УДК 004.42:519.876.5

Егоров Валерий Александрович

магистрант, Арзамасский политехнический институт (филиал)
Нижегородского государственного технического университета
им. Р. Е. Алексеева

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ЭКСПЕРИМЕНТА В PYTHON ДЛЯ АНАЛИЗА САНКЦИОННЫХ ПРОЦЕССОВ

В статье представлена программная реализация вычислительного эксперимента для дискретной модели повторяющихся санкций и контрсанкций. Разработана модульная структура Python-приложения, разделяющая описание параметров, детерминированное ядро, стохастический ансамбль, расчет показателей и экспорт результатов. Предложены проверки входных данных, тесты корректности обновления состояний и средства воспроизводимости случайных экспериментов. Векторизация ансамблевого расчета сокращает время моделирования по сравнению с последовательной обработкой реализаций. Контрольные сценарии подтверждают совпадение программной классификации режимов с аналитическим условием устойчивости. Определены требования к структуре воспроизводимого вычислительного комплекса.

Ключевые слова: Python, программная реализация, вычислительный эксперимент, дискретная модель, санкционное взаимодействие, моделирование систем, векторизация, воспроизводимость.

The paper presents a software implementation of a computational experiment for a discrete model of repeated sanctions and counter-sanctions. A modular Python application separates parameter description, the deterministic simulation core, stochastic ensemble generation, indicator calculation, and result export. Input validation, state-update tests, and reproducibility tools for random experiments are proposed. Vectorized ensemble processing reduces simulation time compared with sequential execution. Control scenarios confirm that the software classification of dynamic regimes agrees with the analytical stability condition and that near-boundary

cases require an extended computational horizon. Requirements for the structure of a reproducible computational system are defined.

Key words: Python, software implementation, computational experiment, discrete model, sanctions interaction, system simulation, vectorization, reproducibility.

Введение

Вычислительный эксперимент связывает математическую модель с анализом ее поведения. Даже простая рекуррентная система требует правильного порядка обновления переменных, контроля параметров и фиксации случайных последовательностей. Ошибка реализации способна дать внешне правдоподобный график, поэтому программа должна рассматриваться как самостоятельный объект верификации.

Модель повторяющихся санкций и контрсанкций описывает взаимную реакцию сторон на изменение воздействия оппонента [2, с. 11010–11014]. Развитие моделей Осипова–Ланчестера, включая модификации коэффициентов взаимодействия, рассмотрено в работе Д. А. Новикова [1, с. 135–140]. Для ее исследования необходимы серии детерминированных и стохастических расчетов. Ручная замена параметров в одном скрипте не обеспечивает воспроизводимости и затрудняет сравнение сценариев.

Цель работы — разработать программную реализацию вычислительного эксперимента в Python. Основное внимание уделяется архитектуре приложения, структурам данных, генерации ансамбля, программным тестам и автоматическому сохранению результатов.

Требования к вычислительному комплексу

Программа должна точно воспроизводить перекрестное обновление состояний и различать уровни воздействия с их приращениями. Классификация режима выполняется по аналитическому критерию, а рассчитанная траектория используется для проверки переходного процесса. До запуска контролируются конечность коэффициентов, допустимость стандартных отклонений, число шагов и количество реализаций.

Воспроизводимость обеспечивается сохранением начальных условий, коэффициентов, горизонта, зерна генератора и версий библиотек. Повторный запуск не должен требовать ручного изменения промежуточных массивов. Данные требования соответствуют рекомендациям по организации вычислительных исследований [6, с. 1–4; 8, с. 1–4].

Архитектура должна допускать расширение модели. Детерминированный расчет, случайный ансамбль, показатели и визуализация разделяются. Это позволяет добавить управление, секторные ограничения или другую модель шума без изменения проверенного базового ядра.

Структура программной реализации

Вычислительный комплекс содержит модули model, simulation, metrics, io и visualization. Первый хранит параметры и функцию перехода, второй выполняет расчеты, третий формирует показатели, четвертый сохраняет результаты, пятый строит графики. Файл сценария задает условия опыта, но не содержит математической логики.

Параметры представлены неизменяемыми структурами данных. Для детерминированного расчета задаются коэффициенты реакции, четыре начальных уровня и горизонт. Стохастическая конфигурация дополнительно содержит стандартные отклонения, число реализаций, порог достижения и зерно генератора. Результат включает траектории, статус, показатели и метаданные среды.

Вычислительное ядро не считывает данные с клавиатуры и не строит графики. Консольный интерфейс или Jupyter Notebook вызывает одну и ту же функцию. Блокнот служит оболочкой для постановки опыта и интерпретации, а не единственным местом хранения кода [5, с. 4–5].

Реализация детерминированного ядра

Пусть x и y обозначают уровни воздействия сторон, а α и β — коэффициенты ответной реакции. Программное ядро реализует систему

$$x_{n+1} = x_n + \alpha(y_n - y_{n-1}), \qquad y_{n+1} = y_n + \beta(x_n - x_{n-1}). \qquad (1)$$

Перед циклом создаются массивы длины N+1 и записываются две пары начальных значений. Оба следующих состояния вычисляются из данных предыдущего шага и только затем помещаются в массивы. Последовательная перезапись x с последующим использованием нового значения при расчете y изменила бы исходную модель.

Предварительное выделение массивов исключает повторное расширение списков и упрощает расчет разностей и максимумов. NumPy предоставляет единый формат численных данных и эффективные массивные операции [3, с. 357–362]. После расчета программа отдельно возвращает уровни и приращения. Устойчивость определяется по затуханию приращений, а не по величине ненулевого предельного уровня.

Стохастический ансамбль

В стохастическом варианте коэффициенты получают независимые случайные добавки:

$$x_{n+1} = x_n + (\alpha + \xi_n)(y_n - y_{n-1}), \qquad y_{n+1} = y_n + (\beta + \eta_n)(x_n - x_{n-1}). \qquad (2)$$

Генерация выполняется через numpy.random.Generator. Зерно передается в конфигурации и включается в отчет. Использование глобального генератора нежелательно, поскольку результат зависит от скрытого порядка вызовов в других частях программы.

Ансамбль из M реализаций обрабатывается векторно: состояния всех реализаций хранятся в одномерных массивах, а цикл выполняется только по времени. Если требуются конечные показатели, полная матрица траекторий не сохраняется. Достаточно текущих состояний и накопителей максимумов. Полные данные записываются лишь для выбранных реализаций.

Программная верификация

Первый уровень проверки включает простые тесты. При нулевых начальных приращениях уровни должны оставаться постоянными. При α=β=0 изменения прекращаются после начального шага. В симметричном случае с

одинаковыми начальными данными массивы x и y совпадают. Такие тесты выявляют ошибки индексации без внешнего эталона.

Второй уровень сравнивает контрольную скалярную и основную векторизованную реализации. Для одинаковых случайных коэффициентов вычисляется среднеквадратическое расхождение

$$E = \sqrt{\frac{1}{2M(N+1)} \sum_{j=1}^{M} \sum_{n=0}^{N} \left[\left(x_{j,n}^{(s)} - x_{j,n}^{(v)} \right)^2 + \left(y_{j,n}^{(s)} - y_{j,n}^{(v)} \right)^2 \right]}. \qquad (3)$$

При совпадающем порядке операций показатель находится на уровне машинного округления. Систематическое отличие указывает на ошибку формы массива, последовательности обновления или генерации коэффициентов. Дополнительно на сетке параметров проверяется совпадение программной классификации с условием $q=\alpha\beta<1$.

Третий уровень контролирует повторяемость. Одинаковое зерно должно формировать идентичные показатели и контрольные суммы, другое зерно — иной ансамбль. Отделение вычислений от ввода-вывода, короткие функции и тестирование компонентов повышают надежность исследовательского программного обеспечения [7, с. 261–272; 8, с. 2–4].

Организация вычислительного эксперимента

Каждый опыт описывается конфигурацией с именем сценария, параметрами модели, горизонтом и перечнем показателей. Набор конфигураций запускается одной командой, поэтому сравнение режимов не требует изменения исходного кода. Для каждого сценария создается отдельный каталог результатов.

Параметры и метаданные сохраняются в JSON, итоговые показатели — в CSV, графики — в отдельном файле. Визуализация выполняется после расчета средствами Matplotlib [4, с. 90–95]. Изменение оформления рисунка не влияет на моделирование. Версии зависимостей фиксируются автоматически; контрольная среда включала Python 3.13, NumPy 2.3 и Matplotlib 3.10.

Результаты контрольных расчетов

В устойчивом сценарии при $\alpha=0{,}8$, $\beta=0{,}7$, начальных уровнях 0 и 1 и горизонте 40 шагов получены конечные значения 4,0909 и 3,8636. Последние приращения равны $1{,}31\cdot10^{-5}$ и $1{,}15\cdot10^{-5}$. Статус затухания совпал с аналитическим выводом $q=0{,}56$.

Для $\alpha=\beta=0{,}99$ после 400 шагов оба уровня достигли 98,2049, а последнее приращение оставалось равным 0,0181. Программа классифицировала режим как устойчивый, но выдала предупреждение о незавершенном переходном процессе. Следовательно, математическая устойчивость не означает быстрого установления.

В стохастическом опыте выполнено 10 000 реализаций при $\alpha=\beta=0{,}95$, стандартных отклонениях 0,12, горизонте 200 шагов и зерне 2026. Последнее приращение оказалось меньше 10^{-4} в 85,52% реализаций. Его медиана составила $1{,}94\cdot10^{-5}$, 95-й процентиль — $2{,}52\cdot10^{-4}$, 99-й — $7{,}29\cdot10^{-4}$. Медиана максимального уровня равна 19,12, а 99-й процентиль — 39,19.

Для 3000 реализаций и 250 шагов медианное время векторизованного потокового расчета составило 0,019 с, последовательного — 0,304 с. Получено ускорение приблизительно в 16 раз при одинаковых случайных коэффициентах. Расхождение результатов находилось на уровне округления. Абсолютное время зависит от оборудования, но преимущество массивной обработки сохраняется.

Ограничения и направления развития

Программа не идентифицирует коэффициенты по реальным данным: α, β и параметры шума задаются исследователем. Поэтому воспроизводимость расчета не доказывает достоверность предметных предпосылок. Нормальное распределение используется как базовый сценарий; модульная структура позволяет заменить его распределением со скачками или временной зависимостью.

При неустойчивых параметрах значения быстро растут, поэтому предусмотрен порог досрочного завершения с сохранением причины остановки. Дальнейшее развитие связано с добавлением управляемых, нелинейных и

многосекторных моделей, параллельным запуском ансамблей и публикацией кода вместе с конфигурациями.

Заключение

Разработана программная реализация вычислительного эксперимента в Python для дискретной модели санкционных процессов. Архитектура разделяет параметры, детерминированное ядро, стохастический ансамбль, показатели, экспорт и визуализацию. Это обеспечивает повторное использование кода и независимость результатов от способа запуска.

Предложены проверки входных данных, тесты свойств, сравнение скалярной и векторизованной версий, контроль зерна и автоматическое сохранение метаданных. Векторизация ускорила контрольный расчет приблизительно в 16 раз без изменения численного результата. Сценарии подтвердили соответствие программы аналитическому условию и необходимость увеличенного горизонта около границы устойчивости.

СПИСОК ЛИТЕРАТУРЫ

1. Новиков Д.А. Эффекты научения в моделях Осипова–Ланчестера // Управление большими системами. 2025. Вып. 116. С. 135–153. DOI: 10.25728/ubs.2025.116.7.

2. Fradkov A.L. International Stability under Iterated Sanctions and Counter-sanctions // IFAC-PapersOnLine. 2023. Vol. 56. No. 2. P. 11010–11014. DOI: 10.1016/j.ifacol.2023.10.800.

3. Array Programming with NumPy / C.R. Harris, K.J. Millman, S.J. van der Walt et al. // Nature. 2020. Vol. 585. P. 357–362. DOI: 10.1038/s41586-020-2649-2.

4. Hunter J.D. Matplotlib: A 2D Graphics Environment // Computing in Science & Engineering. 2007. Vol. 9. No. 3. P. 90–95. DOI: 10.1109/MCSE.2007.55.

5. Ten Simple Rules for Writing and Sharing Computational Analyses in Jupyter Notebooks / A. Rule, A. Birmingham, C. Zuniga et al. // PLOS Computational Biology. 2019. Vol. 15. No. 7. Art. e1007007. DOI: 10.1371/journal.pcbi.1007007.

6. Ten Simple Rules for Reproducible Computational Research / G.K. Sandve, A. Nekrutenko, J. Taylor, E. Hovig // PLOS Computational Biology. 2013. Vol. 9. No. 10. Art. e1003285. DOI: 10.1371/journal.pcbi.1003285.

7. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python / P. Virtanen, R. Gommers, T.E. Oliphant et al. // Nature Methods. 2020. Vol. 17. P. 261–272. DOI: 10.1038/s41592-019-0686-2.

8. Best Practices for Scientific Computing / G. Wilson, D.A. Aruliah, C.T. Brown et al. // PLOS Biology. 2014. Vol. 12. No. 1. Art. e1001745. DOI: 10.1371/journal.pbio.1001745.