['l1n']-d['l3n'],angle=90)
    cp=Line(p1=L1.p1,angle=180).cros_point(Line(p1=L1.p2,angle=180+60))
    L2=Line(p1=L1.p1,p2=cp)
    L3=Line(p1=L1.p2,p2=cp)
    N_angle([L1,L2,L3], [0,0,0]).drawAbaqus(s)
    del L1,L2,L3,cp
    part_builder(p, 'chamfer', (0,0), 'cut')
    del s

    s=model.ConstrainedSketch(name='thread', sheetSize=200.0)
    L1=Line(p1=(d['dn_'],0),len=d['p_n'],angle=90)
    cp=Line(p1=L1.p1,angle=90+60).cros_point(Line(p1=L1.p2,angle=270-60))
    L2=Line(p1=L1.p1,p2=cp)
```

```
    L3=Line(p1=L1.p2,p2=cp)
    N_angle([L1,L2,L3], [0,d['r_n'],0]).drawAbaqus(s)
    del L1,L2,L3,cp
    i=0 # номер витка (0-перший)
    while i*d['p_n']<=d['ln_']: # довжина різьби
        part_builder(p, 'thread', (0,d['ln_']-d['p_n']*i), 'cut')
        i+=1
    del s

def create_coupling3():
    s=model.ConstrainedSketch(name='coupling', sheetSize=200.0)
    L1=Line(p1=(d['dm'],-(d['lm']-d['l1n'])),len=d['lm']-11.1,angle=90)
    L2=Line(p1=L1.p2,len=d['dm']-d['d1_m'],angle=180)
    L3=Line(p1=L2.p2,len=d['lm']-11.1,angle=270)
    L4=Line(p1=L3.p2,p2=L1.p1)
    N_angle([L1,L2,L3,L4], [0,0,0,0]).drawAbaqus(s)
    del L1,L2,L3,L4
    s.ConstructionLine(point1=(0.0, 0.0), angle=90.0)
    p=model.Part(dimensionality=AXISYMMETRIC, name='Part-2',
type=DEFORMABLE_BODY)
    p.BaseShell(sketch=model.sketches['coupling'])
    del s

    s=model.ConstrainedSketch(name='top', sheetSize=200.0)
    L1=Line(p1=(d['dm'],d['l1n']-11.1),len=11.1,angle=90)
    L2=Line(p1=L1.p2,len=d['dm']-d['d1m'],angle=180)
    L3=Line(p1=L2.p2,len=11.1,angle=270)
    L4=Line(p1=L3.p2,p2=L1.p1)
    N_angle([L1,L2,L3,L4], [0,0,0,0]).drawAbaqus(s)
    del L1,L2,L3,L4
    part_builder(p, 'top', (0,0), 'shell')
    del s

    s=model.ConstrainedSketch(name='chamfer', sheetSize=200.0)
    L1=Line(p1=(d['d1m'],d['l1n']-11.1),len=d['d1m']-d['d1_m'],angle=180)
    cp=Line(p1=L1.p1,angle=180+60).cros_point(Line(p1=L1.p2,angle=270))
    L2=Line(p1=L1.p1,p2=cp)
    L3=Line(p1=L1.p2,p2=cp)
    N_angle([L1,L2,L3], [0,0,0]).drawAbaqus(s)
    del L1,L2,L3,cp
    part_builder(p, 'chamfer', (0,0), 'cut')
    del s

    s=model.ConstrainedSketch(name='chamfer', sheetSize=200.0)
    L1=Line(p1=(d['d1m'],d['l1n']),len=2.0,angle=270)
    cp=Line(p1=L1.p1,angle=0).cros_point(Line(p1=L1.p2,angle=90-30))
    L2=Line(p1=L1.p1,p2=cp)
    L3=Line(p1=L1.p2,p2=cp)
    N_angle([L1,L2,L3], [0,0,0]).drawAbaqus(s)
    del L1,L2,L3,cp
    part_builder(p, 'chamfer', (0,0), 'cut')
```

```
    del s

    s=model.ConstrainedSketch(name='chamfer', sheetSize=200.0)
    L1=Line(p1=(d['dm'],d['l1n']),len=3.0,angle=270)
    cp=Line(p1=L1.p1,angle=180).cros_point(Line(p1=L1.p2,angle=90+15))
    L2=Line(p1=L1.p1,p2=cp)
    L3=Line(p1=L1.p2,p2=cp)
    N_angle([L1,L2,L3], [0,0,0]).drawAbaqus(s)
    del L1,L2,L3,cp
    part_builder(p, 'chamfer', (0,0), 'cut')
    del s

    s=model.ConstrainedSketch(name='thread', sheetSize=200.0)
    L1=Line(p1=(d['dm_'],0),len=d['p_m'],angle=90)
    cp1=Line(p1=L1.p1,angle=90-
60).cros_point(Line(p1=(d['dm_']+1.93,0),angle=90))
    L2=Line(p1=L1.p1,p2=cp1)
    cp2=Line(p1=L1.p2,angle=270+60).cros_point(Line(p1=cp1,angle=90))
    L3=Line(p1=cp1,p2=cp2)
    L4=Line(p1=L1.p2,p2=cp2)
    N_angle([L1,L2,L3,L4], [0,0,0,0]).drawAbaqus(s)
    del L1,L2,L3,L4,cp1,cp2
    i=0
    while i*d['p_m']<=d['lm_']-3*d['p_m']/2:#довжина різьби-3*d['p_m']/2
        part_builder(p, 'thread', (0,d['l2m_']-d['p_m']*i), 'cut')
        i+=1
    del s
def createProfile():
    "Створює профіль різьби ніпеля і муфти"
    delCutExtrude()
    # створення профілю різьби ніпеля
    # можна це зробити також за допомогою LinearInstancePattern
    i=0 # номер витка (0-перший)
    while i*d['p_n']<=d['ln_']:#довжина різьби
     s=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
     #s.sketchOptions.setValues(viewStyle=AXISYM)
     model.parts['Part-
1'].projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s)
     s.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
     s.retrieveSketch(sketch=model.sketches['Sketch-1'])
     s.move(objectList=s.geometry.values(),vector=(d['dn_'],d['ln_']-
d['p_n']*i))
     model.parts['Part-1'].Cut(sketch=s)
     del s
     i=i+1
    # створення профілю різьби муфти
    i=0
    while i*d['p_m']<=d['lm_']-3*d['p_m']/2: # довжина різьби-3*d['p_m']/2
     s=model.ConstrainedSketch(name='__profile__',sheetSize=200.)
     #s.sketchOptions.setValues(viewStyle=AXISYM)
```

```
     model.parts['Part-
2'].projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s)
     s.ConstructionLine(point1=(0.0,0.0), point2=(0.0, 10.0))
     s.retrieveSketch(sketch=model.sketches['Sketch-2'])
     s.move(objectList=s.geometry.values(),vector=(d['dm_'],d['l2m_']-
d['p_m']*i))
     model.parts['Part-2'].Cut(sketch=s)
     del s
     i=i+1

def create_nipple_coupling():
    '''Спосіб побудови геометрії за ескізами заготовок деталей'''
    createSketch1()
    createSketch2()
    createSketch3()
    createSketch4()
    # параметри профілю різьби ніпеля
    par={'p_n':d['p_n'],'alf1':30,'alf2':30,'r_n':d['r_n']}
    set_values(sketch='Sketch-1',p=par)
    # параметри профілю різьби муфти
    par={'p_m':d['p_m'],'alf1':30,'alf2':30,'h':2.2-0.275}
    set_values(sketch='Sketch-2',p=par)
    # параметри заготовки ніпеля

par={'ln':d['l1n']+20,'d_n':d['d_n'],'dn':d['dn'],'d1n':d['d1n'],'l1n':d['l
1n'],

'l2n':d['l2n'],'l3n':d['l3n'],'r3n1':d['r3n'],'r3n2':d['r3n'],'fn':30}
    set_values(sketch='Sketch-3',p=par)
    # параметри заготовки муфти

par={'dm':d['dm'],'d1m':d['d1m'],'d1_m':d['d1_m'],'lm':d['lm'],'l1n':d['l1n
']+l_,
         'lf1m':2,'lf2m':3,'f1m':30,'f2m':15,'f_m':30,'l1m':11.1}
    set_values(sketch='Sketch-4',p=par)
    createPart(n='Part-1',s='Sketch-3')
    createPart(n='Part-2',s='Sketch-4')
    createProfile()
def create_nipple_coupling2():
    '''Спосіб побудови геометрії за готовими простими ескізами'''
    create_nipple2()
    create_coupling2()
def create_nipple_coupling3():
    '''Спосіб побудови геометрії за простими ескізами'''
    create_nipple3()
    create_coupling3()

create_nipple_coupling()
#create_nipple_coupling2()
#create_nipple_coupling3()
#createPart3D('Part-1','nipple','Part-3')
```

```
#createPart3D('Part-2','coupling','Part-4')
createPartition(part='Part-2',offset=em3[1])
#createPartition3D(part='Part-4',offset=em3[1])
# ізотропне зміцнення:
#createMaterial('Material-1',et=mat1['el'],pt=mat1['pl'])
#createMaterial('Material-2',et=mat2['el'],pt=mat2['pl'])
# кінематичне зміцнення:
createMaterial('Material-1',et=mat1['el'],pt=mat1['pl'],kinematic=True)
createMaterial('Material-2',et=mat2['el'],pt=mat2['pl'],kinematic=True)
createSectionAssign(n='Section-1',m='Material-1',p='Part-1')
createSectionAssign(n='Section-2',m='Material-2',p='Part-2')
#createSectionAssign3D(n='Section-1',m='Material-1',p='Part-3')
#createSectionAssign3D(n='Section-2',m='Material-2',p='Part-4')
createAssemblyInstance(n='Part-1-1',p='Part-1')
createAssemblyInstance(n='Part-2-1',p='Part-2')
#createAssemblyInstance3D(n='Part-3-1',p='Part-3')
#createAssemblyInstance3D(n='Part-4-1',p='Part-4')
createStep(n='Step-1',pr='Initial')
createStep(n='Step-2',pr='Step-1')
createContactSet(n='Slave',i='Part-1-1',ep=((en1, ), (en2, ), (en3, ),
(en4, ),)) # створюємо набір кромок контакту для ніпеля
createContactSet(n='Master',i='Part-2-1',ep=((em1, ), (em2, ),(em3,),)) #
створюємо набір кромок контакту для муфти
#createContactSet3D(n='Slave',i='Part-3-1',ep=((en1, ), (en3, ), (en4, ),))
# створюємо набір кромок контакту для ніпеля
#createContactSet3D(n='Master',i='Part-4-1',ep=((em1, ), (em2, ),(em3,),))
# створюємо набір кромок контакту для муфти
createContactProperty()
createContact()
#createContact3D()
createBCSet(n='Pressure',i='Part-1-1',ep=(en1, )) # тиск
createBCSet(n='Axis',i='Part-1-1',ep=(en2, )) # закріплення
createBCSet(n='Encastre',i='Part-2-1',ep=(em1, )) # закріплення
#createBCSet3D(n='Pressure',i='Part-3-1',ep=(en1, )) # тиск
#createBCSet3D(n='Encastre',i='Part-4-1',ep=(em1, )) # закріплення
createBC_Pressure([('Step-1',load1),('Step-2',load2)])
createBC_Axis()
createBC_Encastre()
createBC_BoltLoad('Part-2-1',em3,bolt_load)
#createBC_Pressure3D([('Step-1',-1.0),('Step-2',-
276.0e+6*d['d0']**2/d['dn']**2)])
#createBC_Encastre3D()
#createBC_BoltLoad3D('Part-2-1',em3,-0.1)
createMesh()
#createMesh3D()
createEdgesSet(n='Cont',i='Part-2-1',p=((em4, ),) )
createEdgesSet(n='First',i='Part-1-1',p=((enr1, ),) )
createVerticesSet(n='Zero point',i='Part-1-1',p=(((0,0,0), ),) )
createJobSubmit()
```

```
def createResults():
    global myOdb
    myOdb = openOdb(path=model.name + '.odb')
    session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)
    SF_field()
    cont_pres=readODB_set(set='Cont',step='Step-2',var=(('CPRESS',
ELEMENT_NODAL), ),pos=NODAL)
    result1=sum(cont_pres)/len(cont_pres)

result2=min(readODB_set2(set='First',step='',var=('D',''),pos=INTEGRATION_P
OINT))
    result3=max(readODB_set(set='First',step='Step-2',var=(('S',
INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), ),pos=INTEGRATION_POINT))
    result4=readODB_path(path=((d['d1n'],  d['l1n']-d['l2n']/2,
0.0),),step='Step-1',var=(('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'),
)), ),intersections=False)
    myOdb.close()

    # результати з fe-safe
    oodb='results' # назва бази даних результатів
    runFeSafe('Model-1','my',oodb) # виконати fe-safe
    myOdb = openOdb(path=oodb + '.odb') # відкрити базу даних результатів
    session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)

    # отримати логарифм довговічності у множині вузлів Set-1
    var=(('LOGLife-Repeats', ELEMENT_NODAL), )
    x1=readODB_set_(set='SLAVE',var=var)
    x1min = min([x[0][1] for x in x1]) # знайти мінімальне значення

    # отримати відсоток відмов у множині вузлів Set-1
    var=(('%%Failure@Life=5E6-Repeats', ELEMENT_NODAL), )
    x3=readODB_set_(set='SLAVE',var=var)
    x3max = max([x[0][1] for x in x3]) # знайти максимальне значення

    myOdb.close()

writer.writerow([my_iter1,my_iter2,result1,result2,result3,result4,x1min,x3
max])

createResults()

#saveDB('B')
#readDB('B')
```

Лістинг И.3 – gost5286_75.py

```
# -*- coding: cp1251 -*-
if _my_thread_model!=2: from tools import *

zn80={'D':(80,-0.5,0.5), # зовнішній діаметр труби ніпеля
    'D1':(76.5,-0.5,0.5), # зовнішній діаметр упорного торця
```

```
    'd3':(25.0,-0.6,0.6), # внутрішній діаметр ніпеля
    'd4':(36.0,-0.6,0.6), # внутрішній діаметр муфти
    'L2':(240.0,0.0,0.0), # довжина муфти*
    'dsr':(60.080,0.0,0.0), # середній діаметр різьби в основній площині
    'd5':(66.674,0.0,0.0), # діаметр більшої основи конуса ніпеля*
    'd6':(47.674,0.0,0.0), # діаметр меншої основи конуса ніпеля*
    'l3':(76.0,-2.0,0), # довжина конуса ніпеля
    'd7':(68.3,-0.6,0.6), # діаметр конічної виточки в площині торця муфти
    'd8':(61.422,0.0,0.0), # внутрішній діаметр різьби в площині торця
муфти*
    'l4':(82.0,0.0,0.0), # відстань від торця до кінця різьби з повним
профілем муфти (не менше)
    'P':(5.080,0.0,0.0), # крок різьби паралельно осі різьби
    'fi':(atan(0.25/2)*180/pi,0.0,0.0), # кут нахилу
    'H':(4.376,0.0,0.0), # висота гострокутного профілю
    'h1':(2.993,0.0,0.0), # висота профілю різьби
    'h':(2.626,0.0,0.0), # робоча висота профілю
    'l':(0.875,0.0,0.0), # висота зрізу вершин
    'f':(0.508,0.0,0.0), # відтин впадини
    'a':(1.016,0.0,0.0), # площадка*
    'r':(0.508,0.0,0.0), # радіус заокруглень впадин*
    'r_':(0.38,0.0,0.0)} # радіус спряжень (не більше)
zn95={'D':(80, -0.5, 0.5),
    'D1':(76.5,-0.5,0.5),
    'd3':(25.0,-0.6,0.6),
    'd4':(36.0,-0.6,0.6),
    'L2':(240.0,0.0,0.0),
    'dsr':(60.080,0.0,0.0),
    'd5':(66.674,0.0,0.0),
    'd6':(47.674,0.0,0.0),
    'l3':(76.0,-2.0,0),
    'd7':(68.3,-0.6,0.6),
    'd8':(61.422,0.0,0.0),
    'l4':(82.0,0.0,0.0),
    'P':(5.080,0.0,0.0),
    'fi':(atan(0.25/2)*180/pi,0.0,0.0),
    'H':(4.376,0.0,0.0),
    'h1':(2.993,0.0,0.0),
    'h':(2.626,0.0,0.0),
    'l':(0.875,0.0,0.0),
    'f':(0.508,0.0,0.0),
    'a':(1.016,0.0,0.0),
    'r':(0.508,0.0,0.0),
    'r_':(0.38,0.0,0.0)}

zamok={80:zn80,95:zn95} # словник типорозмірів
diameter=80 # типорозмір
d={} # словник усіх розмірів моделі
for x in zamok[diameter].iterkeys():
    d[x]=Dim(zamok[diameter][x]) # копіюємо ключі, а значення перетворюємо
в розміри Dim
```

```
d['D'].v=d['D'].min()/2
d['D1'].v=d['D1'].min()/2
d['d3'].v=d['d3'].max()/2
d['d4'].v=d['d4'].max()/2
d['L2'].v=d['L2'].n/2
d['dsr'].v=d['dsr'].n/2
d['d5'].v=d['d5'].n/2
d['d6'].v=d['d6'].n/2
d['l3'].v=d['l3'].min()
d['d7'].v=d['d7'].max()/2
d['d8'].v=d['d8'].max()/2
d['l4'].v=d['l4'].n
d['P'].v=d['P'].n
d['fi'].v=d['fi'].n
d['H'].v=d['H'].n
d['h1'].v=d['h1'].n
d['h'].v=d['h'].n
d['l'].v=d['l'].n
d['f'].v=d['f'].n
d['a'].v=d['a'].n
d['r'].v=d['r'].n
d['r_'].v=d['r_'].n
#================точки характерних кромок моделі=======================
en1=((d['D'].v+d['d3'].v)/2,d['l3'].v+20.0,0.0) # верхній торець ніпеля
en2=(d['d3'].v,d['l3'].v/2,0.0) # внутрішній циліндр ніпеля
en3=(d['D'].v,d['l3'].v+10.0,0.0) # зовнішній циліндр ніпеля
em1=((d['D'].v+d['d4'].v)/2,d['l3'].v-d['L2'].v,0.0) # нижній торець муфти
em2=(d['D'].v,0.0,0.0) # зовнішній циліндр муфти
em3=(d['d4'].v,d['l3'].v-d['L2'].v+5.0,0.0) # внутрішній циліндр муфти
em4=((d['D'].v+d['d7'].v)/2,d['l3'].v-8.0,0.0) # близько центру Partition
face-1 (для Bolt Load)
mat1=matlib['40'].power(8) # матеріал 1
mat2=Material(E=210000.0e+6,mu=0.28,st=my_iter2*1e+6,sv=750.0e+6,delta=18.0
,psi=60.0).power(8) # матеріал 2
bolt_load=-my_iter1 #-0.1
load1=-1
load2=-155.1e+6 #-1.0e+6/(pi*(d['D'].v/1000)**2-pi*(d['d3'].v/1000)**2) #
або в ньютонах

def createProfile():
    "Створює профіль різьби ніпеля і муфти"
    # створення профілю різьби ніпеля
    createCut(Part='Part-1',Sketch='Sketch-
3',Begin=0,P=d['P'].v,Fi=d['fi'].v,Len=d['l3'].v-
15.875+d['P'].v,X=d['dsr'].v,Y=d['l3'].v-15.875,dx=-1,dy=-1)
    # збіг різьби ніпеля
    createCut(Part='Part-1',Sketch='Sketch-
3',Begin=1,P=d['P'].v,Fi=d['fi'].v,Len=6.35,X=d['dsr'].v,Y=d['l3'].v-
15.875,dx=1,dy=1)
    # створення профілю різьби муфти
```

```
    createCut(Part='Part-2',Sketch='Sketch-
4',Begin=0,P=d['P'].v,Fi=d['fi'].v,Len=d['l4'].v-
16,X=d['dsr'].v,Y=d['l3'].v-15.875,dx=-1,dy=-1)

# параметри заготовки ніпеля
par={'l3':d['l3'].v,'d6':d['d6'].v,'fi':d['fi'].v,'D':d['D'].v,'D1':d['D1']
.v,
     'd3':d['d3'].v}
set_values(sketch='Sketch-1',p=par)
# параметри заготовки муфти
par={'l3':d['l3'].v,'l4':d['l4'].v+2,'fi':d['fi'].v,'D':d['D'].v,'d4':d['d4
'].v,
     'L2':d['L2'].v,'D1':d['D1'].v,'d7':d['d7'].v,'d8':d['d8'].v}
set_values(sketch='Sketch-2',p=par)
# параметри профілю різьби ніпеля
par={'fi':d['fi'].v,'H_21':d['H'].v/2,'H_22':d['H'].v/2,'r':d['r'].v}
set_values(sketch='Sketch-3',p=par)
# параметри профілю різьби муфти
par={'fi':d['fi'].v,'H_21':d['H'].v/2,'H_22':d['H'].v/2,'r':d['r'].v}
set_values(sketch='Sketch-4',p=par)

createPart(n='Part-1',s='Sketch-1')
createPart(n='Part-2',s='Sketch-2')
createProfile()
createPartition(part='Part-2',offset=d['l3'].v-8.0)
createMaterial('Material-1',et=mat1['el'],pt=mat1['pl'])
createMaterial('Material-2',et=mat2['el'],pt=mat2['pl'])
createSectionAssign(n='Section-1',m='Material-1',p='Part-1')
createSectionAssign(n='Section-2',m='Material-2',p='Part-2')
createAssemblyInstance(n='Part-1-1',p='Part-1')
createAssemblyInstance(n='Part-2-1',p='Part-2')
createStep(n='Step-1',pr='Initial')
createStep(n='Step-2',pr='Step-1')
createContactSet(n='Slave',i='Part-1-1',ep=((en1, ), (en2, ), (en3, ),)) #
створюємо набір кромок контакту для ніпеля
createContactSet(n='Master',i='Part-2-1',ep=((em1, ), (em2, ),(em3, ),(em4,
),)) # створюємо набір кромок контакту для муфти
createContactProperty()
createContact()
createBCSet(n='Pressure',i='Part-1-1',ep=(en1, )) # тиск
createBCSet(n='Encastre',i='Part-2-1',ep=(em1, )) # закріплення
createBC_Pressure([('Step-1',load1),('Step-2',load2)])
createBC_Encastre()
createBC_BoltLoad('Part-2-1',em4,bolt_load)
createMesh()
createEdgesSet(n='Cont',i='Part-2-1',p=((((36.725, 74.0, 0)), ),) )
createEdgesSet(n='First',i='Part-1-1',p=(((27.725, 53.045, 0), ),) )
createJobSubmit()

myOdb = openOdb(path=model.name + '.odb')
def createResults():
```

```
    session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)
    SF_field()
    cont_pres=readODB_set(set='Cont',step='Step-2',var=(('CPRESS',
ELEMENT_NODAL), ),pos=NODAL)
    result1=sum(cont_pres)/len(cont_pres)

result2=min(readODB_set2(set='First',step='',var=('D',''),pos=INTEGRATION_P
OINT))
    result3=max(readODB_set(set='First',step='Step-2',var=(('S',
INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), ),pos=INTEGRATION_POINT))
    writer.writerow([my_iter1,my_iter2,result1,result2,result3])

createResults()
myOdb.close()
```

Лістинг И.4 – gost633_80.py

```
# -*- coding: cp1251 -*-
if _my_thread_model!=3: from tools import *

nkt114={'D':(114.3,0.0,0.0), # зовнішній діаметр труби
    'd':(100.3,0.0,0.0), # внутрішній діаметр труби
    'Dm':(132.1,0.0,0.0), # зовнішній діаметр муфти
    'Lm':(156.0,0.0,0.0), # довжина муфти*
    'P':(3.175,0.0,0.0), # крок різьби паралельно осі різьби
    'dsr':(112.566,0.0,0.0), # середній діаметр різьби в основній площині
    'd1':(111.031,0.0,0.0), # зовнішній діаметр різьби в площині торця
труби
    'd2':(107.411,0.0,0.0), # внутрішній діаметр різьби в площині торця
труби
    'L':(65.0,-3.2,3.2), # загальна довжина різьби труби
    'l':(52.3,0.0,0.0), # довжина різьби труби до основної площини (з
повним профілем)
    'l1':(10.0,0.0,0.0), # максимальна довжина збігу різьби труби
    'd3':(111.219,0.0,0.0), # внутрішній діаметр різьби в площині торця
муфти
    'd0':(115.9,0.0,0.8), # діаметр циліндричної виточки муфти
    'l0':(9.5,-0.5,1.5), # глибина виточки муфти
    'A':(6.5,0.0,0.0), # натяг при згвинчуванні вручну
    'fi':(atan(1.0/32)*180/pi,0.0,0.0), # кут нахилу
    'H':(2.75,0.0,0.0), # висота вихідного профілю
    'h1':(1.81,-0.1,0.05), # висота профілю різьби
    'h':(1.734,0.0,0.0), # робоча висота профілю
    'alfa_2':(30.0,-1.0,1.0), # кут нахилу сторони профілю alfa/2
    'r':(0.508,0.0,0.045), # радіус заокруглення вершини профілю
    'r1':(0.432,-0.045,0.0)} # радіус заокруглення впадини профілю
nkt102={'D':(114.3,-0.9,0.9),
    'd':(100.3,0.0,0.0),
    'Dm':(132.1,0.0,0.0),
    'Lm':(156.0,0.0,0.0),
    'P':(3.175,0.0,0.0),
```

```
    'dsr':(112.566,0.0,0.0),
    'd1':(111.031,0.0,0.0),
    'd2':(107.411,0.0,0.0),
    'L':(65.0,-3.2,3.2),
    'l':(52.3,0.0,0.0),
    'l1':(10.0,0.0,0.0),
    'd3':(111.219,0.0,0.0),
    'd0':(115.9,0.0,0.8),
    'l0':(9.5,-0.5,1.5),
    'A':(6.5,0.0,0.0),
    'fi':(atan(0.0625/2)*180/pi,0.0,0.0),
    'H':(2.75,0.0,0.0),
    'h1':(1.81,-0.1,0.05),
    'h':(1.734,0.0,0.0),
    'alfa_2':(30.0,-1.0,1.0),
    'r':(0.508,0.0,0.045),
    'r1':(0.432,-0.045,0.0)}

nkt={114:nkt114,102:nkt102} # словник типорозмірів
diameter=114 # типорозмір
d={} # словник усіх розмірів моделі
for x in nkt[diameter].iterkeys():
    d[x]=Dim(nkt[diameter][x]) # копіюємо ключі, а значення перетворюємо в
розміри Dim

d['D'].v=d['D'].n/2
d['d'].v=d['d'].max()/2
d['Dm'].v=d['Dm'].min()/2
d['Lm'].v=d['Lm'].n/2
d['P'].v=d['P'].n
d['dsr'].v=d['dsr'].n/2
d['d1'].v=d['d1'].n/2
d['d2'].v=d['d2'].n/2
d['L'].v=d['L'].min()
d['l'].v=d['l'].n
d['l1'].v=d['l1'].n
d['d3'].v=d['d3'].n/2
d['d0'].v=d['d0'].min()/2
d['l0'].v=d['l0'].max()
d['A'].v=d['A'].n
d['fi'].v=d['fi'].n
d['H'].v=d['H'].n
d['h1'].v=d['h1'].max()
d['h'].v=d['h'].n
d['alfa_2'].v=d['alfa_2'].n
d['r'].v=d['r'].max()
d['r1'].v=d['r1'].min()
#================точки характерних кромок моделі=======================
en1=((d['D'].v+d['d'].v)/2,d['L'].v+20,0.0) # верхній торець ніпеля
en2=(d['d'].v,d['L'].v/2,0.0) # внутрішній циліндр ніпеля
en3=(d['D'].v,d['L'].v+20-5,0.0) # зовнішній циліндр ніпеля
```

```
em1=(d['Dm'].v-5,d['L'].v-d['A'].v-d['Lm'].v,0.0) # нижній торець муфти
em2=(d['Dm'].v,0.0,0.0) # зовнішній циліндр муфти
mat1=matlib['40'].power(8) # матеріал 1
mat2=matlib['40'].power(8) # матеріал 2
bolt_load=my_iter1
load1=-1
load2=-my_iter2*1e+6

def createProfile():
    '''Створює профіль різьби ніпеля і муфти'''
    #X,Y=const+-n*p
    # створення профілю різьби ніпеля
    x=model.sketches['Sketch-3'].parameters['x'].value # допоміжний
параметр
    dsr=d['D'].v-x # середній діаметр в основній площині
    createCut(Part='Part-1',Sketch='Sketch-
3',Begin=0,P=d['P'].v,Fi=d['fi'].v,Len=d['L'].v-12.7,X=dsr,Y=d['L'].v-
12.7,dx=-1,dy=-1)
    # витки з зрізаними вершинами
    n=createCut(Part='Part-1',Sketch='Sketch-
3',Begin=1,P=d['P'].v,Fi=d['fi'].v,Len=12.7-d['l1'].v,X=d['D'].v-
x,Y=d['L'].v-12.7,dx=1,dy=1)
    # збіг різьби
    createCut(Part='Part-1',Sketch='Sketch-
3',Begin=1,P=d['P'].v,Fi=10.0,Len=d['l1'].v,X=d['D'].v-x,Y=d['L'].v-
12.7+n*d['P'].v,dx=1,dy=1)
    #12.7 - кратне усім стандартним крокам
    #d['D'].v-x - середній діаметр в основній площині
    # створення профіля різьби муфти
    x=model.sketches['Sketch-4'].parameters['x'].value # допоміжний
параметр
    createCut(Part='Part-2',Sketch='Sketch-
4',Begin=0,P=d['P'].v,Fi=d['fi'].v,Len=d['Lm'].v-4,X=dsr,Y=d['L'].v-
12.7,dx=-1,dy=-1)
    #X=d['d3'].v+x,Y=d['L'].v-d['A'].v

# параметри заготовки ніпеля
par={'ln':d['L'].v+20,'D':d['D'].v,'d':d['d'].v,'d2':d['d2'].v,'l':d['L'].v
-12.7,'fi':d['fi'].v}
set_values(sketch='Sketch-1',p=par)
# параметри заготовки муфти
par={'Dm':d['Dm'].v,'Lm':d['Lm'].v,'d3':d['d3'].v,'d0':d['d0'].v,
    'l0':d['l0'].v,'fi':d['fi'].v,'lA':d['L'].v-
d['A'].v,'hk':d['h1'].v+0.5}
set_values(sketch='Sketch-2',p=par)
# параметри профілю різьби ніпеля
par={'fi':d['fi'].v,'P':d['P'].v,'r1':d['r1'].v,'r':d['r'].v}
set_values(sketch='Sketch-3',p=par)
# параметри профілю різьби муфти
par={'fi':d['fi'].v,'P':d['P'].v,'r1':d['r1'].v,'r':d['r'].v}
set_values(sketch='Sketch-4',p=par)
```

```
createPart(n='Part-1',s='Sketch-1')
createPart(n='Part-2',s='Sketch-2')
createProfile()
createMaterial('Material-1',et=mat1['el'],pt=mat1['pl'])
createMaterial('Material-2',et=mat2['el'],pt=mat2['pl'])
createSectionAssign(n='Section-1',m='Material-1',p='Part-1')
createSectionAssign(n='Section-2',m='Material-2',p='Part-2')
createAssemblyInstance(n='Part-1-1',p='Part-1')
createAssemblyInstance(n='Part-2-1',p='Part-2')
createStep(n='Step-1',pr='Initial')
createStep(n='Step-2',pr='Step-1')
createContactSet(n='Slave',i='Part-1-1',ep=((en1, ), (en2, ), (en3, ),)) #
створюємо набір кромок контакту для ніпеля
createContactSet(n='Master',i='Part-2-1',ep=((em1, ), (em2, ),)) #
створюємо набір кромок контакту для муфти
createContactProperty()
createContact()
createBCSet(n='Pressure',i='Part-1-1',ep=(en1, )) # тиск
createBCSet(n='Encastre',i='Part-2-1',ep=(em1, )) # закріплення
createBC_Pressure([('Step-1',load1),('Step-2',load2)])
createBC_Encastre()
createMesh()
model.rootAssembly.translate(instanceList=('Part-2-1', ),
    vector=(0.0, bolt_load*d['P'].v, 0.0)) # моделювання згвинчування
(0(вручну),1,2(станок))

work_list=range(13) # розглядаємо 13 витків ніпеля
n=0
#(56.2047863424959,45.1455709929578,0) - координати робочої сторони першого
витка
for x in work_list: # створюємо Set для кожної робочої сторони витка
    x=(56.2047863424959-n*d['P'].v*tan(radians(d['fi'].v)),
45.1455709929578-n*d['P'].v, 0)
    createEdgesSet(n='work'+str(n),i='Part-1-1',p=((x, ),) )
    n+=1
n=0
nwork_list=range(13) # розглядаємо 13 витків ніпеля
for x in nwork_list: # створюємо Set для кожної неробочої сторони витка
    x=(56.1551769672515-n*d['P'].v*tan(radians(d['fi'].v)),
43.5580709924834-n*d['P'].v, 0)
    createEdgesSet(n='nwork'+str(n),i='Part-1-1',p=((x, ),) )
    n+=1

createJobSubmit()

myOdb = openOdb(path=model.name + '.odb')
def createResults():
    session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)
    for x in range(13):
```

```
        cont_pres=readODB_set2(set='work'+str(x),step='Step-
2',var=('CPRESS',''),pos=NODAL)
        result1=sum(cont_pres)/len(cont_pres)
        cont_pres=readODB_set2(set='nwork'+str(x),step='Step-
2',var=('CPRESS',''),pos=NODAL)
        result2=sum(cont_pres)/len(cont_pres)
        writer.writerow([my_iter1,my_iter2,x,result1,result2])

createResults()
myOdb.close()
```

Лістинг И.5 – main.py

```
# -*- coding: cp1251 -*-
_my_thread_model=1
import csv
csv_file=open("results.csv", "wb")
writer = csv.writer(csv_file,delimiter = ';')
writer.writerow(['boltload','groove','cont','D','first','centr'])
my_iter2=0
my_iter1=0.1
execfile('tools.py')
execfile('gost13877_96.py')
#for my_iter2 in [10*x for x in range(0,3+1,1)]:
#    for my_iter1 in [x/100.0 for x in range(0,30+5,5)]:
#        execfile('tools.py')
#        execfile('gost13877_96.py')
#        print 'my_iters=',my_iter1,my_iter2
#        Mdb()
#        session.viewports['Viewport: 1'].setValues(displayedObject=None)
csv_file.close()
```

Лістинг И.6 – main2.py

```
# -*- coding: cp1251 -*-
_my_thread_model=2
import csv
csv_file=open("results.csv", "wb")
writer = csv.writer(csv_file,delimiter = ';')
writer.writerow(['boltload','sv','cont','Dfirst','Sfirst'])
#my_iter2=0
#my_iter1=0.1
#execfile('tools.py')
#execfile('gost5286_75.py')
for my_iter2 in [100*x for x in range(3,5+1,1)]: # границя плинності
матеріалу муфти, МПа
    for my_iter1 in [x/100.0 for x in range(0,30+5,5)]: # величина
згвинчування, мм
        execfile('tools.py')
        execfile('gost5286_75.py')
```

```
        print 'my_iters=',my_iter1,my_iter2
        Mdb()
        session.viewports['Viewport: 1'].setValues(displayedObject=None)
csv_file.close()
```

Лістинг И.7 – main3.py

```
# -*- coding: cp1251 -*-
_my_thread_model=3
import csv
csv_file=open("results.csv", "wb")
writer = csv.writer(csv_file,delimiter = ';')
writer.writerow(['boltload','press','n','cont_work','cont_nwork'])
#my_iter2=155.1
#my_iter1=0
#execfile('tools.py')
#execfile('gost633_80.py')
for my_iter2 in [0.000001,100,200,300]:
    for my_iter1 in [0,1,2]:
        execfile('tools.py')
        execfile('gost633_80.py')
        print 'my_iters=',my_iter1,my_iter2
        Mdb()
        session.viewports['Viewport: 1'].setValues(displayedObject=None)
csv_file.close()
```

ДОДАТОК І

Пакет ThreadsOCC – прикладна САПР різьбових з’єднань

Код програм доступний також в GitHub (vkopey/ThreadsOCC. URL: http://github.com/vkopey/ThreadsOCC). Вміст пакету:

- gost13877_96params.py – параметри з'єднання ШН за ГОСТ 13877-96;
- myBaseGeom.py – базові класи PythonOCC;
- ccx_inp.py – робота з файлами calculix .inp;
- ccx_out.py – читає дані з файлу результатів CalculiX .frd;
- main.py – головний модуль.

Лістинг І.1 – gost13877_96params.py

```
# -*- coding: utf-8 -*-
from math import *
steel45={'el':((210000000000.0, 0.28), ),
                'pl':((620000000.0, 0.0),
                      (640000000.0, 0.02),
                      (800000000.0, 0.04),
                      (860000000.0, 0.08),
                      (864000000.0, 0.11))}
##
class Dim:
    "Клас описує поняття розміру"
    n=0.0 # номінальний розмір
    ei=0.0 # нижнє відхилення
    es=0.0 # верхнє відхилення
    v=0.0 # дійсне значення
    def __init__(self,n=None,ei=None,es=None,doc=""):
        self.n=n
        self.ei=ei
        self.es=es
        self.__doc__=doc.decode('utf-8')

    def min(self):
        "повертає мінімальний розмір"
        return self.n+self.ei

    def max(self):
        "повертає максимальний розмір"
        return self.n+self.es
##
class Rod(object):
```

```
    d_n=Dim(doc="зовнішній діаметр різьби")
    d2_n=Dim(doc="середній діаметр різьби")
    d1_n=Dim(doc="внутрішній діаметр різьби")
    r_n=Dim(doc="радіус западин різьби")
    dn=Dim(doc="діаметр бурта")
    d1n=Dim(doc="діаметр зарізьбової канавки")
    l1n=Dim(doc="довжина ніпеля")
    l2n=Dim(doc="довжина зарізьбової канавки")
    l3n=Dim(doc="довжина ніпеля без фаски на різьбі")
    l4n=Dim(doc="довжина ніпеля з буртом")
    r3n=Dim(doc="радіус скруглень зарізьбової канавки")
    d_m=Dim(doc="зовнішній діаметр різьби")
    d2_m=Dim(doc="середній діаметр різьби")
    d1_m=Dim(doc="внутрішній діаметр різьби")
    dm=Dim(doc="зовнішній діаметр")
    d1m=Dim(doc="внутрішній діаметр опорної поверхні")
    lm=Dim(doc="довжина муфти")
    d0=Dim(doc="діаметр тіла")
    p_n=Dim(doc="крок різьби")
    p_m=Dim(doc="крок різьби")

    def setModelParams(self,**args):
        self.l_=0 # скорочення муфти при згвинчуванні (0, якщо задано Bolt
Load)
        #====================параметри ніпеля штанги==================
        self.d_n = self.d_n.min() / 2 # зовнішній діаметр різьби/2
        self.d2_n = self.d2_n.min() / 2 # середній діаметр різьби/2
        self.d1_n = self.d1_n.min() / 2 # внутрішній діаметр різьби/2!ei*
        self.r_n = self.r_n.min() # радіус западин різьби
        self.p_n = self.p_n.min() # крок різьби
        self.dn = self.dn.min() / 2 # діаметр бурта/2
        self.d1n = self.d1n.min() / 2 # діаметр зарізьбової канавки/2
        self.l1n = self.l1n.min() # довжина ніпеля
        self.l2n = self.l2n.min() # довжина зарізьбової канавки
        self.l3n = self.l3n.min() # довжина ніпеля без фаски на різьбі
        self.l4n = self.l4n.min() # довжина ніпеля з буртом
        self.r3n = self.r3n.min() # радіус скруглень зарізьбової канавки
        self.d0 = self.d0.min() / 2 # діаметр тіла/2
        #====================параметри муфти==========================
        self.d_m = self.d_m.max() / 2 # зовнішній діаметр різьби/2!es*
        self.d2_m = self.d2_m.max() / 2 # середній діаметр різьби/2
        self.d1_m = self.d1_m.max() / 2 # внутрішній діаметр різьби/2
        self.p_m = self.p_m.min() # крок різьби
        self.dm = self.dm.min() / 2 # зовнішній діаметр/2
        self.d1m = self.d1m.max() / 2 # внутрішній діаметр опорної
поверхні/2
        self.lm = self.lm.min() / 2 # довжина муфти/2

        for k in args:
            self.__dict__[k]=args[k]
```

```
        #================допоміжні параметри=======================
        self.dn_=self.d2_n+0.25*self.p_n/tan(30*pi/180) # зовнішній діаметр
вершин трикутника профілю ніпеля
        self.ln_=self.l1n-self.l2n # z-координата першої западини ніпеля
(довжина різьби ніпеля)
        self.dm_=self.d2_m-0.25*self.p_m/tan(30*pi/180) # внутрішній
діаметр вершин трикутника профілю муфти
        self.lm_=self.lm-11.1 # довжина різьби муфти
        self.l2m_=self.ln_+ceil((self.l2n-11.1)/self.p_m)*self.p_m-
3*self.p_m/2-(self.d2_m-self.d2_n)*tan(30*pi/180) # z-координата першої
западини муфти
        #ceil((self.l2n-11.1)/self.p_m)*self.p_m - перші неробочі витки
муфти
        #-3*self.p_m/2-(self.d2_m-self.d2_n)*tan(30*pi/180) - зміщення
профілю муфти
        #===============точки характерних кромок моделі==================
        self.en1=(self.dn/2, self.l4n, 0.0) # верхній торець штанги (було
l1n+20)
        self.en2=(0.0, self.l4n/2, 0.0) # вісь ніпеля
        self.en3=(self.d1_n/2,0.0,0.0) # нижній торець штанги
        self.en4=(self.dn,self.l4n-5,0.0) # зовнішній циліндр бурта
        self.enr1=(self.d2_n-
0.25*self.p_n/tan(30*pi/180)+self.r_n/sin(30*pi/180)-self.r_n,self.ln_,0.0)
# центр першої западини ніпеля
        self.em1=((self.dm+self.d_m)/2, self.l1n-self.lm+self.l_, 0.0) #
нижній торець муфти (зміщення +self.l_)
        self.em2=(self.dm,self.l1n/2,0.0) # зовнішній циліндр муфти
        self.em3=((self.dm+self.d1m)/2,self.l1n-5,0.) # центр Partition
face-1 (для Bolt Load)
        self.em4=((self.dm+self.d1m)/2,self.l1n,0.) # верхній торець муфти
        self.nn=8 # кількість западин ніпеля для дослідження
        self.mat1=steel45 # матеріал 1
        self.mat2=steel45 # матеріал 2
        self.bolt_load=-0.1
        self.load1=-1*self.d0**2/self.dn**2
        self.load2=-155.1*self.d0**2/self.dn**2

##
class Rod19(Rod):
    d_n=Dim(27, -0.48, -0.376)
    d2_n=Dim(25.35, -0.204, -0.047)
    d1_n=Dim(24.25, 0, -0.415)
    r_n=Dim(0.28, 0, 0.08)
    dn=Dim(38.1, -0.25, 0.13)
    d1n=Dim(23.24, -0.13, 0.13)
    l1n=Dim(36.5, 0, 1.6)
    l2n=Dim(15, 0.2, 1)
    l3n=Dim(32, 0, 1.5)
    l4n=Dim(48, -1, 1.5)
    r3n=Dim(3, 0, 0.8)
    d_m=Dim(27, 0, 0.27)
```

```
    d2_m=Dim(25.35, 0, 0.202)
    d1_m=Dim(24.25, 0, 0.54)
    dm=Dim(41.3, -0.25, 0.13)
    d1m=Dim(27.43, 0, 0.25)
    lm=Dim(102, -1, 1)
    d0=Dim(19.1,-0.41,0.2)
    p_n=Dim(2.54,0,0)
    p_m=Dim(2.54,0,0)
    def __init__(self,**args):
        for k in Rod19.__dict__:
            if Rod19.__dict__[k].__class__==Dim:
               Rod19.__dict__[k].__doc__=Rod.__dict__[k].__doc__ # атрибут
документації
        self.setModelParams(**args)


if __name__=="__main__":
    r=Rod19()
    #r.setModelParams()
    r.dn_
```

Лістинг І.2 – myBaseGeom.py

```
# encoding: utf-8
from OCC import VERSION # версія PythonOCC
print "OCC version",VERSION # версія '0.16.2-0.18.1'

# Геометричний процессор gp - незбережувані базові геометричні об'єкти
# З цими об'єктами працюють за значенням
from OCC.gp import *

# Збережувані базові 3D геометричні об'єкти
# Ці об'єкти STEP-оброблювані і з ними працюють за посиланням
from OCC.Geom import *
# Збережувані базові 2D геометричні об'єкти
# Ці об'єкти STEP-оброблювані і з ними працюють за посиланням
from OCC.Geom2d import *

# Алгоритми для побудови елементарних геометричних об'єктів OCC.Geom
from OCC.GC import *
# Алгоритми для побудови елементарних геометричних об'єктів OCC.Geom2d
from OCC.GCE2d import *
```

Лістинг І.3 – ccx_inp.py

```
# encoding: utf-8
import subprocess

filenameGmsh="gmsh.inp"
filename="model.inp"
nodes={} # вузли
```

```
lines={} # вузли ліній
elements={} # елементи поверхонь

# шаблон нижньої частини файлу .inp
_tmp="""
*MATERIAL, NAME=mat1
*ELASTIC
210000.0, 0.3
*MATERIAL, NAME=mat2
*ELASTIC
210000.0, 0.3
*SOLID SECTION, ELSET=Surface1, MATERIAL=mat1
1.
*SOLID SECTION, ELSET=Surface2, MATERIAL=mat2
1.
*SOLID SECTION, ELSET=Surface3, MATERIAL=mat2
1.
{surfSlaveDefinitions}
{surfMasterDefinitions}
*SURFACE, NAME = surfBoltLoad, TYPE = ELEMENT
{surfBoltLoadDefinitions}
*PRE-TENSION SECTION, NODE={preTNode}, SURFACE=surfBoltLoad
0.0,1.0,0.0
*SURFACE INTERACTION, NAME=Int1
*SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=LINEAR
**1.e7, 3.
*CONTACT PAIR, INTERACTION=Int1, ADJUST=0.0, TYPE=SURFACE TO SURFACE
Slave, Master
*BOUNDARY
{boundLine1},1,2,0.0
*BOUNDARY
{boundLine2},1,1,0.0
*TIME POINTS,NAME=T1
1.0,2.0

*STEP
*STATIC
**1.e-4,1.
*BOUNDARY
{preTNode}, 1, 1, {boltLoad}
*NODE FILE, TIME POINTS=T1
U,
*EL FILE, TIME POINTS=T1
S,
*CONTACT FILE, TIME POINTS=T1
CDIS, CSTR
*END STEP

*STEP
*STATIC
**1.e-4,1.
```

```
**CLOAD
**{loadLine1}, 2, 400.0
*DLOAD
{surfLoadDefinitions}
**CLOAD
**{preTNode}, 1, -4000.0
*BOUNDARY
{preTNode}, 1, 1, {boltLoad}
*NODE FILE, TIME POINTS=T1
U,
*EL FILE, TIME POINTS=T1
S,
*CONTACT FILE, TIME POINTS=T1
CDIS, CSTR
*END STEP
"""

def mesh():
    """Створює сітку у Gmsh"""
    gmshPath=r"e:\Portable\gmsh-4.4.0-Windows64\gmsh.exe"
    subprocess.Popen("%s model.brep -2 -o gmsh.inp -format inp -order 2 -
algo del2d -clscale 0.5 -clcurv"%gmshPath, shell=True).wait() # Gmsh-
>Abaqus

def runCCX():
    """Виконує розрахунок у ccx"""
    ccxPath=r"e:\CL33-win64\bin\ccx\ccx215.exe"
    s=subprocess.check_output([ccxPath, "-i", filename[:-4]], shell=True)
    L=[ln.strip() for ln in s.splitlines()[-10:]] # останні рядки виведення
    if "Job finished" in L:
        return 1 # якщо розрахунок успішний
    return 0

# додати заміну дуже малих чисел на 0?

def appendNode():
    """Додає у файл inp визначення нового вузла і повертає його номер"""
    f=readINP(filename) # прочитати рядки
    lns=[] # список нових рядків файлу
    t='******* E L E M E N T S *************\n'
    prev="" # попередній рядок
    for ln in f: # для кожного рядка
        if ln==t: # якщо закінчилась секція вузлів
            lastNode=int(prev.split(',')[0].strip()) # номер останнього
вузла
            # додати визначення ще одного вузла
            lns.append(str(lastNode+1)+", 0, 0, 0\n")
        prev=ln
        lns.append(ln)
    writeINP(lns) # зберегти рядки
    return lastNode+1
```

```
def elset2nset():
    """Конвертує файл gmsh.inp в CalculiX.inp
    аналогічно утиліти Prool's GMSH.inp to CCX.inp
    Увага! Застосовувати відразу після Gmsh"""
    f=readINP(filenameGmsh) # прочитати рядки
    lns=[] # список нових рядків файлу
    t="" # поточна секція
    t1="*ELEMENT, type=T3D3, ELSET="
    t2="*NSET, NSET="
    nn=[] # вузли лінії
    for ln in f: # для кожного рядка
        # якщо заголовок секції *ELEMENT, type=T3D3, ELSET=
        if ln.startswith(t1):
            if nn: # якщо не перша секція
                # список вузлів і заголовок
                lnn="\n".join(nn)+"\n"+ln.replace(t1, t2)
            else: # якщо перша секція
                # тільки заголовок
                lnn=ln.replace(t1, t2)
            lns.append(lnn)
            t=t1 # позначити, що ми в середині секції
            nn=[] # очистити список вузлів
        # якщо в середині секції
        elif t==t1 and not ln.startswith("*ELEMENT"):
            ns=[x.strip() for x in ln.split(",")] # вузли
            if not nn: # якщо ми на початку секції
                nn+=ns[1:] # додати три вузла
            else: # якщо ми не на початку секції
                nn+=ns[2:] # додати два вузла
        # якщо секція закінчилась
        elif t==t1 and ln.startswith("*ELEMENT"):
            # список вузлів і заголовок
            lnn="\n".join(nn)+"\n"+ln
            lns.append(lnn)
            t="" # позначити, що ми поза секцією
            nn=[] # очистити список вузлів
        # якщо інші рядки
        else:
            lns.append(ln) # залишаємо без змін
    writeINP(lns) # зберегти рядки

def reverseOrient():
    """Змінює орієнтацію елементів у файлі .inp на протилежну.
    Нумерація вузлів елемента CAX6 змінюється так:
    1,2,3,4,5,6 -> 1,3,2,6,5,4
    Увага! Застосовувати після elset2nset(), якщо
    орієнтація елементів неправильна"""
    f=readINP() # прочитати рядки
    lns=[] # список нових рядків файлу
    t="" # поточна секція
```

```
    t1="*ELEMENT, type=CPS6, ELSET="
    for ln in f: # для кожного рядка
        # якщо початок секції
        if ln.startswith(t1):
            t=t1
            lns.append(ln) # залишити без змін
        # якщо в секції
        elif t==t1 and ln.strip()[0] in "0123456789":
            el=[x.strip() for x in ln.split(",")]
            # змінити порядок вузлів
            eln=[el[0],el[1],el[3],el[2],el[6],el[5],el[4]]
            lnn=", ".join(eln)+"\n"
            lns.append(lnn)
        # якщо інші рядки
        else:
            t=""
            lns.append(ln) # залишити без змін
    writeINP(lns) # зберегти рядки

def parse():
    """Парсер файлів calculix .inp
Увага! Застосовувати після elset2nset() і reverseOrient()"""
    f=readINP() # прочитати рядки
    t="" # поточна секція файлу inp
    t1="*NODE"
    t3="*NSET"
    t4="*ELEMENT"
    nset="" # назва лінії (множини вузлів)
    elset="" # назва поверхні (множини елементів)

    for ln in f: # для кожного рядка
        # парсимо в залежності від рядка ln і секції t
        # заголовок секції *NODE
        if ln.startswith(t1):
            t=t1
        # вузол
        elif t==t1 and ln.strip()[0] in "0123456789":
            ns=[e.strip() for e in ln.split(",")]
            nodes[int(ns[0])] = float(ns[1]), float(ns[2]), float(ns[3])
        # заголовок секції *NSET
        elif ln.startswith(t3):
            t=t3
            nset=ln.split("=")[1].strip()
            lines[nset]=[]
        # вузол лінії
        elif t==t3 and ln.strip()[0] in "0123456789":
            lines[nset].append(int(ln.strip()))
        # секція *Element
        elif ln.startswith(t4):
            t=t4
            elset=ln.split(",")[2].split("=")[1].strip()
```

```
            elements[elset]={}
        # вузли елемента
        elif t==t4 and ln.strip()[0] in "0123456789":
            es=[e.strip() for e in ln.split(",")]
            elements[elset][int(es[0])]=int(es[1]), int(es[2]), int(es[3]),
int(es[4]), int(es[5]), int(es[6])

def findLine(x1,y1,x2,y2):
    """Шукає криву за двома точками"""
    r=11 # точність. Можна дати заокруглення до r знаків, якщо не знаходить
    p=[round(x,r) for x in (x1,y1,x2,y2)] # заокруглити
    p1=p[0],p[1]
    p2=p[2],p[3]
    for ln in lines: # для кожної лінії
        ns=[nodes[x] for x in lines[ln]] # список вузлів на лінії
        nsr=[(round(n[0],r),round(n[1],r)) for n in ns] # заокруглити
        #if p1 in nsr and p2 in nsr: # якщо точки в списку <Тут не зовсім
вірно. Див. нижче>
        # якщо крайні точки збігаються
        if p1==nsr[0] and p2==nsr[-1]:
            return ln
        if p2==nsr[0] and p1==nsr[-1]:
            return ln
    return None

def readINP(filename=filename):
    """Повертає рядки файлу .inp"""
    f=open(filename, 'r')
    ls=f.readlines()
    f.close()
    return ls

def writeINP(ls):
    """Записує рядки ls у файл .inp"""
    f=open(filename, 'w')
    f.writelines(ls)
    f.close()

def writeFinalINP(d):
    """Записує кінцевий файл .inp"""
    f=open(filename, 'r')
    s1=f.read()
    f.close()
    s1=replaceElType(s1)
    s2=_tmp.format(**d)
    f=open(filename, 'w')
    f.write(s1+s2)
    f.close()

def replaceElType(s):
    """Замінює тип елементів у тексті s"""
```

```
    old="*ELEMENT, type=CPS6, ELSET="
    new="*ELEMENT, type=CAX6, ELSET="
    return s.replace(old, new)

def partByElement(e):
    """Визначає деталь за елементом"""
    for s in elements: # для кожної деталі
        if e in elements[s]: # якщо елемент серед елементів деталі
            return s # повертає назву деталі
    return None

def partByLine(ln):
    """Визначає деталь за лінією"""
    b=set(lines[ln]) # множина вузлів лінії
    for s in elements: # для кожної деталі
        a=set() # множина вузлів деталі
        for e in elements[s]: # для кожного елементу деталі
            ns=set(elements[s][e]) # множина вузлів елемента
            a.update(ns) #  додати до множини вузлів деталі
        if b.issubset(a): # якщо вузли лінії серед вузлів деталі
            return s # повернути назву деталі
    return None

def findElementSurface(e):
    """Повертає назву лінії і назву сторони елемента на лінії.
    Тільки для елементів типу CAX6 !
    Наприклад, елемент CAX6 має вузли 1,2,3,4,5,6
    тоді сторони елемента нумеруються так:
    s1: 1-2, s2: 2-3, s3: 3-1"""
    s=partByElement(e) # назва деталі
    ns=elements[s][e] # вузли елемента
    for ln in lines: # для кожної лінії
        nsl=lines[ln] # вузли лінії
        nc=set(ns)&set(nsl) # множина спільних вузлів елемента і лінії
        if len(nc)==3: # якщо спільні три вузла (всього 6)
            s1=set(ns[:2])&nc
            s2=set(ns[1:3])&nc
            s3=set((ns[2],ns[0]))&nc
            # якщо обидва кутові вузли на лінії, то
            # повертає назву лінії і назву сторони елемента на лінії
            if len(s1)==2: return (ln,"S1")
            if len(s2)==2: return (ln,"S2")
            if len(s3)==2: return (ln,"S3")
    return (None,None)

def elementsByLine(ln,s=None):
    """Повертає список елементів лінії (код з surfByLineEl)"""
    res=[]
    if s==None: s=partByLine(ln) # назва деталі
    for e in elements[s]: # для кожного елемента деталі
        ls=findElementSurface(e)
```

```
        if ls[0]==ln: # якщо елемент на лінії
            res.append(e)
    return res

def surfByLineEl(ln,s=None):
    """Код визначення *SURFACE за елементами лінії"""
    surdef=""
    if s==None: s=partByLine(ln) # назва деталі

    for e in elements[s]: # для кожного елемента деталі
        ls=findElementSurface(e)
        if ls[0]==ln: # якщо елемент на лінії
            surdef+=str(e)+", "+str(ls[1])+"\n"
    return surdef

def dloadDefs(ln,s,value):
    _=surfByLineEl(ln,s)
    return _.replace('S','P').replace('\n',', %f\n'%value)

def surfDefs(lines,surfname):
    """Код визначення *SURFACE, які відповідають lines"""
    s="""*SURFACE, NAME = {name}, TYPE = ELEMENT\n""".format(name=surfname)
    for ln in lines:
        if ln: #!!!
            s+=surfByLineEl(ln)
    return s

def surfNodeDefs(lines,surfname):
    """Код визначення *SURFACE TYPE=NODE, які відповідають lines"""
    s="""*SURFACE, NAME = {name}, TYPE = NODE\n""".format(name=surfname)
    ls=[ln for ln in lines if ln]
    return s+',\n'.join(ls)

def findLines(pts):
    lns=[]
    for p1,p2 in pts:
        ln=findLine(x1=p1[0],y1=p1[1],x2=p2[0],y2=p2[1])
        lns.append(ln)
    return lns

def nodesByLines(lns):
    lnNodes=set()
    for ln in lns:
        if ln: lnNodes.update(lines[ln])
    return lnNodes

def nearestNode(x,y):
    mn=1e30
    for n in nodes:
        xn,yn,zn=nodes[n]
        d=((x-xn)**2+(y-yn)**2)**0.5
```

```
        if d<mn:
            mn=d
            nnode=n
    return nnode,mn

if __name__=='__main__':
    print "ccx_inp"
    elset2nset()
    #reverseOrient()
    parse()
    print nearestNode(11.13, 23.87)

    # for n in nodes:
    #     print n, nodes[n]

    # for n in lines:
    #     print n, lines[n]

    # for n in lines:
    #     print n, [nodes[x] for x in lines[n]]

    # for s in elements:
    #     print s
    #     for e in elements[s]:
    #         #print e,elements[s][e]
    #         print findElementSurface(e)

    # print findLine(0,0,0.8864911064067351,0)
    #
    # print surfByLineEl("Line1")
    #
    # print surfsDefs(["Line1","Line2","Line3"])

    # for e in elements['Surface2']:
    #     print partByElement(e)

    # print partByLine("Line1")
```

Лістинг І.4 – ccx_out.py

```
# -*- coding: utf-8 -*-
"""!Увага! Читає тільки 1 інкремент з напруженнями,
тому застосовуйте *TIME POINTS
"""
file=r'model.frd'

def parseLines(startLine):
    "Читає дані з блоку що починається з startLine"
    res={} # напруження для кожного вузла
    inside=False # чи всередині блоку
    with open(file,'r') as f:
```

```
        for ln in f: # для економії памяті
            if ln.startswith(startLine): # початок блоку
                inside=True
            if inside: # всередині блоку
                r=parseLine(ln)
                if r!=None: res[r[0]]=r[1]
            if inside and ln==' -3\n': break # кінець блоку
    return res

def parseLine(ln): # парсить рядок з напруженнями вузла
    if not ln.startswith(" -1"): return
    nodeRes=[]
    s=ln.strip()
    for i in range(7): # розбити рядок на 7 частин по 12 символів
        nodeRes.append(s[12*i:12*i+12])
    node=int(nodeRes[0][2:]) # вузол
    r=[float(i) for i in nodeRes[1:]] # значення
    return node,r

def stressMises(Sx,Sy,Sz,Txy,Tyz,Txz):
    "Екв. напруження за критерієм Мізеса"
    #(1/2**0.5)*((Sx-Sy)**2+(Sy-Sz)**2+(Sz-
Sx)**2+6*Txy**2+6*Tyz**2+6*Txz**2)**0.5
    return (Sx*Sx + Sy*Sy + Sz*Sz - Sx*Sy - Sx*Sz - Sy*Sz + 3*Txy*Txy +
3*Txz*Txz + 3*Tyz*Tyz)**0.5

def principalStress(Sx,Sy,Sz,Sxy,Syz,Szx):
    a=((Sx-Sy)**2/4+Sxy**2)**0.5
    S=[(Sx+Sy)/2+a, Sz, (Sx+Sy)/2-a]
    S.sort(reverse=True)
    return S

def principalStress_(Sx,Sy,Sz,Sxy,Syz,Szx):# for test only
    def fn(S,Sx,Sy,Sz,Sxy,Syz,Szx):
        Syz=0.0;Szx=0.0
        #S**3-(Sx+Sy+Sz)*S**2+(Sx*Sy+Sy*Sz+Sz*Sx-Sxy**2-Syz**2-Szx**2)*S-
(Sx*Sy*Sz+2*Sxy*Syz*Szx-Sx*Syz**2-Sy*Szx**2-Sz*Sxy**2)
        return S**3-(Sx+Sy+Sz)*S**2+(Sx*Sy+Sy*Sz+Sz*Sx-Sxy**2)*S-(Sx*Sy*Sz-
Sz*Sxy**2)

    from scipy import arange
    from scipy.optimize import fsolve
    S=fsolve(fn, arange(-1000., 1000., 200.), args=(Sx,Sy,Sz,Sxy,Syz,Szx))
    S=list(set([round(i,2) for i in S]))
    S.sort(reverse=True)
    return S

def FOS(S1_2, S1_1, S2_2, S2_1, S3_2, S3_1):
    """Розраховує коефіцієнт запасу втомної міцності за критерієм Сайнса
S1_2 - головне напруження 1 крок 2 (максимальне навантаження), МПа
S1_1 - головне напруження 1 крок 1 (мінімальне навантаження)
```

```
"""
    sn = 207.0 #границя витривалості
    m = 1.0 #коефіцієнт

    Sm3 = (S3_2 + S3_1) / 2.
    Sa3 = (S3_2 - S3_1) / 2.

    Sm2 = (S2_2 + S2_1) / 2.
    Sa2 = (S2_2 - S2_1) / 2.

    Sm1 = (S1_2 + S1_1) / 2.
    Sa1 = (S1_2 - S1_1) / 2.

    FOS = (sn - m * (Sm1 + Sm2 + Sm3) / 3.) / (((Sa1 - Sa2) ** 2 + (Sa2 -
Sa3) ** 2 + (Sa3 - Sa1) ** 2) / 2.)**0.5
    return FOS

def getResults(nodes):
    "Результати двох кроків для списку вузлів"
    res1=parseLines("    1PSTEP                          2           1
1")
    res2=parseLines("    1PSTEP                          6           1
2")
    res={}
    for node in nodes:
        SM_1=stressMises(*res1[node])
        S1_1,S2_1,S3_1=principalStress(*res1[node])
        SM_2=stressMises(*res2[node])
        S1_2,S2_2,S3_2=principalStress(*res2[node])
        fos=FOS(S1_2, S1_1, S2_2, S2_1, S3_2, S3_1)
        res[node]=[SM_1,SM_2,fos] # список потрібних результатів
    return res

def minFOSnode(res):
    "вузол з найменшим FOS. res - з getResults"
    minVal=1.e30
    for node in res:
        if res[node][2]<minVal:
            minVal=res[node][2]
            minNode=node
    with open('results.txt','w') as f:
        f.write(str(minVal))
    return minNode,minVal

if __name__=='__main__':
    res=parseLines("    1PSTEP                          2           1
1")
    node=1 #res.keys()[0]
    print node,res[node]
    print stressMises(*res[node])
    print principalStress_(*res[node])
```

```
    print principalStress(*res[node])
    res=getResults([node])
    print minFOSnode(res)
```

Лістинг І.5 – main.py

```
#-*- coding: utf-8 -*-
#from __future__ import unicode_literals
import math,sys
from myBaseGeom import *
# Засоби для створення простого GUI
# розкоментувати, якщо потрібна візуалізація
#from OCC.Display.SimpleGui import init_display
#display, start_display, add_menu, add_function_to_menu = init_display()
#display.set_bg_gradient_color(255,255,255,255,255,255) # колір фону

from OCC.BRepBuilderAPI import *
from OCC.TopoDS import *
from OCC.TopExp import *
from OCC.TopAbs import *
from OCC.BRepFilletAPI import BRepFilletAPI_MakeFillet2d
from OCC.ChFi2d import ChFi2d_ChamferAPI,ChFi2d_FilletAPI
from OCC.BRep import BRep_Tool_Pnt
from OCC.BRepAlgoAPI import *
from OCC.BRepAlgo import *
# замість BRepAlgoAPI_Fuse використовувати BRepAlgo_Fuse, щоб результатом
була одна грань
from OCC.BRepBuilderAPI import BRepBuilderAPI_Sewing
from OCC.Precision import precision_Confusion

from gost13877_96params import Rod19
# параметри можуть передаватись через командний рядок:
d={}
for i in sys.argv[1:]: exec i in None,d
d=Rod19(**d)
#d=Rod19(r3n=2.5)
#d=Rod19(d_n=13.12) #12.7,12.84,12.98,13.12,13.26

def translate(f,x1,y1,x2,y2):
    u"""Переміщення з точки в точку"""
    #gp_Trsf2d
    trsf=gp_Trsf()
    trsf.SetTranslation(gp_Pnt(x1,y1,0),gp_Pnt(x2,y2,0)) # переміщення
    f=BRepBuilderAPI_Transform(f,trsf).Shape()
    return f

def rotate(f, angle):
    u"""Поворот навколо осі z на кут angle"""
    trsf=gp_Trsf() # трансформація
    trsf.SetRotation(gp_Ax1(),angle) # поворот
    f=BRepBuilderAPI_Transform(f,trsf).Shape()
```

```
    return f

def poly(pts):
    u"""Полігон зі скругленнями і фасками
    Приклад:
    poly(pts=[[(0,0),1],[(10,0),(1,1)],[(5,5),None]])
    тут 1 - радіус скруглення біля вершини,
    (1,1) - фаска біля вершини,
    None або 0 - без фаски чи скруглення біля вершини"""
    gpts=[gp_Pnt(p[0][0],p[0][1],0) for p in pts] # точки вершин
    es=[] # список ребер (без фасок і скруглень)
    p1=gpts[-1] # остання точка
    for p2 in gpts: # для кожної точки
        e=BRepBuilderAPI_MakeEdge(p1,p2).Edge() # створити ребро
        es.append(e) # додати у список
        p1=p2 # попередня точка

    ecs=[] # список містить ребра фаски чи скруглення або None
    for i,p in enumerate(pts): # для кожної точки
        if i==len(pts)-1: j=0 # індекс другого ребра
        else: j=i+1
        pcf=p[1] # розміри фаски чи скруглення
        if type(pcf) in [tuple,list]: # якщо фаска
            ch=ChFi2d_ChamferAPI(es[i],es[j]) # фаска за двома ребрами
            ch.Perform()
            ec=ch.Result(es[i],es[j],pcf[0],pcf[1]) # ребро фаски
            # es[i],es[j] - нові ребра біля фаски
        elif not pcf: # без фаски чи скруглення
            ec=None
        else: # скруглення
            fl=ChFi2d_FilletAPI(es[i],es[j],gp_Pln()) # скруглення за двома
ребрами
            fl.Perform(pcf)
            ec=fl.Result(gpts[i],es[i],es[j]) # ребро скруглення
            # es[i],es[j] - нові ребра біля скруглення
        ecs.append(ec) # додати ребро фаски чи скруглення у список
    en=[] # список усіх остаточних ребер
    for e,ec in zip(es,ecs):
        en.append(e) # додати ребро
        if ec: en.append(ec) # додати ребро фаски чи скруглення, якщо не
None

#     # ще один спосіб створення полігону
#     mp = BRepBuilderAPI_MakePolygon()
#     for p in gpts:
#         mp.Add(p)
#     mp.Close()
#     w=mp.Wire()

    mw=BRepBuilderAPI_MakeWire() # створити контур
    for e in en: # для кожного ребра
```

```
        mw.Add(e) # додати в контур
    w=mw.Wire() # контур
    f=BRepBuilderAPI_MakeFace(w).Face() # грань

#     # ще один спосіб створення скруглень
#     mf=BRepFilletAPI_MakeFillet2d(f)
#     ex = TopExp_Explorer(f, TopAbs_VERTEX)
#     hasFillet=False
#     while ex.More():
#         v = topods_Vertex(ex.Current())
#         vp=BRep_Tool_Pnt(v)
#         if v.Orientation()==TopAbs_FORWARD:
#             for i in range(len(gpts)):
#                 if vp.IsEqual(gpts[i],1e-6) and r[i]!=0:
#                     print vp.X(),vp.Y(),vp.Z()
#                     hasFillet=True
#                     mf.AddFillet(v,r[i])
#         ex.Next()
#
#     if hasFillet: f=mf.Shape()

    return f

def rect(Lx,Ly,c=[0,0,0,0]):
    u"""Прямокутник з першим кутом (нижній лівий) в 0,0
    c - фаски чи скруглення кутів"""
    pts=[[(0,0),c[0]],[(Lx,0),c[1]],[(Lx,Ly),c[2]],[(0,Ly),c[3]]]
    return poly(pts)

def cut_array(f1,f2,ps):
    u"""Робить масив вирізів поверхнею f1 у поверхні f2.
    Список точок вирізів ps=[(0,0),(0,1),(0,2)]"""
    p0=ps[0]
    for p in ps[1:]:
        f2=BRepAlgoAPI_Cut(f2, f1).Shape()
        f1=translate(f1,p0[0],p0[1],p[0],p[1])
        p0=p
    f2=BRepAlgoAPI_Cut(f2, f1).Shape()
    return f2

def thread_points_array(p0,L,s):
    u"""Масив точок для побудови різьби довжиною L та кроком s. Крок може
бути від'ємний. p0 - початкова точка"""
    ps=[]
    y=p0[1]
    l=0 # поточна довжина різьби
    while l<=L:
        ps.append((p0[0],y))
        y+=s
        l+=abs(s)
    return ps
```

```
def face1():
    u"""Ніпель"""
    h=d.l1n-d.l3n # фаска ніпеля
    f1=rect(d.d_n, d.l1n, [0,(0.577*h,h),0,0]) # ніпель
    f2=rect(d.dn, d.l4n-d.l1n, [0,0,0,0]) # бурт
    f2=translate(f2, 0, 0, 0, d.l1n)
    f3=BRepAlgo_Fuse(f2, f1).Shape()

    f41=rect(d.dn, d.l2n, [0,0,0,0]) # заготовка канавки
    f41=translate(f41, 0, 0, 0, d.l1n-d.l2n)
    f3=BRepAlgo_Fuse(f3, f41).Shape()

    f4=rect(d.dn-d.d1n, d.l2n, [d.r3n,0,0,d.r3n]) # виріз канавки
    f4=translate(f4, 0, 0, d.d1n, d.l1n-d.l2n)
    f5=BRepAlgoAPI_Cut(f3, f4).Shape()

    tn1=math.tan(math.radians(30)) # tan верхнього кута профілю
    tn2=math.tan(math.radians(30)) # tan нижнього кута профілю
    H=2.2 # висота різця
    f6=poly([[(0,0),d.r_n],[(H,-H*tn2),0],[(H,H*tn1),0]]) # різець
    f6=translate(f6, 0, 0, -H, 0)
    #display.DisplayShape(f6,update=True,color='red')

    f6=translate(f6, 0, 0, d.dn_, d.ln_)
    a=thread_points_array((d.dn_,d.ln_), d.l1n-d.l2n, -d.p_n)
    f=cut_array(f6,f5,a) # ніпель з різьбою

    #print f.ShapeType() # COMPOUND
    # отримати поверхню
    ex = TopExp_Explorer(f, TopAbs_SHELL)
    f=topods_Shell(ex.Current())
    return f

def face2():
    u"""Муфта"""
    f1=rect(d.dm-d.d1m, d.lm, [0,0,(3,0.8),(1.15,2)]) # муфта
    f1=translate(f1,0,0,d.d1m,-(d.lm-d.l1n))
    h=d.d1m-d.d1_m # фаска
    f2=rect(d.dm-d.d1_m, d.lm-10, [0,0,0,(h,1.73*h)]) # заготовка різьбової
частини
    f2=translate(f2, 0, 0, d.d1_m, -(d.lm-d.l1n))
    f2=BRepAlgo_Fuse(f2, f1).Shape()

    tn1=math.tan(math.radians(30)) # tan верхнього кута профілю
    tn2=math.tan(math.radians(30)) # tan нижнього кута профілю
    H=2.2 # висота різця
    #f3=poly([[(0,0),0],[(0.3,-0.1),0],[(0.3,-0.11),0],[(0,-0.21),0]])
    f3=poly([[(0,H*tn1),0],[(H,0),(0.275/0.866, 0.275/0.866)],[(0,-
H*tn2),0]]) # різець
    #display.DisplayShape(f3,update=True,color='red')
```

```
    f3=translate(f3, 0, 0, d.dm_, d.l2m_)
    a=thread_points_array((d.dm_, d.l2m_), d.lm_, -d.p_m)
    f=cut_array(f3,f2,a) # муфта з різьбою

    #print f.ShapeType() # COMPOUND !!!
    # ребро для поділу грані на дві частини
    e1=BRepBuilderAPI_MakeEdge(gp_Pnt(d.d1m,d.l1n-5,0),gp_Pnt(d.dm,d.l1n-
5,0)).Edge()

    # отримати грань
    ex = TopExp_Explorer(f, TopAbs_FACE)
    ff=topods_Face(ex.Current())

    # розділити грань ребром
    from OCC.BRepFeat import BRepFeat_SplitShape
    ss=BRepFeat_SplitShape(f)
    ss.Add(e1,ff)
    f=ss.Shape()
    #print f.ShapeType() # COMPOUND

    # отримати поверхню
    ex = TopExp_Explorer(f, TopAbs_SHELL)
    f=topods_Shell(ex.Current())

    return f

def mkCompaund(f1,f2):
    u"""Об'єднує форми для експорту в формат BRep"""
    from OCC.BRep import BRep_Builder
    f=TopoDS_Compound()
    bb=BRep_Builder()
    bb.MakeCompound(f)
    bb.Add(f,f1)
    bb.Add(f,f2)
    return f

def findContEdges(f, exPoints):
    u"Шукає усі ребра грані f крім тих, що задані exPoints. Для контактних
задач. Повертає список ребер у вигляді крайніх точок p1,p2"
    ex = TopExp_Explorer(f, TopAbs_EDGE) # переглядач ребер
    edges=[] # ребра не для контакту як їх крайні точки
    edgesc=[] # ребра для контакту як їх крайні точки
    for p in exPoints:
        x,y=p[1]
        edges.append(findEdge(f,x,y))
    while ex.More():
        e = topods_Edge(ex.Current()) # поточне ребро
        ps=edgeBoundPts(e)
        if (ps[0][0],ps[0][1],ps[1][0],ps[1][1]) not in edges:
            edgesc.append(ps)
```

```
        ex.Next()
    return edgesc

def edgeBoundPts(e):
    u"""Повертає крайні точки ребра"""
    v1=topexp_FirstVertex(e) # перша вершина ребра
    p1=BRep_Tool_Pnt(v1) # точка за вершиною
    v2=topexp_LastVertex(e) # остання вершина ребра
    p2=BRep_Tool_Pnt(v2) # точка за вершиною
    return ((p1.X(),p1.Y()),(p2.X(),p2.Y()))

def drawEdgePts(e):
    u"Рисує крайні точки ребра"
    x1,y1,x2,y2=e
    display.DisplayShape(gp_Pnt(x1,y1,0), color='black')
    display.DisplayShape(gp_Pnt(x2,y2,0), color='black')

def findEdge(f,x,y):
    u"""Шукає ребро поверхні f за точкою на ньому
    Повертає кортеж першої і останньої точок ребер:
    x1,y1,x2,y2"""
    from OCC.BRep import BRep_Tool_Curve
    from OCC.GeomAdaptor import GeomAdaptor_Curve
    from OCC.GeomLib import GeomLib_Tool_Parameter
    px=gp_Pnt(x,y,0)
    #display.DisplayShape(px, color='black')
    ex = TopExp_Explorer(f, TopAbs_EDGE) # переглядач ребер
    while ex.More():
        e = topods_Edge(ex.Current()) # поточне ребро
        c=BRep_Tool_Curve(e) #! зверніть увагу, що результат є кортежем
        gac=GeomAdaptor_Curve(c[0],c[1],c[2]) # крива
(OCC.Geom.Handle_Geom_Curve), перший параметр, другий параметр

        #tp=gac.GetType() # тип кривої 0 - лінія, 1 - коло, ...
        #print type(gac.Line()) # OCC.gp.gp_Lin
        #p = gp_Pnt()
        #gac.D0((c[1]+c[2])/2.0, p) # p - середня точка кривої
        #display.DisplayShape(p, color='black')
        #gac.Line().Contains(px,1e-9) # чи лінія містить точку

        res=GeomLib_Tool_Parameter(c[0],px,1e-9) # ! зверніть увагу, що
        # результат є кортежем. Це не відповідає документації.
        if res[0] and res[1]>=c[1] and res[1]<=c[2]: # якщо точка на кривій
в заданих межах параметрів
            p1=gac.Value(c[1]) # перша точка кривої
            p2=gac.Value(c[2]) # остання точка кривої
            #display.DisplayShape(p1, color='black')
            #display.DisplayShape(p2, color='black')
            return p1.X(),p1.Y(),p2.X(),p2.Y()
        ex.Next()
    return None
```

```
##IsVertexOnLine
##ComputePE
##VertexParameter

def findEdge2(f,x,y):
    u"""Шукає ребро поверхні f за точкою на ньому"""
    from OCC.BRepExtrema import BRepExtrema_ExtPC
    vx=BRepBuilderAPI_MakeVertex(gp_Pnt(x, y, 0)).Vertex() # вершина
    #display.DisplayShape(vx, color='black')
    extr=BRepExtrema_ExtPC() # знаходить відстані від вершини до ребра
    ex = TopExp_Explorer(f, TopAbs_EDGE) # переглядач ребер
    while ex.More(): # для кожного ребра
        e = topods_Edge(ex.Current()) # поточне ребро
        extr.Initialize(e)
        extr.Perform(vx)
        if extr.IsDone(): # якщо екстремуми знайдені
            for i in range(1,extr.NbExt()+1): # для кожного екстремуму
                if extr.SquareDistance(i)<1e-6: # квадрат відстані до
першого екстремуму
                    p=edgeBoundPts(e)
                    return p[0][0],p[0][1],p[1][0],p[1][1]
        ex.Next()
    return None

def drawMesh(es):
    u"""Рисує сітку елементів es"""
    for e in es: # для кожного елемента
        for s in ccx_inp.elements: # для кожної деталі
            # якщо елемент не цієї деталі, то пропустити
            if not e in ccx_inp.elements[s]: continue
            ns=ccx_inp.elements[s][e] # вузли елемента
            ps=[] # точки gp_Pnt
            for n in ns: # для кожного вузла
                x,y,z=ccx_inp.nodes[n] # координати вузла
                ps.append(gp_Pnt(x,y,z)) # додати точку
                display.DisplayShape(ps[-1]) # показати точку
                display.DisplayMessage(ps[-1],str(n),message_color=(0,0,0))
            poly=BRepBuilderAPI_MakePolygon(ps[0],ps[1],ps[2],True).Shape()
            display.DisplayShape(poly) # показати полігон елемента

def visualize(shape):
    u"""Налаштовує параметри візуалізації і показує форму"""
    #from OCC.Display.OCCViewer import Viewer3d
    display.View_Top()
    display.set_bg_gradient_color(255,255,255,255,255,255)
    # сітка
    for x in range(0,50,10):
        for y in range(0,50,10):
            #for z in range(3):
            display.DisplayShape(gp_Pnt(x,y,0))
```

```
    display.DisplayShape(shape,update=True)

def saveBrep(shape,filename):
    u"""Зберегти форму у форматі BRep"""
    from OCC.BRepTools import breptools_Write # функція PythonOCC для
запису BRep
    breptools_Write(shape,filename) # зберегти в PythonOCC у форматі BRep

############################################################

f1=face1() # ніпель
f2=face2() # муфта
f=mkCompaund(f1,f2)
saveBrep(f,"model.brep")
#visualize(f) # візуалізація форми

p={} # словник характерних точок моделі
p[1]=[f2, d.em1[:2]] # нижній торець муфти
p[2]=[f1, d.en1[:2]] # верхній торець ніпеля
p[3]=[f1, d.em4[:2]] # точка на контактній поверхні бурта
p[4]=[f2, d.em4[:2]] # точка на контактній поверхні бурта
p[5]=[f1, d.en2[:2]] # вісь ніпеля
p[6]=[f2, d.em3[:2]] # лінія поділу муфти
p[7]=[f1, d.en3[:2]] # нижній торець ніпеля
p[8]=[f1, d.en4[:2]] # зовнішній циліндр бурта
p[9]=[f2, d.em2[:2]] # #зовнішній циліндр муфти

import ccx_inp
ccx_inp.mesh()
ccx_inp.elset2nset()
lastNode=ccx_inp.appendNode()
#ccx_inp.reverseOrient()
ccx_inp.parse()

def linFinder():
    """Додає лінії у словник характерних точок"""
    for i in p:
        e=findEdge(p[i][0],*p[i][1])
        p[i].append(ccx_inp.findLine(*e))
linFinder()
#print p[6][2] # назва лінії
#drawEdgePts(findEdge(f2,*p[9][1]))

ce1=findContEdges(f=f1, exPoints=[p[7], p[5], p[2], p[8]])
ce2=findContEdges(f=f2, exPoints=[p[9], p[1], p[6]])
master=ccx_inp.findLines(ce1)
#print master # !warning some None
slave=ccx_inp.findLines(ce2)
print len(master), len(set(master)) # чомусь не рівні?
print len(slave), len(set(slave))
```

```
c={}
c["surfSlaveDefinitions"]=ccx_inp.surfDefs(set(slave),"Slave")
#c["surfSlaveDefinitions"]=ccx_inp.surfNodeDefs(set(slave),"Slave")
c["surfMasterDefinitions"]=ccx_inp.surfDefs(set(master),"Master")
c["surfBoltLoadDefinitions"]=ccx_inp.surfByLineEl(p[6][2],"Surface3")
c["surfLoadDefinitions"]=ccx_inp.dloadDefs(p[2][2],"Surface1",d.load2)
c["boundLine1"]=p[1][2]
c["loadLine1"]=p[2][2]
c["boundLine2"]=p[5][2]
c["preTNode"]=lastNode
c["boltLoad"]=d.bolt_load

ccx_inp.writeFinalINP(c)
if not ccx_inp.runCCX(): print "ccx error"
import ccx_out
res=ccx_out.getResults(ccx_inp.nodesByLines(master))
print "FOS=",ccx_out.minFOSnode(res)[1] # мінімальні FOS на поверхні ніпеля
res=ccx_out.getResults(ccx_inp.nodesByLines([p[4][2]]))
print sum([res[n][1] for n in res])/len(res) # середні напруження на бурті
на другому кроці

"""
# візуалізація елементів
lnels=set() # множина елементів на контактних лініях
for ln in slave+master:
    if not ln: continue
    lnels.update(set(ccx_inp.elementsByLine(ln)))
#та на лінії BoltLoad
lnels.update(set(ccx_inp.elementsByLine(p[6][2],"Surface2")))
#та на лінії навантаження
lnels.update(set(ccx_inp.elementsByLine(p[2][2],"Surface1")))
drawMesh(lnels)
#drawMesh(ccx_inp.elements['Surface1'])
"""
#start_display()
```

ДОДАТОК Й

Пакет для геометричного моделювання різьб з відхиленнями за допомогою FreeCAD API

Код програм доступний також в GitHub (vkopey/Thread-turning-simulator. URL: http://github.com/vkopey/Thread-turning-simulator). Вміст пакету:

– main.py – головний модуль для побудови геометричної моделі;

– dimsNKT.py – параметри з'єднання гладких НКТ.

Лістинг Й.1 – main.py

```
# -*- coding: utf-8 -*-
import sys
# якщо freecad 64 біт, то python повинен бути 2.7 64 !
FREECADPATH = "e:\\FreeCAD 0.17x64\\bin"
sys.path.append(FREECADPATH) # шлях до бібліотек FreeCAD
import FreeCAD as App # модуль для роботи з програмою
import FreeCADGui as Gui # модуль для роботи з GUI
import Part # workbench-модуль для створення і керування BRep об'єктами
import numpy as np
from dims import d

##
def show(name): # показує форму
    doc.addObject("Part::Feature",name).Shape=globals()[name]

def helixPoints(r,h,p,fi,n): # конічна гвинтова лінія
    "r-менший радіус, h-висота, p-крок, fi - кут (рад), n-кількість точок на витку"
    a=np.tan(fi)*p/(2*pi)
    b=p/(2*pi)
    k=h/p # кількість витків
    N=int(n*k) # загальна кількість точок
    t=np.linspace(0,h/b,N) # масив значень t
    x=(r+a*t)*np.cos(-t) # масив значень x
    z=(r+a*t)*np.sin(-t) # масив значень y
    y=b*t # масив значень y
    # (OY - вісь гвинтової лінії)
    return zip(t,x,y,z)

def perror(points):
    epoints=[] # точки з відхиленнями
    d=0.2 # параметр трикутного розподілу
    for t,x,y,z in points:
        x=x+np.random.triangular(-d,0,d)
```

```
34          y=y+np.random.triangular(-d,0,d)
35          z=z+np.random.triangular(-d,0,d)
36          epoints.append((t,x,y,z))
37      return epoints
38
39  def rebuildSketch(dim, sk):
40      "Перебудовує ескіз і повертає грань. dim - розміри ескізу sk"
41      doc=App.getDocument("Sketches") # об’єкт документа
42      sketch=doc.__getattribute__(sk) # ескіз або doc.<назва ескізу>
43      # для кожної пари (обмеження ескізу ID, значення розміру)
        for k,v in dim.iteritems():
44          sketch.setDatum(k,v) # установити значення розміру
45      doc.recompute() # перебудувати документ
46      w=sketch.Shape.Wires[0] # отримати перший цикл
47      f=Part.Face(w) # створити грань
48      return f
49
50  def revolve(f):
51      """Створює тіло шляхом обертання грані f навколо осі Y"""
52      return f.revolve(App.Vector(0,0,0), App.Vector(0,1,0)) # тіло
53
54  def helix(r,h,p,fi):
55      "Гвинтова лінія. r-радіус, h-висота, p-крок, fi-кут конуса (рад.)"
56      w=Part.makeLongHelix(p,h,r,np.degrees(fi)) # !!! але не makeHelix
57      #FreeCAD-master\src\Mod\Part\App\TopoShape.cpp
58      w.rotate(App.Vector(0,0,0),App.Vector(1,0,0),-90)
59      return w
60
61  def makeThread(f,h,s): # різьба ніпеля/муфти
62      s2=h.makePipeShell(f.Wires,True,True) # тіло спіралі; дозволяє
    кілька профілів!
63      s=s.cut(s2) # булева операція вирізання
64      return s
65
66  def helix2(points): # гвинтова лінія з відхиленнями
67      pts=[(x,y,z) for t,x,y,z in points]
68      h=Part.makePolygon(pts) # полінія
69      return h
70
71  def makeThread2(f,h,s): # різьба з відхиленнями
72      w=f.Wires[0] # цикл інструмента
73      points=[(v.X,v.Y,v.Z) for v in h.Vertexes] # точки на гвинт. лінії
74      W=[] # профілі в різних точках лінії
75      for x,y,z in points: # для кожної точки
76          w_=w.copy() # копія циклу інструмента
77          t=np.arctan2(-z,x) # кут повороту різця відносно заготовки
78          w_.rotate(App.Vector(0,0,0),App.Vector(0,1,0),np.degrees(t)) #
    повернути інструмент навколо осі Y
79          w_.translate(App.Vector(x,y,z)) # перемістити інструмент
80          W.append(w_.copy()) # додати профіль до списку W
81          if len(W)>1:
```

```
            st=Part.makeLoft([W[-2],W[-1]],True) # створити тіло за двома сусідніми профілями
            s=s.cut(st) # виріз в заготовці тілом st

    return s,W

#S={} # форми
print "OCC", Part.OCC_VERSION
pi=np.pi
App.open(u"D:/3/Sketches.FCStd") # відкрити документ

## ніпель
fA0=rebuildSketch(dim={6:d.fi, 9:d.H}, sk='Sketch') # грань інструмента
fA0.translate(d._v2) # перемістити інструмент в поч. позицію
sA0=revolve(rebuildSketch(dim={20:d.fi, 16:d.d3/2}, sk='Sketch001')) # тіло заготовки
hA0=helix(r=d._r, h=d.l4-12, p=d.P, fi=d.fi) # гвинтова лінія
hA0.translate((0,d.l3-d.l4,0)) # перемістити лінію
sA1=makeThread(fA0, hA0, sA0) # ніпель з різьбою

## муфта
fB2=rebuildSketch(dim={10:(pi-d.fi), 9:d.H}, sk='Sketch002') # tool
fB2.translate(d._v2)
#f2.rotate(App.Vector(0,0,0),App.Vector(0,1,0),1)
sB2=revolve(rebuildSketch(dim={28:d.fi, 17:d.d3/2}, sk='Sketch003'))
hB2=helix(r=d._r, h=d.l4, p=d.P, fi=d.fi)
hB2.translate((0,d.l3-d.l4,0))
sB3=makeThread(fB2, hB2, sB2)

## ніпель без відхилень
fA3=rebuildSketch(dim={6:d.fi, 9:d.H}, sk='Sketch') # tool
fA3.translate((-d.H/2+np.tan(d.fi)*d.P/2, d.P/2, 0))
points=helixPoints(r=d._r, h=d.l4-12, p=d.P, fi=d.fi, n=50)
hA3=helix2(points)
hA3.translate((0,d.l3-d.l4,0))
sA3,W=makeThread2(fA3, hA3, sA0)

## ніпель з відхиленнями
fA3.rotate(App.Vector(0,0,0),App.Vector(0,1,0),-10) # утворити γ
points=perror(points) # точки гвинт лінії з відхиленнями
hA3=helix2(points) # гвинтова лінія з відхиленнями
hA3.translate((0,d.l3-d.l4,0)) # перемістити
sA4,W=makeThread2(fA3, hA3, sA0) # ніпель з різьбою

# булеві операції над плоскими перерізами
f=Part.makePlane(400,400,App.Vector(-200,-200,0),App.Vector(0,0,1)) # площина XY
f1=sA3.common(f).Faces[0] # осьовий перетин ніпеля
f2=sA4.common(f).Faces[0] # осьовий перетин муфти
f3=f1.cut(f2).fuse(f2.cut(f1)) # операція XOR
print f3.Area # XOR-площа
```

```
130
131 ##
132 # Наступні команди потрібні тільки для візуалізації створених форм
133 Gui.showMainWindow() # показати головне вікно
134 Gui.activateWorkbench("PartWorkbench")
135 doc=App.newDocument() # створити новий документ
136 # показати форми
137 show('fA0')
138 show('sA1')
139 show('sB3')
140 show('fA3')
141 show('hA3')
142 show('sA3')
143 show('sA4')
144 show('f1')
145 show('f2')
146 show('f3')
147 for w in W[:10]:
148     Part.show(w)
149
150 doc.recompute() # перебудувати
151 Gui.exec_loop() # головний цикл програми
```

Лістинг Й.2 – dimsNKT.py

```
# -*- coding: utf-8 -*-
from math import atan, degrees, tan

class Dim:
    "Клас описує поняття розміру"
    n=0.0 # номінальний розмір
    ei=0.0 # нижнє відхилення
    es=0.0 # верхнє відхилення
    v=0.0 # дійсне значення
    def __init__(self,n,ei,es,doc):
        "конструктор"
        self.v=n
        self.n=n
        self.ei=ei
        self.es=es
        self.__doc__=doc.decode('utf-8')
    def min(self):
        "повертає мінімальний розмір"
        return self.n+self.ei
    def max(self):
        "повертає максимальний розмір"
        return self.n+self.es

nkt114={'D':Dim(114.3,0.0,0.0,"зовнішній діаметр труби"),
    'd':Dim(100.3,0.0,0.0,"внутрішній діаметр труби"),
```

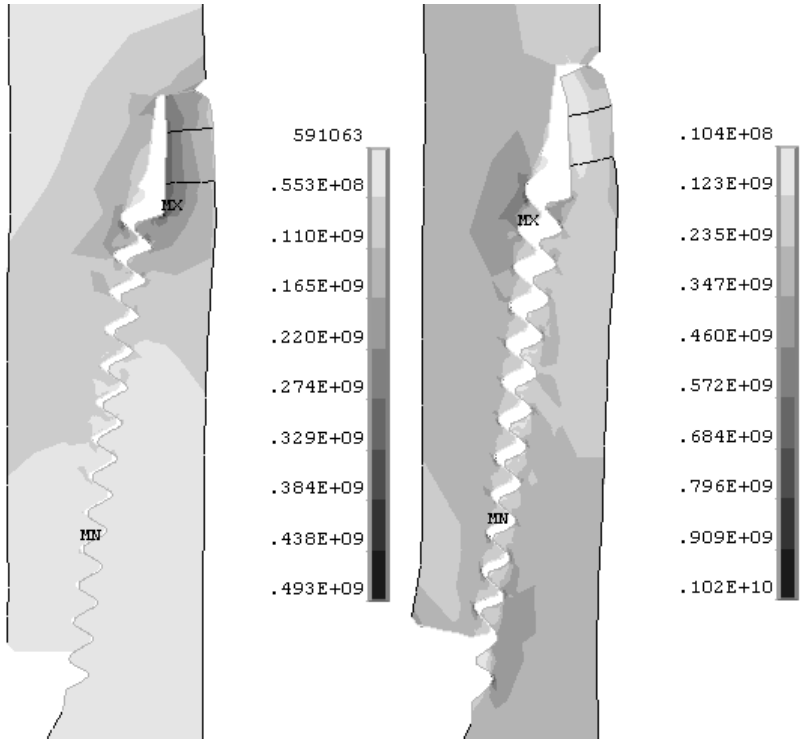
```
    'Dm':Dim(132.1,0.0,0.0,"зовнішній діаметр муфти"),
    'Lm':Dim(156.0,0.0,0.0,"довжина муфти*"),
    'P':Dim(3.175,0.0,0.0,"крок різьби паралельно осі різьби"),
    'dsr':Dim(112.566,0.0,0.0,"середній діаметр різьби в основній
площині"),
    'd1':Dim(111.031,0.0,0.0,"зовнішній діаметр різьби в площині торця
труби"),
    'd2':Dim(107.411,0.0,0.0,"внутрішній діаметр різьби в площині торця
труби"),
    'L':Dim(65.0,-3.2,3.2,"загальна довжина різьби труби"),
    'l':Dim(52.3,0.0,0.0,"довжина різьби труби до основної площини (з
повним профілем)"),
    'l1':Dim(10.0,0.0,0.0,"максимальна довжина збігу різьби труби"),
    'd3':Dim(111.219,0.0,0.0,"внутрішній діаметр різьби в площині торця
муфти"),
    'd0':Dim(115.9,0.0,0.8,"діаметр циліндричної виточки муфти"),
    'l0':Dim(9.5,-0.5,1.5,"глибина виточки муфти"),
    'A':Dim(6.5,0.0,0.0,"натяг при згвинчуванні вручну"),
    'fi':Dim(atan(1.0/32),0.0,0.0,"кут нахилу (рад)"),
    'H':Dim(2.75,0.0,0.0,"висота вихідного профілю"),
    'h1':Dim(1.81,-0.1,0.05,"висота профілю різьби"),
    'h':Dim(1.734,0.0,0.0,"робоча висота профілю"),
    'alfa_2':Dim(30.0,-1.0,1.0,"кут нахилу сторони профілю alfa/2"),
    'r':Dim(0.508,0.0,0.045,"радіус заокруглення вершини профілю"),
    'r1':Dim(0.432,-0.045,0.0,"радіус заокруглення впадини профілю")}

nkt33={'D':Dim(33.4,0.0,0.0,"зовнішній діаметр труби"),
    'd':Dim(22.6,0.0,0.0,"внутрішній діаметр труби"),
    'Dm':Dim(42.2,0.0,0.0,"зовнішній діаметр муфти"),
    'Lm':Dim(84.0,0.0,0.0,"довжина муфти*"),
    'P':Dim(2.54,0.0,0.0,"крок різьби паралельно осі різьби"),
    'dsr':Dim(32.065,0.0,0.0,"середній діаметр різьби в основній площині"),
    'd1':Dim(32.382,0.0,0.0,"зовнішній діаметр різьби в площині торця
труби"),
    'd2':Dim(29.568,0.0,0.0,"внутрішній діаметр різьби в площині торця
труби"),
    'L':Dim(29.0,-2.5,2.5,"загальна довжина різьби труби"),
    'l':Dim(16.3,0.0,0.0,"довжина різьби труби до основної площини (з
повним профілем)"),
    'l1':Dim(8.0,0.0,0.0,"максимальна довжина збігу різьби труби"),
    'd3':Dim(31.210,0.0,0.0,"внутрішній діаметр різьби в площині торця
муфти"),
    'd0':Dim(35.0,0.0,0.8,"діаметр циліндричної виточки муфти"),
    'l0':Dim(8.0,-0.5,1.5,"глибина виточки муфти"),
    'A':Dim(5.0,0.0,0.0,"натяг при згвинчуванні вручну"),
    'fi':Dim(atan(1.0/32),0.0,0.0,"кут нахилу (рад)"),
    'H':Dim(2.2,0.0,0.0,"висота вихідного профілю"),
    'h1':Dim(1.412,-0.1,0.05,"висота профілю різьби"),
    'h':Dim(1.336,0.0,0.0,"робоча висота профілю"),
    'alfa_2':Dim(30.0,-1.0,1.0,"кут нахилу сторони профілю alfa/2"),
    'r':Dim(0.432,0.0,0.045,"радіус заокруглення вершини профілю"),
```

```
    'r1':Dim(0.356,-0.045,0.0,"радіус заокруглення впадини профілю")}

D=nkt114 #nkt33 вибрати типорозмір
class Dims(object): pass
d=Dims() # атрибутами є об'єкти класу Dim, що зручніше ніж словник D
for key,value in D.iteritems():
    setattr(d,key,value)

# дійсні розміри ескіза:
"""
d.D.v=d.D.n/2
d.d.v=d.d.max()/2
d.Dm.v=d.Dm.min()/2
d.Lm.v=d.Lm.n/2
d.P.v=d.P.n
d.dsr.v=d.dsr.n/2
d.d1.v=d.d1.n/2
d.d2.v=d.d2.n/2
d.L.v=d.L.min()
d.l.v=d.l.n
d.l1.v=d.l1.n
d.d3.v=d.d3.n/2
d.d0.v=d.d0.min()/2
d.l0.v=d.l0.max()
d.A.v=d.A.n
d.fi.v=d.fi.n
d.H.v=d.H.n
d.h1.v=d.h1.max()
d.h.v=d.h.n
d.alfa_2.v=d.alfa_2.n
d.r.v=d.r.max()
d.r1.v=d.r1.min()
"""
d.D.v=d.D.n/2
d.d.v=d.d.n/2
d.Dm.v=d.Dm.n/2
d.Lm.v=d.Lm.n/2
d.P.v=d.P.n
d.dsr.v=d.dsr.n/2
d.d1.v=d.d1.n/2
d.d2.v=d.d2.n/2
d.L.v=d.L.n
d.l.v=d.l.n
d.l1.v=d.l1.n
d.d3.v=d.d3.n/2
d.d0.v=d.d0.n/2
d.l0.v=d.l0.n
d.A.v=d.A.n
d.fi.v=d.fi.n
d.H.v=d.H.n
d.h1.v=d.h1.n
```

```
d.h.v=d.h.n
d.alfa_2.v=d.alfa_2.n
d.r.v=d.r.max()
d.r1.v=d.r1.min() #0.3 - для FEA без радіусів при вершині

# приклад використання:
#print d.D.v
#print d.D.n
#print d.D.min()
#print d.D.__doc__

##
# Або доступ до дійсного значення без атрибута v:
class F(float, Dim): pass
d=Dims() # атрибутами є дійсні розміри ескізу
for key,value in D.iteritems():
    setattr(d,key,F(value.v))
    getattr(d,key).v=value.v
    getattr(d,key).n=value.n
    getattr(d,key).ei=value.ei
    getattr(d,key).es=value.es
    getattr(d,key).__doc__=value.__doc__

# приклад використання:
#print d.D
#print d.D.n
#print d.D.min()
#print d.D.__doc__

# допоміжні параметри:
# Увага! не враховано відхилення h1
d.tanFi=tan(d.fi) # тангенс кута нахилу (зміна радіуса конуса на одиницю
довжини)
d._l=d.L-d.A # відстань від 0 до верхнього торця муфти
d._l2=d.Lm-d._l # відстань від 0 до нижнього торця муфти
d._r=d.dsr-d.tanFi*(d.l+d._l2) # радіус середнього діаметра в нижньому
торці муфти
x1,y1 = -d.H/2, 0 # вектор переміщення в 0,0 сер. діам. різця ніпеля
x2,y2 = -d.H/2+d.tanFi*d.P/2, d.P/2 # -//- муфти
d._v1 = x1+d._r, y1-d._l2, 0  # поч. положення різця ніпеля
d._v2 = x2+d._r, y2-d._l2, 0 # поч. положення різця муфти
```

ДОДАТОК К

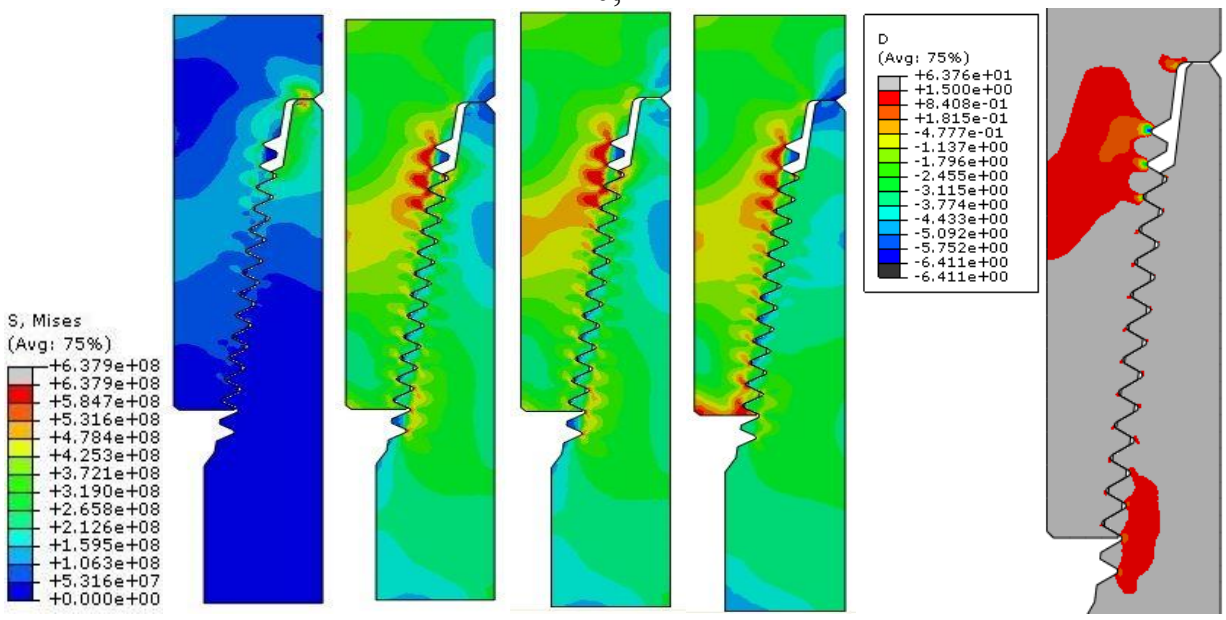
Результати моделювання замкового РЗ



а б

а) – навантаження 0 Н; б) – навантаження 1 МН

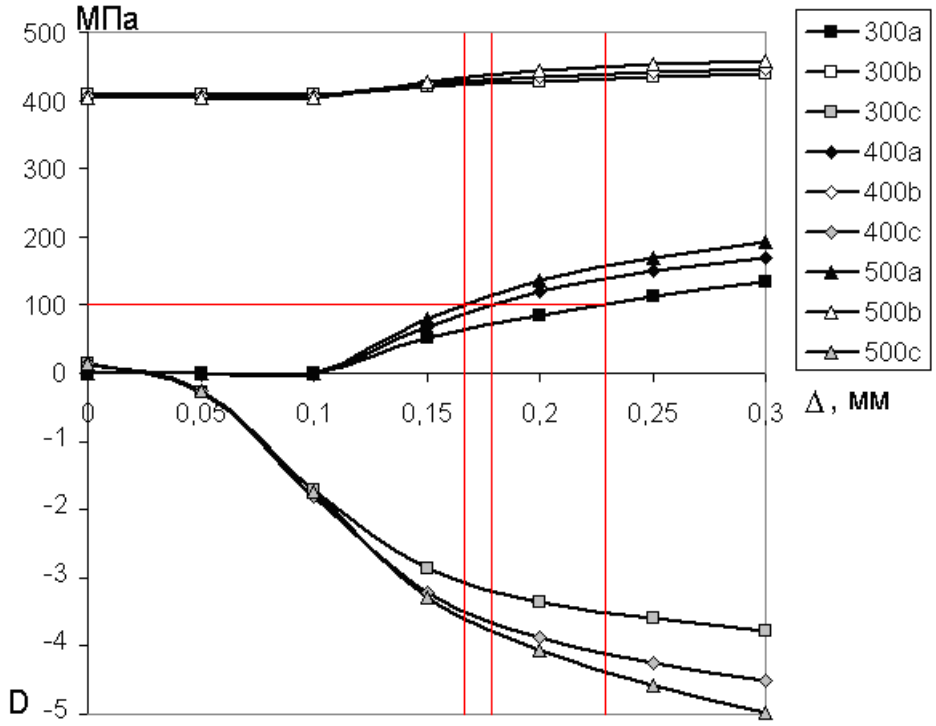
Рисунок К.1 – Розподіл напружень $\sigma_м$ (Па) в пружній моделі РЗ З-66 замка ЗН-80 з Δ=0,1мм



а б в г д

а – навантаження 0 МН, Δ=0,1 мм; б – навантаження 1 МН, Δ=0,1 мм; в – навантаження 1 МН, Δ=0,2 мм; г – навантаження 1 МН, Δ=0,1 мм, муфта з пластичнішого матеріалу

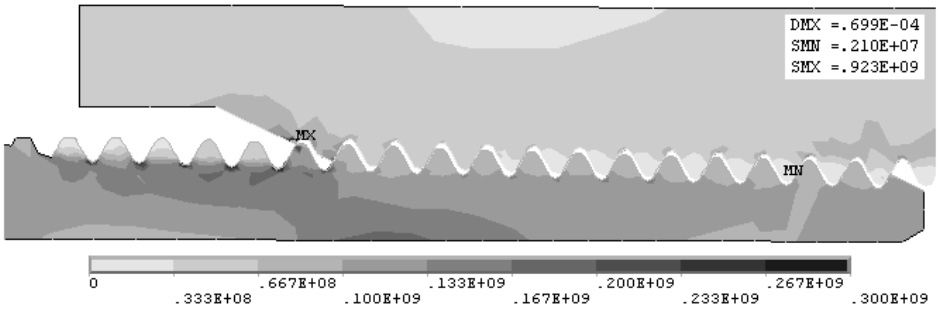
Рисунок К.2 – Розподіл напружень $\sigma_М$ (Па) та коефіцієнта D для циклу навантаження $\sigma_р$=0...155 МПа (д) в пружно-пластичній моделі РЗ З-66 замка ЗН-80



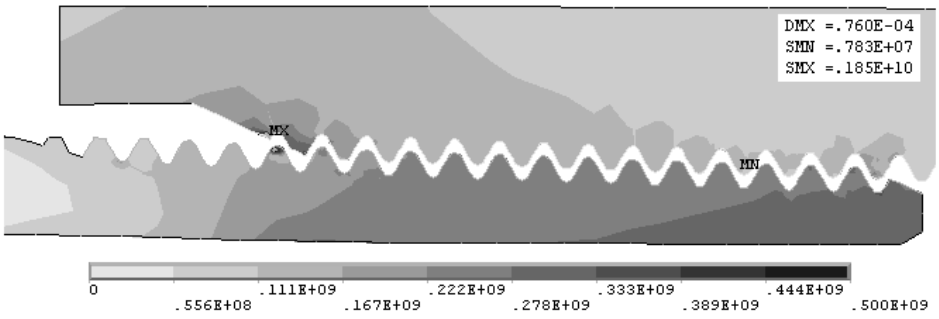
a – величина контактного тиску в місці стику під час максимального навантаження; b – напруження $\sigma_м$ в першій робочій западині різьби ніпеля під час максимального навантаження; c – коефіцієнт запасу втомної міцності *D* в першій робочій западині різьби ніпеля

Рисунок К.3 – Діаграма для визначення оптимальної величини згвинчування Δ (мм) РЗ замка ЗН-80 для циклу навантаження $\sigma_р$=0...155 МПа і значень $\sigma_т$ матеріалу муфти 300, 400 і 500 МПа

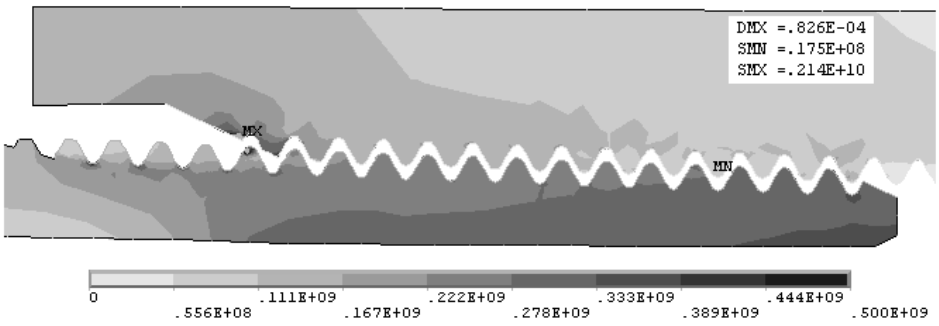
ДОДАТОК Л
Результати моделювання РЗ гладких НКТ умовним діаметром 114 мм (ГОСТ 633-80) з пружною моделлю матеріалу



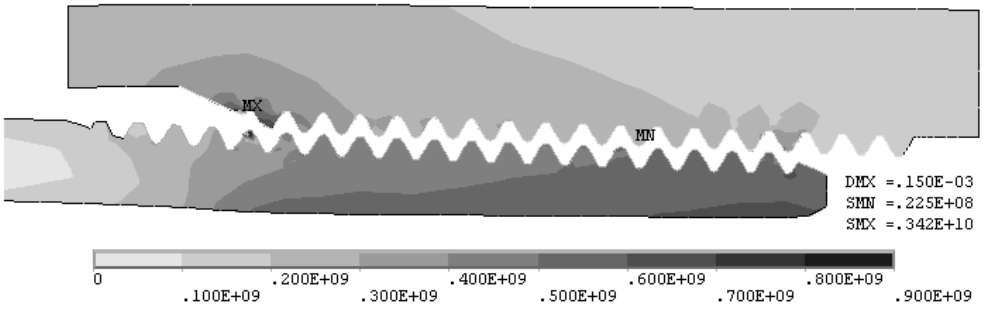
а



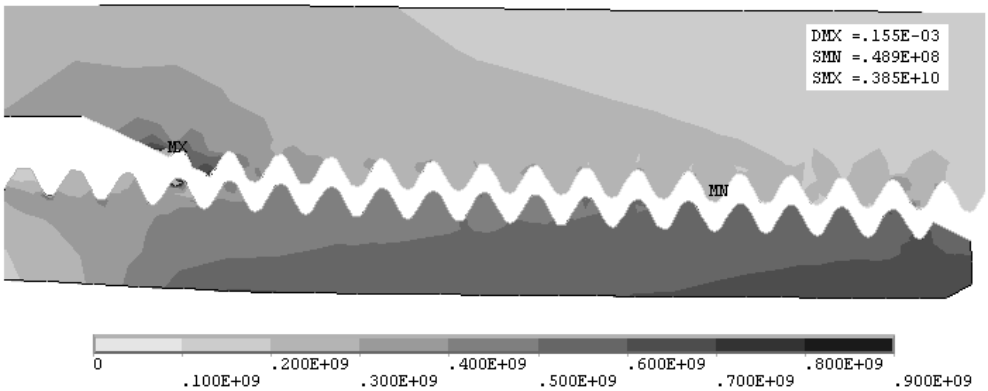
б



в



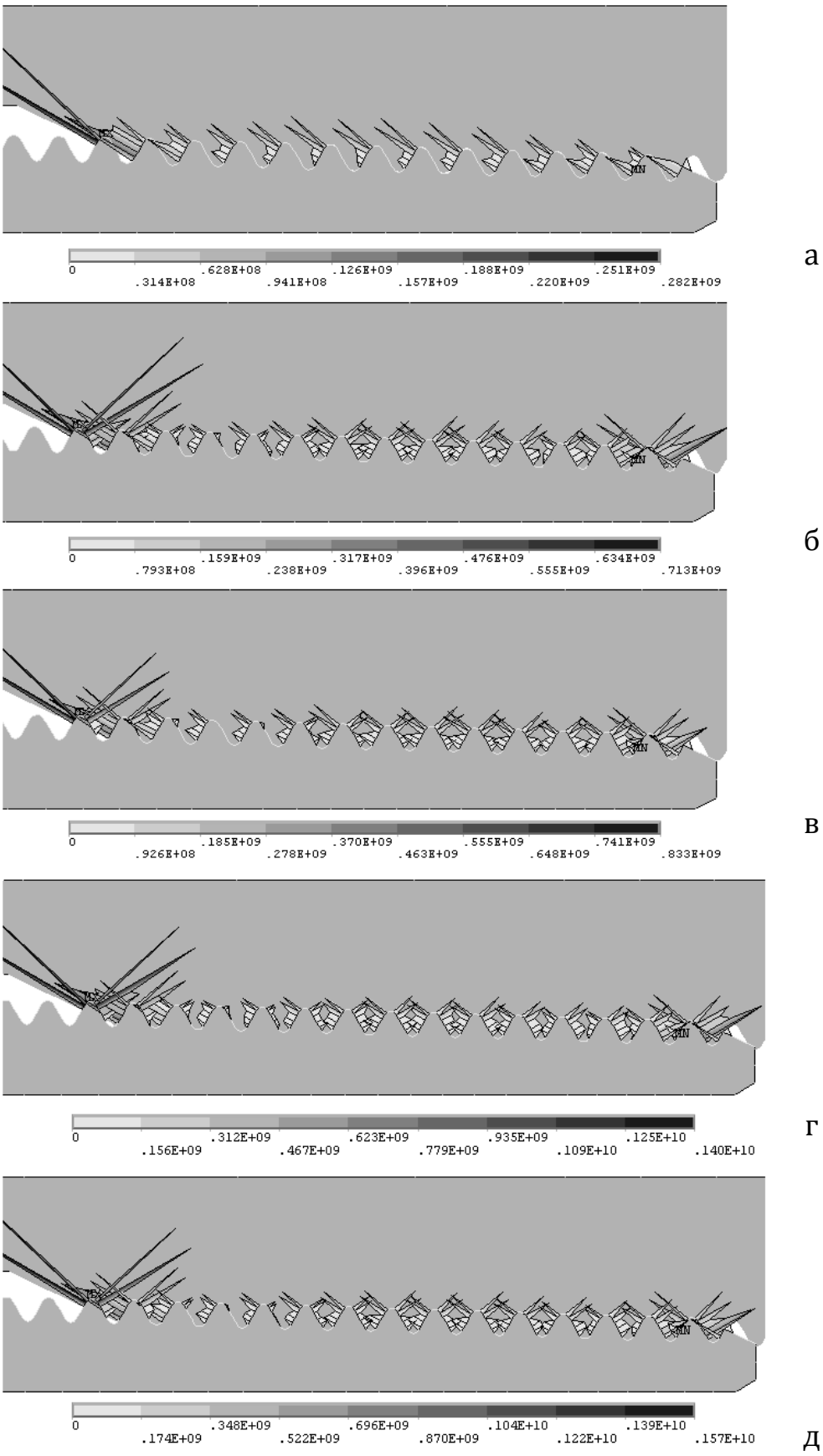
г



д

а) - *A*=6,35 мм; б, в) - *A*=3,175 мм; г, д) - *A*=0 мм; б, г) - $\sigma_р$=0 МПа; а, в, д) - $\sigma_р$=100 МПа

Рисунок Л.1 – Розподіл $\sigma_м$ (Па) в муфтовому РЗ НКТ



а) - A=6,35 мм; б, в) - A=3,175 мм; г, д) - A=0 мм;
б, г) - $\sigma_р$=0 МПа; а, в, д) - $\sigma_р$=100 МПа

Рисунок Л.2 – Значення контактного тиску на витках РЗ НКТ

ДОДАТОК М

Input-файли Abaqus для відтворення гармонічного і динамічного аналізу

Лістинг М.1 – Частина Input-файлу Abaqus для відтворення гармонічного аналізу

```
**Властивості матеріалу
*Material, name=Material-1
*Density
 7.8e-09,
*Elastic
210000., 0.28
** Залежність напруження-деформація (табл.1)
*Plastic
314.,         0.
 349., 0.0013
...
** Властивості контактної взаємодії, коефіцієнт тертя
*Surface Interaction, name=IntProp-1
1.,
*Friction, slip tolerance=0.005
 0.06,
*Surface Behavior, pressure-overclosure=HARD
** Контактні взаємодії поверхонь
*Contact Pair, interaction=IntProp-1, small sliding, type=SURFACE TO
SURFACE
_PickedSurf50, _PickedSurf49
** Перший крок навантажування - згвинчування
*Step, name=Step-1, nlgeom=YES
*Static
1., 1., 1e-05, 1.
** Гранична умова на нижньому торці муфти Uy=URx=URz=0
*Boundary
_PickedSet54, YSYMM
** Навантаження - тиск на верхньому торці -1 МПа
*Dsload
_PickedSurf53, P, -1.
** Зменшувати перекриття контактних поверхонь для імітації згвинчування
*Contact Interference, shrink
_PickedSurf50, _PickedSurf49
** Виведення результатів
*Restart, write, frequency=0
*Output, field, variable=PRESELECT
*Output, history, variable=PRESELECT
*End Step
** Другий крок - статичний осьовий розтяг
*Step, name=Step-2, nlgeom=YES
*Static
1., 1., 1e-05, 1.
```

```
** Навантаження - тиск на верхньому торці -155.1 МПа
*Dsload
_PickedSurf53, P, -155.1
** Виведення результатів
*Restart, write, frequency=0
*Output, field, variable=PRESELECT
*Output, history, variable=PRESELECT
*End Step
** Третій крок - гармонічний аналіз в діапазоні 10000..20000 Гц
*Step, name=Step-3, nlgeom=NO, perturbation
*Steady State Dynamics, direct, real only, frequency scale=LINEAR, friction
damping=NO
10000., 20000., 500, 1.
** Гармонічне навантаження - дійсна частина тиску на верхньому торці -10
МПа
*Dsload, real
Surf-4, P, -10.
** Виведення результатів
*Output, field, variable=PRESELECT
*Output, history, variable=PRESELECT
*End Step
```

Лістинг М.2 – Частина Input-файлу Abaqus для відтворення неявного динамічного аналізу

```
** Опис гармонічного навантаження частотою 55000 рад/с, амплітудою -10 МПа
і середнім значенням -155.1 МПа
*Amplitude, name=Amp-1, definition=PERIODIC
1,          55000.,              0.,          -155.1
             0.,            -10.
** Крок тривалістю 0,005с неявного динамічного аналізу з прямим
інтегруванням
*Step, name=Step-3, nlgeom=YES, inc=1000
*Dynamic,direct,initial=NO
5e-06,0.005,
** Навантаження - змінний тиск на верхньому торці за законом Amp-1
*Dsload, op=NEW, amplitude=Amp-1
Surf-4, P, 1.
*Dsload, op=NEW
*Restart, write, frequency=0
*Output, field, variable=PRESELECT, frequency=1
*Output, history, variable=PRESELECT, frequency=1
*End Step
```

ДОДАТОК Н

VBA-макрос для обчислення коефіцієнта запасу втомної міцності за критерієм Сайнса в SOLIDWORKS Simulation

Код доступний також в GitHub (vkopey/FOS: VBA macro for SOLIDWORKS API for fatigue safety factor calculation by Sines' method. URL: http://github.com/vkopey/FOS).

Лістинг Н.1 – calcFOS.bas

```
Attribute VB_Name = "Macro11"
'Необхідно додати в Tools/References... SolidWorks Simulation 2018 type
library
Dim swApp As Object
Dim COSMOSWORKS As Object
Dim CWObject As CosmosWorksLib.CwAddincallback
Dim ActDoc As CosmosWorksLib.CWModelDoc
Dim StudyMngr As CosmosWorksLib.CWStudyManager
Dim Study As CosmosWorksLib.CWStudy
Dim errCode As Long 'код помилки
Dim CWResult As CosmosWorksLib.cwResults
'Dim LBCMgr As CosmosWorksLib.CWLoadsAndRestraintsManager
'Dim lr As CosmosWorksLib.CWLoadsAndRestraints
Dim Face As SldWorks.Face2

' Розраховує коефіцієнт запасу втомної міцності за критерієм Сайнса
' S1_2 - головне напруження 1 крок 2 (максимальне навантаження), МПа
' S1_1 - головне напруження 1 крок 1 (мінімальне навантаження)
Public Function FOS(S1_2, S1_1, S2_2, S2_1, S3_2, S3_1 As Double)
    sn = 207 '000000 ' границя витривалості
    m = 1 ' коефіцієнт

    Sm3 = (S3_2 + S3_1) / 2
    Sa3 = (S3_2 - S3_1) / 2

    Sm2 = (S2_2 + S2_1) / 2
    Sa2 = (S2_2 - S2_1) / 2

    Sm1 = (S1_2 + S1_1) / 2
    Sa1 = (S1_2 - S1_1) / 2

    FOS = (sn - m * (Sm1 + Sm2 + Sm3) / 3) / Sqr(((Sa1 - Sa2) ^ 2 + (Sa2 -
Sa3) ^ 2 + (Sa3 - Sa1) ^ 2) / 2)
End Function

Sub main()
```

```
Dim s1(1 To 2), s2(1 To 2), s3(1 To 2) As Double ' головні напруження
Set swApp = Application.SldWorks ' об'єкт Solidworks
Set CWObject = swApp.GetAddInObject("SldWorks.Simulation") ' об'єкт
Simulation
Set COSMOSWORKS = CWObject.COSMOSWORKS
Set ActDoc = COSMOSWORKS.ActiveDoc() ' активний документ COSMOSWORKS
Set StudyMngr = ActDoc.StudyManager() ' менеджер задач
Set Part = swApp.ActiveDoc ' активна деталь

For Each N In Array(6313, 6334, 6349, 198, 186) ' вузол
For i = 1 To 2
StudyMngr.ActiveStudy = i - 1
Set Study = StudyMngr.GetStudy(i - 1) ' задача

'Study.MeshAndRun
'errCode = Study.CreateMesh(0, 4.7, 0.25) ' створити сітку
'runError = Study.RunAnalysis ' виконати задачу
Set CWResult = Study.results ' результати

MaxStep = CWResult.GetMaximumAvailableSteps()
sn = CWResult.GetStress(0, MaxStep, Nothing, 3, errCode) ' масив напружень
в вузлах

s1(i) = sn((N - 1) * 12 + 7) ' перше головне напруження (МПа)
s2(i) = sn((N - 1) * 12 + 8) ' друге
s3(i) = sn((N - 1) * 12 + 9) ' третє
'Debug.Print i, s1(i), s2(i), s3(i)
Next i
Debug.Print N, FOS(s1(2), s1(1), s2(2), s2(1), s3(2), s3(1))
Next N

End Sub
```

ДОДАТОК О

Варіанти конструкції пресового з’єднання полімерного стержня зі сталевою оболонкою

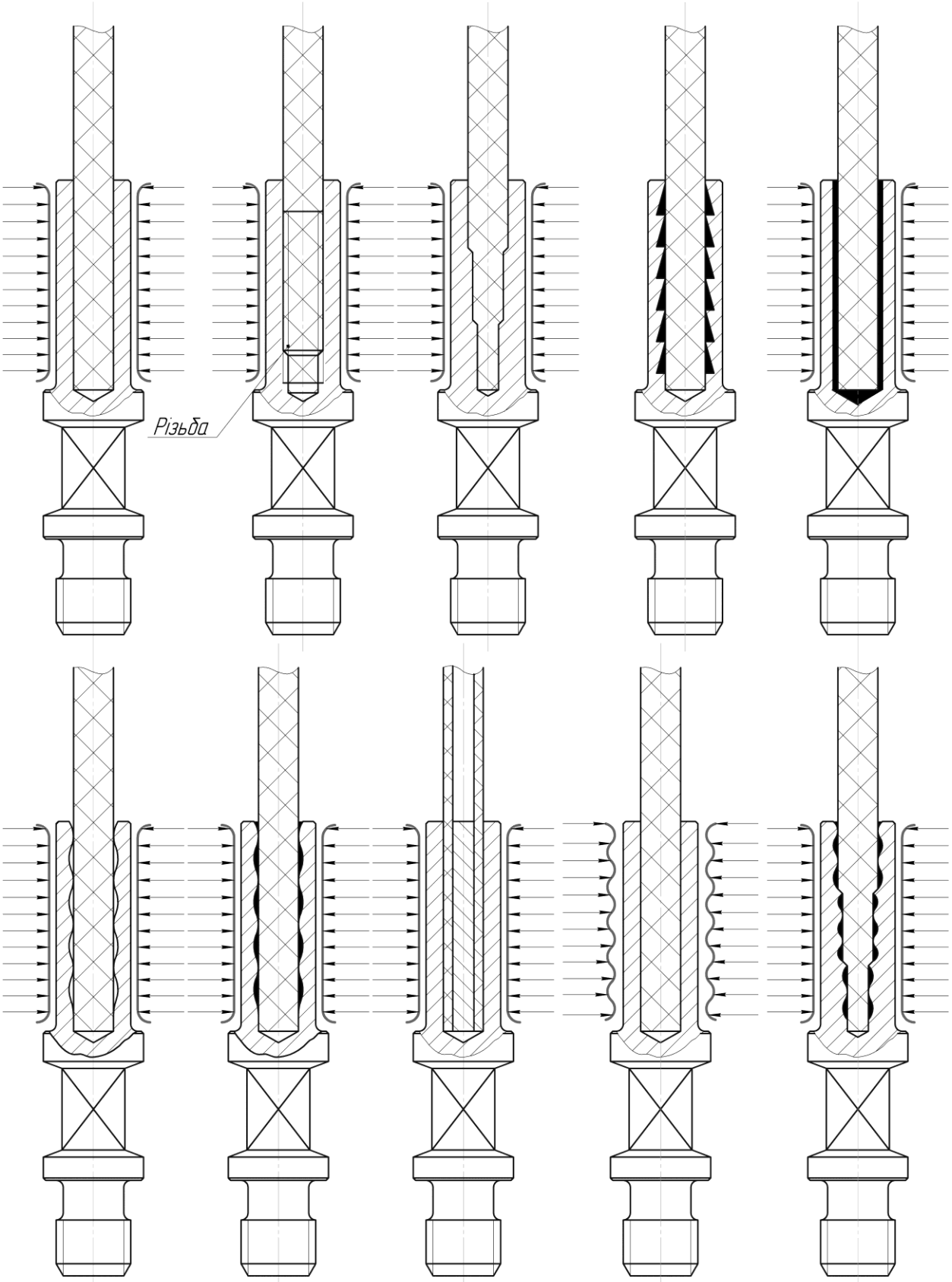


Рисунок О.1 – Варіанти конструкції з’єднання

ДОДАТОК П

Макрос Abaqus/CAE для розрахунку з'єднання полімерного стержня зі сталевою оболонкою

Код доступний також в GitHub (vkopey/Fiberglass-Sucker-Rod-Connection: FEM model of fiberglass sucker rod connection. URL: http://github.com/vkopey/Fiberglass-Sucker-Rod-Connection).

Лістинг П.1 – main.py

```
# -*- coding: cp1251 -*-
'''Макрос Abaqus CAE для розрахунку з'єднання полімерного стержня зі
сталевою оболонкою'''
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *

class Material:
    '''Клас описує поняття матеріалу
    В Abaqus визначається істинна діаграма деформування (див. Stress and
strain measures)
    E - модуль пружності, Па
    mu - коефіцієнт Пуассона
    st - границя текучості, Па
    et - деформація для st
    sb - істинна границя міцності, Па (sv - умовна границя)
    eb - істинна деформація, яка відповідає границі міцності
    delta - відносне видовження
    psi - відносне звуження
    '''
    def __init__(self,E,mu,st,sv,delta,psi):
        '''конструктор'''
        self.E=E # модуль пружності
        self.mu=mu # коефіцієнт Пуассона
        self.st=st # границя текучості
        self.et=st/E # деформація для st
        self.delta=delta/100.0 # відносне видовження після розриву
```

```
        self.psi=psi/100.0 # відносне звуження після розриву
        k=0.4 # коефіцієнт(eb=(0.1...0.4,0.2...0.8)delta)
        self.sv=sv # границя міцності
        self.sb=sv*(1+k*self.delta) # істинна границя міцності
        self.eb=log(1+k*self.delta) # істинна деформація, яка відповідає
границі міцності
        # істинне напруження і деформація в момент руйнування
        self.sk=0.8*self.sv/(1-self.psi) # 0.8-коефіцієнт руйнуючого
навантаження
        #self.sk=self.sv*(1+1.35*self.psi)
        self.ek=log(1/(1-self.psi))
    def bilinear(self):
        '''Повертає словник еластичних і пластичних властивостей'''
        return {'el':((self.E,self.mu),),
                'pl':((self.st,0.0), # білінійна залежність
                (self.sb,self.eb))} # або (self.sk,self.ek)
    def e(self,s,n):
        '''Степенева залежність e(s)
        n - степінь
        '''
        return self.et*(s/self.st)**n
    def power(self,k):
        '''Повертає словник еластичних і пластичних властивостей
        k - кількість ліній для апроксимації пластичної ділянки (2,4,8...)
        '''
        #n визначається з умови проходження через точку (eb+et,sb),
n=6...10
        n=log((self.eb+self.et)/self.et)/log(self.sb/self.st)
        ds=self.sb-self.st
        #степенева залежність
        k_=float(k)
        s_e=[(self.st+i/k_*ds,self.e(self.st+i/k_*ds,n)-self.et) for i in
range(0,k+1)]
        #s_e=[(self.st+i*ds,self.e(self.st+i*ds,n)-self.et) for i in
[0.0,0.25,0.5,0.75,1.0]]
        s_e.append((self.sk,self.ek)) # додати точку руйнування
        return {'el':((self.E,self.mu),),
                'pl':tuple(s_e)}

def set_values(part,feature,par):
    '''
    Присвоює значення параметрам
    Приклад:
    par={'h1':0.0002,'h2':0.00004}
    set_values(part='Al',feature='Shell planar-1',par=par)
    '''
    p=model.parts[part] # деталь
    f=p.features[feature] # елемент
    s=model.ConstrainedSketch(name='__edit__', objectToCopy=f.sketch) #
тимчасовий ескіз
```

```
    p.projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s,
upToFeature=f) # спроектувати
    for k,v in par.iteritems(): # для всіх параметрів
        s.parameters[k].setValues(expression=str(v)) # установити значення
    f.setValues(sketch=s) # установити ескіз
    del s # знищити
    p.regenerate() # оновити деталь

def mesh_all():
    '''Будує сітку скінченних елементів'''
    ra=model.rootAssembly # збірка
    # елементи збірки
    reg=(ra.instances['Nipple-1'],ra.instances['Rod-1'],ra.instances['Tool-
1']) #!
    ra.deleteMesh(regions=reg) # знищити сітку
    ra.generateMesh(regions=reg) # створити сітку

def JobSubmit():
    ''' Виконує задачу '''
    myJob = mdb.jobs['Model-1'] # задача
    myJob.submit() # виконати задачу
    # Чекати поки задача не буде розв'язана
    myJob.waitForCompletion()

def readODB_set(set,step,var,pos=NODAL):
    '''Повертає список результатів у вузлах заданої множини вказаного кроку
    set - множина
    step - номер кроку
    var - змінна:
    (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
    (('U', NODAL, ((COMPONENT, 'U2'), )), )
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set(set='Set-1',step='Step-1',var=var)
    '''
    # отримати дані
    if pos==NODAL:
        dat=session.xyDataListFromField(odb=myOdb, outputPosition=NODAL,
variable=var, nodeSets=(set.upper(),)) # дані
    if pos==INTEGRATION_POINT:
        dat=session.xyDataListFromField(odb=myOdb,
outputPosition=INTEGRATION_POINT, variable=var, elementSets=(set.upper(),))
# дані

    step_number=myOdb.steps[step].number # номер кроку

    nframes=[] # містить кількість фреймів в кожному кроці
    for k in myOdb.steps.keys(): # для всіх кроків
        nframes.append(len(myOdb.steps[k].frames))

    nstart_frame=nframes[step_number-2] # номер початкового фрейму кроку
```

```
    nend_frame=int(sum(nframes[step_number-2:step_number])) # номер
кінцевого фрейму кроку
    res=[] # список результатів
    for x in dat: # для всіх вузлів
        #x.data - це ((час,значення),(час,значення)...)
        res.append(x.data[nstart_frame:nend_frame]) # дані тільки з
вказаного кроку

    # видалити тимчасові дані
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set2(set,step,var,pos=NODAL):
    '''Повертає список середніх результатів в вузлах заданої множини
вказаного кроку
    (менш універсальна альтернатива readODB_set())
    set - множина
    step - крок
    var - змінна:
    ('S','Mises')
    ('S','Pressure')
    ('U','Magnitude')
    ('U','U1')
    ('CPRESS','')
    ('D','') # коефіцієнт запасу втомної міцності
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set2(set='Cont',step='Step-1',var=('S','Mises'))
    '''
    if pos==NODAL:
        s=myOdb.rootAssembly.nodeSets[set.upper()] # множина вузлів
    if pos==INTEGRATION_POINT:
        s=myOdb.rootAssembly.elementSets[set.upper()] # множина елементів
    m=[] # список середніх результатів з усіх вузлів множини
    for f in myOdb.steps[step].frames: # для кожного фрейму
        fo=f.fieldOutputs[var[0]].getSubset(region=s,position=pos) # дані
        #openOdb(r'C:/Temp/Model-1.odb').steps['Step-
1'].frames[4].fieldOutputs['CPRESS'].getSubset(position=NODAL,
region=openOdb(r'C:/Temp/Model-
1.odb').rootAssembly.nodeSets['CONT']).values[0].data
        res=[] # список результатів
        for v in fo.values: # для кожного вузла/елемента
            if var[1]=='Mises': res.append(v.mises) # додати до списку
результатів
            if var[1]=='S11': res.append(v.data.tolist()[0])
            if var[1]=='S22': res.append(v.data.tolist()[1])
            if var[1]=='S33': res.append(v.data.tolist()[2])
            if var[1]=='S12': res.append(v.data.tolist()[3])
            if var[1]=='Pressure': res.append(v.press)
            if var[0]=='U' and var[1]=='Magnitude': res.append(v.magnitude)
            if var[1]=='U1': res.append(v.data.tolist()[0])
```

```
            if var[1]=='U2': res.append(v.data.tolist()[1])
            if var[0]=='CPRESS': res.append(v.data)
        m.append((f.frameValue, sum(res)/len(res)))  # додати середнє з
усіх вузлів
    return m # повертае список значень

def findmax(data):
    '''Повертає максимальне значення в форматі (час, значення)'''
    max=(0,0)
    for x in data:
        if x[1]>max[1]:
            max=x
    return max

import csv
csv_file=open("results.csv", "wb") # відкрити csv файл
writer = csv.writer(csv_file,delimiter = ';') # установити розділювач
writer.writerow(['deltax','st','lc','step_time','S22','step_time','cpress']
) # записати рядок
model=mdb.models['Model-1'] # модель

for deltax in [0.1,0.15,0.2,0.25,0.3]: # цикл для зміни глибини переміщення
штампів
    for lc in [100.0,140.0,180.0]: # цикл для зміни довжини обтискання
        for st in [300.0,400.0,500.0]: # цикл для зміни границі плинності
сталі
            # установити значення геометричних параметрів
            set_values(part='Nipple',feature='Shell planar-
1',par={'l1':lc+20,'r1':17})
            set_values(part='Tool',feature='Wire-
1',par={'l1':lc,'l2':lc+10})
            set_values(part='Rod',feature='Shell planar-
1',par={'l1':lc+70})
            model.rootAssembly.regenerate() # оновити
            #створити матеріал

mat_steel=Material(E=210000.0,mu=0.28,st=st,sv=600.0,delta=21.0,psi=56.0)
            # визначити матеріали
            model.materials['Material-
1'].elastic.setValues(table=((mat_steel.E, mat_steel.mu), ))
            model.materials['Material-
1'].plastic.setValues(table=mat_steel.power(8)['pl'])
            model.materials['Material-2'].elastic.setValues(table=((0.1e5,
0.5e5, 0.1e5, 0.22, 0.22, 0.22, 0.04e5, 0.04e5, 0.2e5), ))
            mesh_all() # створити сітку
            model.boundaryConditions['BC-4'].setValues(u1=-deltax)
#гранична умова
            #model.loads['Load-1'].setValues(magnitude=press)
            JobSubmit() # виконати задачу
            myOdb = openOdb(path=model.name + '.odb') # відкрити базу даних
результатів
```

```
            session.viewports['Viewport:
1'].setValues(displayedObject=myOdb)

            # отримати напруження
            x1=readODB_set2(set='Up',step='Step-
2',var=('S','S22'),pos=INTEGRATION_POINT)
            x1max=findmax(x1) # знайти максимальне з усіх фреймів

            # отримати контактний тиск
            x2=readODB_set2(set='Nip',step='Step-1',var=('CPRESS',''))
            x2max=findmax(x2) # знайти максимальне з усіх фреймів

            # отримати вертикальне переміщення
            #x3=readODB_set2(set='Bot',step='Step-2',var=('U','U2'))

            ## отримати напруження
            #var=(('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
            #print readODB_set(set='Up',step='Step-2',var=var)
            ## отримати вертикальне переміщення
            #var=(('U', NODAL, ((COMPONENT, 'U2'), )), )
            #print readODB_set(set='Bot',step='Step-2',var=var)

            # записати дані у файл
            writer.writerow([deltax, st, lc, x1max[0], x1max[1], x2max[0],
x2max[1]])
            myOdb.close() # закрити базу даних результатів
csv_file.close() # закрити файл csv
```

ДОДАТОК Р

Система автоматизованого проектування металополімерних з'єднань на основі вільного програмного забезпечення

Код доступний також в GitHub (vkopey/Fiberglass-Sucker-Rod-Connection: FEM model of fiberglass sucker rod connection. URL: http://github.com/vkopey/Fiberglass-Sucker-Rod-Connection).

Таблиця Р.1 – Структура вхідного файлу CalculiX моделі з'єднання тіла склопластикової штанги зі сталевою головкою

Секція файлу	Пояснення
*NODE, NSET=NALL 1, 0.000000,5.000000 ...	Визначення вузлів сітки у форматі: номер вузла, x, y
*NSET, NSET=Line1 52 ...	Визначення множини вузлів Line1 у форматі: номер вузла
*ELEMENT, type=CAX4, ELSET=Rod 1, 1,2,3,4 ...	Визначення множини елементів Rod типу CAX4 (4-вузловий осесиметричний елемент) у форматі: номер елемента, вузол 1, вузол 2, вузол 3, вузол 4
*SURFACE, NAME = Line2, TYPE = ELEMENT 33, S4 ...	Визначення поверхні Line2 з граней елементів у форматі: номер елемента, грань елемента
*ELSET,ELSET=Elset1 2 ...	Визначення множини елементів для отримання результатів у форматі: номер елемента
*MATERIAL, NAME=mat1 *ELASTIC,TYPE=ENGINEERING CONSTANTS 210000.0, 0.3	Визначення матеріалу з пружними властивостями у форматі: модуль пружності, коефіцієнт Пуассона або, якщо TYPE=ENGINEERING CONSTANTS: значення параметрів ортотропного матеріалу
*SOLID SECTION, ELSET=Rod, MATERIAL=mat1 1.	Визначення матеріалів для множини елементів Rod
*SURFACE INTERACTION, NAME=Int1 *SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=LINEAR 1.E7, 3. *FRICTION 0.2, 47000 *CONTACT PAIR, INTERACTION=Int1, ADJUST=Line4, TYPE=SURFACE TO SURFACE Line4, Line2	Визначення взаємодії контактних поверхонь, характеристик поверхні і тертя, контактних поверхонь. Тут параметр ADJUST дозволяє перемістити вибрані вузли Line4 підпорядкованої поверхні Line4 на початку першого кроку так, щоб вони контактували з головною поверхнею Line2.

Кінець таблиці Р.1

Секція файлу	Пояснення
*BOUNDARY Line1,1,2,0.0	Визначення граничної умови, яка забороняє переміщення по x і y на вузлах Line1
*AMPLITUDE,NAME=A1 0.0,0.0, ...	Визначення залежності A1 величини від часу у форматі: час, значення величини
*STEP,NLGEOM *STATIC 1E-4,1.0	Визначення першого кроку статичної задачі тривалістю 1 с
*NODE FILE U, *EL FILE S,	Визначення результатів першого кроку, які будуть зберігатись у файл результатів jobname.frd - переміщення і напруження
*END STEP	Кінець визначення першого кроку
*STEP,NLGEOM *STATIC,DIRECT 0.1,1.0	Визначення другого кроку статичної задачі тривалістю 1 с. Параметр DIRECT означає, що підкроки будуть тривати 0,1 с.
*BOUNDARY,AMPLITUDE=A1 Line3,2,2,10.0	Визначення граничної умови, яка переміщує по y вузли Line2 на величину y(t)=10*A1(t) відповідно залежності A1
*EL PRINT,ELSET=Elset1 S	Визначення результатів (напруження в елементах Elset1) другого кроку, які будуть зберігатись у файл результатів jobname.dat
*END STEP	Кінець визначення другого кроку

Лістинг Р.1 – fibrod_regular.py – код програми

```
#encoding: utf-8
# скінченно-елементна модель з'єднання з прямокутною регулярною сіткою
from __future__ import division
import os
import numpy as np
import matplotlib.pyplot as plt
from math import pi,sin

class RodConnector:
    """Параметри з'єднання"""
    r0=11.0 # радіус штанги
    r1=18.0 # радіус ніпеля
    l0=150.0 # довжина штанги
    l1=150.0 # довжина ніпеля
    l3=1.875 # зміщення штанги відносно ніпеля вверх
    # рекомендується l3=(l1/n1[1])/2
    n0=3,40 # кількість елементів штанги по x,y
    n1=2,40 # кількість елементів ніпеля по x,y
    y1,y2=10.0, 140.0 # границі вертикальної зони деформування
    args=[0.12, 1.0, 1.0] # параметри деформування для функції dx()
    def dx(self,y):
        """Закон деформування - залежність зміщень (в напрямку до осі)
```

```
        точок внутрішньої поверхні ніпеля від ординати точки y"""
        a,b,c=self.args
        #0.1 # рівномірне
        #0.01*y # клинопідібне
        #0.2+0.01*y # рівномірне+клинопідібне
        #1.-0.0005*(y-70)**2 # параболічне
        #1.0+0.5*sin(2*pi*y/30.0+0.0) # гармонічне
        #1.0+0.01*y*sin(2*pi*y/30.0+0.0) # гармонічне зі змінною амплітудою
        #0.1+(0.2-0.001*y)*sin(2*pi*y/30.0+0.0) # гармонічне зі змінною
амплітудою (обернене)
        return self.args[0]

class Model:
    """Скінченно-елементна модель з'єднання з прямокутною регулярною сіткою
    Увага! Тут нумерація індексів з 0, а у inp-файлі з 1"""
    def __init__(self):
        self.ccx=r"c:\CalculiXLauncher-03.1-beta_Windows-
32bit\bin\ccx\ccx212.exe" # повний шлях до розв'язувача
        self.dirccx=os.path.dirname(self.ccx)
        os.chdir(self.dirccx)
        self.filename='fiber' # назва файлу моделі
        self.reset()

    def reset(self): # очистити сітку
        self.N=[] # вузли
        self.E=[] # елементи
        self.indE={} # індекси елементів деталей

    def indN(self,part):
        """Індекси вузлів деталі part"""
        s=set()
        for i in self.indE[part]:
            s.update(set(self.E[i]))
        return list(s)

    def mesh(self,x,y,lx,ly,nx,ny):
        """Будує регулярну сітку на прямокутній області
        x,y # лівий нижній кут
        lx,ly # ширина, висота
        nx,ny - кількість елементів в ширину і висоту"""
        newE=[] # індекси нових елементів
        dx, dy = lx/nx, ly/ny # ширина, висота елемента
        for j in range(ny):
            for i in range(nx):
                #print i,j
                e=[] # елемент
                for dn in [(0,0),(dx,0),(dx,dy),(0,dy)]: # додати 4 вузли
(проти годинникової стрілки)
                    n=[i*dx+dn[0]+x, j*dy+dn[1]+y] # вузол
                    if n not in self.N: # якщо такого вузла ще немає
                        self.N.append(n) # додати вузол
```

```
                    e.append(self.N.index(n)) # додати номер вузла в
елемент
                self.E.append(e) # додати елемент
                newE.append(len(self.E)-1) # додати індекс елемента
        return newE

    def getVLineNodes(self,part,x,y1,y2):
        """Повертає індекси вузлів вертикальної лінії"""
        L=[] # вузли однієї лінії
        for i in self.indN(part): # для індексів вузлів деталі part
            n=self.N[i] # вузол
            if n[0]==x and n[1]>=y1 and n[1]<=y2: # якщо належить лінії
                L.append([i,n])
        L.sort(key=lambda x: x[1][1]) # сортувати по y
        L=[i[0] for i in L] # тільки індекси
        return L

    def getHLineNodes(self,part,y,x1,x2):
        """Повертає індекси вузлів горизонтальної лінії"""
        L=[] # вузли однієї лінії
        for i in self.indN(part): # для індексів вузлів деталі part
            n=self.N[i] # вузол
            if n[1]==y and n[0]>=x1 and n[0]<=x2: # якщо належить лінії
                L.append([i,n])
        L.sort(key=lambda x: x[1][0]) # сортувати по x
        L=[i[0] for i in L] # тільки індекси
        return L

    def getElements(self,Nodes):
        """Повертає список елементів і їх граней за впорядкованим списком
вузлів лінії"""
        n=0
        E=[] # список елементів
        P={(0,1):'S1',(1,2):'S2',(2,3):'S3',(0,3):'S4'} # грані елемента
        while n<len(Nodes)-1: # кожний вузол крім останнього
            n1,n2=Nodes[n],Nodes[n+1] # пара сусідніх вузлів
            # шукаємо цю пару серед усіх елементів
            for i,e in enumerate(self.E):
                if n1 in e and n2 in e:
                    p=sorted((e.index(n1),e.index(n2))) # вузли грані
                    E.append((i,P[tuple(p)]))
            n+=1
        return E

    def deform(self):
        """Деформування сітки лінії
        Увага! Величина деформування повинна бути меншою розміру елемента
        Увага! Деформувати сітку тільки після визначення потрібних вузлів
функціями getHLineNodes, getVLineNodes"""
        L=self.getVLineNodes(part='nipple',x=rc.r0,y1=0,y2=rc.l1)
        for i in L: # для кожного вузла лінії
```

```
            x,y=self.N[i]
            if y>=rc.y1 and y<=rc.y2: # межі деформування
                self.N[i][0]=x-rc.dx(y) # змістити вузол вліво на величину
dx

    def draw(self):
        """Рисує сітку"""
        N=self.N
        for i,e in enumerate(self.E):# [:4]
            X=[N[e[0]][0],N[e[1]][0],N[e[2]][0],N[e[3]][0],N[e[0]][0]]
            Y=[N[e[0]][1],N[e[1]][1],N[e[2]][1],N[e[3]][1],N[e[0]][1]]
            if i in self.indE['nipple']:
                plt.plot(X, Y,'ko-')
            else:
                plt.plot(X, Y,'ro-')
        plt.axes().set_aspect('equal')
        plt.show()

    def writeINP(self,Lines):
        """Створює inp-файл для CalculiX
        Увага! Нумерація індексів у файлі з 1, тому усі індекси збільшуємо
на 1"""
        Line1,Line2,Line3,Line4,Line5=Lines # списки вузлів потрібних ліній

        s="*NODE, NSET=NALL\n"
        for i,n in enumerate(self.N):
            x,y=n
            s+="%d, %f,%f\n"%(i+1,x,y)

        s+="*NSET, NSET=Line1\n"
        for i in Line1:
            s+="%d\n"%(i+1)

        s+="*NSET, NSET=Line2\n"
        for i in Line2:
            s+="%d\n"%(i+1)

        s+="*NSET, NSET=Line3\n"
        for i in Line3:
            s+="%d\n"%(i+1)

        s+="*NSET, NSET=Line4\n"
        for i in Line4:
            s+="%d\n"%(i+1)

        s+="*NSET, NSET=Line5\n"
        for i in Line5:
            s+="%d\n"%(i+1)

        s+="*ELEMENT, type=CAX4, ELSET=Rod\n" # або CAX4R
        for i,e in enumerate(self.E):
```

```
            if i==self.indE['nipple'][0]:
                s+="*ELEMENT, type=CAX4, ELSET=Nipple\n"
            s+="%d, %d,%d,%d,%d\n"%(i+1,e[0]+1,e[1]+1,e[2]+1,e[3]+1)

        s+="*SURFACE, NAME = Line2, TYPE = ELEMENT\n"
        for i,j in self.getElements(Line2):
            s+="%d, %s\n"%(i+1, j)

        s+="*SURFACE, NAME = Line4, TYPE = ELEMENT\n"
        for i,j in self.getElements(Line4):
            s+="%d, %s\n"%(i+1, j)

        Elset1=set() # елементи, де потрібні будуть результати
        for i,j in self.getElements(Line3)+self.getElements(Line4):
            Elset1.add(i)
        s+="*ELSET,ELSET=Elset1\n"
        for i in Elset1:
            s+="%d\n"%(i+1)

        s+="""
*MATERIAL, NAME=mat1
*ELASTIC,TYPE=ENGINEERING CONSTANTS
10000.0,50000.0,10000.0,0.22,0.22,0.22,4000.0,4000.0
20000.0,295.0
*MATERIAL, NAME=mat2
*ELASTIC
210000.0, 0.3
*SOLID SECTION, ELSET=Rod, MATERIAL=mat1
1.
*SOLID SECTION, ELSET=Nipple, MATERIAL=mat2
1.
*SURFACE INTERACTION, NAME=Int1
*SURFACE BEHAVIOR, PRESSURE-OVERCLOSURE=LINEAR
1.E7, 3.
*FRICTION
0.2, 47000
*CONTACT PAIR, INTERACTION=Int1, ADJUST=Line4, TYPE=SURFACE TO SURFACE
Line4, Line2
*BOUNDARY
Line5,1,1,0.0
*BOUNDARY
Line1,1,2,0.0
*AMPLITUDE,NAME=A1
0.0,0.0,0.1,0.1,0.2,0.2,0.3,0.3,
0.4,0.4,0.5,0.5,0.6,0.6,0.7,0.7,
0.8,0.8,0.9,0.9,1.0,1.0
*STEP,NLGEOM
*STATIC
1E-4,1.0
*NODE FILE
U,
```

```
*EL FILE
S,
*END STEP

*STEP,NLGEOM
*STATIC,DIRECT
0.01,1.0
*BOUNDARY
**BOUNDARY,AMPLITUDE=A1
Line3,2,2,5.0
*NODE FILE
U,
*EL FILE
S,
** *NODE PRINT,NSET=Line3
** RF
*EL PRINT,ELSET=Elset1
S
*END STEP
"""

        f=open(self.dirccx+"\\"+self.filename+".inp", 'w')
        f.write(s)
        f.close()

    def runCCX(self):
        """Розраховує задачу в CalculiX"""
        import subprocess
        s=subprocess.check_output([self.ccx, "-i", self.dirccx+ "\\"
+self.filename], shell=True)
        L=[ln.strip() for ln in s.splitlines()[-10:]] # останні рядки
виведення
        if "Job finished" in L:
            return 1 # якщо розрахунок успішний
        return 0 # якщо розрахунок не успішний

    def readResult1(self, Nodes):
        """Читає результати для елементів штанги в місці контакту"""
        elset=[str(i+1) for i,j in self.getElements(Nodes)]
        f=open(self.dirccx+"\\"+self.filename+".dat",'r')
        L=f.readlines()
        f.close()
        Y=[] # значення
        inc1=True
        for s in L: # для кожного рядка
            w=s.strip().split(' ') # слова
            w=[i for i in w if i.strip()!=''] # без пустих слів
            if w==[]: continue # пропустити пусті рядки
            if w[0]=='stresses':
                if not inc1: break # тільки для першого інкременту
                inc1=False
```

```
            if w[0] in elset: # якщо елемент у списку
                if w[1]=='3': # integration point (1 для CAX4R)
                    Y.append(float(w[2])) # !!! напруження sxx
        return min(Y) # бо зі знаком -

    def readResult2(self, Nodes):
        """Читає результати для елементів штанги в місці верхнього торця"""
        elset=[str(i+1) for i,j in self.getElements(Nodes)]
        f=open(self.dirccx+"\\"+self.filename+".dat", 'r')
        L=f.readlines()
        f.close()
        T=[] # час
        Y=[] # значення
        n=len(elset) # кількість елементів
        for s in L: # для кожного рядка
            w=s.strip().split(' ') # слова
            w=[i for i in w if i.strip()!=''] # без пустих слів
            if w==[]: continue # пропустити пусті рядки
            #print w
            if w[0]=='stresses': #'forces'
                T.append(float(w[-1]))
            if w[0]==elset[0]: # якщо перший елемент списку
                if w[1]=='3': # integration point (1 для CAX4R)
                    Y.append(float(w[3])/n)
            if w[0] in elset[1:]: # якщо не перший елемент списку
                if w[1]=='3': # integration point
                    Y[-1]=Y[-1]+float(w[3])/n # середнє по елементам
        plt.plot(T,Y,'ko-')
        plt.show()

        # D=[(b-a)/(T[1]-T[0]) for a,b in zip(Y[:-1],Y[1:])] # похідна
dy/dt
        # print D
        # dmin=200 # мінімальне допустиме (малий наклон дотичної на рис.
означає руйнування з'єднання)
        # for i,d in enumerate(D): # для усіх значень dy/dt
        #     if d<dmin: # якщо менше допустимого
        #         return T[i],Y[i] # то це точка руйнування
        Ymax=max(Y)
        t=T[Y.index(Ymax)]
        if t==T[-1]: print "Увага! Можливо знайдено максимальне напруження.
Збільшіть величину осьової деформації"
        return t, Ymax

    def run(self, runCCX=True):
        """Створює модель і виконує задачу"""
        self.reset() # очистити
        # створити сітку
        self.indE['rod']=self.mesh(x=0.0, y=rc.l3, lx=rc.r0, ly=rc.l0,
nx=rc.n0[0], ny=rc.n0[1])
        print len(self.N),len(self.E)
```

```
        self.indE['nipple']=self.mesh(x=rc.r0, y=0.0, lx=rc.r1-rc.r0,
ly=rc.l1, nx=rc.n1[0], ny=rc.n1[1])

        Line1=self.getHLineNodes('nipple', y=0, x1=0, x2=rc.r1) # низ
        Line2=self.getVLineNodes('nipple', x=rc.r0, y1=0, y2=rc.l1) #
контакт
        Line3=self.getHLineNodes('rod', y=rc.l0+rc.l3, x1=0, x2=rc.r0) #
верх
        Line4=self.getVLineNodes('rod', x=rc.r0, y1=rc.l3, y2=rc.l0+rc.l3)
# контакт
        Line5=self.getVLineNodes('rod', x=0, y1=rc.l3, y2=rc.l0+rc.l3) #
вісь
        self.deform() # !!! деформувати сітку тільки після визначення
потрібних вузлів функціями getHLineNodes, getVLineNodes
        self.writeINP([Line1,Line2,Line3,Line4,Line5])
        if runCCX==False:
            self.draw() # тільки нарисувати сітку і вийти
            return
        if self.runCCX(): # розрахувати і повернути результати
            sxx=self.readResult1(Nodes=Line4)
            t,syy=self.readResult2(Nodes=Line3)
            print sxx,t,syy
            return sxx,t,syy

    def optimize(self):
        """Виконує послідовність задач для оптимізації"""
        open('result.csv', 'w').close() # створити файл результатів
        for x in [0.10,0.11,0.12,0.13,0.14,0.15]: # значення параметрів
моделі
            rc.args[0]=x # змінити значення параметрів моделі
            sxx,t,syy=self.run() # розрахувати і отримати результати
            with open('result.csv', 'a') as f: # додати результати у файл
                f.write("%f;%f;%f;%f\n"%(x,sxx,t,syy))

##
rc=RodConnector()
mr=Model()
#mr.run(False) # тільки створити модель і показати сітку
mr.run() # виконати задачу
#mr.optimize() # виконати послідовність задач
```

ДОДАТОК С

Програма для обчислення КІН за результатами моделювання МСЕ

Код доступний також в GitHub (vkopey/SIF: Calculation of stress intensity factor (SIF) and fatigue life by FEM results. URL: http://github.com/vkopey/SIF).

Лістинг С.1 – Код програми

```
# -*- coding: utf-8 -*-
"""Розрахунок КІН за результатами моделювання МСЕ"""
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

E=2.1e11
G=7.9e10 # модуль зсуву, Па
#G=E/(2+2*nu)
nu=0.28 # коефіцієнт Пуассона
r=0.2/1000 # віддаль від вістря тріщини до місця заміру, м
sigma=56.0e6 # кільцеві напруження, Па
d=5.5/1000 # товщина стінки труби, м

def K(V):
    """Коефіцієнт інтенсивності напружень (КІН), Па*м**0.5
    V - половина величини розкриття тріщини, м"""
    k=3-4*nu # для плоского деформування
    #return np.sqrt(2*np.pi)*2*G*V/((1+k)*np.sqrt(r))
    return np.sqrt(2*np.pi/r)*V*E/(4-4*nu**2) # [Meinhard Kuna]

def K_(s):
    """Коефіцієнт інтенсивності напружень (КІН), Па*м**0.5
    s - напруження біля тріщини, Па"""
    return s*np.sqrt(2*np.pi*r)

def Y(K,a):
    """Поправочна функція"""
    return K/(sigma*np.sqrt(np.pi*a))

def Kt(Y,sigma,a):
    """теоретичне значення КІН, Па*м**0.5
    Y - значення поправочної функції
    sigma - напруження, Па
    a - глибина тріщини, м"""
    return Y*sigma*np.sqrt(np.pi*a)

def Kt1(sigma,b):
```

```
    """Теоретичний КІН [Муракамі, т.2, с.556, табл.9.33]
    R - внутрішній радіус труби
    t - товщина стінки труби
    b - глибина тріщини
    a - половина довжини тріщини
    p - внутрішній тиск
    sigma - кільцеві напруження в трубі
    """
    R=62.0/2000
    t=5.5/1000
    Rt=50.0/1000 # радіус кругового фронту тріщини
    a=(Rt**2-(Rt-b)**2)**0.5
    p=sigma*t/R
    Q=1+1.464*(b/a)**1.65
    R0=R+t
    G0=np.interp(b/t, [0.2, 0.5, 0.8], [1.147, 1.584, 2.298])
    G1=np.interp(b/t, [0.2, 0.5, 0.8], [0.685, 0.839, 1.099])
    G2=np.interp(b/t, [0.2, 0.5, 0.8], [0.521, 0.600, 0.739])
    G3=np.interp(b/t, [0.2, 0.5, 0.8], [0.432, 0.480, 0.568])
    Fe=(t/R)*(R*R/(R0*R0-
R*R))*(2*G0+2*G1*b/R0+3*G2*(b/R0)**2+4*G3*(b/R0)**3)
    return Fe*(p*R/t)*(np.pi*b/Q)**0.5

def v(K):
    """Швидкість росту тріщини, м/цикл
    K - КІН, Па*м**0.5"""
    K=K/1.0e6 # перевести МПа*м**0.5
    # параметри діаграми втомного руйнування сталі 20Н2М 3% NaCl [ Копей Б.
Штанги... моногр. рис. 4.5]:
    C=9.14e-13 # м/(МПа*м**0.5)
    n=3.6
    #C=6.0e-17 # повітря
    #n=6.04
    return C*K**n

a=np.array([0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5])/1000 # глибина тріщини, м

def N(sigma,f,popt):
    """Циклічна довговічність, цикли
    sigma - напруження, Па"""
    a_=np.linspace(a.min(), a.max()) # розміри тріщини, м
    return np.trapz(1/v( Kt(f(a_/d,*popt),sigma,a_) ), a_) # інтеграл по a

def N_(sigma,f,popt): # інший спосіб інтегрування для перевірки
    A=a.min()
    dN=1000.0
    #dA=0.01/1000 # або
    N=0.0
    while A<=a.max():
        K=Kt(f(A/d,*popt),sigma,A)
        V=v(K)
```

```
        dA=V*dN
        #dN=dA/V # або
        A=A+dA
        N=N+dN
    return N

def N1(sigma,material):
    """Кількість циклів з рівняння втоми [Трощенко т.1, с.51]"""
    D={"сталь"      :dict(s=210.0e6, N=3.0e6, m=9.00), # середня сталь
       "сталь45"    :dict(s=253.0e6, N=1.8e6, m=8.72), # гладкий зразок,
сталь 45, відпал, на повітрі, R=-1 [Трощенко т.1, с.456]
       "сталь45NaCl":dict(s=100.0e6, N=2.5e7, m=2.76), # гладкий зразок,
сталь 45, відпал, 3% NaCl, R=-1 [Трощенко т.1, с.456]
       "20H2M"      :dict(s=190.0e6, N=2.5e6, m=5.86), # моно. Рисунок 4.5
       "20H2MNaCl"  :dict(s=60.00e6, N=2.0e7, m=2.3), # моно. Рисунок 4.5
       "20H2MNaCl_" :dict(s=125.0e6, N=1.0e6, m=1.86), # моно. Рисунок 3.15
       "20H2M_"     :dict(s=370.0e6, N=1.0e6, m=6.09), # Трощенко с.672
       "20H2MNaCl__":dict(s=50.0e6, N=5.0e6, m=3.32, m1=10.3) # Трощенко
с.672
       }
    d=D[material]
    s=d['s'] # ордината точки перелому кривої втоми в лог. коорд.
    N=d['N'] # абсциса точки перелому кривої втоми в лог. коорд.
    m=d['m'] # показник нахилу кривої втоми в лог. коорд.
    psi=0.2 # коефіцієнт чутливості матеріалу до асиметрії циклу
    s=2*s/(1+psi) # перевід з R=-1 в R=0 (максимальне, а не амплітуда)
    if sigma<s:
        if d.has_key('m1'): return (N*s**d['m1'])/sigma**d['m1'] # якщо
крива має 2 переломи
        #else: return 1e8 # включити тільки для візуалізації !
    return (N*s**m)/sigma**m

def S1(n):
    """Напруження з рівняння втоми (обернене до N1 - перевірено) n -
кількість циклів"""
    s=253.0e6
    N=1.8e6
    m=8.72
    sigma=((N*s**m)/n)**(1/m) # ампл. напруження для R=-1, що відповідає n
    psi=0.2 # коефіцієнт чутливості матеріалу до асиметрії циклу
    sigma=sigma/(1+psi) # ампл. напруження для R=0, що відповідає n (з
залежності sa=sa_eq-psi*sm)
    return 2*sigma # макс. напруження для R=0, що відповідає n

e=a/d # відносна глибина тріщини
# половина величини розкриття тріщини, мкм:
# без бандажу
V1=[0.19311,0.30551,0.40903,0.52202,0.65405,0.79376,0.94576,1.09433,1.24091
]
# з бандажем 84 мм
```

```
V2=[0.08682, 0.17087, 0.24629, 0.30835, 0.36911, 0.42522, 0.47333, 0.52429,
0.58763]
# з бандажем 74 мм
#V2=[0.09031, 0.17794, 0.25781, 0.32571, 0.39187, 0.4531, 0.50657, 0.55936,
0.62274]
# з бандажем 84 мм no panetration
#V2=[0.18499, 0.29258, 0.39114, 0.49768, 0.62166, 0.75185, 0.89302,
1.03042, 1.16669]
V1=np.array(V1)/1000000 # в метрах
V2=np.array(V2)/1000000

##
Y1=Y(K(V1),a)
def f1(x,a,b,c): # модель
    return a*x**2+b*x+c
popt1, pcov1 = curve_fit(f1, e, Y1) # апроксимувати нелінійним методом
найменших квадратів
print popt1 # коефіцієнти a,b,c
print "R^2=", np.corrcoef(Y1, f1(e,*popt1))[0, 1]**2 # коефіцієнт
детермінації
print "N(%fПа)="%sigma, np.trapz(1/v(K(V1)), a) # інтегрувати задану
масивом функцію методом трапецій

##
Y2=Y(K(V2),a)
def f2(x,a,b,c,d): # модель
    return a*x**3+b*x**2+c*x+d
popt2, pcov2 = curve_fit(f2, e, Y2) # апроксимувати нелінійним методом
найменших квадратів
print popt2 # коефіцієнти
print "R^2=", np.corrcoef(Y2, f2(e,*popt2))[0, 1]**2 # коефіцієнт
детермінації
print "N(%fПа)="%sigma, np.trapz(1/v(K(V2)),a)

##
plt.figure()
plt.plot(K(V1), v(K(V1))*1e6,'ko-') # кінетична діаграма втомного
руйнування
#np.savetxt('KDVRa.csv', v(K(V1))*1e6, delimiter=';')
_=np.loadtxt('KDVRa.csv', delimiter=';')
plt.plot(K(V1), _, 'rs-')
plt.xlabel(u'$K, Пам^{1/2}$'),plt.ylabel(u'$v, мкм/цикл$')
plt.show()

##
plt.figure()
# перевірка - мають збігатись:
plt.plot(a, K(V1), 'ko-')
plt.plot(a, Kt(f1(a/d,*popt1),sigma,a), 'r^-')
plt.plot(a, Kt1(sigma,a), 'rs-')
plt.xlabel('$a$'),plt.ylabel(u'$K$')
```

```
plt.show()

##
plt.figure()
S=np.linspace(80.0e6, 500.0e6)
plt.plot([np.log10(N(s,f1,popt1)) for s in S], S, 'k-')
#plt.plot([N(s,f1,popt1) for s in S], S, 'k-')

plt.plot([np.log10(N(s,f2,popt2)) for s in S], S, 'k--')
#plt.plot([N(s,f2,popt2) for s in S], S, 'k--')

# зберегти масиви
#np.savetxt('_.csv', [np.log10(N(s,f2,popt2)) for s in S], delimiter=';')
#нарисувати збережені np.savetxt графіки
_=np.loadtxt('74.csv', delimiter=';')
plt.plot(_, S, 'k:')
_=np.loadtxt('84np.csv', delimiter=';')
plt.plot(_, S, 'r-.')
_=np.loadtxt('73a.csv', delimiter=';')
plt.plot(_, S, 'r-')
_=np.loadtxt('84a.csv', delimiter=';')
plt.plot(_, S, 'r--')

# графіки втоми за рівнянням втоми
import scipy
N1=scipy.vectorize(N1)
S=np.linspace(80.0e6, 500.0e6)
for k,_ in enumerate(["20H2M","20H2MNaCl"]):
    plt.plot(np.log10(N1(S,_)), S, 'k-.', linewidth=k+1)

plt.xlabel('$log(N)$'),plt.ylabel(u'$Па$')
plt.grid()
plt.show()

##
plt.figure()
plt.plot(e, Y1, 'ko') # нарисувати емпіричну залежність
plt.plot(e, f1(e,*popt1), 'k-') # нарисувати апроксимовану залежність
plt.plot(e, Y2, 'k^')
plt.plot(e, f2(e,*popt2), 'k-')
plt.xlabel('$\epsilon$'),plt.ylabel('$Y$')
plt.show()
```

ДОДАТОК Т

Макрос Abaqus/CAE для оптимізації конструкції ШН за критерієм втомної міцності з fe-safe

Код доступний також в GitHub (vkopey/Abaqus-feSafe-optimization: Abaqus CAE macro for design optimization with fe-safe. URL: http://github.com/vkopey/Abaqus-feSafe-optimization).

Лістинг Т.1 – rod3D.py

```
# -*- coding: cp1251 -*-
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
import subprocess

def set_values(part,feature,par):
    '''
    Присвоює значення параметрам
    Приклад:
    par={'h1':0.0002,'h2':0.00004}
    set_values(part='Al',feature='Shell planar-1',par=par)
    '''
    p=model.parts[part] # деталь
    f=p.features[feature] # елемент
    s=model.ConstrainedSketch(name='__edit__', objectToCopy=f.sketch) #
тимчасовий ескіз
    p.projectReferencesOntoSketch(filter=COPLANAR_EDGES, sketch=s,
upToFeature=f) # спроектувати
    for k,v in par.iteritems(): # для всіх параметрів
        s.parameters[k].setValues(expression=str(v)) # установити значення
    f.setValues(sketch=s) # установити ескіз
    del s # знищити
    p.regenerate() # оновити деталь

def mesh_all(lst):
```

```
    '''Будує сітку скінченних елементів
    lst - список назв елементів збірки'''
    ra=model.rootAssembly #збірка
    # елементи збірки
    reg=[ra.instances[i] for i in lst]
    #reg=(ra.instances['Part-1-1'],ra.instances['Part-2-2'])
    ra.deleteMesh(regions=reg) # знищити сітку
    ra.generateMesh(regions=reg) # створити сітку

def JobSubmit(job):
    '''Виконує задачу'''
    myJob = mdb.jobs[job] # задача
    myJob.submit() # виконати задачу
    # Чекати поки задача не буде розв'язана
    myJob.waitForCompletion()

def readODB_set(set,step,var,pos=NODAL):
    '''Повертає список результатів у вузлах заданої множини вказаного кроку
    set - множина
    step - номер кроку
    var - змінна:
    (('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
    (('U', NODAL, ((COMPONENT, 'U2'), )), )
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set(set='Set-1',step='Step-1',var=var)
    '''
    #отримати дані
    if pos==NODAL:
        dat=session.xyDataListFromField(odb=myOdb, outputPosition=NODAL,
variable=var, nodeSets=(set.upper(),)) # дані
    if pos==INTEGRATION_POINT:
        dat=session.xyDataListFromField(odb=myOdb,
outputPosition=INTEGRATION_POINT, variable=var, elementSets=(set.upper(),))
# дані

    step_number=myOdb.steps[step].number # номер кроку

    nframes=[] # містить кількість фреймів в кожному кроці
    for k in myOdb.steps.keys(): # для всіх кроків
        nframes.append(len(myOdb.steps[k].frames))

    nstart_frame=nframes[step_number-2] # номер початкового фрейму кроку
    nend_frame=int(sum(nframes[step_number-2:step_number])) # номер
кінцевого фрейму кроку
    res=[] # список результатів
    for x in dat: # для всіх вузлів
        #x.data - це ((час,значення),(час,значення)...)
        res.append(x.data[nstart_frame:nend_frame]) # дані тільки з
вказаного кроку

    # видалити тимчасові дані
```

```
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set_(set,var):
    '''Повертає список результатів в вузлах заданої множини
    (для змінних fe-safe)
    set - множина
    var - змінна:
    (('LOGLife-Repeats', ELEMENT_NODAL), )
    (('FOS@Life=Infinite', ELEMENT_NODAL), )
    (('%%Failure@Life=5E6-Repeats', ELEMENT_NODAL), )
    Приклад: readODB_set_(set='Set-1',var=var)
    '''
    # отримати дані
    dat=session.xyDataListFromField(odb=myOdb, outputPosition=NODAL,
variable=var, nodeSets=(set.upper(),)) # дані

    res=[] # список результатів
    for x in dat: # для всіх вузлів
        #x.data - це ((час,значення),(час,значення)...)
        res.append(x.data) # дані

    # видалити тимчасові дані
    for k in session.xyDataObjects.keys():
        del session.xyDataObjects[k]
    return res # повертає список значень

def readODB_set2(set,step,var,pos=NODAL):
    '''Повертає список середніх результатів в вузлах заданої множини
вказаного кроку
    (менш універсальна альтернатива readODB_set())
    set - множина
    step - крок
    var - змінна:
    ('S','Mises')
    ('S','Pressure')
    ('U','Magnitude')
    ('U','U1')
    ('CPRESS','')
    ('D','') # коефіцієнт запасу втомної міцності
    ('LOGLife-Repeats', '')
    ('FOS@Life=Infinite', '')
    ('%%Failure@Life=5E6-Repeats', '')
    pos - позиція: NODAL - для вузлів,INTEGRATION_POINT - для елементів
    Приклад: readODB_set2(set='Cont',step='Step-1',var=('S','Mises'))
    '''
    if pos==NODAL:
        s=myOdb.rootAssembly.nodeSets[set.upper()] # множина вузлів
    if pos==INTEGRATION_POINT:
        s=myOdb.rootAssembly.elementSets[set.upper()] # множина елементів
```

```
    m=[] # список середніх результатів з усіх вузлів множини
    for f in myOdb.steps[step].frames: # для кожного фрейму
        fo=f.fieldOutputs[var[0]].getSubset(region=s,position=pos) # дані
        #openOdb(r'C:/Temp/Model-1.odb').steps['Step-
1'].frames[4].fieldOutputs['CPRESS'].getSubset(position=NODAL,
region=openOdb(r'C:/Temp/Model-
1.odb').rootAssembly.nodeSets['CONT']).values[0].data
        res=[] # список результатів
        for v in fo.values: # для кожного вузла/елемента
            if var[1]=='Mises': res.append(v.mises) # додати до списку
результатів
            if var[1]=='S11': res.append(v.data.tolist()[0])
            if var[1]=='S22': res.append(v.data.tolist()[1])
            if var[1]=='S33': res.append(v.data.tolist()[2])
            if var[1]=='S12': res.append(v.data.tolist()[3])
            if var[1]=='Pressure': res.append(v.press)
            if var[0]=='U' and var[1]=='Magnitude': res.append(v.magnitude)
            if var[1]=='U1': res.append(v.data.tolist()[0])
            if var[1]=='U2': res.append(v.data.tolist()[1])
            if var[0]=='CPRESS': res.append(v.data)
        m.append((f.frameValue, sum(res)/len(res)))  # додати середнє з
усіх вузлів
    return m # повертае список значень

def findmax(data):
    '''Повертає максимальне значення в форматі (час, значення)'''
    max=(0,0)
    for x in data:
        if x[1]>max[1]:
            max=x
    return max

def runFeSafe(input_odb,input_stlx,output_odb):
    '''Виконує аналіз втомної міцності у fe-safe
    input_odb - назва вхідного файлу результатів Abaqus (без розширення
.odb)
    input_stlx - назва файлу моделі fe-safe (без розширення .stlx)
    output_odb - назва вихідного файлу результатів Abaqus (без розширення
.odb)
    '''
    s=r'd:\Program Files\Safe_Technology\fe-safe\version.6.2\exe\fe-
safe_cl.exe -s j=c:\1\{iodb}.odb b=c:\1\{istlx}.stlx o=c:\1\{oodb}.odb'
    s=s.format(iodb=input_odb, istlx=input_stlx, oodb=output_odb)
    # виконує обчислення в fe-safe та чекає завершення
    subprocess.Popen(s).communicate()

def writeLDFfile(filename,lst):
    '''Змінює вміст файлу визначення навантаження LDF fe-safe
    filename - ім'я файлу
    lst - список рядків-даних
    '''
```

```
    f=open(filename, "w")
    s="""
# .ldf file created by fe-safe compliant product 6.2-01[mswin]

INIT
transitions=Yes
END

# Block number 1
BLOCK n=1, scale=1
lh=0 1 , ds=1, scale=1
lh=0 {x} , ds=2, scale=1
END

""" # шаблон файлу
    s=s.format(x=lst[0]) # вміст файлу з шаблону
    f.write(s)
    f.close()


import csv
csv_file=open("results.csv", "wb") # відкрити csv файл
writer = csv.writer(csv_file,delimiter = ';')
writer.writerow(['r','load','LogLife','FOS','%Failure']) # записати рядок
model=mdb.models['Model-3'] # модель

for rad in [0.067,0.25,0.5,0.75,1.0]: # цикл для зміни значення параметру r
    # установити значення геометричного параметра r
    set_values(part='Part-1',feature='Solid revolve-1',par={'r':rad})
    #set_values(part='Part-1',feature='Shell planar-1',par={'r':rad})
    model.rootAssembly.regenerate() # оновити модель
    mesh_all(['Part-1-1']) # створити сітку скінченних елементів
    #model.loads['Load-2'].setValues(magnitude=load)
    JobSubmit('Job-3') # виконати задачу

    for load in [0.1,0.15,0.2,0.25,0.3]: # цикл для зміни значення
навантаження згину
        writeLDFfile('model3.ldf',[str(load)]) # змінити LDF файл fe-safe
        oodb='results'+str(int(rad*1000))+'_'+str(int(load*100)) # назва
бази даних результатів
        runFeSafe('Job-3','model3',oodb) # виконати fe-safe

        #myOdb = openOdb(path=model.name + '.odb') # відкрити базу даних
результатів
        myOdb = openOdb(path=oodb + '.odb') # відкрити базу даних
результатів
        session.viewports['Viewport: 1'].setValues(displayedObject=myOdb)

        # отримати логарифм довговічності у множині вузлів Set-1
        var=(('LOGLife-Repeats', ELEMENT_NODAL), )
        x1=readODB_set_(set='Set-1',var=var)
```

```
        x1min = min([x[0][1] for x in x1]) # знайти мінімальне значення

        # отримати коефіцієнт запасу у множині вузлів Set-1
        var=(('FOS@Life=Infinite', ELEMENT_NODAL), )
        x2=readODB_set_(set='Set-1',var=var)
        # x2=[((3.0, 1.62),), ((3.0, 2.0),), ...,((час,значення),)]
        x2min = min([x[0][1] for x in x2]) # знайти мінімальне значення

        # отримати відсоток відмов у множині вузлів Set-1
        var=(('%%Failure@Life=5E6-Repeats', ELEMENT_NODAL), )
        x3=readODB_set_(set='Set-1',var=var)
        x3max = max([x[0][1] for x in x3]) # знайти максимальне значення

        '''
        # отримати напруження
        x1=readODB_set2(set='Up',step='Step-
2',var=('S','S22'),pos=INTEGRATION_POINT)
        x1max=findmax(x1) # знайти максимальне з усіх фреймів

        # отримати контактний тиск
        x2=readODB_set2(set='Nip',step='Step-1',var=('CPRESS',''))
        x2max=findmax(x2) # знайти максимальне з усіх фреймів

        # отримати вертикальне переміщення
        #x3=readODB_set2(set='Bot',step='Step-2',var=('U','U2'))

        ## отримати напруження
        #var=(('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), )
        #print readODB_set(set='Up',step='Step-2',var=var)
        ## отримати вертикальне переміщення
        #var=(('U', NODAL, ((COMPONENT, 'U2'), )), )
        #print readODB_set(set='Bot',step='Step-2',var=var)
        '''
        # записати дані у файл
        writer.writerow([rad,load,x1min,x2min,x3max])
        myOdb.close() # закрити базу даних результатів

csv_file.close() # закрити файл csv
```

ДОДАТОК У

Засоби оптимізації конструкції протектора

Таблиця У.1 – План експерименту і результати симуляції

№	b	r1	r2	L/2	a	y2	y1	y2n	y1n
1	0,02514	0,02159	0,02159	0,036591	0,37664	4204,253	2,443136	-0,746227364	0,537861
2	0,02514	0,02159	0,02159	0,036591	0,67002	3618,773	1,362398	-1,418270929	-1,07477
3	0,02514	0,02159	0,02159	0,053409	0,37664	5811,499	2,397281	1,098653089	0,469438
4	0,02514	0,02159	0,02159	0,053409	0,67002	5003,99	1,311918	0,171751743	-1,1501
5	0,02514	0,02159	0,03841	0,036591	0,37664	4110,578	2,452658	-0,853752246	0,552068
6	0,02514	0,02159	0,03841	0,036591	0,67002	3537,188	1,374097	-1,511918895	-1,05732
7	0,02514	0,02159	0,03841	0,053409	0,37664	5717,824	2,41894	0,991127185	0,501756
8	0,02514	0,02159	0,03841	0,053409	0,67002	4922,403	1,32	0,078102744	-1,13804
9	0,02514	0,03841	0,02159	0,036591	0,37664	4205,495	2,920309	-0,74480156	1,249878
10	0,02514	0,03841	0,02159	0,036591	0,67002	3619,83	1,64	-1,417057616	-0,66055
11	0,02514	0,03841	0,02159	0,053409	0,37664	5812,742	2,734789	1,100078846	0,973053
12	0,02514	0,03841	0,02159	0,053409	0,67002	5005,047	1,51	0,172965011	-0,85453
13	0,02514	0,03841	0,03841	0,036591	0,37664	4111,82	2,9415	-0,852326442	1,281498
14	0,02514	0,03841	0,03841	0,036591	0,67002	3538,245	1,652804	-1,510705582	-0,64144
15	0,02514	0,03841	0,03841	0,053409	0,37664	5719,066	2,754014	0,992552931	1,00174
16	0,02514	0,03841	0,03841	0,053409	0,67002	4923,46	1,519298	0,079316023	-0,84065
17	0,03986	0,02159	0,02159	0,036591	0,37664	4689,444	2,46011	-0,189299732	0,563189
18	0,03986	0,02159	0,02159	0,036591	0,67002	4034,958	1,343304	-0,940552945	-1,10326
19	0,03986	0,02159	0,02159	0,053409	0,37664	6296,69	2,398508	1,655579618	0,471267
20	0,03986	0,02159	0,02159	0,053409	0,67002	5420,173	1,298565	0,649468637	-1,17002
21	0,03986	0,02159	0,03841	0,036591	0,37664	4651,76	2,461645	-0,232555768	0,565478
22	0,03986	0,02159	0,03841	0,036591	0,67002	4002,121	1,356735	-0,978244266	-1,08322
23	0,03986	0,02159	0,03841	0,053409	0,37664	6259,006	2,390605	1,612323617	0,459475
24	0,03986	0,02159	0,03841	0,053409	0,67002	5387,337	1,314174	0,611777672	-1,14673
25	0,03986	0,03841	0,02159	0,036591	0,37664	4689,444	3,070677	-0,189299744	1,474251
26	0,03986	0,03841	0,02159	0,036591	0,67002	4034,958	1,647086	-0,940552945	-0,64997
27	0,03986	0,03841	0,02159	0,053409	0,37664	6296,69	2,922793	1,655579618	1,253585
28	0,03986	0,03841	0,02159	0,053409	0,67002	5420,174	1,580314	0,649469108	-0,74961
29	0,03986	0,03841	0,03841	0,036591	0,37664	4651,76	3,06592	-0,232555768	1,467153
30	0,03986	0,03841	0,03841	0,036591	0,67002	4002,121	1,665305	-0,978244266	-0,62279
31	0,03986	0,03841	0,03841	0,053409	0,37664	6259,006	2,920903	1,612323617	1,250764
32	0,03986	0,03841	0,03841	0,053409	0,67002	5387,337	1,597182	0,611778142	-0,72444
33	0,015	0,03	0,03	0,045	0,523333	4217,64	1,795891	-0,730861214	-0,42793
34	0,05	0,03	0,03	0,045	0,523333	5396,368	1,949208	0,622143574	-0,19916
35	0,0325	0,01	0,03	0,045	0,523333	4865,889	2,469018	0,013233251	0,57648
36	0,0325	0,05	0,03	0,045	0,523333	4865,905	2,506527	0,013250767	0,632449
37	0,0325	0,03	0,01	0,045	0,523333	4914,347	1,894558	0,068855389	-0,28071
38	0,0325	0,03	0,05	0,045	0,523333	4791,199	1,931681	-0,072500796	-0,22531
39	0,0325	0,03	0,03	0,025	0,523333	3086,55	2,051168	-2,029184368	-0,04702
40	0,0325	0,03	0,03	0,065	0,523333	6645,228	1,801189	2,055650192	-0,42003
41	0,0325	0,03	0,03	0,045	0,174444	5732,07	4,200831	1,007480051	3,160619
42	0,0325	0,03	0,03	0,045	0,872222	3999,708	0,980021	-0,981014215	-1,64534
43	0,0325	0,03	0,03	0,045	0,523333	4865,889	1,904474	0,013232918	-0,26591
44	0,0325	0,03	0,03	0,045	0,523333	4865,889	1,906308	0,013232918	-0,26317

Таблиця У.2 – Значення даних *y*, регресійної моделі *f*(**x**) та відхилення

№	$\omega_1=1$, $\omega_2=-1$			$\omega_1=1$, $\omega_2=0$			$\omega_1=0$, $\omega_2=-1$		
	y	f(x)	f(x)-y	y	f(x)	f(x)-y	y	f(x)	f(x)-y
1	1,284088	1,232064	-0,05202	0,537861	0,51763	-0,02023	0,746227	0,746764	0,000537
2	0,343499	0,279873	-0,06363	-1,07477	-1,08826	-0,01349	1,418271	1,421052	0,002781
3	-0,62922	-0,68258	-0,05336	0,469438	0,432628	-0,03681	-1,09865	-1,09818	0,000477
4	-1,32185	-1,31336	0,008485	-1,1501	-1,17327	-0,02317	-0,17175	-0,17138	0,000369
5	1,40582	1,38297	-0,02285	0,552068	0,51763	-0,03444	0,853752	0,844012	-0,00974
6	0,454603	0,430779	-0,02382	-1,05732	-1,08826	-0,03095	1,511919	1,507112	-0,00481
7	-0,48937	-0,53167	-0,0423	0,501756	0,432628	-0,06913	-0,99113	-0,99672	-0,00559
8	-1,21614	-1,16246	0,053683	-1,13804	-1,17327	-0,03523	-0,0781	-0,08111	-0,00301
9	1,994679	1,944897	-0,04978	1,249878	1,256379	0,006502	0,744802	0,746764	0,001963
10	0,756512	0,68058	-0,07593	-0,66055	-0,69362	-0,03308	1,417058	1,421052	0,003994
11	-0,12703	0,030256	0,157282	0,973053	1,105156	0,132103	-1,10008	-1,09818	0,001903
12	-1,02749	-0,91266	0,114836	-0,85453	-0,84485	0,009679	-0,17297	-0,17138	0,001582
13	2,133824	2,095803	-0,03802	1,281498	1,256379	-0,02512	0,852326	0,844012	-0,00831
14	0,869264	0,831486	-0,03778	-0,64144	-0,69362	-0,05218	1,510706	1,507112	-0,00359
15	0,009187	0,181162	0,171975	1,00174	1,105156	0,103416	-0,99255	-0,99672	-0,00416
16	-0,91997	-0,76175	0,158218	-0,84065	-0,84485	-0,0042	-0,07932	-0,08111	-0,00179
17	0,752488	0,83272	0,080232	0,563189	0,576713	0,013525	0,1893	0,183867	-0,00543
18	-0,16271	-0,11947	0,04324	-1,10326	-1,02918	0,074082	0,940553	0,939701	-0,00085
19	-1,18431	-1,23259	-0,04828	0,471267	0,491711	0,020444	-1,65558	-1,66107	-0,00549
20	-1,81949	-1,86338	-0,04389	-1,17002	-1,11418	0,055837	-0,64947	-0,65273	-0,00327
21	0,798034	0,927898	0,129864	0,565478	0,576713	0,011235	0,232556	0,234957	0,002402
22	-0,10498	-0,02429	0,080685	-1,08322	-1,02918	0,054041	0,978244	0,979604	0,00136
23	-1,15285	-1,13741	0,015436	0,459475	0,491711	0,032236	-1,61232	-1,60577	0,006554
24	-1,75851	-1,7682	-0,00969	-1,14673	-1,11418	0,032547	-0,61178	-0,60862	0,003159
25	1,663551	1,545553	-0,118	1,474251	1,361492	-0,11276	0,1893	0,183867	-0,00543
26	0,290581	0,281236	-0,00934	-0,64997	-0,58851	0,061461	0,940553	0,939701	-0,00085
27	-0,40199	-0,51976	-0,11776	1,253585	1,210268	-0,04332	-1,65558	-1,66107	-0,00549
28	-1,39908	-1,46267	-0,06359	-0,74961	-0,73973	0,009872	-0,64947	-0,65273	-0,00326
29	1,699708	1,640731	-0,05898	1,467153	1,361492	-0,10566	0,232556	0,234957	0,002402
30	0,355458	0,376413	0,020956	-0,62279	-0,58851	0,034275	0,978244	0,979604	0,00136
31	-0,36156	-0,42458	-0,06302	1,250764	1,210268	-0,0405	-1,61232	-1,60577	0,006554
32	-1,33621	-1,36749	-0,03128	-0,72444	-0,73973	-0,0153	-0,61178	-0,60862	0,00316
33	0,302929	0,166522	-0,13641	-0,42793	-0,36358	0,064349	0,730861	0,738782	0,007921
34	-0,8213	-0,85902	-0,03772	-0,19916	-0,16838	0,030781	-0,62214	-0,62335	-0,0012
35	0,563247	0,434041	-0,12921	0,57648	0,859609	0,283129	-0,01323	-0,0127	0,000536
36	0,619198	0,751464	0,132266	0,632449	0,664302	0,031854	-0,01325	-0,0127	0,000554
37	-0,34956	-0,30379	0,045775	-0,28071	-0,26598	0,014725	-0,06886	-0,06817	0,000681
38	-0,15281	-0,02618	0,126627	-0,22531	-0,26598	-0,04067	0,072501	0,071385	-0,00112
39	1,982167	1,92202	-0,06015	-0,04702	-0,02198	0,025037	2,029184	2,034029	0,004844
40	-2,47568	-2,25199	0,223686	-0,42003	-0,35983	0,0602	-2,05565	-2,05062	0,005029
41	2,153139	2,143922	-0,00922	3,160619	3,175576	0,014957	-1,00748	-1,00654	0,000941
42	-0,66433	-0,91896	-0,25464	-1,64534	-1,72875	-0,08341	0,981014	0,981145	0,000131
43	-0,27914	-0,16499	0,114156	-0,26591	-0,26598	-7,2E-05	-0,01323	-0,0127	0,000536
44	-0,27641	-0,16499	0,11142	-0,26317	-0,26598	-0,00281	-0,01323	-0,0127	0,000536

Код програми та SOLIDWORKS-модель доступні в GitHub (vkopey/protector-for-sucker-rod: SOLIDWORKS model of protector for sucker rod and programs for ortimization. URL: http://github.com/vkopey/protector-for-sucker-rod).

Лістинг У.1 – Код VBA-макросу для обчислення середньої площі тертя

```
Sub Makro()

    Const swDocPART = 1
    Const swDocASSEMBLY = 2
    Const swDocDRAWING = 3

    Dim swApp As Object
    Dim Part As Object
    Dim face As Object
    'Dim massProps As Variant
    Dim R1, R2, L, a, b, f As Double
    Dim i As Integer
    R1min = 10: R1max = 50: R1step = 5
    R2min = 10: R2max = 50: R2step = 5
    Lmin = 20: Lmax = 60: Lstep = 10
    amin = 10: amax = 50: astep = 5
    bmin = 15: bmax = 50: bstep = 5
    fmin = 1: fmax = 10: fstep = 2
    MyPath = CurDir
    MyPath = "C:\Protector"

    Set swApp = CreateObject("SldWorks.Application")
    Set Part = swApp.OpenDoc(MyPath + "\ModelVBA.SLDPRT", swDocPART)
    If Part Is Nothing Then
       Exit Sub
    Else
       Set Part = swApp.ActivateDoc("ModelVBA.SLDPRT")
    End If
    i = 3
    For R1 = R1min To R1max Step R1step
    For R2 = R2min To R2max Step R2step
    For L = Lmin To Lmax Step Lstep
    For a = amin To amax Step astep
    For b = bmin To bmax Step bstep
    For f = fmin To fmax Step fstep
    Part.Parameter("D1@Filet1").SystemValue = R1 / 1000
    Part.Parameter("D1@Filet2").SystemValue = R2 / 1000
    Part.Parameter("D1@Extrude2").SystemValue = L / 1000
    Part.Parameter("D1@c_sketch").SystemValue = (a * 3.14) / 180
    Part.Parameter("D3@schemfer").SystemValue = b / 1000
```

```
    Part.Parameter("D1@w_sketch").SystemValue = 0.056 - (f / 1000)
    Part.EditRebuild
    Cells(i, 1).Value = R1: Cells(2, 1).Value = R1
    Cells(i, 2).Value = R2: Cells(2, 2).Value = R2
    Cells(i, 3).Value = L: Cells(2, 3).Value = L
    Cells(i, 4).Value = a: Cells(2, 4).Value = a
    Cells(i, 5).Value = b: Cells(2, 5).Value = b
    Cells(i, 6).Value = f: Cells(2, 6).Value = f

    'massProps = Part.GetMassProperties
    'Cells(i, 7).Value = 2 * massProps(3)

    Set face = Part.GetEntityByName(1, 2)
    Cells(i, 8).Value = face.GetArea
   Set face = Part.GetEntityByName(2, 2)
    Cells(i, 9).Value = 2 * face.GetArea
    i = i + 1
    Next f
    Next b
    Next a
    Next L
    Next R2
    Next R1
End Sub
```

ДОДАТОК Ф
Абстрактна модель ІС

Таблиця Ф.1 – Опис класів елементів ІС на інших рівнях ієрархії

Рівень, біполярні множини ознак	Назва класу, приклади елементів та їх інформаційної продукції
Рівень 1.1 0: {1, 2, 3, 4} 1: {5, 6, 7, 8}	*Етап проектів.* Центр етапу відповідає максимуму *А*, *М*. Підтримка проектування виробу. Дослідно-конструкторська та дослідно-технологічна робота. Системи підтримки прийняття проектних рішень. Системи автоматизованого проектування (системи CAD і CAE). *Етап реалізацій.* Центр етапу відповідає максимуму *О*, *Ре*. Підтримка виробництва і експлуатації. Управління ланцюгом постачань (SCM-системи). Створення супроводжуючої документації (IETM-системи), навчання персоналу.
Рівень 1.2 0: {8, 1, 2, 3} 1: {4, 5, 6, 7}	*Теоретичний етап.* Теоретичні методи. Науково дослідна робота над створенням продукції. *Практичний етап.* Емпіричні методи. Застосування результатів науково-дослідної роботи.
Рівень 1.3 0: {7, 8, 1, 2} 1: {3, 4, 5, 6}	*Етап концепцій.* Центр етапу відповідає максимуму *І, В*. Абстрактні моделі, гіпотези, пошук рішень. *Етап технологічний.* Центр етапу відповідає максимуму *Р, К*. Конкретні рішення. Інтегровані автоматизовані системи управління виробництвом. Управління ресурсами підприємства (ERP-системи).
Рівень 1.4 0: {6, 7, 8, 1} 1: {2, 3, 4, 5}	*Етап експлуатації і вимог.* Експлуатаційні моделі виробу. Інтегровані автоматизовані системи управління експлуатацією. Дослідження експлуатації. Експлуатаційна документація. Управління відносинами з клієнтами (CRM-системи). *Етап конструкторсько-технологічний.* Конструкторсько-технологічні моделі. Конструкторсько-технологічна документація. Технологічна підготовка (системи CAD і CAM). Поставлення продукції на виробництво (підготовка виробництва, освоєння виробництва).
Рівень 2.1 0: {1, 2, 5, 6} 1: {3, 4, 7, 8}	*Класи.* Подання об’єктів абстрактно, як їх класи. {1, 2} - *етап науково-дослідницький* (пошуково-аналітичний, дослідження та обґрунтування розроблення). {5, 6} - *етап серійного виробництва і дослідження збуту.* *Об’єкти.* Подання об’єктів конкретно, як екземпляри класів. {3, 4} - *етап підготовки виробництва* (швидке прототипування, дослідне виробництво, CAM-системи). {7, 8} - *етап збуту і експлуатації*.

Кінець таблиці Ф.1

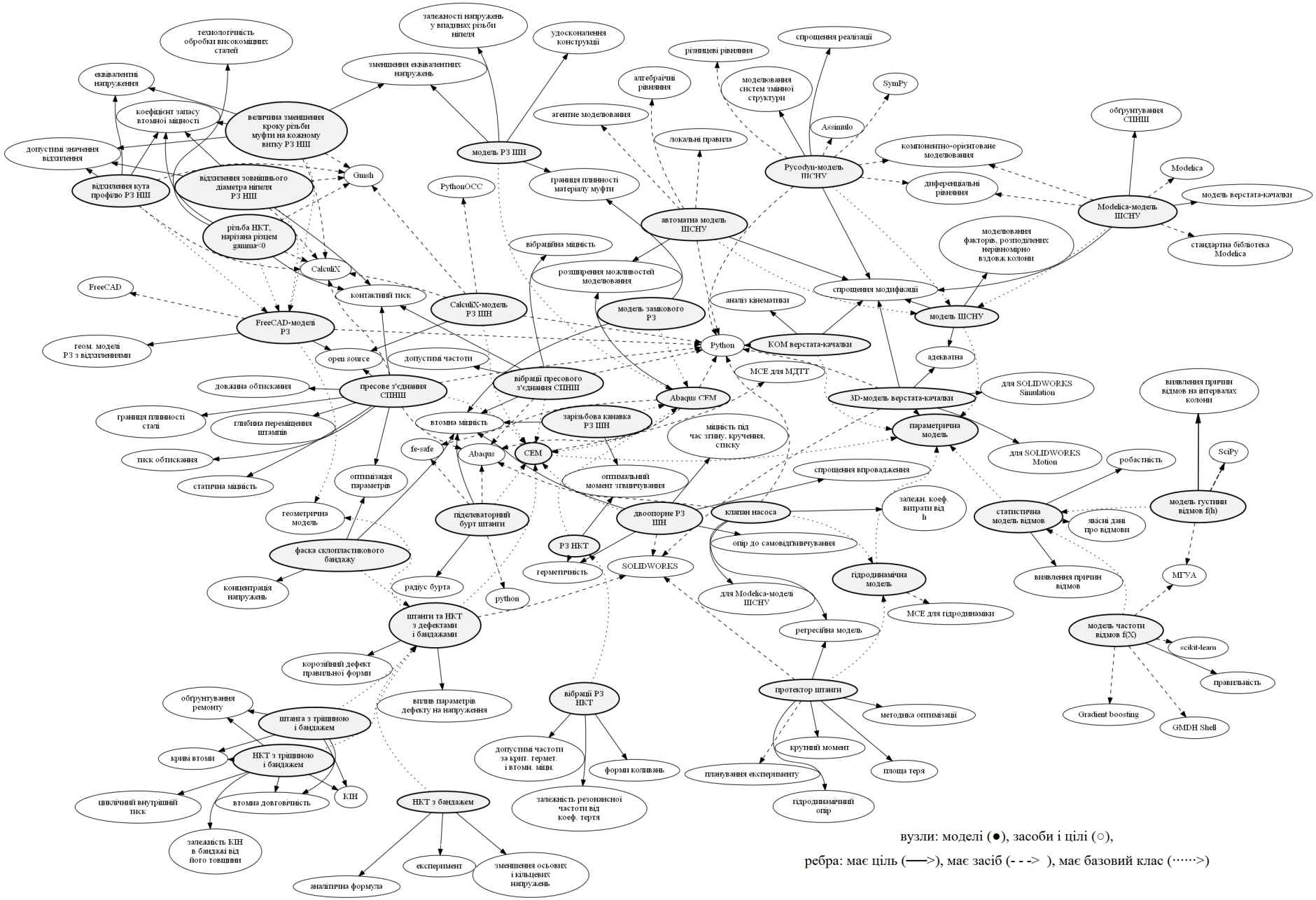
Рівень, біполярні множини ознак	**Назва класу, приклади елементів та їх інформаційної продукції**
Рівень 2.2 0: {8, 1, 4, 5} 1: {2, 3, 6, 7}	*Ірраціональні етапи.* Стохастичні, еврістичні методи, еволюційні алгоритми, нейронні мережі, когнітивне моделювання, МАІС. {8, 1} - *етап генерації ідей і вимог* (виявлення проблеми, розробки варіантів і моделей прийняття рішення, ескізний проект). {4, 5} - *етап прийняття рішення і контролю* (організаційно-планова підготовка виробництва). *Раціональні етапи.* Детерміновані методи, МФПС, експертні системи логічного виведення. {2, 3} - *етап аналізів і мотивів* **(**пошук рішення і оцінка альтернатив, технічний проект). {6, 7} - *етап реалізації та її оцінки* (експлуатація, економічні методи оцінювання, аналіз витрат і вигод)**.**
Рівень 3.1 0: {1, 3, 5, 7} 1: {2, 4, 6, 8}	*Процеси.* Орієнтація на процеси в ІС. Цілі нечіткі. Каузальний підхід. *Цілі.* Орієнтація на цілі в ІС. Процес не важливий. Аксіологічний підхід. Початок ЖЦ завжди має ознаку 0, а кінець - 1 (рис. 2).
0: {1а, 2а, 3а, 4а, 5а, 6а, 7а, 8а} 1: {1б, 2б, 3б, 4б, 5б, 6б, 7б, 8б}	*Потік 1.* Загальні перспективи. Проект першої черги. Статичні властивості об’єктів. Статичні моделі або дискретні динамічні моделі. Зосередження на цілях. *Потік 2.* Точні прогнози. Проект другої черги. Динамічні властивості об’єктів. Динамічні неперервні моделі. Зосередження на методах досягнення цілей.
0: {1а, 2б, 3б, 4а, 5а, 6б, 7б, 8а} 1: {1б, 2а, 3а, 4б, 5б, 6а, 7а, 8б}	*Властивості.* Опис і моделі властивостей об’єктів. *Відношення.* Опис і моделі відношень між об’єктами.
0: {1а, 2а, 3б, 4б, 5б, 6б, 7а, 8а} 1: {1б, 2б, 3а, 4а, 5а, 6а, 7б, 8б}	*Зміст.* Еврістичні методи оцінки перспектив або прогнозів. Зміст понять. Абстрактні поняття. *Обсяг.* Еврістичні методи висування вимоги і прийняття рішення. Обсяг понять. Конкретні поняття.
0: {1а, 2а, 3а, 4а, 5б, 6б, 7б, 8б} 1: {1б, 2б, 3б, 4б, 5а, 6а, 7а, 8а}	*Кількість.* Детерміновані методи аналізу і синтезу. Формально-логічні методи. Кількісні методи. Об’єктивні оцінки. *Якість.* Детерміновані методи постановки цілей або оцінки результатів. Методи експертного оцінювання. Якісні методи. Суб’єктивні оцінки.

Таблиця Ф.2 – Бінарні відношення між елементами класів рівня 2.1

{1,2},{1,2}	{3,4},{3,4}	{5,6},{5,6}	{7,8},{7,8}	1.Тотожність
{1,2},{3,4}	{3,4},{5,6}	{5,6},{7,8}	{7,8},{1,2}	2.Пряма послідовність
{1,2},{5,6}	{3,4},{7,8}	{5,6},{1,2}	{7,8},{3,4}	3.Антипослідовність
{1,2},{7,8}	{3,4},{1,2}	{5,6},{3,4}	{7,8},{5,6}	4.Обернена послідовн.

Таблиця Ф.3 – Бінарні відношення між елементами класів *I, B, ..., Ре*

I,I	B,B	A,A	M,M	P,P	K,K	O,O	Ре,Ре	Тотожність 1
I,B	B,I	A,M	M,A	P,K	K,P	O,Ре	Ре,O	Синхронність 2
I,A	B,M	A,P	M,K	P,O	K,Ре	O,I	Ре,B	Пряма послідовність 1
I,M	B,A	A,K	M,P	P,Ре	K,O	O,B	Ре,I	Пряма послідовність 2
I,P	B,K	A,O	M,Ре	P,I	K,B	O,A	Ре,M	Антипослідовність 1
I,K	B,P	A,Ре	M,O	P,B	K,I	O,M	Ре,A	Антипослідовність 2
I,O	B,Ре	A,I	M,B	P,A	K,M	O,P	Ре,K	Обернена послідовність 1
I,Ре	B,O	A,B	M,I	P,M	K,A	O,K	Ре,P	Обернена послідовність 2



вузли: моделі (●), засоби і цілі (○),

ребра: має ціль (——>), має засіб (- - ->), має базовий клас (·····>)

Рисунок Ф.1 – Граф елементів ІС

ДОДАТОК Х

Код експертної системи з проблем надійності та довговічності різьбових з’єднань

Файл assertedFacts.csv, що доступний в GitHub (vkopey/TreadsKB: Expert system for problems of reliability and durability of threaded connections. URL: http://github.com/vkopey/TreadsKB), згенерований експертною системою автоматично та містить факти бази знань (461 факт) у вигляді триплетів (суб’єкт; предикат; об’єкт). Для зменшення об’єму цього коду в лістингу Х.1 використані такі позначення:

```
1 – Base\Фактор\Конструкційні елементи;
2 – Base\Фактор\Покриття;
3 – Base\Фактор\Пошкодження;
4 – Base\Фактор\Геометричні параметри;
5 – Base\Фактор\Матеріал;
6 – Base\Фактор\Складання;
7 – Base\Фактор\Деталі;
8 – Base\Фактор\Міцність;
9 – Base\Фактор\Жорсткість;
10 – Base\Фактор\Технологія;
11 – Base\Фактор\Тертя;
12 – Base\Фактор\Загальні напрямки підвищення надійності;
13 – Base\Фактор\Середовище;
14 – Base\Фактор\Напруження;
15 – Base\Фактор\Температура;
16 – Base\Фактор\Масштабний фактор;
17 – Base\Фактор\Навантаження;
18 – Base\Фактор\Концентрація напружень;
19 – Base\Фактор\Пошкодження при транспортуванні;
20 – Base\Фактор\Стопоріння;
21 – Base\Фактор\Згвинчуваність;
22 – Base\Фактор\Герметичність;
23 – Base\Фактор\Маса;
24 – Base\Фактор\Залишкові напруження;
25 – Base\Фактор\Рухомість з'єднання;
26 – Base\Фактор\Змащення;
27 – Base\Фактор\Пошкодження при транспортуванні;
28 – Base\Фактор\Розкриття стику.
```

Редактор коду експертної системи доступний в GitHub (vkopey/TreePyKB: Knowledge base editor. URL: http://github.com/vkopey/TreePyKB). Готовий до виконання код експертної системи, зібраний в один модуль generatedKBcode.py, доступний повністю у GitHub (vkopey/TreadsKB: Expert system for problems of

reliability and durability of threaded connections. URL: http://github.com/vkopey/TreadsKB). Він містить усі факти бази знань, що відповідають триплетам в лістингу Х.1, правила і машину виведення та запити. Код цього модуля з невеликою частиною фактів бази знань наведено у лістингу Х.2.

Лістинг Х.1 – assertedFacts.csv (триплети бази знань)

```
1\Гайка з осьовими прорізями;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Герметизуючі елементи;isCause;22
1\Зарізьбова канавка;isCause;8\За статичного навантаження\Низька
1\Зарізьбова канавка;isCause;8\За циклічного навантаження\Висока
1\Опорні поверхні\Конічні;isCause;14\Згину\Зменшення
1\Опорні поверхні\Сферичні;isCause;14\Згину\Зменшення
1\Різьба\Асиметрична різьба;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Збіг;isCause;8\За статичного навантаження\Висока
1\Різьба\Збіг;isCause;8\За циклічного навантаження\Низька
1\Різьба\Змінний середній діаметр різьби гайки;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Крок\Змінний крок;isCause;8\За циклічного навантаження\Висока
1\Різьба\Крок\Неоднаковий крок\Збільшення кроку гайки;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Крок\Неоднаковий крок\Зменшення кроку болта;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Напрям гвинтової лінії\Ліва;Not;1\Різьба\Напрям гвинтової лінії\Права
1\Різьба\Обтиск останніх витків різьби муфти;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Податливі витки\Перші витки гайки;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Розтиск останніх витків різьби ніпеля;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Розтиск перших витків різьби муфти;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Фаски\Зріз перших витків різьби муфти;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Різьба\Форма профілю\Кругла;isCause;8\За циклічного навантаження\Висока
1\Різьба\Форма профілю\Прямокутна;isCause;18\Нерівномірне навантаження по виткам різьби
1\Різьба\Форма профілю\Спеціальна;isCause;8\За циклічного навантаження\Висока
1\Різьба\Форма профілю\Трикутна;isCause;3\Залишкова деформація\Деформація тіла гайки
1\Різьба\Форма профілю\Упорна;isCause;18\Нерівномірне навантаження по виткам різьби
```

1\Різьба\Форма профілю\Упорна;isCause;3\Залишкова деформація\Деформація тіла гайки\Зменшення
1\Різьба\Характер поверхні\Конічна;isCause;22
1\Розташування різьбової частини болта\Вільні витки болта;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Розташування різьбової частини болта\Різьба болта утоплена в гайку;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
1\Форма головки болта\Двухрадіусна галтель;isCause;18\Під головкою болта\Низька
1\Форма головки болта\Оптимальна;isCause;8\За циклічного навантаження\Висока
1\Центруючі елементи;isCause;14\Згину\Зменшення
2;isCause;3\Заїдання\Зменшення
2\Багатошарове мідь-нікель, хром;isCause;15\Робоча температура\Максимальна\600
2\Багатошарове мідь-нікель;isCause;15\Робоча температура\Максимальна\600
2\Електрохімічна обробка;isCause;5\Насичення атомарним воднем
2\Зносостійкі;isCause;12\Захист від механічного спрацювання деталей
2\Кадмієве з хроматуванням;isCause;11\На різьбі\Мале
2\Кадмієве з хроматуванням;isCause;15\Робоча температура\Максимальна\200
2\Кадмієве з хроматуванням;isCause;2\Електрохімічна обробка
2\Корозійностійкі;isCause;3\Корозійна\Низька
2\Мідне;isCause;11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Збільшується
2\Мідне;isCause;15\Робоча температура\Максимальна\600
2\Нікелеве;isCause;11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Збільшується
2\Нікелеве;isCause;15\Робоча температура\Максимальна\900
2\Оксидне анодізаційне з хроматуванням;isCause;15\Робоча температура\Максимальна\200
2\Оксидне з кислих розчинів;isCause;15\Робоча температура\Максимальна\200
2\Оксидне;isCause;11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Збільшується
2\Оксидне;isCause;15\Робоча температура\Максимальна\200
2\Оксидне;isCause;24\Стиску
2\Олов'яне;isCause;11\На різьбі\Мале
2\Олов'яне;isCause;15\Робоча температура\Максимальна\150
2\Оптимальне;isCause;8\За циклічного навантаження\Висока
2\Пластичне;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
2\Свинцеве;isCause;11\На різьбі\Мале
2\Срібне;isCause;11\На різьбі\Мале
2\Срібне;isCause;15\Робоча температура\Максимальна\600
2\Травлення під час нанесення покриттів;isCause;5\Насичення атомарним воднем
2\Фосфатне;isCause;15\Робоча температура\Максимальна\200
2\Фосфатне;isCause;2\Травлення під час нанесення покриттів
2\Цинкове з хроматуванням;isCause;15\Робоча температура\Максимальна\300
2\Цинкове;isCause;11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Збільшується
2\Цинкове;isCause;15\Робоча температура\Максимальна\200
2\Цинкове;isCause;2\Електрохімічна обробка

3\Від високошвидкісного навантаження;SubClassOf;3
3\Втомна тріщина;isCause;3\Корозійна
3\Втомна тріщина;SubClassOf;3
3\Втомна тріщина\В останньому витку муфти;SubClassOf;3\Втомна тріщина
3\Втомна тріщина\В першому витку ніпеля;SubClassOf;3\Втомна тріщина
3\Втомна тріщина\Під головкою болта;SubClassOf;3\Втомна тріщина
3\Заїдання;isCause;11\На різьбі\Велике
3\Заїдання;isCause;11\На упорному бурті\Велике
3\Заїдання;isCause;14\Кручення
3\Заїдання;isCause;3\Знос
3\Заїдання;Not;3\Заїдання\Зменшення
3\Заїдання;SubClassOf;3
3\Залишкова деформація;isCause;3\Заїдання
3\Залишкова деформація;isCause;3\Самовідгвинчування
3\Залишкова деформація;SubClassOf;3
3\Залишкова деформація\Деформація тіла гайки;Not;3\Залишкова деформація\Деформація тіла гайки\Зменшення
3\Залишкова деформація\Деформація тіла гайки;SubClassOf;3\Залишкова деформація
3\Залишкова деформація\Зріз витків;Not;3\Залишкова деформація\Зріз витків\Зменшення
3\Залишкова деформація\Зріз витків;SubClassOf;3\Залишкова деформація
3\Залишкова деформація\Зріз стержня;Not;3\Залишкова деформація\Зріз стержня\Зменшення
3\Залишкова деформація\Зріз стержня;SubClassOf;3\Залишкова деформація
3\Залишкова деформація\Обрив стержня;SubClassOf;3\Залишкова деформація
3\Знос;isCause;22\Низька
3\Знос;isCause;3\Залишкова деформація\Зріз витків
3\Знос;SubClassOf;3
3\Знос\Абразивний;SubClassOf;3\Знос
3\Знос\Адгезійний;SubClassOf;3\Знос
3\Знос\Втомний;SubClassOf;3\Знос
3\Знос\Ерозійний;SubClassOf;3\Знос
3\Знос\Зменшення;Not;3\Знос
3\Знос\Механохімічний;SubClassOf;3\Знос
3\Знос\Фретинг-корозійний;SubClassOf;3\Знос
3\Знос\Фретинговий;SubClassOf;3\Знос
3\Корозійна;isCause;18
3\Корозійна;isCause;22\Низька
3\Корозійна;isCause;3\Заїдання
3\Корозійна;isCause;3\Знос
3\Корозійна;isCause;8\Низька
3\Корозійна;SubClassOf;3
3\Корозійна\Низька;Not;3\Корозійна
3\Корозійне розтріскування;isCause;3\Крихкий злам
3\Корозійне розтріскування;isCause;8\За статичного навантаження\Низька
3\Корозійне розтріскування;SubClassOf;3
3\Крихкий злам;SubClassOf;3
3\Крихкий злам\Уповільнене крихке руйнування;SubClassOf;3\Крихкий злам
3\Крихкий злам\Уповільнене крихке руйнування\Низьке;Not;3\Крихкий злам\Уповільнене крихке руйнування

3\Розгерметизація;SubClassOf;3
3\Розгерметизація\Запобігання;Not;3\Розгерметизація
3\Самовідгвинчування;isCause;3\Залишкова деформація\Зріз витків
3\Самовідгвинчування;SubClassOf;3
3\Самовідгвинчування\Зменшення;Not;3\Самовідгвинчування
4\Допуск розміру\Великий;isCause;4\Посадка\З зазором
4\Допуск розміру\Малий;isEffect;6\Селекційне складання
4\Допуск розміру\Малий;Not;4\Допуск розміру\Великий
4\Допуск форми\Перекос опорних поверхонь;isCause;14\Згину
4\Допуск форми\Перекос опорних поверхонь;isCause;3\Крихкий злам\Уповільнене крихке руйнування
4\Допуск форми\Перекос опорних поверхонь\Високоміцних сталей;isCause;8\За циклічного навантаження\Низька
4\Допуск форми\Перекос опорних поверхонь\Зменшення;Not;4\Допуск форми\Перекос опорних поверхонь
4\Допуск форми\Перекос опорних поверхонь\Зменшення;Not;4\Допуск форми\Перекос опорних поверхонь\Великий
4\Допуск форми\Перекос осей;isCause;14\Згину
4\Допуск форми\Перекос осей;Not;4\Допуск форми\Перекос осей\Малий
4\Допуск форми\Чутливість до перекосу;isCause;8\За статичного навантаження\Низька
4\Допуск форми\Чутливість до перекосу;isCause;8\За циклічного навантаження\Низька
4\Допуск форми\Чутливість до перекосу;Not;4\Допуск форми\Нечутливість до перекосу
4\Посадка\З зазором;isCause;25
4\Посадка\З зазором\Зменшення діаметральних зазорів;isCause;18\Низька
4\Посадка\З зазором\Зменшення діаметральних зазорів;isCause;3\Заїдання
4\Посадка\З зазором\Зменшення діаметральних зазорів;isCause;8\За циклічного навантаження\Висока
4\Посадка\З натягом;isCause;25\Низька
4\Посадка\З натягом;isCause;3\Заїдання
4\Посадка\З натягом;isCause;6\Момент згвинчування\Неоптимальний\Великий
4\Посадка\Оптимальна;isCause;18\Низька
4\Посадка\Оптимальна;isCause;8\За циклічного навантаження\Висока
4\Посадка\Перехідна;isCause;25\Низька
4\Розмір\Відношення діаметру до кроку\Велике;isCause;18\Нерівномірне навантаження по виткам різьби
4\Розмір\Діаметр зарізьбової канавки\Великий;isCause;8\За статичного навантаження\Висока
4\Розмір\Діаметр зарізьбової канавки\Великий;isCause;8\За циклічного навантаження\Низька
4\Розмір\Діаметр зарізьбової канавки\Великий;Not;4\Розмір\Діаметр зарізьбової канавки\Малий
4\Розмір\Діаметр різьби\Великий;isCause;3\Залишкова деформація\Зріз витків\Зменшення
4\Розмір\Діаметр різьби\Великий\З концентратором напружень;isCause;8\За циклічного навантаження\Низька
4\Розмір\Довжина гайки\Велика;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке

4\Розмір\Довжина гайки\Велика;isCause;3\Залишкова деформація\Зріз витків\Зменшення
4\Розмір\Довжина головки болта\Велика;isCause;18\Під головкою болта\Низька
4\Розмір\Довжина зарізьбової канавки\Велика;isCause;9\Болта\Мала
4\Розмір\Довжина згвинчування\Мала;isCause;3\Залишкова деформація\Зріз витків
4\Розмір\Крок різьби\Великий;isCause;3\Залишкова деформація\Зріз витків\Зменшення
4\Розмір\Крок різьби\Оптимальний;isCause;8\За циклічного навантаження\Висока
4\Розмір\Кут профілю\Великий;isCause;3\Залишкова деформація\Деформація тіла гайки
4\Розмір\Кут профілю\Великий;Not;4\Розмір\Кут профілю\Малий
4\Розмір\Кут профілю\Малий;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
4\Розмір\Кут профілю\Малий;isCause;3\Залишкова деформація\Зріз витків
4\Розмір\Перекриття витків\Велике;isCause;8
4\Розмір\Перекриття витків\Мале;isCause;3\Залишкова деформація\Зріз витків
4\Розмір\Перекриття витків\Мале;Not;4\Розмір\Перекриття витків\Велике
4\Розмір\Радіус впадин різьби\Збільшення;isCause;18\Низька
4\Розмір\Радіус впадин різьби\Збільшення;isCause;3\Крихкий злам\Уповільнене крихке руйнування\Низьке
4\Розмір\Типорозмір різьби\Великий;isCause;16\Великі розміри
4\Розмір\Типорозмір різьби\Великий;isCause;23\Велика
4\Розмір\Типорозмір різьби\Великий;Not;4\Розмір\Типорозмір різьби\Малий
4\Розмір\Товщина гайки\Велика;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
4\Розмір\Товщина гайки\Велика;isCause;8
4\Розмір\Товщина гайки\Велика;Not;4\Розмір\Товщина гайки\Мала
4\Розмір\Товщина гайки\Мала;isCause;1\Різьба\Форма профілю\Упорна
4\Шорсткість\Висока;isCause;18
4\Шорсткість\Висока;isCause;3\Заїдання
4\Шорсткість\Висока;isCause;3\Знос
4\Шорсткість\Висока;isCause;3\Корозійна
4\Шорсткість\Низька;isCause;18\Низька
4\Шорсткість\Низька;isCause;3\Самовідгвинчування\Зменшення
4\Шорсткість\Низька;Not;4\Шорсткість\Висока
4\Шорсткість\Низька\Леговані сталі;isCause;18\Низька
4\Шорсткість\Низька\Леговані сталі;isCause;8\За циклічного навантаження\Висока
5\Алюмінієві сплави;isCause;23\Мала
5\Алюмінієві сплави\Гайки;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
5\Бериллієві сплави;isCause;10\Метод виготовлення\Накатування
5\Бериллієві сплави;isCause;23\Мала
5\Бериллієві сплави;isCause;3\Корозійна\Низька
5\Бериллієві сплави;isCause;5\Алюмінієві сплави\Гайки
5\Бериллієві сплави;isCause;5\Токсичність
5\Бериллієві сплави;isCause;5\Чутливість до концентрації напружень
5\Бесемерівська сталь;isCause;5\Наявність азоту
5\Бесемерівська сталь;isCause;5\Наявність фосфору

5\Високоміцні сталі болтів;isCause;15\Робоча температура\Висока
5\Високоміцні сталі болтів;isCause;3\Залишкова деформація\Зріз стержня\Зменшення
5\Високоміцні сталі болтів;isCause;3\Корозійне розтріскування
5\Високоміцні сталі болтів;isCause;3\Самовідгвинчування\Зменшення
5\Високоміцні сталі болтів;isCause;4\Допуск форми\Чутливість до перекосу
5\Високоміцні сталі болтів;isCause;5\Пластичність\Низька
5\Високоміцні сталі болтів;isCause;8\За циклічного навантаження\Висока
5\Високоміцні сталі болтів\Границя міцності\1100-1400;isCause;15\Робоча температура\Максимальна\400
5\Високоміцні сталі болтів\Границя міцності\1100-1400;isCause;8\За циклічного навантаження\Висока
5\Високоміцні сталі болтів\Границя міцності\1100-1600;isCause;8\За осьового навантаження
5\Високоміцні сталі болтів\Границя міцності\1800-2100;isCause;8\За зрізуючого навантаження
5\Високоміцні сталі болтів\За високих моментів згвинчування і агресивного середовища;isCause;3\Корозійне розтріскування
5\Високоміцні сталі болтів\За високих моментів згвинчування і агресивного середовища;And;5\Вуглець фосфор азот на границях зерен
5\Високоміцні сталі болтів\За високих моментів згвинчування і агресивного середовища;And;6\Момент згвинчування\Неоптимальний\Великий
5\Високоміцні сталі болтів\За високих моментів згвинчування і агресивного середовища;And;13\Корозійне
5\Високоміцні сталі болтів\Зарізьбова канавка;isCause;8
5\Високоміцні сталі болтів\Зниження водневої крихкості;isCause;8
5\Високоміцні сталі болтів\Кадміювання;isCause;8
5\Високоміцні сталі болтів\Накатування;isCause;10\Метод виготовлення\Накатування\Стійкість інструмента\Низька
5\Високоміцні сталі болтів\Накатування;isCause;8\За циклічного навантаження\Висока
5\Високоміцні сталі болтів\Полірування;isCause;8
5\Високоміцні сталі болтів\Спеціальна термічна обробка високоміцних болтів з легованних сталей;isCause;8
5\Високоміцні сталі болтів\Сталь07Х16Н6;isCause;15\Робоча температура\Мінімальна\-196
5\Високоміцні сталі болтів\Сталь07Х16Н6;isCause;5\Пластичність\Висока
5\Високоміцні сталі болтів\Сталь07Х16Н6;isCause;5\Холодноламкість\Низька
5\Високоміцні сталі болтів\Сталь07Х16Н6;isCause;8\За циклічного навантаження\Висока
5\Високоміцні сталі болтів\Сталь07Х16Н6;SubClassOf;5\Високоміцні сталі болтів
5\Високоміцні сталі болтів\Сталь07Х16Н6\Обробка холодом;isCause;5\Чутливість до концентрації напружень\Низька
5\Високоміцні сталі болтів\Сталь1Х15Н4АМ3-Ш;isCause;5\Холодноламкість\Низька
5\Високоміцні сталі болтів\Сталь1Х15Н4АМ3-Ш;isCause;8\За циклічного навантаження\Висока
5\Високоміцні сталі болтів\Сталь1Х15Н4АМ3-Ш;SubClassOf;5\Високоміцні сталі болтів

5\Високоміцні сталі болтів\Сталь1Х15Н4АМ3-Ш\Обробка холодом;isCause;5\Чутливість до концентрації напружень\Низька
5\Вуглець у поверхневих шарах;Not;5\Вуглець у поверхневих шарах\Зменшення
5\Вуглець у поверхневих шарах\Неоптимальний;isCause;8\За циклічного навантаження\Низька
5\Вуглець у поверхневих шарах\Оптимальний;isCause;8\За циклічного навантаження\Висока
5\Вуглець у поверхневих шарах\Оптимальний;Not;5\Вуглець у поверхневих шарах\Неоптимальний
5\Вуглець фосфор азот на границях зерен;isCause;3\Корозійне розтріскування
5\Газонасочений шар;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Жароміцні сплави\Висока робоча температура;isCause;3\Заїдання
5\Жароміцні сплави\Висока робоча температура;isCause;4\Допуск форми\Чутливість до перекосу
5\Жароміцні сплави\Висока робоча температура;isCause;5\Чутливість до концентрації напружень
5\Коефіцієнт лінійного розширення\Гайки\Великий;isCause;3\Заїдання\Зменшення
5\Корозійностійкі матеріали;isCause;3\Корозійна\Низька
5\Корозійностійкі матеріали\Сталі;isCause;3\Заїдання
5\Корозійностійкі матеріали\Сталі;SubClassOf;5\Корозійностійкі матеріали
5\Кремнієві сталі;isCause;3\Заїдання\Зменшення
5\Крихкий поверхневий шар;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Крихкість;isCause;3\Крихкий злам
5\Матеріали з метастабільною структурою і малою пластичністю;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Матеріали з метастабільною структурою і малою пластичністю;isCause;5\Пластичність\Низька
5\Надвисокоміцні сталі;isCause;3\Залишкова деформація\Зріз стержня\Зменшення
5\Надвисокоміцні сталі;isCause;5\Пластичність\Низька
5\Насичення атомарним воднем;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Насичення атомарним воднем;isCause;5\Крихкість
5\Наявність азоту;isCause;5\Крихкість
5\Наявність фосфору;isCause;5\Крихкість
5\Ніобій;isCause;5\Холодноламкість\Низька
5\Пластичність\Висока\Гайки;isCause;18\Нерівномірне навантаження по виткам різьби\Низьке
5\Пластичність\Низька;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Пластичність\Низька;isCause;5\Чутливість до концентрації напружень
5\Пластичність\Низька;Not;5\Пластичність\Висока
5\Пластмасси;isCause;22
5\Пластмасси;isCause;3\Розгерметизація\Запобігання
5\Пластмасси;isCause;5\Корозійностійкі матеріали
5\Пластмасси;isCause;5\Электричний опір\Великий
5\Повзучість;isCause;3\Залишкова деформація
5\Повзучість;isCause;3\Самовідгвинчування
5\Протекторний захист;isCause;3\Корозійна\Низька
5\Співвідношення матеріалів\Неоптимальне;isCause;3\Заїдання

```
5\Співвідношення матеріалів\Пластична гайка;isCause;18\Нерівномірне
навантаження по виткам різьби\Низьке
5\Співвідношення матеріалів\Різні механічні властивості болта і
гайки;isCause;3\Залишкова деформація\Зріз витків
5\Старіння сталі;isCause;5\Пластичність\Низька
5\Старіння сталі\Низьковуглецевої;isCause;8
5\Структура матеріалу\Покращення;isCause;8\За циклічного
навантаження\Висока
5\Термообробка\В захисній атмосфері;isCause;5\Вуглець у поверхневих
шарах\Зменшення
5\Термообробка\Відпуск;SubClassOf;5\Термообробка
5\Термообробка\Відпуск\Збільшення температури відпуску;isCause;4\Допуск
форми\Нечутливість до перекосу
5\Термообробка\Відпуск\Недостатній після гартування;isCause;5\Крихкість
5\Термообробка\Гартування;SubClassOf;5\Термообробка
5\Термообробка\Гартування\Перегрів при гартуванні;isCause;5\Крихкість
5\Термообробка\Хіміко-термічна обробка;isCause;8\За циклічного
навантаження\Висока
5\Термообробка\Хіміко-термічна обробка;SubClassOf;5\Термообробка
5\Термообробка\Хіміко-термічна обробка\Азотування\Високий момент
згвинчування;isCause;3\Крихкий злам
5\Технологічні дефекти матеріалів;isCause;18
5\Технологічні дефекти матеріалів;isCause;5\Крихкість
5\Титанові сплави;isCause;11\На різьбі\Велике
5\Титанові сплави;isCause;11\На упорному бурті\Велике
5\Титанові сплави;isCause;15\Робоча температура\Максимальна\350-550
5\Титанові сплави;isCause;23\Мала
5\Титанові сплави;isCause;24\Розтягу\Чутливість
5\Титанові сплави;isCause;3\Залишкова деформація\Зріз витків\Зменшення
5\Титанові сплави;isCause;3\Крихкий злам\Уповільнене крихке
руйнування\Низьке
5\Титанові сплави;isCause;3\Крихкий злам\Уповільнене крихке руйнування
5\Титанові сплави;isCause;4\Допуск форми\Нечутливість до перекосу
5\Титанові сплави;isCause;5\Корозійностійкі матеріали
5\Титанові сплави;isCause;5\Пластичність\Низька
5\Титанові сплави;isCause;5\Чутливість до концентрації напружень
5\Титанові сплави;isCause;8
5\Титанові сплави;isCause;8\За зрізуючого навантаження
5\Холодноламкість;isCause;8\За статичного навантаження\Низька
5\Холодноламкість;Not;5\Холодноламкість\Низька
5\Чутливість до концентрації напружень;isCause;18
5\Чутливість до концентрації напружень;Not;5\Чутливість до концентрації
напружень\Низька
6\Згвинчування в нагрітому стані;isCause;4\Посадка\З натягом
6\Контргайка затянута великим моментом;isCause;18\Нерівномірне навантаження
по виткам різьби\Низьке
6\Контроль затягування;isCause;6\Момент згвинчування\Оптимальний
6\Контроль затягування\Контроль за видовженням болта;SubClassOf;6\Контроль
затягування
6\Контроль затягування\Контроль за кутом повороту;SubClassOf;6\Контроль
затягування
```

```
6\Контроль затягування\Контроль за моментом
згвинчування;SubClassOf;6\Контроль затягування
6\Момент згвинчування\Неоптимальний;isCause;3\Втомна тріщина
6\Момент згвинчування\Неоптимальний\Великий;isCause;22
6\Момент згвинчування\Неоптимальний\Великий;isCause;3\Залишкова деформація
6\Момент згвинчування\Неоптимальний\Великий;isCause;8\За циклічного
навантаження\Висока
6\Момент згвинчування\Неоптимальний\Великий;Not;6\Момент
згвинчування\Неоптимальний\Малий
6\Момент згвинчування\Неоптимальний\Малий;isCause;3\Самовідгвинчування
6\Момент згвинчування\Оптимальний;isCause;8\За циклічного
навантаження\Висока
6\Момент згвинчування\Оптимальний;Not;6\Момент згвинчування\Неоптимальний
6\Очищення різьби;isCause;11\На різьбі\Мале
6\Очищення різьби;isCause;21\Добра
6\Попереднє пластичне деформування перших витків;isCause;18\Нерівномірне
навантаження по виткам різьби\Низьке
6\Селекційне складання;isCause;4\Посадка\З зазором\Зменшення діаметральних
зазорів
6\Тертя при згвинчуванні;isCause;6\Момент
згвинчування\Неоптимальний\Великий
6\Уникнення перекосів;isCause;4\Допуск форми\Перекос осей\Малий
6\Фіксація болта при згвинчуванні;isCause;14\Кручення\Зменшення
6\Часте згвинчування розгвинчування;isCause;3\Знос
7\Болт\Болт під розвертку;isCause;14\Зрізу\Зменшення
7\Гайка\Гайка розтягу-стиску;isCause;18\Нерівномірне навантаження по виткам
різьби\Низьке
7\Гвинтова вставка;isCause;12\Зменшення механічного спрацювання витків
7\Гвинтова вставка;isCause;18\Нерівномірне навантаження по виткам
різьби\Низьке
7\Захисний ковпачок;isCause;19\Зменшення
7\З'єднувані деталі\Мала кількість;isCause;3\Самовідгвинчування\Зменшення
7\Компенсатори температурних деформацій;isCause;17\Температурне\Зменшення
7\Протектори;isCause;3\Знос\Зменшення
7\Сферична шайба;isCause;14\Згину\Зменшення
7\Шайба\З низьковуглецевої сталі;isCause;4\Допуск форми\Перекос опорних
поверхонь\Зменшення
7\Шайба\Пружинна;isCause;3\Самовідгвинчування\Зменшення
7\Шайба\Сферична;isCause;14\Згину\Зменшення
8\За зрізуючого навантаження;isCause;3\Залишкова деформація\Зріз
стержня\Зменшення
8\За зрізуючого навантаження;SubClassOf;8
8\За осьового навантаження;SubClassOf;8
8\За статичного навантаження;SubClassOf;8
8\За статичного навантаження\Висока;Not;8\За статичного навантаження\Низька
8\За статичного навантаження\Низька;isCause;3\Залишкова деформація
8\За статичного навантаження\Низька;isCause;3\Крихкий злам
8\За циклічного навантаження;SubClassOf;8
8\За циклічного навантаження\Висока;Not;8\За циклічного навантаження\Низька
8\За циклічного навантаження\Низька;isCause;3\Втомна тріщина
8\Низька;Not;8
```

9\Болта;isCause;14\Розтягу\Болта
9\Болта;Not;9\Болта\Мала
9\З'єднуваних деталей;isCause;14\Розтягу\Болта\Зменшення
9\З'єднуваних деталей;Not;9\З'єднуваних деталей\Мала
9\З'єднуваних деталей\Мала;isCause;3\Крихкий злам\Уповільнене крихке руйнування
9\З'єднуваних деталей\Мала;isCause;3\Самовідгвинчування
9\З'єднуваних деталей\Мала;isCause;8\За циклічного навантаження\Низька
10\Метод виготовлення\Накатування;isCause;24\Стиску
10\Метод виготовлення\Накатування;isCause;3\Крихкий злам\Уповільнене крихке руйнування\Низьке
10\Метод виготовлення\Накатування\Стійкість інструмента\Низька;Not;10\Метод виготовлення\Накатування\Стійкість інструмента
10\Метод виготовлення\Обкатування після нарізання;isCause;24\Стиску
10\Метод виготовлення\Піскоструминна обробка;isCause;3\Крихкий злам\Уповільнене крихке руйнування\Низьке
10\Метод виготовлення\Полірування;isCause;3\Крихкий злам\Уповільнене крихке руйнування\Низьке
10\Попередній статичний розтяг високоміцних болтів;isCause;8\За циклічного навантаження\Висока
10\Режими обробки\Оптимальні;isCause;24\Стиску
10\Режими обробки\Оптимальні;isCause;4\Допуск розміру\Малий
10\Режими обробки\Оптимальні;isCause;8
10\Технологічні дефекти;isCause;18
10\Технологічні дефекти;isCause;3\Крихкий злам\Уповільнене крихке руйнування
10\Технологічні дефекти;isCause;8\За статичного навантаження\Низька
11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Збільшується;Not;11\Зміна коефіцієнта тертя під час повторних згвинчуваннях\Зменшується
11\На різьбі\Велике;isCause;21\Погана
11\На різьбі\Велике;isCause;3\Заїдання
11\На різьбі\Велике;Not;11\На різьбі\Мале
11\На різьбі\Мале;isCause;21\Добра
11\На упорному бурті\Велике;isCause;21\Погана
11\На упорному бурті\Велике;isCause;3\Заїдання
11\На упорному бурті\Велике;Not;11\На упорному бурті\Мале
11\На упорному бурті\Мале;isCause;21\Добра
12\Вибір оптимальних допусків і шорсткості;isCause;18\Низька
12\Вибір оптимальної технології;isCause;24\Стиску
12\Вибір оптимальної технології;isCause;4\Шорсткість\Низька
12\Запобігання заїдань;isCause;3\Заїдання\Зменшення
12\Запобігання розгерметизації;isCause;22
12\Захист від корозійно-втомного і статичного руйнування;isCause;8
12\Захист від корозійного руйнування;isCause;3\Корозійна\Низька
12\Захист від механічного спрацювання деталей;isCause;3\Знос\Зменшення
12\Захист від середовища;isCause;13\Корозійне\Захист
12\Захист при транспортуванні;isCause;19\Зменшення
12\Зменшення концентрації напружень;isCause;18\Низька
12\Підбір матеріалів і термообробки;isCause;3\Корозійна\Низька
12\Підбір матеріалів і термообробки;isCause;8

12\Попередження самовідгвинчування;isCause;3\Самовідгвинчування\Зменшення
12\Центрування різьби запобігання згину;isCause;14\Згину\Зменшення
13\Інгібіторний захист;isCause;3\Корозійна\Низька
13\Корозійне;isCause;3\Корозійна
13\Корозійне;isCause;3\Корозійне розтріскування
13\Корозійне;isCause;8\За циклічного навантаження\Низька
13\Корозійне;Not;13\Корозійне\Захист
14\Згину;isCause;3\Втомна тріщина
14\Згину\Зменшення;Not;14\Згину
14\Зрізу;isCause;3\Залишкова деформація\Зріз стержня
14\Зрізу;Not;14\Зрізу\Зменшення
14\Кручення;Not;14\Кручення\Зменшення
14\Розтягу\Болта;isCause;3\Крихкий злам\Уповільнене крихке руйнування
14\Розтягу\Болта;Not;14\Розтягу\Болта\Зменшення
15\Робоча температура\Висока;isCause;17\Температурне
15\Робоча температура\Висока;isCause;3\Заїдання
15\Робоча температура\Висока;isCause;5\Повзучість
15\Робоча температура\Висока;isCause;5\Чутливість до концентрації напружень
15\Робоча температура\Висока;isCause;8\За циклічного навантаження\Низька
15\Робоча температура\Висока;Not;15\Робоча температура\Низька
15\Робоча температура\Низька;isCause;5\Пластичність\Низька
15\Робоча температура\Низька;isCause;5\Холодноламкість
15\Температура за залишкових напруженнях стиску\Висока;isCause;8\За циклічного навантаження\Низька
16\Великі розміри;isCause;8\За циклічного навантаження\Низька
16\Великі розміри;Not;16\Малі розміри
17\Вібрація;isCause;3\Самовідгвинчування
17\Згинаюче;isCause;14\Згину
17\Згинаюче;isCause;22\Низька
17\Згинаюче;isCause;3\Втомна тріщина
17\Зрізаюче;isCause;14\Зрізу
17\Зрізаюче;isCause;3\Залишкова деформація\Зріз стержня
17\Крутне;isCause;14\Кручення
17\Крутне\Догвинчування;isCause;3\Залишкова деформація\Зріз витків
17\Крутне\Догвинчування;isCause;3\Самовідгвинчування\Зменшення
17\Крутне\Догвинчування;SubClassOf;17\Крутне
17\Крутне\Розгвинчування;isCause;3\Самовідгвинчування
17\Крутне\Розгвинчування;SubClassOf;17\Крутне
17\Осьове\Розтягу;isCause;14\Розтягу
17\Осьове\Розтягу;isCause;3\Залишкова деформація\Зріз витків
17\Осьове\Розтягу;isCause;3\Залишкова деформація\Обрив стержня
17\Осьове\Розтягу;Not;17\Осьове\Стиску
17\Температурне;isCause;17\Осьове\Розтягу
17\Температурне;Not;17\Температурне\Зменшення
17\Тиск;isCause;13\Корозійне
17\Удар;isCause;3\Від високошвидкісного навантаження
17\Циклічне;isCause;3\Втомна тріщина
17\Швидкісне\Низьковуглецева сталь;isCause;8
18;isCause;3\Крихкий злам\Уповільнене крихке руйнування
18;isCause;8\За статичного навантаження\Низька
18;isCause;8\За циклічного навантаження\Низька

```
18;Not;18\Низька
18\Нерівномірне навантаження по виткам різьби;isCause;18
18\Нерівномірне навантаження по виткам різьби;isCause;3\Втомна тріщина\В
першому витку ніпеля
18\Нерівномірне навантаження по виткам різьби;Not;18\Нерівномірне
навантаження по виткам різьби\Низьке
18\Під головкою болта;isCause;3\Втомна тріщина\Під головкою болта
18\Під головкою болта;Not;18\Під головкою болта\Низька
18\Під головкою болта;SubClassOf;18
20;isCause;3\Самовідгвинчування\Зменшення
20\Жорстке;SubClassOf;20
20\Клеєм;SubClassOf;20
20\Фрикційне;SubClassOf;20
21\Погана;isCause;14\Кручення
21\Погана;isCause;3\Знос
21\Погана;Not;21\Добра
22;isCause;13\Корозійне\Захист
22;Not;22\Низька
22\Низька;isCause;3\Розгерметизація
23\Мала;Not;23\Велика
24\Розтягу;isCause;8\За циклічного навантаження\Низька
24\Розтягу;Not;24\Стиску
24\Розтягу\Чутливість;isCause;8\За циклічного навантаження\Низька
24\Стиску;isCause;8\За циклічного навантаження\Висока
24\Стиску\За високої температури;And;15\Робоча температура\Висока
24\Стиску\За високої температури;And;24\Стиску
24\Стиску\За високої температури;isCause;8\За циклічного
навантаження\Низька
25;isCause;3\Самовідгвинчування
25;Not;25\Низька
26;isCause;11\На різьбі\Мале
26;isCause;11\На упорному бурті\Мале
26;isCause;12\Зменшення механічного спрацювання витків
26;isCause;3\Заїдання\Зменшення
26\Відсутність;Not;26
27;isCause;18
27;Not;19\Зменшення
28;isCause;22\Низька
28;isCause;8\За циклічного навантаження\Низька
```

Лістинг Х.2 – generatedKBcode.py

```
# -*- coding: CP1251 -*-
import os
import copy

class Property(object):
    '''Клас, який описує властивість об'єкта'''
```

```
    def __init__(self, subj, name, inverseName='', functional=False,
symmetric=False, transitive=False):
        '''Конструктор'''
        self.subj=subj # суб'єкт властивості
        self.name=name # назва властивості
        self.inverseName=inverseName # назва інверсної властивості
        self.functional=functional # властивість функціональна
        self.symmetric=symmetric # властивість симетрична
        self.transitive=transitive # властивість транзитивна
        self.set=set() # множина значень властивості
        self.subj.__setattr__(self.name,self) # установити атрибут
властивості для суб'єкта

    def add(self,*args):
        '''Додає об'єкт або кортеж об'єктів в множину'''
        for obj in args: # для всіх об'єктів в args
            if self.functional: # якщо властивість функціональна
                self.set.clear() # очистити множину
            self.set.add(obj) # додати об'єкт в множину

    def get(self, showTransitive=False):
            '''Повертає множину значень властивості
            (showTransitive=True - транзитивної властивості)'''
            def getTransitive(subj,s=set()):
                '''Повертає множину значень транзитивної властивості.
Рекурсивна'''
                if hasattr(subj, self.name): # якщо subj має атрибут
self.name
                    # для усіх об'єктів в властивості з назвою self.name
                    for obj in subj.__dict__[self.name].set:
                        if obj not in s: # якщо obj немає в множині s
                            s.add(obj) # додати об'єкт в множину
                            s=getTransitive(obj,s) # рекурсія
                return s # повертає множину
            # якщо властивість транзитивна і показувати транзитивні
            if self.transitive and showTransitive:
                # множина значень транзитивної властивості
                s=getTransitive(self.subj)
                s.discard(self.subj) # вилучити self.subj з множини, якщо є
                return s
            # інакше повернути множину значень властивості
            else: return self.set

KB={} # словник бази знань
programDir="D:\!My_doc\Python_projects\TreePyKB"

class X(object):
    '''Базовий клас онтології'''
    def __init__(self): # конструктор
        # ці властивості дозволяють створювати нові об'єкти за допомогою
логічних операцій
```

```
        Property(subj=self,name='And') # властивість 'And'
        Property(subj=self,name='Or') # властивість 'Or'
        Property(subj=self,name='Not',symmetric=True) # властивість 'Not'
        Property(subj=self,name='SubClassOf',transitive=True) # властивість
'SubClassOf'
        self.doc="" # довідка
KB[r"""Base"""]=X; del X

class X(KB[r"""Base"""]):
    '''Клас, який описує посилання на джерело'''
    def __init__(self): # конструктор
        KB[r"""Base"""].__init__(self)
        # властивість 'є посиланням'
        Property(subj=self,name='isReference',inverseName='hasReference')
KB[r"""Base\Джерело"""]=X; del X

class X(KB[r"""Base"""]):
    '''Клас, який описує залежність'''
    def __init__(self, xy=None, relative=None, xName='x', yName='y'):
        KB[r"""Base"""].__init__(self)
        # властивість 'є залежністю'
        Property(subj=self,name='isDependence',inverseName='hasDependence')
        self.xy=xy # дані у вигляді [(x1,y1),(x2,y2),(x3,y3)]
        self.relative=relative # відносна залежність (збільшує, зменшує, є
екстремум)
        self.xName=xName; self.yName=yName # назви осей
    def plot(self):
        '''Рисує графік залежності'''
        from matplotlib import rcParams, pyplot # бібліотека matplotlib
        rcParams['text.usetex']=False
        rcParams['font.sans-serif'] = ['Arial']
        rcParams['font.serif'] = ['Arial'] # шрифт для вводу кирилиці
        x,y=[p[0] for p in self.xy],[p[1] for p in self.xy] # розділити
дані
        pyplot.plot(x,y,'b-') #крива
        pyplot.title(unicode('')) # заголовок
        pyplot.xlabel(unicode(self.xName)) # надпис осі x
        pyplot.ylabel(unicode(self.yName)) # надпис осі y
        pyplot.grid(True) # сітка
        pyplot.show() # показати рисунок
    def interp(self,x,reverse=False):
        '''Знаходить значення Y лінійною інтерполяцією
        якщо reverse=True, то знаходить значення X'''
        # якщо reverse=True, поміняти X і Y місцями
        if reverse: data=[(p[1],p[0]) for p in self.xy]
        else: data=self.xy
        lines=[] # список ліній залежності
        p1=data[0] # перша точка
        for p2 in data[1:]: # для усіх точок крім першої
            lines.append((p1,p2)) # створити лінію з пар сусідніх точок
            p1=p2
```

```
        results=[] # список результатів
        for line in lines: # для всіх ліній залежності
            # координати точок лінії
            x1,y1,x2,y2=line[0][0],line[0][1],line[1][0],line[1][1]
            if x>=x1 and x<=x2 or x<=x1 and x>=x2: # якщо x в межах [x1,x2]
                results.append((x-x1)*(y2-y1)/(x2-x1)+y1) # знайти точку
перетину x з лінією
        return results
KB[r"""Base\Залежність"""]=X; del X

class X(KB[r"""Base"""]):
    def __init__(self):
        KB[r"""Base"""].__init__(self)
        # властивість 'параметр'
        Property(subj=self,name='parameter')
KB[r"""Base\Модель"""]=X; del X

class X(KB[r"""Base\Модель"""]):
    def __init__(self):
        KB[r"""Base\Модель"""].__init__(self)
    def create(self):
        "цю функцію необхідно викликати після створення об'єктів КВ"

        self.d={ # словник геометричних параметрів
        'd_n':"зовнішній діаметр різьби ніпеля",
        'd2_n':"середній діаметр різьби ніпеля",
        'd1_n':"внутрішній діаметр різьби ніпеля",
        'r_n':"радіус западин різьби ніпеля",
        'dn':"діаметр бурта ніпеля",
        'd1n':"діаметр зарізьбової канавки ніпеля",
        'l1n':"довжина ніпеля",
        'l2n':"довжина зарізьбової канавки ніпеля",
        'l3n':"довжина ніпеля без фаски на різьбі",
        'l4n':"довжина ніпеля з буртом",
        'r3n':"радіус скруглень зарізьбової канавки ніпеля",
        'd_m':"зовнішній діаметр різьби муфти",
        'd2_m':"середній діаметр різьби муфти",
        'd1_m':"внутрішній діаметр різьби муфти",
        'dm':"зовнішній діаметр муфти",
        'd1m':"внутрішній діаметр опорної поверхні муфти",
        'lm':"довжина муфти",
        'd0':"діаметр тіла штанги",
        'p_n':"крок різьби ніпеля",
        'p_m':"крок різьби муфти"
        }

        self.p={ # словник інших параметрів
        'delta_ln':"величина збільшення довжини ніпеля",
        'material1':"назва матеріалу з бібліотеки матеріалів",
        'material2':"назва матеріалу з бібліотеки матеріалів",
        'bolt_load':"осьова деформація муфти під час згвинчування (мм)",
```

```
        'sigma1':"напруження в тілі штанги для кроку 1 (Па)",
        'sigma2':"напруження в тілі штанги для кроку 2 (Па)"
        }

    def createAbaqusModel(self):
        """Створює модель Abaqus. Створює тимчасовий файл з даними для
передачі їх скрипту Abaqus. Виконує скрипт в Abaqus"""
        import os,pickle,tempfile,subprocess
        path=os.path.join(os.getcwd(), self.name) # шлях до каталогу
        data={'d':self.d,'p':self.p,'path':path}
        name=os.path.join(tempfile.gettempdir(),"data4AbaqusScript.tmp")
        f=open(name, "wb") # відкрити бінариний файл для запису
        pickle.dump(data,f) # законсервувати дані у файлі
        f.close() # закрити файл

        print "Abaqus CAE started. Please wait"
        # виконує скрипт в Abaqus та чекає завершення
        subprocess.Popen(r'C:\SIMULIA\Abaqus\6.11-3\exec\abq6113.exe cae
noGUI=gost13877_96AbaqusMain.py').communicate()
        #os.system(r'start /WAIT abaqus cae noGUI=gost13877_96Abaqus.py')
        print "Abaqus CAE finished"
KB[r"""Base\Модель\ГОСТ 13877-96"""]=X; del X

class X(KB[r"""Base\Модель\ГОСТ 13877-96"""]):
    def __init__(self):
        KB[r"""Base\Модель\ГОСТ 13877-96"""].__init__(self)
    def create(self):
        "цю функцію необхідно викликати після створення об'єктів КВ"
        KB[r"""Base\Модель\ГОСТ 13877-96"""].create(self)

        d={
        'd_n':(27, -0.48, -0.376),
        'd2_n':(25.35, -0.204, -0.047),
        'd1_n':(24.25, 0, -0.415),
        'r_n':(0.28, 0, 0.08),
        'dn':(38.1, -0.25, 0.13),
        'd1n':(23.24, -0.13, 0.13),
        'l1n':(36.5, 0, 1.6),
        'l2n':(15, 0.2, 1),
        'l3n':(32, 0, 1.5),
        'l4n':(48, -1, 1.5),
        'r3n':(3, 0, 0.8),
        'd_m':(27, 0, 0.27),
        'd2_m':(25.35, 0, 0.202),
        'd1_m':(24.25, 0, 0.54),
        'dm':(41.3, -0.25, 0.13),
        'd1m':(27.43, 0, 0.25),
        'lm':(102, -1, 1),
        'd0':(19.1,-0.41,0.2),
        'p_n':(2.54,0,0),
        'p_m':(2.54,0,0)
```

```
        }

        for k in d:
            self.d[k]=[self.d[k]]+list(d[k])

        p={
        'delta_ln':0,
        'material1':"40fesafe",
        'material2':"40fesafe",
        'bolt_load':-0.1,
        'sigma1':1,
        'sigma2':170.0e+6
        }

        for k in p:
            self.p[k]=[self.p[k]]+[p[k]]
KB[r"""Base\Модель\ГОСТ 13877-96\ШН 19"""]=X; del X

class X(KB[r"""Base"""]):
    "Клас описує поняття розміру"
    def __init__(self):
        KB[r"""Base"""].__init__(self)
    def create(self,doc,n,ei,es,v):
        "створює об'єкт"
        self.doc=doc # довідка
        self.n=n # номінальний розмір
        self.ei=ei # нижнє відхилення
        self.es=es # верхнє відхилення
        self.v=v # дійсне значення
    def min(self):
        "повертає мінімальний розмір"
        return self.n+self.ei
    def max(self):
        "повертає максимальний розмір"
        return self.n+self.es
KB[r"""Base\Параметр"""]=X; del X

class X(KB[r"""Base"""]):
    '''Клас, який описує факт (триплет) у вигляді
    суб'єкт-предикат-об'єкт'''
    def __init__(self): # конструктор
        KB[r"""Base"""].__init__(self)
        # властивість 'має посилання'
        Property(subj=self,name='hasReference',inverseName='isReference')
        # властивість 'має залежність'
        Property(subj=self,name='hasDependence',inverseName='isDependence')
    def create(self,subjName,propName,objName):
        self.subjName=subjName # назва суб'єкта
        self.propName=propName # назва предиката (властивість)
        self.objName=objName # назва об'єкта
        # додати значення в властивість, якщо немає
```

```
        KB[self.subjName].__dict__[self.propName].add(KB[self.objName])
KB[r"""Base\Факт"""]=X; del X

class X(KB[r"""Base"""]): # успадковує клас Base
    '''Клас, який описує фактор'''
    def __init__(self): # конструктор
        KB[r"""Base"""].__init__(self)
        # транзитивна властивість 'є причиною'

Property(subj=self,name='isCause',inverseName='isEffect',transitive=True)
        # транзитивна властивість 'є наслідком'

Property(subj=self,name='isEffect',inverseName='isCause',transitive=True)
KB[r"""Base\Фактор"""]=X; del X

############################
KB[r"""Base\Джерело\И.А.Биргер, Г.Б.Иосилевич. Резьбовые и фланцевые
соединения"""]=KB[r"""Base\Джерело"""]()

KB[r"""Base\Модель\ГОСТ 13877-96\ШН
19\bolt_load\0.1"""]=KB[r"""Base\Модель\ГОСТ 13877-96\ШН 19"""]()

KB[r"""Base\Модель\ГОСТ 13877-96\ШН
19\bolt_load\0.1\delta_ln\10"""]=KB[r"""Base\Модель\ГОСТ 13877-96\ШН
19"""]()

KB[r"""Base\Фактор\Концентрація напружень"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Міцність"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Міцність\За циклічного
навантаження"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Міцність\За циклічного
навантаження\Низька"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Технологія"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Технологія\Технологічні
дефекти"""]=KB[r"""Base\Фактор"""]()

KB[r"""Base\Фактор\Концентрація
напружень"""].__dict__["isCause"].add(KB[r"""Base\Фактор\Міцність\За
циклічного навантаження\Низька"""])

KB[r"""Base\Фактор\Технологія\Технологічні
дефекти"""].__dict__["isCause"].add(KB[r"""Base\Фактор\Концентрація
напружень"""])
############################

# правила виведення
```

```
def allFactsSet(showTransitive=True):
    '''Повертає множину всіх фактів (аксіом) бази знань.
    Факти у вигляді кортежу (суб'єкт, предикат, об'єкт)'''
    factsSet=set() # множина фактів
    for k in KB.values(): # для усіх концептів
        if k.__class__.__name__==r"""Base\Фактор""": # якщо це фактор
            for p in k.__dict__: # для усіх атрибутів
                if k.__dict__[p].__class__.__name__=='Property': # якщо
атрибут властивість
                    for obj in k.__dict__[p].get(showTransitive): # для
усіх значень властивості
                        factsSet.add((k,p,obj)) # додати факт в множину
фактів
    return factsSet

for k,v in KB.iteritems():
    v.name=k # для всіх об'єктів в KB задає атрибут name і присвоює йому
значення ключа в KB
    v.codes={} # властивість codes - словник вихідних кодів
    if v.__class__.__name__=='type': # якщо це клас
        v.__name__=k # для всіх класів в KB присвоює __name__ значення
ключа в KB

assertedFacts=allFactsSet(False) # введені факти
print len(assertedFacts)
import csv
writer = csv.writer(open("assertedFacts.csv", "wb"),delimiter = ';') #
відкрити csv файл
for x in assertedFacts:
    writer.writerow([x[0].name,x[1],x[2].name]) # записати в файл csv

while True: # цикл для застосування правил
    beforeFacts=allFactsSet() # кількість фактів до
    print str(len(beforeFacts))+' facts. Iteration for apply rules...'

    # логічне виведення для інверсних і симетричних властивостей
    for k in KB.values(): # для усіх концептів
        for p,pv in k.__dict__.iteritems(): # для усіх атрибутів
            if pv.__class__.__name__=='Property': # якщо атрибут
властивість
                for obj in pv.get(True): # для усіх значень властивості

                    if pv.inverseName!='': # якщо є інверсна властивість
                        # додати його в інверсну властивість об'єкта
                        obj.__dict__[pv.inverseName].add(pv.subj)

                    if pv.symmetric: # якщо властивість симетрична
                        # додати його в аналогічну властивість об'єкта
                        obj.__dict__[pv.name].add(pv.subj)

    # правила виведення:
```

```
    # SubClassOf(?x, ?y) & isCause(?y, ?p) -> isCause(?x, ?p)
    # SubClassOf(?x, ?y) & isEffect(?y, ?p) -> isEffect(?x, ?p)
    for k in KB.values(): # для усіх концептів
        if k.__class__.__name__==r"""Base\Фактор""": # якщо це фактор
            for bk in k.SubClassOf.get(True): # для усіх базових концептів
(класів)
                for x in bk.isCause.get(True):
                    k.isCause.add(x)
                for x in bk.isEffect.get(True):
                    k.isEffect.add(x)

    # правила виведення:
    # Not(?x, ?nx) & isCause(?nx, ?ny) & Not(?y, ?ny) -> isCause(?x, ?y)
    # Not(?x, ?nx) & isEffect(?nx, ?ny) & Not(?y, ?ny) -> isEffect(?x, ?y)
    for k in KB.values(): # для усіх концептів
        if k.__class__.__name__==r"""Base\Фактор""": # якщо це фактор
            for nk in k.Not.get(): # для усіх заперечень концепту
                for e in nk.isCause.get(True): # для усіх наслідків
заперечення
                    for ne in e.Not.get(): # для усіх заперечень наслідків
заперечення
                        k.isCause.add(ne) # концепт є їх причиною
                for e in nk.isEffect.get(True): # для усіх причин
заперечення
                    for ne in e.Not.get(): # для усіх заперечень причин
заперечення
                        k.isEffect.add(ne) # концепт є їх наслідком

    afterFacts=allFactsSet() # кількість фактів після
    # перервати цикл, якщо кількість фактів до і після застосування правил
рівна
    if beforeFacts==afterFacts: break

allFacts=allFactsSet()  # усі факти
print len(allFacts)
import csv
writer = csv.writer(open("allFacts.csv", "wb"),delimiter = ';') # відкрити
csv файл
for x in allFacts:
    writer.writerow([x[0].name,x[1],x[2].name]) # записати в файл csv

# блок запитів до бази знань
print "Відповідь на запит:"
for x in KB[r"""Base\Фактор\Міцність\За циклічного
навантаження\Низька"""].isEffect.get(True):
        print x.name
```

ДОДАТОК Ц

Приклад PLM системи різьбових зєднань

Код доступний також в GitHub (vkopey/ThreadsPLM. URL: http://github.com/vkopey/ThreadsPLM). Вміст пакету:

- main.py – головний модуль,
- tools.py – утіліти,

класи:

- class_Agent.py,
- class_Property.py,
- class_Datalog.py,
- class_Dependence.py,
- class_DependenceMulti.py,
- class_Fact.py,
- class_Factor.py,
- class_Model.py,
- class_Parameter.py,

агенти:

- Datalog_inference.py,
- Dependence_dependence1.py,
- Dependence_dependence2.py,
- Dependence_dependence3.py,
- DependenceMulti_dependence4.py,
- Fact_конц_ напр_ спричинює зменшення цикл_ міцності.py,
- Factor_відсутність корозійного пошкодження.py,
- Factor_збільшення довжини зарізьбової канавки.py,
- Factor_збільшення радіуса скруглень зарізьбової канавки.py,
- Factor_збільшення циклічної міцності.py,
- Factor_зменшення циклічної міцності.py,

- Factor_концентрація напружень.py,
- Factor_корозійна язва.py,
- Factor_корозійне пошкодження.py,
- Model_ШН19ГОСТ13877-96.py,
- Model_ШН19ГОСТ13877-96_r3n=2.5.py,
- Parameter_довжина зарізьбової канавки.py,
- Parameter_коефіцієнт запасу втомної міцності.py,
- Parameter_логарифм циклічної довговічності.py,
- Parameter_радіус скруглень зарізьбової канавки.py,
- Property_factorHigh.py,
- Property_factorLow.py,
- Property_isCause.py,
- Property_isContraryOf.py,
- Property_isEffect.py,
- Property_objecT.py,
- Property_source.py,
- Property_subClassOf.py,
- Property_subject.py,
- Property_Xpar.py,
- Property_Ypar.py.

Лістинг Ц.1 – main.py

```
#-*- coding: utf-8 -*-
"Це приклад роботи з системою"
from tools import *

files=getFiles()
createClasses(files)
createKB(files)
#loadKB()

applyRules(n=10)
#saveKB()
```

Лістинг Ц.2 – tools.py

```
#-*- coding: utf-8 -*-
import sys,os,re
import numpy as np
KB={}
systemencoding=sys.getfilesystemencoding() #'mbcs'

def getFiles(ex={'main.py','tools.py'} ):
    "Повертає список файлів поточного каталогу в довільному порядку"
    files=os.listdir(os.getcwdu())
    files=[f for f in files if f[-3:]=='.py' and f not in ex]
    #print 'files:',files
    return files

def createClasses(files):
    files=[f for f in files if f.startswith('class_')]
    while files:
        f=files.pop(0)
        try:
            execfile(f.encode(systemencoding), globals()) #KB
        except NameError:
             files.append(f)

def createKB(files):
    "Створює систему агентів з модулів files"
    files=[f for f in files if not f.startswith('class_')]
    for f in files:
        cls,_,name=f.partition('_')
        name=name[:-3].encode('utf-8')
        #if name in KB: continue
        obj=globals()[cls](name)
        obj.__dict__['KB']=KB
        KB[obj.__name__]=obj
        execfile(f.encode(systemencoding), obj.__dict__)

def getClassObjects(clsName):
    return [k for k in KB if KB[k].__class__.__name__==clsName]

def KBToPropFacts(prop):
    "Повертає усі факти-триплети KB з властивістю prop (prop має бути set)"
    facts=set()
    for k in KB:
        if hasattr(KB[k], prop):
            if KB[prop].FunctionalProperty:
                facts.add((k, prop, KB[k].__dict__[prop]))
            else:
                for v in KB[k].__dict__[prop]:
                    facts.add((k, prop, v))
    return facts
```

```
def KBToFacts():
    "Повертає усі факти-триплети KB"
    facts=set()
    props=getClassObjects('Property')
    for p in props:
        facts.update(KBToPropFacts(p))
    return facts

def factToKB(subj,pred,obj):
    "Оновлює значення властивості об`єкта за триплетом"
    #потрібна також перевірка на Domain і Range!
    if not(KB.has_key(pred) and KB[pred].__class__==Property): return
#Property in KB[pred].__class__.__mro__
    if KB.has_key(subj) and hasattr(KB[subj],pred):
        if KB[pred].FunctionalProperty:
            KB[subj].__dict__[pred]=obj
        else:
            KB[subj].__dict__[pred].add(obj)

def factsToKB(facts):
    "Оновлює всю KB за списком фактів-триплетів"
    for s,p,o in facts:
        factToKB(s,p,o)

def runDatalog(facts, rules, predicates):
    """виконує логічне виведення в pyDatalog. Повертає список триплетів.
    facts - список Datalog-фактів,
    rules - список Datalog-правил,
    predicates - список предикатів, для яких будуть шукатись факти"""

    #if not predicates:
    #    predicates={p for s,p,o in facts}|{r.split('(')[0] for r in rules}

    from pyDatalog.pyDatalog import assert_fact, load, ask, clear
    code='\n'.join(facts)+'\n'+'\n'.join(rules) # факти і правила
    load(code)
    allFacts=set()
    for pred in predicates:
        # try:
        res=ask('%s(X,Y)'%pred).answers
        # except AttributeError: #Predicate without definition
        #     continue
        for subj,obj in res:
            allFacts.add((subj.encode('utf-8'),pred.encode('utf-
8'),obj.encode('utf-8')))
            print subj,pred,obj
    clear()
    return allFacts

def subClasses(n, s=set()):
    "Множина усіх підоб’єктів об’єкта n. Рекурсивна."
```

```
    for k in KB:
        if n in KB[k].subClassOf:
            s.add(k)
            s.update(subClasses(k, s))
    return s

def setAtrSubClasses(name, attr, value, ch=False):
    "Установлює значення value усім атрибутам attr усіх підкласів об'єкта
name, якщо атрибута немає"
    for k in subClasses(name,set()):
        if not hasattr(KB[k],attr):
            setattr(KB[k],attr,value)
            ch=True
    return ch

def applyRules(lst=KB,n=5):
    "Застосовує правила до кожного агента з lst n раз, поки є зміни"
    ch=True
    while ch and n>0: # поки правила створюють зміни
        ch=False
        print '*ApplyRules*'
        for k in lst.keys():
            if KB[k].active and KB[k].rule(): ch=True
        n-=1

def saveKB(lst=KB,new=True):
    import shelve
    from dill import Pickler
    shelve.Pickler = Pickler
    KBp=shelve.open("shelve.dat")
    if new: KBp.clear()
    for k in lst:
        KBp[k]=KB[k]
    KBp.close()

def loadKB():
    import shelve
    from dill import Unpickler
    shelve.Unpickler = Unpickler
    KBp=shelve.open("shelve.dat")
    for k in KBp:
        KB[k]=KBp[k]
    KBp.close()

def find(regexp=".*", lst=KB):
    """Повертає список назв, які відповідають регулярному виразу"""
    res=[]
    po=re.compile(regexp, re.IGNORECASE | re.UNICODE)
    for k in lst:
        mo=po.search(k.decode('utf-8'))
        if mo:
```

```
            res.append(k)
            print k
    res.sort() # сортувати
    return res

def createFile(cls,name):
    fname=cls+'_'+name+'.py'
    fname=fname.decode('utf-8') # в Юнікод
    if os.path.exists(fname):
        print 'Error! File exists'
        return
    t='''#-*- coding: utf-8 -*-
"""%s_%s"""
'''%(cls,name)
    with open(fname,'w') as f:
        f.write(t)
    return fname

def autoKey(start):
    """Генерує унікальний ключ для KB шляхом додавання цілого числа до
start"""
    k=start+'0'
    n=0
    while k in KB.keys():
        n+=1
        k=start+str(n)
    return k

def isSibling(a,b):
    "Словники рівні або відрізняється знач. не більше ніж одного парам."
    if a==b: return True
    if set(a.keys()) ^ set(b.keys()): return False # різниця ключів
    d=set(a.items()) ^ set(b.items()) # різниця
    return len(d)==2

def query1():
    for v in KB["концентрація напружень"].isEffect:
        print v
    #print KB["isCause"].__doc__

def query2():
    import matplotlib.pyplot as plt
    d=KB['dependence2']
    print d.linregress()
    d.plot(plt)

def query3():
    import networkx as nx
    G = nx.DiGraph()
    for i in KB:
        if KB[i].__class__.__name__=='Factor':
```

```
            for j in KB[i].isCause:
                G.add_edge(i.decode('utf-8'), j.decode('utf-8'),
label='isCause')

    pr=nx.pagerank(G) # for DiGraph only
    pr={k:'%s\n%f'%(k,v) for k,v in pr.iteritems()}
    G=nx.relabel_nodes(G, pr)
    nx.drawing.nx_agraph.write_dot(G,'graph.dot')
    os.system(r'"d:\Program Files\Graphviz2.38\bin\dot.exe" -Tsvg graph.dot
-o graph.svg') # конвертуємо в SVG
    #nx.write_graphml_lxml(G, "graph.graphml")
##
def query4():
    def proToEdges(pro):
        if KB[pro].FunctionalProperty:
            G.add_edge(i.decode('utf-8'), KB[i].__dict__[pro].decode('utf-
8'), label=pro)
            return
        for j in KB[i].__dict__[pro]:
            G.add_edge(i.decode('utf-8'), j.decode('utf-8'), label=pro)

    import networkx as nx
    G = nx.MultiDiGraph()
    for i in KB:
        if KB[i].__class__.__name__=='Factor':
            proToEdges('isCause')
            proToEdges('isEffect')
            proToEdges('subClassOf')
            proToEdges('isContraryOf')
        if KB[i].__class__.__name__=='Fact':
            proToEdges('objecT')
            proToEdges('subject')
        if KB[i].__class__.__name__=='Parameter':
            proToEdges('factorHigh')
            proToEdges('factorLow')
        if KB[i].__class__.__name__=='Dependence':
            proToEdges('Xpar')
            proToEdges('Ypar')
            proToEdges('source')
    nx.drawing.nx_agraph.write_dot(G,'graph.dot')
    replaceLongStr('graph.dot')
    os.system(r'"d:\Program Files\Graphviz2.38\bin\dot.exe" -Tsvg graph.dot
-o graph.svg') # конвертуємо в SVG
##
def replaceLongStr(file='graph.dot'):
    def repl(mo):
        s=mo.group(1) # рядок знайденої групи
        n = 20 # кількість символів у рядку
        s=r'\n'.join([s[i:i+n] for i in range(0, len(s), n)])
        return s
    import re
```

```
    all=open(file,'r').read()
    po=re.compile(r'(".*?")', re.UNICODE|re.DOTALL)
    all=re.sub(po, repl, all.decode('utf-8'))
    f=open(file,'w')
    f.write(all.encode('utf-8'))
    f.close()
```

Лістинг Ц.3 – class_Agent.py

```
#-*- coding: utf-8 -*-
class Agent(object):
    "Інтелектуальний агент"
    def __init__(self, name):
        self.__name__=name
        self.active=True

    def rule(self):
        "Правило поведінки агента. Повертає 1, якщо його застосування
призвело до змін"
        return False #у іншому випадку повертає False або нічого не
повертає
```

Лістинг Ц.4 – class_Property.py

```
#-*- coding: utf-8 -*-
class Property(Agent):
    """Property. Базовий клас властивостей"""
    def __init__(self, name):
        super(Property,self).__init__(name)
        self.inverseOf=None
        self.domain=set()
        self.range=set()
        self.SymmetricProperty=False
        self.TransitiveProperty=False
        self.FunctionalProperty=False

    def datalogRules(self):
        r=set()
        if self.inverseOf: r.add('%s(X,Y) <= %s(Y,X)'%(self.inverseOf,
self.__name__))
        if self.SymmetricProperty: r.add('%s(X,Y) <=
%s(Y,X)'%(self.__name__, self.__name__))
        return r
```

Лістинг Ц.5 – class_Datalog.py

```
#-*- coding: utf-8 -*-
class Datalog(Agent):
    """Datalog."""
    def __init__(self, name):
        super(Datalog,self).__init__(name)
```

```
        self.oldfacts=set()

    def getDatalogFacts(self,facts):
        "Повертає список datalog-фактів зі списку фактів-триплетів"
        dfacts=[]
        for k, p, v in facts:
            df='+%s(u"%s",u"%s")'%(p,k.decode('utf-8'),v.decode('utf-8'))
            dfacts.append(df)
        return dfacts

    def getDatalogRules(self):
        r=set()
        for k in KB:
            if hasattr(KB[k],'datalogRules'):
                r.update(KB[k].datalogRules())
        return r

    def rule(self):
        props=getClassObjects('Property')
        facts=KBToFacts()
        dfacts=self.getDatalogFacts(facts)
        drules=self.getDatalogRules()
        allFacts=runDatalog(dfacts, drules, props)
        factsToKB(allFacts)
        n=allFacts-self.oldfacts
        self.oldfacts=allFacts
        return len(n)
```

Лістинг Ц.6 – class_Dependence.py

```
#-*- coding: utf-8 -*-
class Dependence(Agent):
    """Dependence. Базовий клас статистичних залежностей"""
    def __init__(self, name):
        super(Dependence,self).__init__(name)
        self.Xpar=None # незалежна змінна
        self.Ypar=None # залежна змінна
        self.X=[]
        self.Y=[]
        self.source=None

    def plot(self,plt):
        try:
            a=self.linregress()
            plt.plot(self.X,self.Y,'ko')
            x=np.array([min(self.X),max(self.X)])
            y=a[0]*x+a[1]
            plt.plot(x,y,'k-')
            plt.xlabel('X')
            plt.ylabel('Y')
            plt.show()
```

```
22          except:
23              pass
24
25      def linregress(self):
26          from scipy import stats
27          return stats.linregress(self.X, self.Y)
28
29      def fromModels(self): # повертає точки з моделей
30          if KB.get(self.source).__class__.__name__!='Model': return
31          models=getClassObjects('Model')
32          for k in models:
33              m=KB[k]
34              if not m.isSibling(self.source): continue
35              if self.Xpar not in m.paramsIn: continue
36              if self.Ypar not in m.paramsOut: continue
37              y=m.paramsOut[self.Ypar]
38              if y==None: continue
39              x=m.paramsIn[self.Xpar]
40              try:
41                  i=self.X.index(x)
42                  self.Y[i]=y # додати значення Y
43              except ValueError:
44                  self.X.append(x) # додати точку
45                  self.Y.append(y)
46          XY=zip(self.X, self.Y)
47          XY.sort()
48          self.X=[x for x,y in XY]
49          self.Y=[y for x,y in XY]
50
51      def modelExist(self, x): # модель зі значенням x існує?
52          models=getClassObjects('Model')
53          for m in models:
54              if KB[m].paramsIn.get(self.Xpar)==x:
55                  return True
56          return False
57
58      def toModels(self): # динамічно створити моделі, якщо y=None
59          if KB.get(self.source).__class__.__name__!='Model': return
60          for x,y in zip(self.X,self.Y):
61              if y!=None or self.modelExist(x): continue
62              k=autoKey("model")
63              KB[k]=KB[self.source].__class__(k)
64              KB[k].paramsIn[self.Xpar]=x
65              print 'Model created'
66
67      def rule(self):
68          self.fromModels()
69          self.toModels()
70          if None in self.Y: return
71          if not len(self.Y)>1: return
72          slope, intercept, r_value, p_value, std_err=self.linregress()
```

```
73          #print slope, intercept, r_value, p_value, std_err
74          if r_value**2<0.5: return False
75
76          try:
77              if slope>0: # додатна кореляція
78                  KB[KB[self.Xpar].factorHigh].isCause.add(
                        KB[self.Ypar].factorHigh)
79              else:
80                  KB[KB[self.Xpar].factorLow].isCause.add(
                        KB[self.Ypar].factorHigh)
81          except KeyError:
82              pass
```

Лістинг Ц.7 – class_DependenceMulti.py

```
#-*- coding: utf-8 -*-
class DependenceMulti(Agent):
    """DependenceMulti"""
    def __init__(self, name):
        super(DependenceMulti,self).__init__(name)
        self.Xpars=None # незалежні змінні
        self.Ypars=None # залежні змінні
        self.X=[]
        self.Y=[]
        self.source=None

    def toDataFrame(self):
        import pandas as pd
        XY=self.X+self.Y
        return pd.DataFrame(data = zip(*XY), columns=self.Xpars+self.Ypars)

    def fromDataFrame(self,df):
        for i,xp in enumerate(self.Xpars):
            self.X[i]=[x for x in df[xp]]
        for i,yp in enumerate(self.Ypars):
            self.Y[i]=[y for y in df[yp]]

    def linregress(self, yp): # лінійна регресія для параметра yp
        X=np.array(self.X).T
        Y=self.Y[self.Ypars.index(yp)]
        from sklearn import linear_model
        reg = linear_model.LinearRegression()
        reg.fit(X, Y)
        return reg.coef_, reg.score(X, Y)

    def byModels(self):
        if KB.get(self.source).__class__.__name__!='Model': return
        df=self.toDataFrame()
        model=KB[self.source].__class__('tmp')
        print 'Temporary model created'
```

```
        for i in df.index: # for all points
            for xp in self.Xpars:
                model.paramsIn[xp]=df[xp][i]
                print xp,'=',df[xp][i]
            for yp in self.Ypars:
                model.paramsOut[yp]=None
            model.rule()
            for yp in self.Ypars:
                df[yp][i]=model.paramsOut[yp]
                print yp,'=',df[yp][i]
        self.fromDataFrame(df)

    def simModel(self,x,yp): # симуляція моделі
        model=KB[self.source].__class__('tmp')
        print 'Temporary model created', x
        for i,xp in enumerate(self.Xpars):
            model.paramsIn[xp]=x[i]
        model.rule()
        print 'y=',model.paramsOut[yp]
        return model.paramsOut[yp]

    def optimize(self,yp="коефіцієнт запасу втомної міцності",
bounds=[(2.5,3.5),(12.7, 13.26)], maximize=True):
        from scipy.optimize import differential_evolution # мінімізація
        s = -1 if maximize else 1
        #res=minimize(lambda x,yp: s*self.simModel(x,yp), x0=[sum(p)/2.0
for p in bounds], args=(yp,), method="L-BFGS-B", bounds=bounds,
options=dict(maxfun=10,eps=0.1))
        res=differential_evolution(lambda x,yp: s*self.simModel(x,yp),
args=(yp,), bounds=bounds, maxiter=5, popsize=5, tol=0.1)
        return res

    def rule(self):
        if None not in [item for sublist in self.Y for item in sublist]:
return
        self.byModels()
```

Лістинг Ц.8 – class_Fact.py

```
#-*- coding: utf-8 -*-
class Fact(Agent):
    """Fact. Базовий клас фактів"""
    def __init__(self, name):
        super(Fact,self).__init__(name)
        self.subject=None
        self.predicate=None
        self.objecT=None
        self.sources=set()
    def rule(self):
        for k in KB:
            if k!=self.predicate: continue
```

```
            if KB[k].__class__.__name__!='Property': continue
            if self.subject not in KB[k].domain: continue
            if self.objecT not in KB[k].range: continue
            try:
                if KB[k].FunctionalProperty:
                    KB[self.subject].__dict__[k]=self.objecT
                else:
                    KB[self.subject].__dict__[k].add(self.objecT)
                break
            except KeyError:
                pass
```

Лістинг Ц.9 – class_Factor.py

```
1  #-*- coding: utf-8 -*-
2  class Factor(Agent):
3      """Factor. Базовий клас факторів"""
4      def __init__(self, name):
5          super(Factor,self).__init__(name)
6          self.subClassOf=set()
7          self.isCause=set()
8          self.isEffect=set()
9          self.isContraryOf=set()
10         # 'subClassOf(O,B) :- subClassOf(A,B),subClassOf(O,A)'
11     # def rule(self):
12     #     "Правило установлює атрибути в підкласах, якщо їх немає"
13     #     from tools import setAtrSubClasses
14     #     return any( [setAtrSubClasses(__name__,"isCause",set()),
15     #                  setAtrSubClasses(__name__,"isEffect",set()) ])
16
17     # def rule(self):
18     #     "Тільк для тестування прямого виведення. Ви можете
   протестувати цей алгоритм шляхом виключення агентів Datalog
   (active=False)."
19     #     for k in KB:
20     #         if KB[k].__class__.__name__!='Factor': continue
21     #         if k not in self.isCause: continue
22     #         if self.__name__ not in KB[k].__dict__['isEffect']:
23     #             KB[k].__dict__['isEffect'].add(self.__name__)
24     #             return True
```

Лістинг Ц.10 – class_Model.py

```
#-*- coding: utf-8 -*-
class Model(Agent):
    """Model.ШН19 ГОСТ 13877-96"""
    def __init__(self, name):
        super(Model,self).__init__(name)
        self.paramsIn={"радіус скруглень зарізьбової канавки":3.5,
                       "зовнішній радіус різьби ніпеля":13.26}
```

```
        self.paramsOut={'коефіцієнт запасу втомної міцності':None}
    def parseOutput(self,s,t):
        "Читає результат у змінну t: parseOutput(s,t='FOS')"
        t=t+'='
        for r in s.splitlines():
            if r.startswith(t):
                return float(r.partition(t)[2])
    def isSibling(self,model):
        "Моделі відрізняється знач. не більше ніж одного парам."
        a=self.paramsIn
        b=KB[model].paramsIn
        return isSibling(a,b)
    def rule(self):
        if None not in self.paramsOut.values(): return
        from subprocess import check_output
        p=r"e:\Anaconda2\python.exe"
        m=r"e:\!Kopey_Documents\Python_projects\ThreadsOCC\main.py"
        cnvName={"радіус скруглень зарізьбової канавки":"r3n",
                 "зовнішній радіус різьби ніпеля":"d_n",
                 "коефіцієнт запасу втомної міцності":"FOS"}
        args=[cnvName[k]+'='+str(v) for k,v in self.paramsIn.iteritems()]
        s=check_output([p, m]+args, shell=True)
        for k in self.paramsOut:
            self.paramsOut[k]=self.parseOutput(s,cnvName[k])
```

Лістинг Ц.11 – class_Parameter.py

```
#-*- coding: utf-8 -*-
class Parameter(Agent):
    """Parameter. Базовий клас ..."""
    def __init__(self, name):
        super(Parameter,self).__init__(name)
        self.factorHigh=None
        self.factorLow=None
```

Лістинг Ц.12 – Datalog_inference.py

```
#-*- coding: utf-8 -*-
"""inference. Логічне виведення за допомогою Datalog"""
```

Лістинг Ц.13 – Dependence_dependence1.py

```
#-*- coding: utf-8 -*-
"""dependence1 ШН19 ГОСТ 13877-96"""
Xpar="довжина зарізьбової канавки"
Ypar="логарифм циклічної довговічності"
X=[15.0, 25.0, 35.0, 45.0]
Y=[4.3, 5.4, 6.1, 7.0]
source="Копей В.Б. Технологічний аудит та резерви виробництва. - № 6/2 (8).
- 2012. - С.7-8."
```

Лістинг Ц.14 – Dependence_dependence2.py

```
#-*- coding: utf-8 -*-
"""dependence2 ШН19 ГОСТ 13877-96"""
Xpar="радіус скруглень зарізьбової канавки"
Ypar="коефіцієнт запасу втомної міцності"
X=[0.5, 1.5, 2.5, 3.5]
#Y=[None, None, None, None]
Y=[-41.5474014244, -18.9878897227, -13.7151432304, -11.3376607349]
source="ШН19ГОСТ13877-96"
```

Лістинг Ц.15 – Dependence_dependence3.py

```
#-*- coding: utf-8 -*-
"""dependence3 ШН19 ГОСТ 13877-96"""
Xpar="зовнішній радіус різьби ніпеля"
Ypar="коефіцієнт запасу втомної міцності"
X=[12.7, 12.84, 12.98, 13.12, 13.26]
#Y=[None, None, None, None, None]
Y=[-12.6285730685, -12.1345249073, -12.0526604823, -11.982904361, -
11.9369383163]
source="ШН19ГОСТ13877-96"
```

Лістинг Ц.16 – DependenceMulti_dependence4.py

```
#-*- coding: utf-8 -*-
"""dependence4 ШН19 ГОСТ 13877-96"""
Xpars=["радіус скруглень зарізьбової канавки",
      "зовнішній радіус різьби ніпеля"]
X=[[00.5, 01.5, 02.5, 03.5, 00.50, 01.50, 02.50, 03.50],
   [12.7, 12.7, 12.7, 12.7, 13.26, 13.26, 13.26, 13.26]]
Ypars=["коефіцієнт запасу втомної міцності"]
#Y=[[None,None,None,None,None,None,None,None]]
Y=[[-36.0,-18.1,-14.2,-12.1,-41.5,-19.0,-13.7,-11.3]]
source="ШН19ГОСТ13877-96"
```

Лістинг Ц.17 – Fact_конц_ напр_ спричинює зменшення цикл_ міцності.py

```
#-*- coding: utf-8 -*-
"""Fact.конц_ напр_ спричинює зменшення цикл_ міцності"""
subject="концентрація напружень"
predicate="isCause"
objecT="зменшення циклічної міцності"
```

Лістинг Ц.18 – Factor_відсутність корозійного пошкодження.py

```
#-*- coding: utf-8 -*-
"""відсутність корозійного пошкодження"""
isContraryOf={"корозійне пошкодження"}
```

Лістинг Ц.19 – Factor_збільшення довжини зарізьбової канавки.py

```
#-*- coding: utf-8 -*-
"""збільшення довжини зарізьбової канавки"""
```

Лістинг Ц.20 – Factor_збільшення радіуса скруглень зарізьбової канавки.py

```
#-*- coding: utf-8 -*-
"""збільшення радіуса скруглень зарізьбової канавки"""
```

Лістинг Ц.21 – Factor_збільшення циклічної міцності.py

```
#-*- coding: utf-8 -*-
"""збільшення циклічної міцності"""
isContraryOf={'зменшення циклічної міцності'}
```

Лістинг Ц.22 – Factor_зменшення циклічної міцності.py

```
#-*- coding: utf-8 -*-
"""зменшення циклічної міцності"""
isEffect={'концентрація напружень'}
```

Лістинг Ц.23 – Factor_концентрація напружень.py

```
#-*- coding: utf-8 -*-
"""концентрація напружень"""
```

Лістинг Ц.24 – Factor_корозійна язва.py

```
#-*- coding: utf-8 -*-
"""корозійна язва"""
subClassOf={"корозійне пошкодження"}
```

Лістинг Ц.25 – Factor_корозійне пошкодження.py

```
#-*- coding: utf-8 -*-
"""корозійне пошкодження"""
isCause={"концентрація напружень"}
```

Лістинг Ц.26 – Model_ШН19ГОСТ13877-96.py

```
#-*- coding: utf-8 -*-
"""Model ШН19 ГОСТ 13877-96"""
```

Лістинг Ц.27 – Model_ШН19ГОСТ13877-96_r3n=2.5.py

```
#-*- coding: utf-8 -*-
"""Model_ШН19ГОСТ13877-96_r3n=2.5"""
paramsIn["радіус скруглень зарізьбової канавки"]=2.5
#paramsIn['d_n']=13.26
```

Лістинг Ц.28 – Parameter_довжина зарізьбової канавки.py

```
#-*- coding: utf-8 -*-
"""довжина зарізьбової канавки ШН19 ГОСТ 13877-96"""
factorHigh="збільшення довжини зарізьбової канавки"
factorLow="зменшення довжини зарізьбової канавки"
```

Лістинг Ц.29 – Parameter_коефіцієнт запасу втомної міцності.py

```
#-*- coding: utf-8 -*-
"""коефіцієнт запасу втомної міцності"""
factorHigh="збільшення циклічної міцності"
factorLow="зменшення циклічної міцності"
```

Лістинг Ц.30 – Parameter_логарифм циклічної довговічності.py

```
#-*- coding: utf-8 -*-
"""логарифм циклічної довговічності"""
factorHigh="збільшення циклічної міцності"
factorLow="зменшення циклічної міцності"
```

Лістинг Ц.31 – Parameter_радіус скруглень зарізьбової канавки.py

```
#-*- coding: utf-8 -*-
"""радіус скруглень зарізьбової канавки ШН19 ГОСТ 13877-96"""
factorHigh="збільшення радіуса скруглень зарізьбової канавки"
factorLow="зменшення радіуса скруглень зарізьбової канавки"
```

Лістинг Ц.32 – Property_factorHigh.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'factorHigh'"""
inverseOf="factorLow"
domain={"Parameter"}
range={"Factor"}
FunctionalProperty=True
```

Лістинг Ц.33 – Property_factorLow.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'factorLow'"""
inverseOf="factorHigh"
domain={"Parameter"}
range={"Factor"}
FunctionalProperty=True
```

Лістинг Ц.34 – Property_isCause.py

```
#-*- coding: utf-8 -*-
"""isCause. Властивість 'є причиною'"""
inverseOf="isEffect"
domain={"Factor"}
range={"Factor"}

datalogRules_=KB['isCause'].datalogRules
def datalogRules():
    r={'isCause(X,Y) <= isCause(B,Y) & subClassOf(X,B)',
       'isEffect(X,Y) <= isEffect(B,Y) & subClassOf(X,B)'}
    return r|datalogRules_()
```

Лістинг Ц.35 – Property_isContraryOf.py

```
#-*- coding: utf-8 -*-
"""isContraryOf. Властивість 'є протилежністю'"""
domain={"Factor"}
range={"Factor"}
def datalogRules():
    r={'isCause(X, Y) <= isContraryOf(X, NX) & isCause(NX, NY) &
isContraryOf(Y, NY)',
       'isEffect(X, Y) <= isContraryOf(X, NX) & isEffect(NX, NY) &
isContraryOf(Y, NY)'}
    return r
```

Лістинг Ц.36 – Property_isEffect.py

```
#-*- coding: utf-8 -*-
"""isEffect. Властивість 'є наслідком'"""
inverseOf="isCause"
domain={"Factor"}
range={"Factor"}
```

Лістинг Ц.37 – Property_objecT.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'objecT'"""
domain={"Fact"}
range={"Factor"}
FunctionalProperty=True
```

Лістинг Ц.38 – Property_source.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'source'"""
domain={"Dependence"}
range={"Model","str"}
FunctionalProperty=True
```

Лістинг Ц.39 – Property_subClassOf.py

```
#-*- coding: utf-8 -*-
"""subClassOf. Властивість 'є підкласом'"""
domain={"Factor"}
range={"Factor"}

# def datalogRules():
#     r={'isCause(X,Y) <= isCause(B,Y), subClassOf(X,B)'}
#     return r
```

Лістинг Ц.40 – Property_subject.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'subject'"""
domain={"Fact"}
range={"Factor"}
FunctionalProperty=True
```

Лістинг Ц.41 – Property_Xpar.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'Xpar'"""
domain={"Dependence"}
range={"Parameter"}
FunctionalProperty=True
```

Лістинг Ц.42 – Property_Ypar.py

```
#-*- coding: utf-8 -*-
"""Property. Властивість 'Ypar'"""
domain={"Dependence"}
range={"Parameter"}
FunctionalProperty=True
```

ДОДАТОК Ч
Акти впровадження результатів роботи

ЗАТВЕРДЖУЮ
Головний інженер
НГВУ "Долинанафтогаз"
_____ Яремко І.Я.
12. 07 2016 р.

АКТ

про впровадження інформаційної системи підтримки життєвого циклу свердловинних штангових насосних установок (СШНУ)

Комісія у складі:

Голова - головний інженер Яремко І.Я.;

Члени комісії – головний механік НГВУ "Долинанафтогаз" Петрів М.В., провідний інженер ЦІТС, к.т.н. Кузьмін О.О., доцент кафедри КМВ ІФНТУНГ, к.т.н. Копей В.Б., завідувач кафедри НГО ІФНТУНГ, д.т.н. Копей Б.В. цим Актом засвідчує, що інформаційна система підтримки життєвого циклу СШНУ, яка розроблена Копеєм В.Б. та Копеєм Б.В., впроваджена в НГВУ "Долинанафтогаз" з метою підвищення якості експлуатації СШНУ та забезпечення її надійності. Нижче перелічені впроваджені компоненти інформаційної системи та основні поточні результати їх впровадження:

1. Головний модуль інформаційної системи та експертна система з проблем надійності і довговічності різьбових з'єднань. Використані для підбору оптимальних методів і засобів забезпечення надійності різьбових з'єднань насосних штанг.

2. Динамічна модель СШНУ. Виявлено відповідність результатів розрахунку моделі та експериментальної динамограми СШНУ. Модель використано для уточнення компонування штангової колони.

3. Параметричні скінченно-елементні моделі різьбових з'єднань нафтового обладнання (насосних штанг, насосно-компресорних труб, замкові

з’єднання) та програмні засоби їх побудови. Шляхом моделювання отримані рекомендації щодо вибору оптимальних моментів згвинчування насосних штанг в залежності від умов їх експлуатації.

4. Параметричні скінченно-елементні моделі з’єднань склопластикового тіла насосної штанги зі сталевим ніпелем. Шляхом моделювання отримані допустимі навантаження на склопластикові насосні штанги з врахуванням особливостей конструкції з’єднання.

5. Параметричні скінченно-елементні моделі протекторів для насосних штанг. Шляхом розрахунку гідродинамічних моделей отримані сили гідродинамічного опору протекторів різної конструкції та подані рекомендації щодо області їх раціонального використання.

Члени комісії

головний механік
НГВУ "Долинанафтогаз" Петрів М.В.
провідний інженер ЦІТС к.т.н. Кузьмін О.О.
доцент кафедри КМВ ІФНТУНГ, к.т.н. Копей В.Б.
завідувач кафедри НГО ІФНТУНГ, д.т.н Копей Б.В.

«12» 07 2016 р.

ЗАТВЕРДЖУЮ
Проректор з наукової роботи
проф. _____ Чудик І. І.
"05" березня 2020 р.

АКТ

Впровадження у навчальний процес результатів дисертаційної роботи Копея Володимира Богдановича на тему "Науково-методологічні основи автоматизованого проектування обладнання штангової свердловинної насосної установки", яка подається на здобуття наукового ступеня доктора технічних наук

Наукові та прикладні результати дисертації доцента кафедри комп'ютеризованого машинобудування Копея В. Б. використовуються для підготовки фахівців освітньо-кваліфікаційного рівня магістр спеціальності 131 "Прикладна механіка" освітньо-професійної програми "Комп'ютеризовані і роботизовані технології машинобудування" під час вивчення дисциплін "Інформаційне забезпечення САПР", "Методи моделювання та симуляції кінематики та динаміки машин" і під час виконання магістерських робіт.

Директор інституту інженерної механіки
канд. техн. наук, професор _____ Л. І. Романишин

Завідувач кафедри
комп'ютеризованого машинобудування,
д-р техн. наук, професор _____ В. Г. Панчук