Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Институт: ИВТИ Кафедра: УИТ

Направление подготовки: 27.03.04 Управление в технических системах

ОТЧЕТ по практике

Наименование практики: Производственная практика: научно-исследовательская работа

СТУДЕНТ

/ Ермакова П.А /
(подпись) (*Фамилия и инициалы*)

Группа А-02-21
(номер учебной группы)

ПРОМЕЖУТОЧНАЯ АТТЕСТАЦИЯ ПО ПРАКТИКЕ

(отлично, хорошо, удовлетворительно, неудовлетворительно, зачтено, не зачтено)

/ /
(подпись) (*Фамилия и инициалы члена комиссии*)

/ /
(подпись) (*Фамилия и инициалы члена комиссии*)

Москва
2024

Оглавление

Глава 1. Структурная декомпозиция моделей линейных разнотемповых динамических систем.

1.1. Методика декомпозиции моделей линейных разнотемповых динамических систем

Когда в системе присутствуют переменные, меняющиеся с разной скоростью, их называют разнотемповыми или многотемповыми системами. Эти системы образуют отдельный класс в классификации динамических систем и широко применяются на практике. Их отличительная черта – возможность описания моделями различных порядков на разных временных интервалах наблюдения.

Пусть математическое описание динамической системы представлено системой обыкновенных дифференциальных уравнений в нормальной форме Коши:

$$\dot{\vec{x}} = \vec{\phi}(\vec{x}, t) \qquad (1.1.1)$$

Декомпозиция модели предполагает замену исходной системы (1.1.1) на систему дифференциально-алгебраических уравнений [1, 2]

$$\begin{cases} \dot{\vec{\bar{x}}} = \tilde{\vec{\phi}}(\vec{\bar{x}}) \\ \vec{\Phi}(\vec{x}, t) = 0 \end{cases} \qquad (1.1.2)$$

Упрощенная модель определяется подсистемой дифференциальных уравнений (1.1.2), полученной из исходной модели путем исключения переменных и учета их алгебраических связей. Решение системы (1.1.2) будет отличаться от решения исходной модели (1.1.1), иначе модель (1.1.1) являлась бы моделью завышенного порядка. Поэтому при декомпозиции указывают допустимую погрешность решения системы (1.1.2) по сравнению с исходной моделью (1.1.1). Погрешность позволяет получить величину временного интервала применимости упрощенной модели (1.1.2).

Под интервалом справедливости упрощенной модели будем понимать интервал на временной оси наблюдения процессов, на котором погрешность воспроизведения ею процессов исходной модели не превышает допустимого значения δ^0 [3] . Его левая граница определяется моментом времени, когда некоторая модальная функция с указанной погрешностью становится равной нулю.

Пусть исходная модель определена на временном интервале $t \in (0, T]$, а упрощенная модель – $t \in [\Gamma_1, T]$, где T – правая граница интервала наблюдения процессов, а Γ_1 – левая граница интервала применимости упрощенной модели (1.1.2).

В явной форме значение Γ_v для простых вещественных и комплексно-сопряженных собственных значений.:

$$\Gamma_v = \frac{-1}{\alpha_v} \ln(\delta^0), \qquad (1.1.3)$$

где δ^0 - допустимое значение погрешности, α_v - собственное значение системы (1.1.1).

Правая граница интервала справедливости определяется временем переходного процесса T, под которым подразумевается время установления процессов с заданной точностью относительно их предельных значений. Будем определять время переходного процесса T как момент, начиная с которого все модальные функции затухнут с погрешностью не более, чем на величину δ^0. Тогда значение T будет определяться наибольшим из найденных выше граничных значений, то есть:

$$T = \Gamma_n$$

При этом общий интервал справедливости упрощенной модели соответствует временному отрезку $t \in [\Gamma_1, \Gamma_n]$.

Более подробно методика декомпозиции моделей линейных разнотемповых динамических систем на основе структурных свойств их решений приведена в [3].

1.2. Иллюстрация методики на примере математической модели 4 порядка

Рассмотрим изложенную выше методику декомпозиции моделей линейных разнотемповых систем на примере следующей модели 4 порядка, заданная в соответствии с нижеприведенной формулой :

$$\dot{\vec{x}} = A\vec{x} \qquad (1.2.1)$$

Получим системную матрицу A с заданными свойствами (собственные значения равны заданным) с помощью преобразования подобия:

$$A = T^{-1} * B * T, \qquad (1.2.2)$$

где T — произвольная матрица с $det \neq 0$, B – диагональная матрица, состоящая из собственных чисел.

Для этого сформируем матрицу T:

$$T = \begin{pmatrix} -2 & -10 & -5 & 14 \\ 12 & -6 & 10 & -2 \\ -6 & 11 & 3 & -11 \\ 14 & 2 & -7 & -11 \end{pmatrix}$$

Пусть имеются собственные числа вида:

$$\vec{\lambda} = (-10, -5, -1, -0.5)$$

Запишем диагональную матрицу B:

$$B = \begin{pmatrix} -10.0000 & 0 & 0 & 0 \\ 0 & -5.0000 & 0 & 0 \\ 0 & 0 & -1.0000 & 0 \\ 0 & 0 & 0 & -0.5000 \end{pmatrix}$$

В результате преобразования (1.2.2) системная матрица A принимает вид:

$$A = \begin{pmatrix} -3.0504 & -44.6371 & -14.7046 & 56.9748 \\ -1.4677 & -89.0323 & -25.8790 & 110.2661 \\ -3.4798 & -10.1452 & -6.7681 & 15.2601 \\ -1.2984 & -66.4516 & -19.4315 & 82.3508 \end{pmatrix}$$

В соответствии с (1.2.1) исходная модель имеет вид:

$$\begin{cases} \dot{x}_1 = -3.0504\,x_1 - 44.6371\,x_2 - 14.7046\,x_3 + 56.9748\,x_4 \\ \dot{x}_2 = -1.4677\,x_1 - 89.0323\,x_2 - 25.8790\,x_3 + 110.2661\,x_4 \\ \dot{x}_3 = -3.4798\,x_1 - 10.1452\,x_2 - 6.7681\,x_3 + 15.2601\,x_4 \\ \dot{x}_4 = -1.2984\,x_1 - 66.4516\,x_2 - 19.4315\,x_3 + 82.3508\,x_4 \end{cases}$$

Найдем решение системы, в соответствии с методикой, в виде:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} C_1 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 \\ 0 & 0 & C_3 & 0 \\ 0 & 0 & 0 & C_4 \end{pmatrix} (\vec{\alpha}_1 \vec{\alpha}_2 \vec{\alpha}_3 \vec{\alpha}_4) * \begin{pmatrix} e^{\lambda_1} \\ e^{\lambda_2} \\ e^{\lambda_3} \\ e^{\lambda_4} \end{pmatrix} \qquad (1.2.3)$$

Для этого рассчитаем собственные числа и собственные вектора системы (1.2.3):

$$\lambda = \begin{pmatrix} -10.0000 & 0 & 0 & 0 \\ 0 & -5.0000 & 0 & 0 \\ 0 & 0 & -1.0000 & 0 \\ 0 & 0 & 0 & -0.5000 \end{pmatrix}$$

$$\alpha = \begin{pmatrix} 0.379009 & 0.318930 & -0.393187 & 0.263195 \\ 0.734644 & 0.755812 & -0.705996 & 0.701854 \\ 0.096839 & -0.042371 & -0.068777 & 0.229452 \\ 0.554322 & 0.570295 & -0.585016 & 0.620871 \end{pmatrix}$$

Из рассчитанных собственных чисел видно, что преобразование (1.2.1) действительно дало модель с заданными свойствами.

Из уравнения (1.2.3) видно, что для расчета коэффициентов C_i можно воспользоваться следующей формулой:

$$C = \alpha^{-1} \vec{x}_0,$$

где α^{-1} - матрица собственных векторов, $\vec{x}_0$ - вектор начальных условий.

Пусть имеется вектор начальных условий $\vec{x}_0$, заданный следующим образом:

$$\vec{x}_0^T = (2.0000 \; -1.5000 \; 1.0000 \; -2.5000)$$

Тогда в соответствии с формулой, приведенной выше, получим вектор-столбец $\vec{C}$:

$$\vec{C}=\begin{pmatrix}35.0181\\-21.1266\\2.4331\\-13.5930\end{pmatrix}$$

Запишем полученное решение исходной модели в соответствии с (1.2.3):

$$\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}=\begin{pmatrix}C_1&0&0&0\\0&C_2&0&0\\0&0&C_3&0\\0&0&0&C_4\end{pmatrix}\left(\vec{\alpha}_1\vec{\alpha}_2\vec{\alpha}_3\vec{\alpha}_4\right)*\begin{pmatrix}e^{\lambda_1}\\e^{\lambda_2}\\e^{\lambda_3}\\e^{\lambda_4}\end{pmatrix}=\begin{pmatrix}13.2722&-6.7379&-0.9567&-3.5776\\25.7258&-15.9677&-1.7177&-9.5403\\3.3911&0.8952&-0.1673&-3.1190\\19.4113&-12.0484&-1.4234&-8.4395\end{pmatrix}\begin{pmatrix}e^{\lambda_1}\\e^{\lambda_2}\\e^{\lambda_3}\\e^{\lambda_4}\end{pmatrix}=$$

$$=\begin{pmatrix}13.2722e^{\lambda_1 t}&-6.7379e^{\lambda_2 t}&-0.9567e^{\lambda_3 t}&-3.5776e^{\lambda_4 t}\\25.7258e^{\lambda_1 t}&-15.9677e^{\lambda_2 t}&-1.7177e^{\lambda_3 t}&-9.5403e^{\lambda_4 t}\\3.3911e^{\lambda_1 t}&+0.8952e^{\lambda_2 t}&-0.1673e^{\lambda_3 t}&-3.1190e^{\lambda_4 t}\\19.4113e^{\lambda_1 t}&-12.0484e^{\lambda_2 t}&-1.4234e^{\lambda_3 t}&-8.4395e^{\lambda_4 t}\end{pmatrix}$$

В соответствии с методикой, изложенной выше, для дальнейших преобразований необходимо выразить $e^{\lambda_i t}$ из (1.2.3). Для этого обозначим коэффициенты при $e^{\lambda_i t}$ в решении (1.2.3) как матрицу H:

$$H=\begin{pmatrix}C_1&0&0&0\\0&C_2&0&0\\0&0&C_3&0\\0&0&0&C_4\end{pmatrix}\left(\vec{\alpha}_1\vec{\alpha}_2\vec{\alpha}_3\vec{\alpha}_4\right)$$

Тогда, чтобы выразить $e^{\lambda_i t}$, воспользуемся следующей формулой:

$$\begin{pmatrix}e^{\lambda_1 t}\\e^{\lambda_2 t}\\e^{\lambda_3 t}\\e^{\lambda_4 t}\end{pmatrix}=H^{-1}\begin{pmatrix}x_1\\x_2\\x_3\\x_4\end{pmatrix}\qquad(1.2.4)$$

Для того чтобы воспользоваться формулой (1.2.4), найдем H^{-1}:

$$H^{-1}=\begin{pmatrix}0.068966&0.344828&0.172414&-0.482759\\0.250000&-0.125000&0.208333&-0.041667\\-3.000000&5.500000&1.500000&-5.500000\\0.307692&0.043956&-0.153846&-0.241758\end{pmatrix}$$

В соответствии с (1.2.4) запишем получившиеся уравнения:

$$\begin{pmatrix} e^{-10t} \\ e^{-5t} \\ e^{-t} \\ e^{-0.5t} \end{pmatrix} = \begin{pmatrix} 0.068966 & 0.344828 & 0.172414 & -0.482759 \\ 0.250000 & -0.125000 & 0.208333 & -0.041667 \\ -3.000000 & 5.500000 & 1.500000 & -5.500000 \\ 0.307692 & 0.043956 & -0.153846 & -0.241758 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

Пусть для удобства $e^{\lambda_i t} = e_i$, тогда уравнения, изложенные выше, примут вид:

$$\begin{cases} e_1 = 0.068966\,x_1 + 0.344828\,x_2 + 0.172414\,x_3 - 0.482759\,x_4 \\ e_2 = 0.250000\,x_1 - 0.125000\,x_2 + 0.208333\,x_3 - 0.041667\,x_4 \\ e_3 = -3.000000\,x_1 + 5.500000\,x_2 + 1.500000\,x_3 - 5.500000\,x_4 \\ e_4 = 0.307692\,x_1 + 0.043956\,x_2 - 0.153846\,x_3 - 0.241758\,x_4 \end{cases} \quad (1.2.5)$$

В соответствии с методикой, после момента времени $t = \Gamma_1$ введем голономную связь $e_1 = 0$, тогда модель (1.2.5) примет вид:

$$\begin{cases} 0 = 0.068966\,x_1 + 0.344828\,x_2 + 0.172414\,x_3 - 0.482759\,x_4 \\ e_2 = 0.250000\,x_1 - 0.125000\,x_2 + 0.208333\,x_3 - 0.041667\,x_4 \\ e_3 = -3.000000\,x_1 + 5.500000\,x_2 + 1.500000\,x_3 - 5.500000\,x_4 \\ e_4 = 0.307692\,x_1 + 0.043956\,x_2 - 0.153846\,x_3 - 0.241758\,x_4 \end{cases} \quad (1.2.6)$$

Выразим x_1 из (1.2.6):

$$x_1 = -\frac{0.344828}{0.068966}x_2 - \frac{0.172414}{0.068966}x_3 + \frac{0.482759}{0.068966}x_4$$

$$x_1 = -4.999971000203 \cdot x_2 - 2.4999855001015 \cdot x_3 + 6.9999565003045 \cdot x_4$$

После подстановки выраженного x_1 в исходную модель получаем следующую модель пониженного порядка:

$$\begin{cases} \dot{x}_2 = -81.6938425630021 \cdot x_2 - 22.209771281501 \cdot x_3 + 99.9922638445031 \cdot x_4 \\ \dot{x}_3 = 7.2536990865064 \cdot x_2 + 1.9313495432532 \cdot x_3 - 9.09834862975959 \cdot x_4 \\ \dot{x}_4 = -59.9596376533364 \cdot x_2 - 16.1855188266682 \cdot x_3 + 73.2620564800046 \cdot x_4 \end{cases} \quad (1.2.7)$$

Рассчитаем левые и правые границы временного интервала справедливости модели пониженного порядка. Для этого воспользуемся формулой (1.1.3). Пусть $\delta^0 = 0.05$, тогда:

$$\Gamma_1 = \frac{1}{-10} * \ln(0.05) = 0.299573$$

$$\Gamma_4 = \frac{1}{-0.5} * \ln(0.05) = 5.99146$$

Таким образом, общий интервал справедливости упрощенной (вырожденной) модели может быть определен как $[\Gamma_1; \Gamma_4]$.

1.3. Проверка близости процессов в исходной и упрощенной моделях путем моделирования в SimInTech

Проверим моделированием в ПО SimInTech, что процессы вырожденной модели близки к процессам исходной модели.

Для того чтобы рассматривать вырожденную и исходную систему в границах интервала справедливости модели пониженного порядка, найдем начальные условия на границе Γ_1. Для этого соберем схему, представленную ниже на Рис.1.

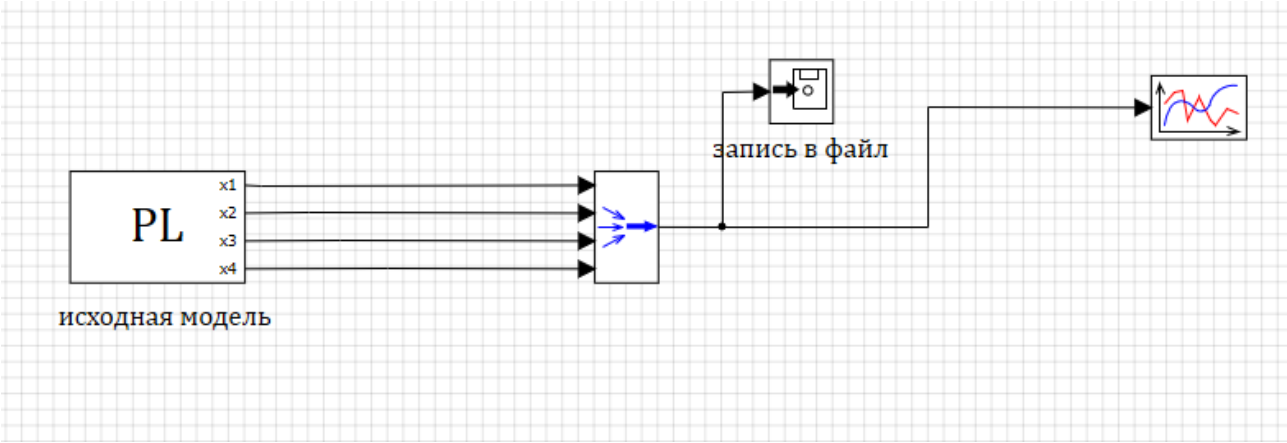

Рисунок 1: Схема для нахождения начальных условий на границе

В блоке PL(блок «Язык программирования») зададим систему дифференциальных уравнений исходной модели, а также зададим начальные условия. Содержание блока представлено на Рис.2. После моделирования получим новые значения начальных условий:

$$\vec{x}_0^T = (-4.62903746 - 11.75958163 - 2.44099544 - 10.03510235)$$

```
init x1 = 2, x2 = -1.5, x3 = 1, x4 = -2.5;
output x1,x2,x3,x4;

x1' = -3.0504*x1-44.6371*x2-14.70464*x3+56.9748*x4;
x2' = -1.46774*x1-89.03226*x2-25.87903*x3+110.26613*x4;
x3' = -3.47984*x1-10.14516*x2-6.76815*x3+15.26008*x4;
x4' = -1.29839*x1-66.45161*x2-19.43145*x3+82.35081*x4;
```

Рисунок 2: Блок PL для исходной модели

Далее составим схему для сравнения процессов в исходной и вырожденных моделях на интервале$[\Gamma_1;\Gamma_4]$. . Схема представлена на Рис.3.

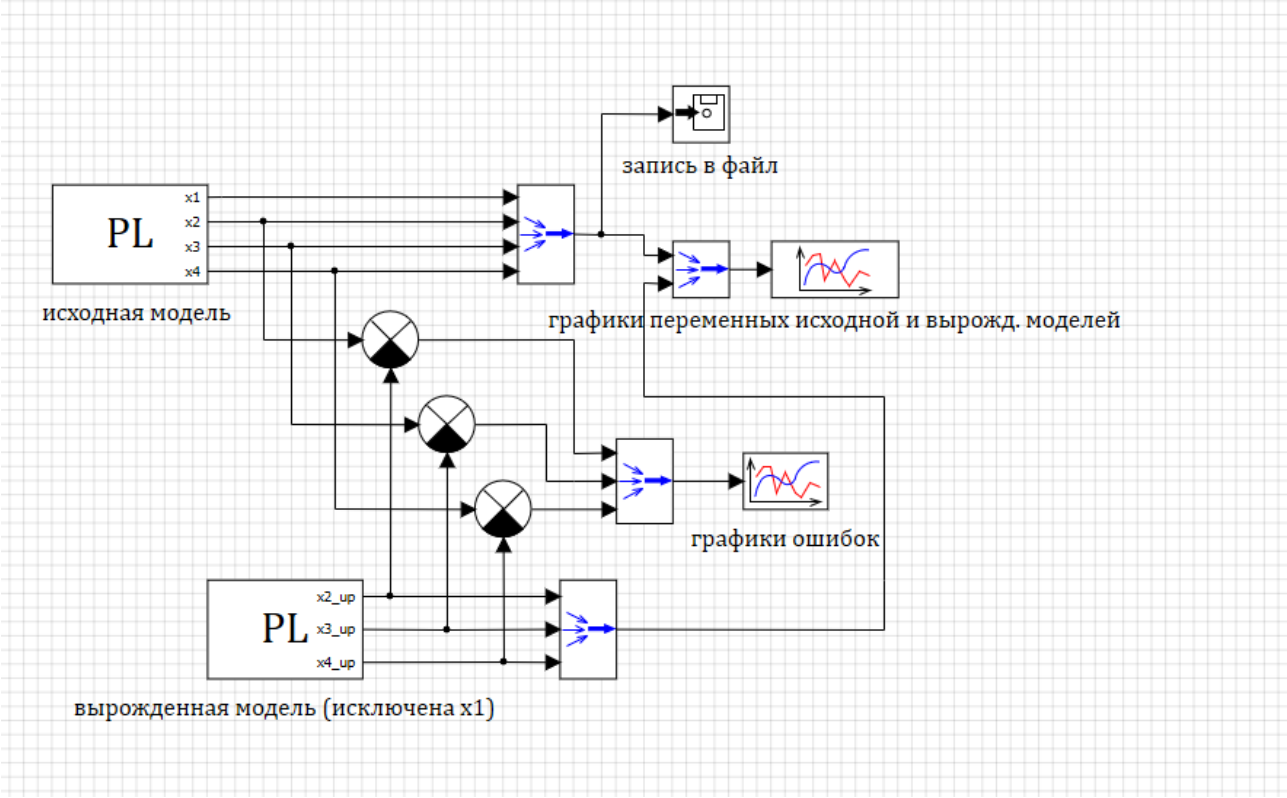

Рисунок 3: Схема для проверки близости процессов

Содержимое блоков PL представлено ниже на Рис.4, Рис.5:

```
init x1 = -4.62903746, x2 =  -11.75958163, x3 = -2.44099544 , x4 = -10.03510235;


output x1,x2,x3,x4;

x1' = -3.0504*x1-44.6371*x2-14.70464*x3+56.9748*x4;
x2' = -1.46774*x1-89.03226*x2-25.87903*x3+110.26613*x4;
x3' = -3.47984*x1-10.14516*x2-6.76815*x3+15.26008*x4;
x4' = -1.29839*x1-66.45161*x2-19.43145*x3+82.35081*x4;
```

Рисунок 4: Блок PL для исходной системы

```
init x2_up =  -11.75958163, x3_up = -2.44099544 , x4_up = -10.03510235;
output x2_up,x3_up,x4_up;

x2_up' = -81.69355*x2_up-22.20968*x3_up+99.99194*x4_up;
x3_up' = 7.25403*x2_up+1.93145*x3_up-9.09879*x4_up;
x4_up' = -59.95968*x2_up-16.18548*x3_up+73.2621*x4_up;
```

Рисунок 5: Блок PL для вырожденной системы

После моделирования получим графики абсолютных значений отклонений переменных вырожденной модели от переменных исходной модели (рис. 6):

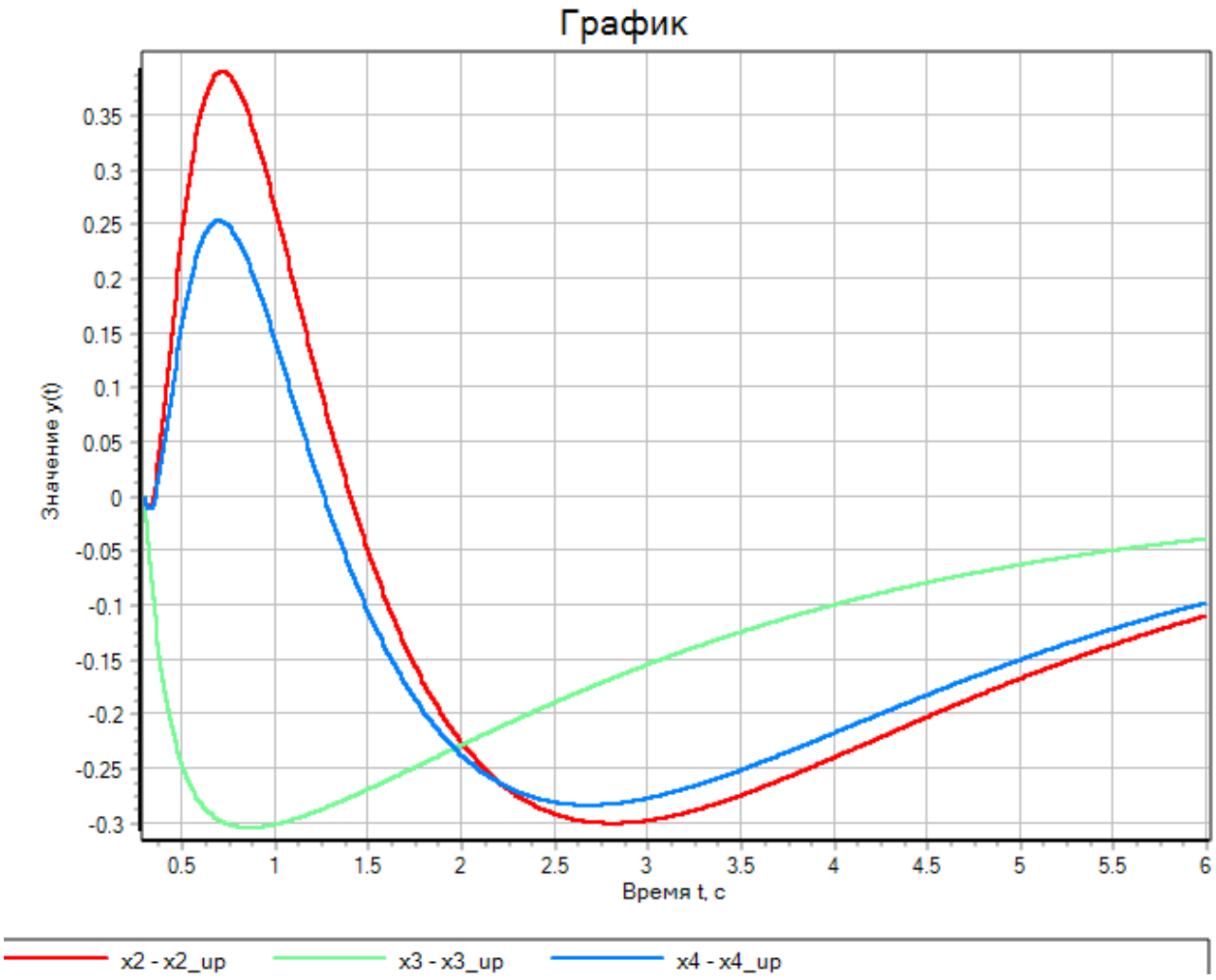

Рисунок 6: Графики разностей между переменными исходной и вырожденной моделей

Глава 2. Разработка программного обеспечения для осуществления декомпозиции моделей линейных разнотемповых систем .

Требуется разработать программное средство, реализующее методику [3] структурной декомпозиции математических моделей линейных разнотемповых динамических систем.

2.1. Выбор средств разработки и их описание

Разработка программного обеспечения для исследования декомпозиции многотемповых линейных систем будет проводится на базе языка программирования Python 3 [4]. Выбор Python 3 для разработки программного обеспечения имеет множество достоинств. Рассмотрим несколько из них:

1. Читаемость кода: Python 3 имеет простую и читаемую синтаксическую структуру, что делает код легким для понимания и поддержки.

2. Стандартные библиотеки: Python 3 поставляется с обширным набором стандартных библиотек, которые охватывают множество задач, таких как работа с файлами, сетевое программирование, обработка текстов и многого другого.

3. Обширное сообщество: Python имеет огромное сообщество разработчиков, которые активно делятся своими наработками и помогают друг другу решать проблемы.

4. Кроссплатформенность: Программы, написанные на Python 3, могут работать на различных операционных системах без необходимости вносить изменения в код.

5. Активное развитие: Python 3 продолжает активно развиваться и совершенствоваться, получая новые функции и улучшения

производительности. Это гарантирует, что язык остается актуальным и конкурентоспособным.

Таким образом, Python 3 сочетает в себе удобство использования, широкие возможности, поддержку сообщества и постоянное развитие, что делает его идеальным выбором для разработки программного обеспечения, включая исследования в области декомпозиции многотемповых линейных систем.

2.2. Разработка программного обеспечения

Для получения программного алгоритма в соответствии с методикой изложенной в (1.1) необходимо реализовать следующие функции:

1) Рассчитать собственные числа и собственные вектора.

2) Рассчитать коэффициенты C.

3) Рассчитать инверсную матрицу собственных векторов.

4) Рассчитать новую системную матрицу пониженного порядка.

5) Рассчитать границы временного интервала справедливости модели пониженного порядка.

6) Рассчитать новые начальные условия в момент времени левой границы временного интервала справедливости.

7) Рассчитать новые решения, используя новые начальные условия.

В соответствии с этим была разработана программа, листинг которой представлен в приложении.

Рассмотрим интерфейс полученной программы, представленный на Рис.7. В интерфейсе имеется возможность выбрать размерность системной матрицы A. Также имеются окошки для ввода элементов самой матрицы, начальных условий и ошибки. После нажатия на кнопку «Понизить порядок»

рассчитывается новая модель, а также выводятся графики и границы временного интервала справедливости модели.

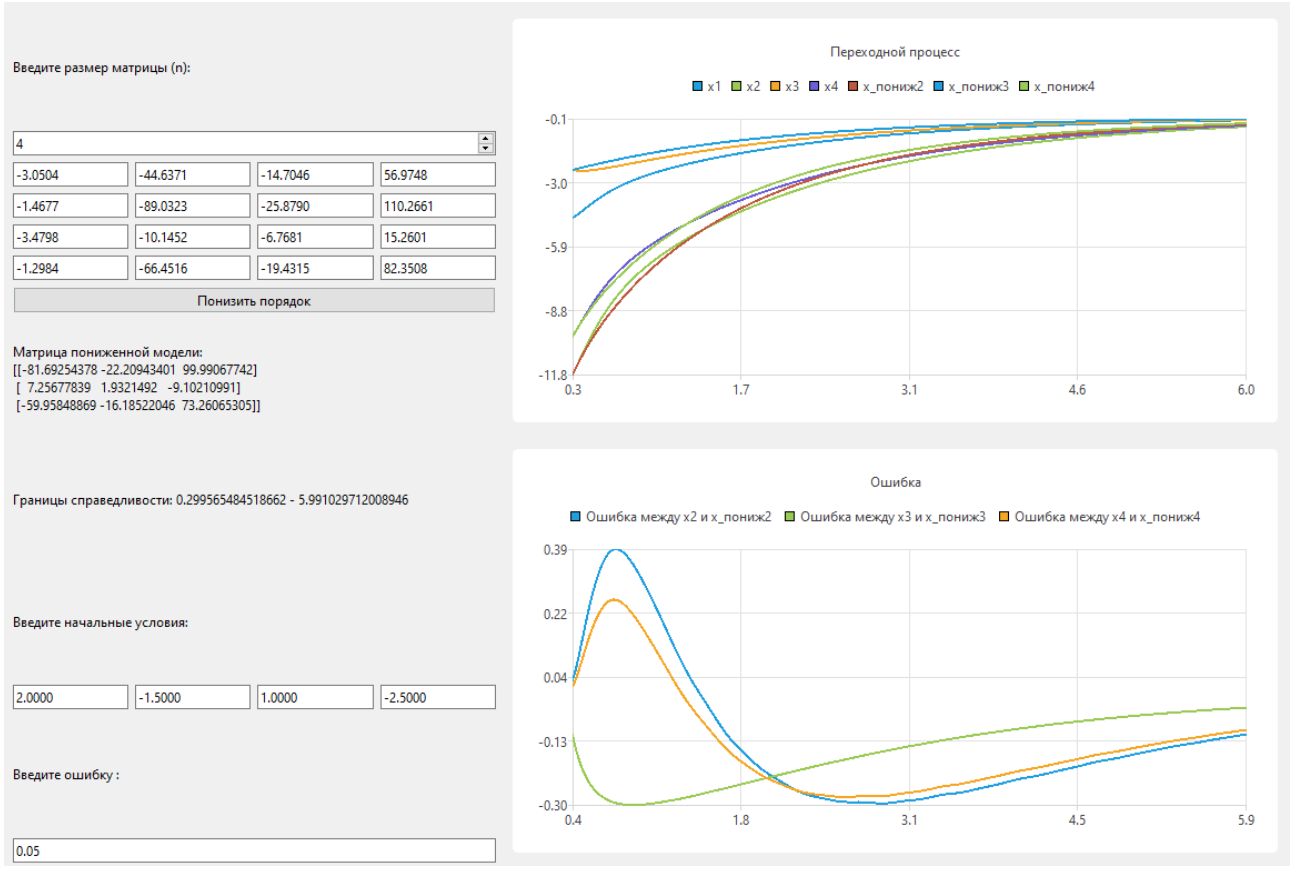

Рисунок 7: Интерфейс программы

На Рис.8 и Рис.9 представлены более подробно окна для ввода размерности модели, системной матрицы, начальных условий, а также ошибки.

Рисунок 8: Ввод системной матрицы

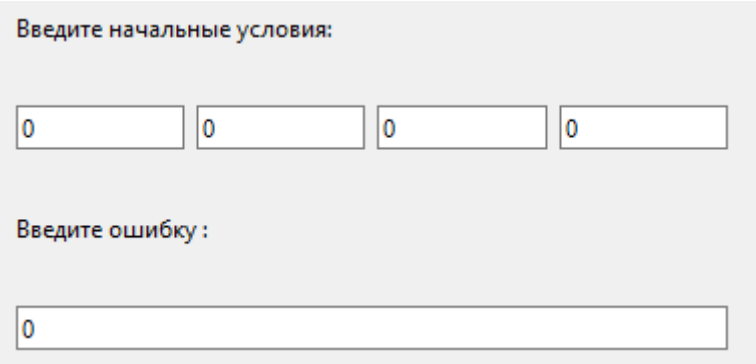

Рисунок 9: Ввод начальных условий и ошибки

2.3. Пример декомпозиции на основе разработанного ПО

Проверим работоспособность полученной программы. Введем системную матрицу A исходной модели, полученной ранее в п. 1.2, и сравним результаты ее декомпозиции на основе разработанного ПО с приведенными в главе 1 (рис. 10). Начальные условия для исходной модели и допустимое значение погрешности вырожденной модели (δ^0) также возьмем равными заданным в главе 1 (рис. 11)

Рисунок 10.Ввод матрицы системы

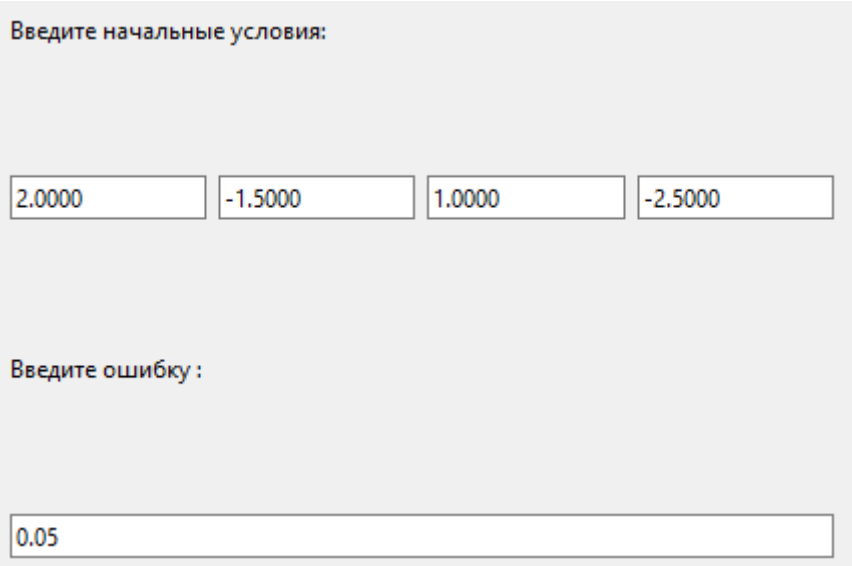

Рисунок 11.Ввод начальных условий и ошибки

После нажатия на кнопку «Понизить порядок» выводятся матрица вырожденной модели, а также границы временного интервала справедливости вырожденной модели.

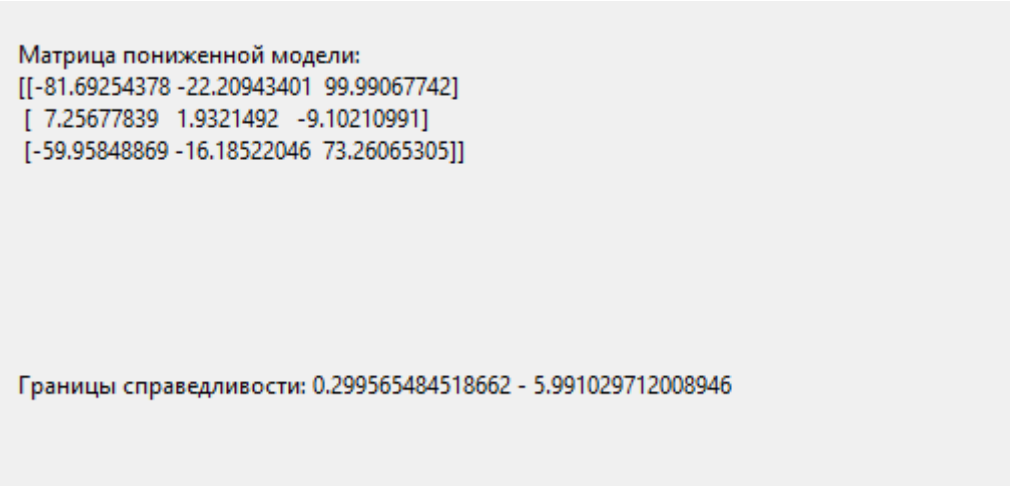

Рисунок 12.матрица модели пониженного порядка и границы ее справедливости.

Справа от ввода данных появляются графики переходных процессов и ошибки между моделями. Они представлены на Рис.13, Рис.14.

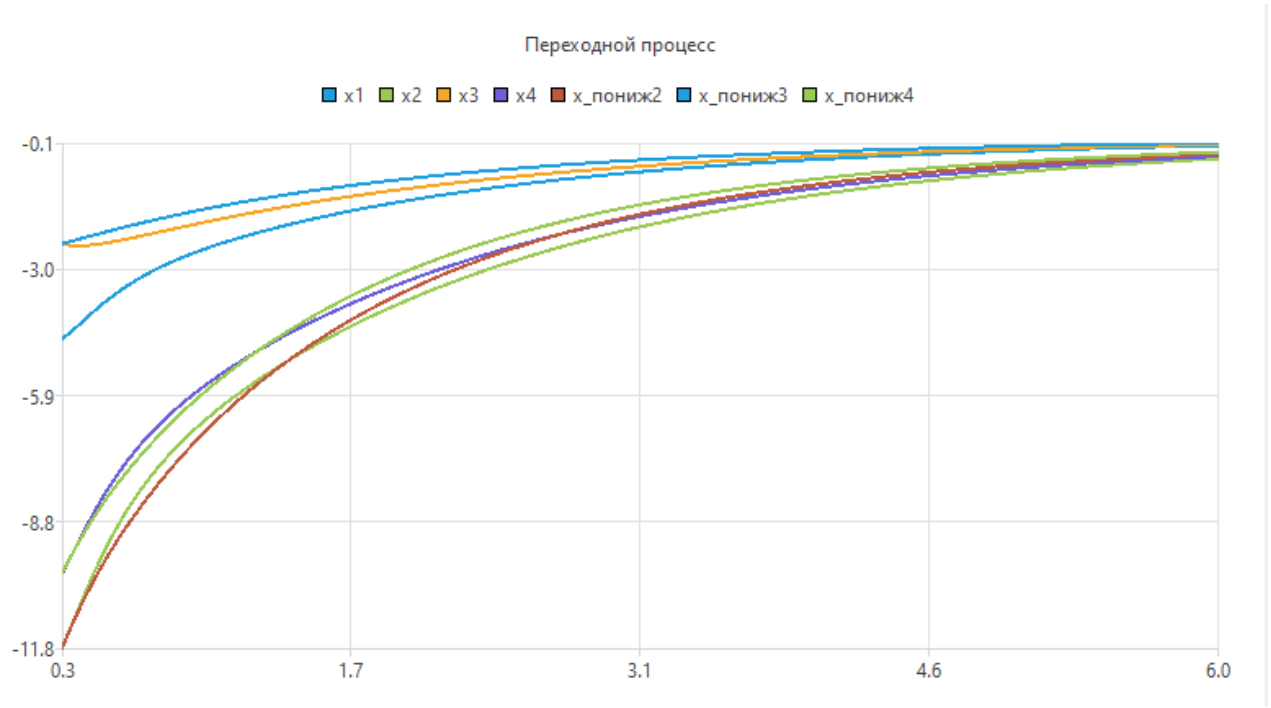

Рисунок 13.Процессы в исходной и вырожденной моделях

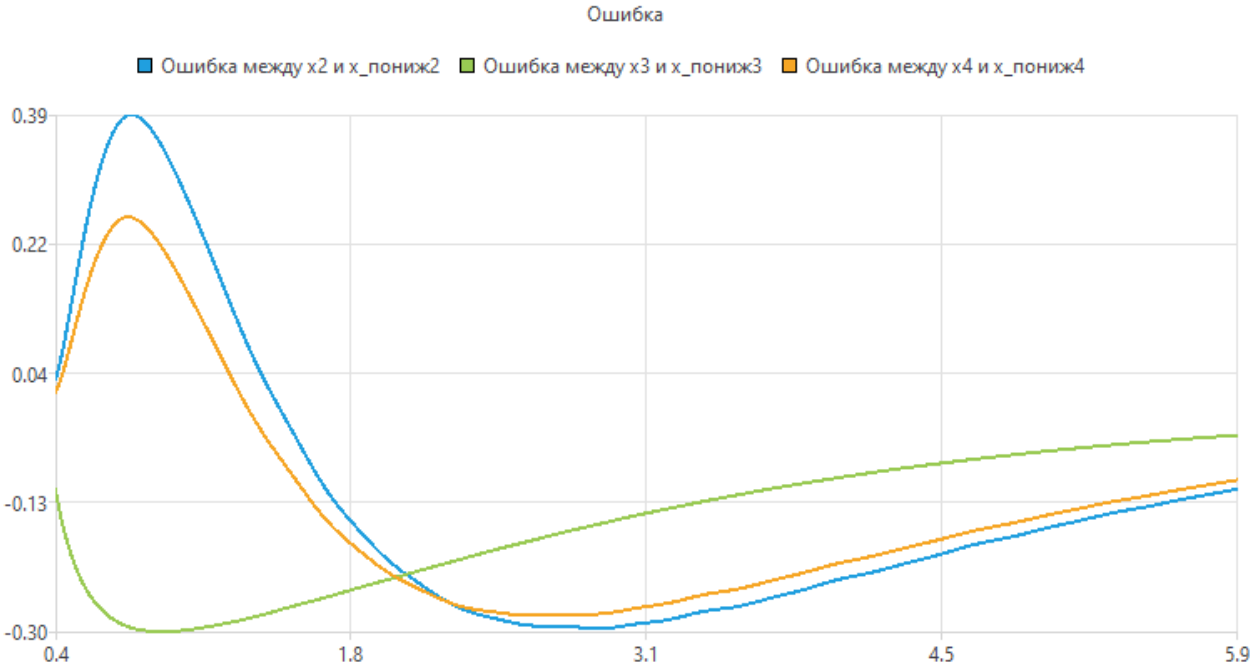

Рисунок 14.Графики отклонений переменных вырожденной модели от переменных исходной (графики ошибок)

Если сравнить графики ошибок между моделями, полученные путем имитационного моделирования в SimInTech (см. Рис. 6) и с помощью разработанного ПО, то можно сделать вывод о корректной работе последнего.

Список литературы

1. Воронов А.А. Введение в динамику сложных управляемых систем. - М.: Наука. Главная редакция физико-математической литературы, 1985. — 352-с.

2. Халил Х.К. Нелинейные системы. - М.-Ижевск: НИЦ «Регулярная и хаотическая динамика», Институт компьютерных исследований 2009 — 832-с.

3. Державин О.М., Сидорова Е.Ю. Методика декомпозиции моделей многотемповых динамических систем, Известия Тульского государственного университета. Технические науки. 2023 г. Вып. 12.-С. 3-9.

4. Прохоренок. Дронов: Python 3 и PyQt 5. Разработка приложений. 2-е издание/Н.А. Прохоренок, В.А. Дронов. - BHV, 2019. - 832 с.

Приложение

```
import sys
import numpy as np
from PyQt6.QtWidgets import QApplication, QWidget, QVBoxLayout, QGridLayout,
QLabel, QLineEdit, QPushButton, QMainWindow, QSpinBox, QHBoxLayout
from PyQt6.QtGui import QPalette, QColor
from PyQt6.QtCharts import QChart, QChartView, QLineSeries
import sympy as sp
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import math as m

class Matrix:
  def __init__(self, n, values, initial_conditions, err):
    self.size = n
    self.err = err
    self.values = values
    self.determinant = None
    self.eigenvalues = np.zeros(n)
    self.eigenvectors = np.zeros((n, n))
    self.initial_conditions = initial_conditions
    self.coefficients = np.zeros(n)
    self.inverse_eig = None
    self.inverse = None
    self.index_max_eigval = None
    self.matr_with_x = []
    self.matr_resh = np.zeros((n-1, n-1))
    self.gran = []

  def calc_gran(self):
    for i in range(self.size):
```

```
      gran_value = (1 / self.eigenvalues[i]) * m.log(self.err)
      print(gran_value)
      self.gran.append(gran_value)

  def is_determinant_negative(self):
    self.determinant = np.linalg.det(self.values)
    return self.determinant < 0

  def calculate_eigenvalues_and_eigenvectors(self):
    self.eigenvalues, self.eigenvectors = np.linalg.eig(self.values)

  def calculate_inverse(self):
    self.inverse_eig = np.linalg.inv(self.eigenvectors)

  def calculate_coefficients(self):
    self.calculate_inverse()
    if self.inverse_eig is not None:
      self.coefficients = self.inverse_eig @ self.initial_conditions.T

  def calculate_inverse_from_eigenvectors(self):
    diagonal_matrix = np.diag(self.coefficients.flatten())
    self.inverse = np.linalg.inv(self.eigenvectors @ diagonal_matrix)

  def find_index_of_max_eigenvector(self):
    self.index_max_eigval = np.argmax(abs(self.eigenvalues))

  def expr_max_exp(self):
    self.find_index_of_max_eigenvector()
    x_resh = sp.symbols(f'x{self.index_max_eigval + 1}')
    xi = sp.solve(sum(self.inverse[self.index_max_eigval][i] * sp.symbols(f'x{i +
1}') for i in range(self.size)), x_resh)[0]
```

```
    for i in range(self.size):
      temp_expr = sum(self.values[i][j] * sp.symbols(f'x{j + 1}') for j in
range(self.size))
      self.matr_with_x.append(temp_expr)
    substitutedequations = [eq.subs(x_resh, xi) for eq in self.matr_with_x]
    simplifiedequations = [sp.simplify(eq) for eq in substitutedequations]
    print("\nПолученные выражения:")
    for i, eq in enumerate(simplifiedequations[1:]):
      coefficients = [float(eq.coeff(sp.symbols(f'x{j+1}'))) for j in range(1,
self.size)]
      if i < self.matr_resh.shape[0]:
        self.matr_resh[i, :len(coefficients)] = coefficients

  def construct_equations_matrix(self):
    equations_matrix = []
    for i in range(self.size):
      equation = sum(self.coefficients[j] * self.eigenvectors[j, i] *
sp.exp(self.eigenvalues[j]) for j in range(self.size))
      equations_matrix.append(equation)
    return equations_matrix

  def calc(self):
    self.calculate_eigenvalues_and_eigenvectors()
    self.calculate_coefficients()
    self.calculate_inverse_from_eigenvectors()
    self.expr_max_exp()
    self.calc_gran()
    equations_matrix = self.construct_equations_matrix()

def system_of_equations(t, y, values):
  x = y
  dx_dt = values @ x
```

```
  return dx_dt


def plot_transient_response2(temp_sol, temp_sol2, size_matr_1, size_matr_2,
chart_view):
  chart = QChart()
  for i in range(size_matr_1):
    series = QLineSeries()
    for j in range(len(temp_sol.t)):
      series.append(temp_sol.t[j], temp_sol.y[i][j])
    series.setName(f'x{i+1}')
    chart.addSeries(series)
  for i in range(size_matr_2):
    series = QLineSeries()
    for j in range(len(temp_sol2.t)):
      series.append(temp_sol2.t[j], temp_sol2.y[i][j])
    series.setName(f'x_пониж{i+2}')
    chart.addSeries(series)
  chart.createDefaultAxes()
  chart.setTitle('Переходной процесс')
  chart_view.setChart(chart)
  chart_view.setRubberBand(QChartView.RubberBand.RectangleRubberBand)


def moving_average(data, window_size):
  return np.convolve(data, np.ones(window_size)/window_size, mode='valid')


def plot_error_response(solution1_new, solution2_new, size_matr_2, chart_view,
window_size=10):
  errors = solution1_new.y[1:size_matr_2+1, :] - solution2_new.y
  chart = QChart()
  for i in range(size_matr_2):
    smoothed_errors = moving_average(errors[i], window_size)
    smoothed_t = moving_average(solution1_new.t, window_size)
```

```
    series = QLineSeries()
    for j in range(len(smoothed_t)):
      series.append(smoothed_t[j], smoothed_errors[j])
    series.setName(f'Ошибка между x{i+2} и x_пониж{i+2} ')
    chart.addSeries(series)
  chart.createDefaultAxes()
  chart.setTitle('Ошибка')
  chart_view.setChart(chart)
  chart_view.setRubberBand(QChartView.RubberBand.RectangleRubberBand)
# Приложение для матриц

class MatrixApp(QMainWindow):
  def __init__(self):
    super().__init__()
    self.setWindowTitle("Matrix Exponent App")
    self.setGeometry(100, 100, 1200, 600)
    self.flag = True
    self.flag2 = True
    self.central_widget = QWidget()
    self.setCentralWidget(self.central_widget)


    self.layout = QHBoxLayout()


    self.input_layout = QVBoxLayout()
    self.grid_layout = QGridLayout()


    self.size_input = QSpinBox()
    self.size_input.setRange(2, 10)
    self.size_input.setValue(3)


    self.input_layout.addWidget(QLabel("Введите размер матрицы (n):"))
```

```
    self.input_layout.addWidget(self.size_input)
    self.entries = []
    self.initial_conditions_entries = []
    self.error_entries = []
    self.values_matrix = np.zeros((int(self.size_input.text()),
int(self.size_input.text())))
    self.values_matrix_initial_cond = np.zeros((1, int(self.size_input.text())))
    self.error_value = None

    self.size_input.valueChanged.connect(self.create_matrix_inputs)
    self.plot_button = QPushButton("Понизить порядок")
    self.chart_view1 = QChartView()
    self.chart_view2 = QChartView()

    self.chart_layout = QVBoxLayout()
    self.plot_button.clicked.connect(self.plot_exponents)

    self.input_layout.addLayout(self.grid_layout)
    self.input_layout.addWidget(self.plot_button)

    self.chart_layout.addWidget(self.chart_view1)
    self.chart_layout.addWidget(self.chart_view2)

    self.layout.addLayout(self.input_layout)
    self.layout.addLayout(self.chart_layout)

    # Добавление QLabel для пониженной матрицы и границ
    self.reduced_matrix_label = QLabel()
    self.bounds_label = QLabel()
    self.input_layout.addWidget(self.reduced_matrix_label)
    self.input_layout.addWidget(self.bounds_label)
```

```
    self.central_widget.setLayout(self.layout)

def create_matrix_inputs(self):
    for i in reversed(range(len(self.entries))):
        for j in range(len(self.entries[i])):
            self.grid_layout.itemAtPosition(i, j).widget().deleteLater()
        self.entries.pop()

    for i in range(self.size_input.value()):
        row_entries = []
        for j in range(self.size_input.value()):
            entry = QLineEdit()
            entry.setPlaceholderText(f"Элемент ({i+1},{j+1})")
            entry.setText("0")
            row_entries.append(entry)
            self.grid_layout.addWidget(entry, i, j)
        self.entries.append(row_entries)

    self.create_initial_conditions_inputs()
    self.create_error_input()

def create_initial_conditions_inputs(self):
    if (self.flag):
        self.initial_conditions_label = QLabel("Введите начальные условия:")
        self.input_layout.addWidget(self.initial_conditions_label)
        self.flag = False
    else:
        self.initial_conditions_label.deleteLater()
        self.initial_conditions_label = QLabel("Введите начальные условия:")
        self.input_layout.addWidget(self.initial_conditions_label)
```

```
    for entry in self.initial_conditions_entries:
      entry.deleteLater()
    self.initial_conditions_entries.clear()

    n = self.size_input.value()
    self.initial_conditions_grid = QGridLayout()
    for j in range(n):
      entry = QLineEdit()
      entry.setPlaceholderText(f"Начальное условие {j+1}")
      entry.setText("0")
      self.initial_conditions_entries.append(entry)
      self.initial_conditions_grid.addWidget(entry, 0, j)
    self.input_layout.addLayout(self.initial_conditions_grid)

  def read_matrix_data(self):
    n = self.size_input.value()
    self.values_matrix = np.zeros((n, n))
    self.values_matrix_initial_cond = np.array([np.zeros(n)])
    self.error_value = 0.0

    for i in range(n):
      for j in range(n):
        try:
          self.values_matrix[i][j] = float(self.entries[i][j].text())
        except ValueError:
          self.values_matrix[i][j] = 0.0

    initial_conditions = []
    for j in range(n):
      try:
        initial_conditions.append(float(self.initial_conditions_entries[j].text()))
```

```
      except ValueError:
        initial_conditions.append(0.0)
    self.values_matrix_initial_cond = np.array([initial_conditions])

    try:
      self.error_value = float(self.error_entry.text())
    except ValueError:
      self.error_value = 0.0

  def create_error_input(self):
    if (self.flag2):
      self.error_label = QLabel("Введите ошибку :")
      self.input_layout.addWidget(self.error_label)
      self.flag2 = False
    else:
      self.error_label.deleteLater()
      self.error_label = QLabel("Введите ошибку :")
      self.input_layout.addWidget(self.error_label)
    if self.error_entries:
      self.error_entries[0].deleteLater()
    self.error_entries.clear()
    self.error_entry = QLineEdit()
    self.error_entry.setPlaceholderText("Значение ошибки")
    self.error_entry.setText("0")
    self.error_entries.append(self.error_entry)
    self.input_layout.addWidget(self.error_entry)

  def plot_exponents(self):
    self.read_matrix_data()
    matrix_1 = Matrix(int(self.size_input.text()), self.values_matrix,
self.values_matrix_initial_cond, self.error_value)
```

```
    matrix_1.calc()
    new_values = matrix_1.matr_resh
    matrix_2 = Matrix(int(self.size_input.text())-1, new_values,
self.values_matrix_initial_cond[:, 1:], self.error_value)
    matrix_2.calc()
    print(matrix_1.values)
    t_span = (0, 10)
    t_eval = np.linspace(t_span[0], t_span[1], 400)
    solution1 = solve_ivp(system_of_equations, t_span,
matrix_1.initial_conditions.flatten(), t_eval=t_eval, args=(matrix_1.values,))
    solution2 = solve_ivp(system_of_equations, t_span,
matrix_2.initial_conditions.flatten(), t_eval=t_eval, args=(matrix_2.values,))

    initial_conditions_new = None
    print("Solution 1:")
    min_gran = min(matrix_1.gran)
    print(min_gran)
    for i, t in enumerate(solution1.t):
      if abs(t - min_gran) <= 0.02:
        initial_conditions_new = np.array([solution1.y[:, i]])
        print(f"t = {t:.2f}, y = {solution1.y[:, i]}")
        break

    a_new = Matrix(int(self.size_input.text()), self.values_matrix,
initial_conditions_new, self.error_value)
    a_new.calc()

    new_values1 = a_new.matr_resh
    b_new = Matrix(int(self.size_input.text())-1, new_values1,
initial_conditions_new[:, 1:], self.error_value)
    b_new.calc()
    t_span = (min(matrix_1.gran), max(matrix_1.gran))
```

```
    t_eval = np.linspace(t_span[0], t_span[1], 400)
    solution1_new = solve_ivp(system_of_equations, t_span,
a_new.initial_conditions.flatten(), t_eval=t_eval, args=(a_new.values,))
    solution2_new = solve_ivp(system_of_equations, t_span,
b_new.initial_conditions.flatten(), t_eval=t_eval, args=(b_new.values,))

    plot_transient_response2(solution1_new, solution2_new,
int(self.size_input.text()), int(self.size_input.text())-1, self.chart_view1)
    plot_error_response(solution1_new, solution2_new,
int(self.size_input.text())-1, self.chart_view2)

    self.reduced_matrix_label.setText(f"Матрица пониженной модели:\n{new_values}")
    self.bounds_label.setText(f"Границы справедливости: {min_gran} -
{max(matrix_1.gran)}")

if __name__ == "__main__":
  app = QApplication(sys.argv)
  window = MatrixApp()
  window.show()
  sys.exit(app.exec())
```