Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО

Е.П. Тарадаев

ИСПОЛЬЗОВАНИЕ СРЕДЫ WOLFRAM MATHEMATICA ДЛЯ РЕШЕНИЯ ЧИСЛЕННЫХ ЗАДАЧ

Учебное пособие

Санкт-Петербург
2020

УДК 519.6(075.8)

Тарадаев Е.П. Использование среды Wolfram Mathematica для решения численных задач : учебное пособие / Е.П. Тарадаев. – СПб., 2020. – 57 с.

Пособие по учебной дисциплине «Численные методы» специальности 16.03.01 «Техническая физика». Пособие составлено в соответствии с образовательным стандартом высшего образования Федерального государственного автономного образовательного учреждения высшего образования «Санкт-Петербургский политехнический университет Петра Великого» по направлению подготовки бакалавров 16.03.01 «Техническая физика».

Данное пособие предназначено студентам, выполняющим практические задания или лабораторные работы по курсу «Численные методы». В пособии приводятся краткие теоретические сведения по основным разделам курса. Обращается внимание на методы и алгоритмы решения основных прикладных задач.

В пособии приведены примеры методов численного решения задач в системе Mathematica по основным разделам линейной алгебры, аппроксимации и интерполяции функций, численного решения дифференциальных уравнений. Применение среды Wolfram Mathematica позволяет просто и наглядно, без ненужного технического программирования, реализовать вычислительные схемы численных методов. Пособие может служить основой для проведения дистанционных лабораторных работ или подготовки студентов заочного отделения физико-математических факультетов по данному курсу.

Оглавление

Введение

Решение множества практических задач методами математического моделирования обычно содержат погрешности, и поэтому полученные решения всегда являются приближением реальных процессов. Назовем основные источники погрешностей.

Погрешность исходных данных. Уже на этапе формирования модели как дискретной, так и непрерывной принимаются определенные допущения, задаются параметры процессов, константы модели и т.д. Эти неопределенности привносят неустранимый вклад в погрешность результата математического анализа. Для адекватной оценки результатов моделирования уже на этапе формирования модели следует иметь оценки погрешности исходных данных.

Погрешность дискретизации. Дискретизация математической модели, а как следствие замена непрерывных операторов на конечно-разностные аппроксимации, приводит к появлению ошибки дискретизации. Например, замена оператора дифференцирования dy/dx на его конечно-разностную аппроксимацию $y'(x) = (y(x+h) - y(x))/h$. Эта ошибка хоть может быть сколь угодно малой посредством выбора параметров дискретизации (например,шагов h), но она останется в том или ином видее.

Погрешность округления. Неизбежные округления, возникающие в процессе моделирования, зачастую вызываются вычислительным инструментом моделирования. Например, для хранения и обработки чисел в памяти компьютера отводиться ограниченное количество ячеек памяти. Это приводит к тому, что множество вещественных чисел, которые можно представить в компьютере дискретно, а значит, машинное представление любого вещественного числа содержит погрешность, именуемою погрешностью округления. В ходе реализации алгоритма расчёта математической модели эти погрешности могут накапливаться, что, в частности, и определит погрешность результата. Назовем алгоритм устойчивым, если в ходе его реализации погрешность результата остается ограниченной.

Важнейший параметр вычислительной системы — машинный эпсилон ϵ_M. Он характеризует относительную ошибку представления вещественных чисел в компьютерном представлении. Любое число в диапазоне $[1 - \epsilon_M, 1 + \epsilon_M]$ в машинном представлении не отличимо от 1.

Стоит отметить, что влияние ошибок округления на результат моделирования зависит не только от метода моделирования, но и от свойств самой задачи. Небольшое возмущение исходных данных может приводить к появлению неограниченного роста погрешности решения. Здесь возникает понятие обусловленности задачи. По сути, обусловленность показывает, как поведет себя решение при малом отклонении исходных данных.

Предлагаемые лабораторные работы нацелены на определение влияния методов решения численных задач, погрешностей исходных данных, величины разрядной сетки компьютера на погрешность получаемого решения.

1. Экспериментальное исследование численных методов решения систем линейных алгебраических уравнений

1.1 Цель работы

Изучение прямых и итерационных методов решения систем алгебраических уравнений с позиции получаемых решений и сходимости итерационных процессов.

1.2 Методы решения систем алгебраических уравнений

Одной из основных задач линейной алгебры является нахождение решения системы алгебраических уравнений (СЛАУ). Под СЛАУ понимается система уравнений, каждое уравнение в которой является линейным алгебраическим уравнением первой степени.

Общий вид системы линейных алгебраических уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases} \tag{1.1}$$

Система называется однородной, если все её свободные члены равны нулю ($b_1 = b_2 = \ldots b_n = 0$), иначе — неоднородной. Систему (1.1) можно записать в матричном виде:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{1.2}$$

или

$$A\overrightarrow{x} = \overrightarrow{b}, \tag{1.3}$$

где A - матрица размером $n \times n$, x - искомый вектор, $\overrightarrow{b}$ - заданный вектор.

Из курса линейной алгебры известно, что если определитель матрицы A отличен от нуля, то решение $\overrightarrow{x}$ существует и единственно. Однако, при реализации любого численного алгоритма выполнение этого условия еще не гарантирует нахождение достоверного решения. Решить систему (1.3) возможно, по крайней мере, двумя способами — методом

Крамера или методом Гаусса. Для большинства вычислительных задач характерным является большой порядок матрицы A. При больших n первый способ, основанный на вычислении определителей, требует порядка $n!$ арифметических действий, в то время как метод Гаусса — только $O(n^3)$ действий. Поэтому метод Гаусса в различных вариантах широко используется при решении задач линейной алгебры [1].

Методы численного решения СЛАУ (1.3) можно разделить на две группы: прямые и итерационные методы. В прямых методах решение $\overrightarrow{x}$ системы (1.3) находится за заранее известное конечное число арифметических действий. Примером прямого метода является метод Гаусса. Итерационные методы (методы последовательных приближений) состоят в том, что решение $\overrightarrow{x}$ СЛАУ (1.3) находится как предел при $k \to \infty$ последовательных приближений $\overrightarrow{x_k}$, где k — номер итерации. Как правило, за конечное число итераций этот предел не достигается [2].

Однако, при реализации любого численного алгоритма нахождение точного решения проблематично. В общем случае, численно полученное решение $\overrightarrow{x^*}$ отличается от точного решения $\overrightarrow{x}$.

Источниками ошибки численного решения являются:

- неопределенность в задании исходных данных (матрицы A и вектора $\overrightarrow{b}$),
- ошибки представления чисел в ЭВМ,
- дополнительные возмущения, возникающие при реализации вычислительного алгоритма посредством арифметики конечной разрядности (устойчивость численного алгоритма).

Проблема оценки величины ошибки δx численного решения $\overrightarrow{x^*}$ является едва ли не центральной проблемой реализации любого численного метода

$$\delta x = \frac{\| \overrightarrow{x^*} - \overrightarrow{x} \|}{\| \overrightarrow{x^*} \|}. \tag{1.4}$$

1.2.1 Понятие нормы вектора и нормы матрицы

Нормой элемента x линейного векторного пространства называется вещественное число $\|x\|$ такое, что

1) $\|x\| \geq 0$; $\|x\| = 0$ тогда и только тогда, когда $x = 0$,
2) $\|\lambda x\| = |\lambda| \cdot \|x\|$ для любого λ,
3) $\|x + y\| \leq \|x\| + \|y\|$.

Для вектора $\overrightarrow{x} = (x_1, x_2, ..., x_n)$ рассматриваются следующие нормы:

- $\|\overrightarrow{x}\|_1 = \sum\limits_{i=1}^{n} |x_i|$, что также имеет название метрика L1 или манхэттенское расстояние.
- $\|\overrightarrow{x}\|_2 = \sqrt{\sum\limits_{i=1}^{n} |x_i|^2}$, что также имеет название метрика L2 или евклидова норма.
- $\|\overrightarrow{x}\|_\infty = \max |x_i|$, что носит название максимальная норма.

Пусть R — основное поле и $R^{n \times n}$ — линейное пространство всех матриц с n строками и n столбцами, состоящих из элементов R. На пространстве матриц задана норма, если

каждой матрице $A \in R^{n\times n}$ ставится в соответствие неотрицательное действительное число $\|A\|$, называемое ее нормой, так, что

1) $\|A\| > 0$, если $A \neq 0$, и $\|A\| = 0$, если $A = 0$,
2) $\|A + B\| \leq \|A\| + \|B\|, \quad A, B \in K^{n\times n}$,
3) $\|\alpha A\| = |\alpha|\|A\|, \quad \alpha \in K, \quad A \in R^{n\times n}$.
4) $\|AB\| \leq \|A\|\|B\|$ для всех матриц A и B в $R^{n\times n}$.

Для матрицы $A = (a_{ij})$ порядка $n \times n$ рассматриваются нормы:

- Октаэдрическая (первая) норма $\|A\|_1 = \max\limits_{i=1\ldots n} \sum\limits_{j=1}^{n} |a_{ij}|$.
- Фробениусова (евклидова) норма матрицы $\|A\|_F = \sqrt{\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} |a_{ij}|}$.
- Равномерная $\|A\|_\infty = \max\limits_{j=1\ldots n} \sum\limits_{i=1}^{n} |a_{ij}|$.
- Полезной оказывается и спектральная норма $\|A\|_s = \max\limits_{i} \sqrt{\lambda_i}$, где λ_i - собственные числа матрицы $A^T \cdot A$.

1.2.2 Число обусловленности

Все методы оценки величины ошибки решения системы (1.3) используют значение числа обусловленности матрицы системы. Последнее определяется разными способами, но, по сути, всегда является коэффициентом пропорциональности между вариацией исходных данных (значений элементов матрицы и вектора правой части) и соответствующей вариацией решения.

В общем случае, в силу наличия ошибок округления и возможных погрешностях исходных данных, всегда решается возмущённая задача

$$(A + \Delta A)\overrightarrow{x} = \overrightarrow{b}, \tag{1.5}$$

и полученное численное решение (вектор $\overrightarrow{x^*}$) удовлетворяет не исходному уравнению (1.3), а возмущённому уравнению (1.5). Это уравнение можно записать в следующем виде:

$$A\overrightarrow{x^*} = \overrightarrow{b} + \overrightarrow{r}, \tag{1.6}$$

где $\overrightarrow{r}$ — вектор невязки. Поскольку решение $\overrightarrow{x^*}$ является точным для задачи с внесенным в нее возмущением, то можем записать $\Delta A\overrightarrow{x^*} = \overrightarrow{b} - A\Delta\overrightarrow{x^*}$. Подставив (1.6) в это выражение, получим

$$\Delta A \cdot \overrightarrow{x^*} = -\overrightarrow{r}.$$

Из этого выражения и свойств нормы вытекают следующие выражения:

$$\| \overrightarrow{r} \| \leq \| \Delta A \| \cdot \| \overrightarrow{x^*} \|$$

и

$$\frac{\| \overrightarrow{r} \|}{\| \overrightarrow{x^*} \|} \leq \| \Delta A \| .$$

Величина нормы матрицы возмущения $\| \Delta A \|$ реально всегда мала по сравнению с нормой $\| A \|$, следовательно, значение $\| \overrightarrow{r} \|$ также всегда мало и не может быть объективным критерием точности получаемого решения [3]. Действительно,

$$\overrightarrow{r} = A\overrightarrow{x^*} - \overrightarrow{b} = A\overrightarrow{x^*} - A\overrightarrow{x} = A(\overrightarrow{x^*} - \overrightarrow{x}),$$

$$\overrightarrow{r} = A\Delta\overrightarrow{x},$$

$$\| \Delta\overrightarrow{x} \| = \| A^{-1} \| \cdot \| \overrightarrow{r} \| . \tag{1.7}$$

Отсюда хорошо видно, что при малых значениях нормы вектора $\overrightarrow{r}$ можно получить значительную погрешность решения, так как величина нормы обратной матрицы может быть очень большой

$$\delta x = \frac{\| \overrightarrow{x^*} - \overrightarrow{x} \|}{\| \overrightarrow{x} \|} \leq \frac{\| A^{-1} \| \cdot \| \overrightarrow{b} \|}{\| \overrightarrow{x} \|} \cdot \frac{\| \overrightarrow{r} \|}{\| \overrightarrow{b} \|}. \tag{1.8}$$

Если в правой части выражения (1.8) заменить $\| \overrightarrow{x} \|$ на $\| \overrightarrow{x^*} \|$ оценка точности решения пострадает слабо. Тогда

$$\delta x \leq \mu_1 \frac{\| \overrightarrow{r} \|}{\| \overrightarrow{b} \|}. \tag{1.9}$$

Здесь

$$\mu_1 = \frac{\| A^{-1} \| \cdot \| \overrightarrow{b} \|}{\| \overrightarrow{x^*} \|} \tag{1.10}$$

— естественное число обусловленности. Очевидно, что такое число может быть получено только после решения задачи. В силу этого факта, оно не получило широкого распространения.

Зависимость числа обусловленности от нормы вектора решения можно исключить, если провести простые преобразования (1.7)

$$\delta x \leq \| A^{-1} \| \cdot \| A \| \frac{\| \overrightarrow{r} \|}{\| \overrightarrow{x} \| \cdot \| A \|}$$

с учетом неравенства

$$\| \overrightarrow{b} \| \leq \| A \| \cdot \| \overrightarrow{x} \|$$

приводят к «легко» вычисляемой оценке погрешности численного решения

$$\delta x \leq \mu_2 \frac{\| \overrightarrow{r} \|}{\| \overrightarrow{b} \|}, \tag{1.11}$$

где

$$\mu_2 = \| A^{-1} \| \cdot \| A \| \tag{1.12}$$

— стандартное число обусловленности. Стоит отметить, что выражения (1.9) и (1.11) являются всего лишь оценками возможной ошибки решения. Они почти всегда дают завышенную оценку, иногда — сильно завышенную, ещё реже — заниженную.

Необходимость вычисления нормы обратной матрицы $\| A^{-1} \|$ существенным образом затрудняет вычисление оценки погрешности. Нахождение обращенной матрицы A^{-1} слишком «трудоёмкий» процесс, требующий большого количества операций для вычисления оценки возможной ошибки. Для практического применения не всегда нужно знать точное значение $\| A^{-1} \|$, а вполне можно обойтись оценкой нормы обратной матрицы.

1.2.3 Оценка величины нормы обратной матрицы

Обращение матрицы является самостоятельной и важной задачей, представляющей интерес не только для нахождения ошибок решения или решения СЛАУ. Алгоритм Гауса в сочетании с LU — разложением (представление матрицы A в виде произведения двух матриц, $A = LU$, где L — нижняя треугольная матрица, а U — верхняя треугольная матрица.) позволяет относительно просто получить обратную матрицу.

Однако при наличии LU — разложения матрицы A, задача обращения требует около n^3 дополнительных операций и примерно в 4 раза увеличит общее число операций.

Грубая оценка величины $\| A^{-1} \|$ может быть получена на основании выражения

$$\| A^{-1} \| \cong 1.5 \max_{1 \le i \le k} \frac{\| \overrightarrow{z_i} \|}{\| \overrightarrow{y_i} \|}, \tag{1.13}$$

где $\overrightarrow{z_i}$ — решение задачи $A\overrightarrow{z_i} = \overrightarrow{y_i}$, а $\overrightarrow{y_i}$ — псевдослучайные векторы. Несмотря на то, что замена исходной задачи на задачу (1.13) довольно груба, оценка нормы обратной матрицы вполне приемлема. При этом можно обойтись сравнительно небольшим количеством ($k \sim 3$) решением задачи (1.13).

Можно использовать и хорошо зарекомендовавшую себя процедуру DECOMP [4]. В этой процедуре идея предыдущего метода усовершенствована так, что единственный вектор $\overrightarrow{z}$ ищется из системы $A\overrightarrow{z} = \overrightarrow{e}$. Здесь $\overrightarrow{e}$ — вектор, имеющий компоненты ± 1, строится по принципу максимизации модулей компонентов вектора решения.

1.2.4 Анализ чувствительности решения к возмущениям исходных данных и составная формула определения ошибки

В численном анализе распространены две формы записи машинного представления задачи с неточно заданными исходными данными.

Первая:

$$A(E+P)\overrightarrow{x} = (E+D)\overrightarrow{b}, \tag{1.14}$$

где P и D — матрицы возмущения. Поскольку возмущения носят, как правило, случайный характер, то компоненты этих матриц можно записать в виде: $p_{i,j} = \varepsilon_A \cdot \alpha, \alpha \in [-1,+1]$, α — случайное число. $d_{i,j} = \varepsilon_b \cdot \beta$, при $i = j$ и $d_{i,j} = 0$, при $i \neq j$, $\beta \in [-1,+1]$, β — случайное число. ε_A и ε_b — максимальные относительные ошибки в задании элементов матрицы A и вектора $\overrightarrow{b}$.

Вторая:

$$(A+M)\overrightarrow{x} = (E+D)\overrightarrow{b}, \tag{1.15}$$

где: $m_{i,j} = a_{i,j}\alpha\varepsilon_A$ и $d_{i,j}$- определяются по приведенному выше выражению.

Ясно, что модель (1.14) приводится к виду (1.15), если $AP = M$, т.е. при $P = A^{-1}M$. Но отсюда следует, что при «плохих» матрицах A (при «больших» A^{-1}) результаты исследования чувствительности решения системы по (1.14) и (1.15) могут существенно различаться при одинаковых значениях ε_A.

Задавая в эксперименте различные значения $\varepsilon_A = k_{\varepsilon,A}\epsilon_M$, где ϵ_M — машинный эпсилон, $k_{\varepsilon,A} = 1, 10, 10^2, ...10^9, ...$ и $\varepsilon_b = k_{\varepsilon,b}\epsilon_M$, можно оценить приемлемую, с позиций требуемой точности решения, неопределенность в задании исходных данных. Это позволяет дать разумное обоснование требуемой точности измерения (вычисления) компонент исходных данных задачи (1.3). Одновременно, зная реальные величины ε_a и ε_b , можно оценить достоверность полученного решения. Последнему служит, так называемая, составная формула оценки ошибки.

В случае, если модель описывается выражением (1.14), то для численного решения $\overrightarrow{x^*}$ можно записать

$$A(E+P)\overrightarrow{x^*} = (E+D)\overrightarrow{b} + \overrightarrow{r}.$$

В предположении малости компонент матриц P, D и вектора $\overrightarrow{r}$ формула для оценки составной ошибки имеет следующий вид:

$$\delta x = \| P \| + \frac{\| A^{-1}D\overrightarrow{b} \|}{\| \overrightarrow{x^*} \|} + \frac{\| A^{-1}\overrightarrow{r} \|}{\| \overrightarrow{x^*} \|}. \tag{1.16}$$

Достоинством этой формулы является большая реалистичность, в сравнении с выражением (1.7), оценки погрешности задач с плохо обусловленными матрицами и возможность учета погрешности исходных данных. Аналогичную формулу можно вывести и для выражения (1.15):

$$\delta x = \frac{\| A^{-1}M\overrightarrow{x} \|}{\| \overrightarrow{x^*} \|} + \frac{\| A^{-1}D\overrightarrow{b} \|}{\| \overrightarrow{x^*} \|} + \frac{\| A^{-1}\overrightarrow{r} \|}{\| \overrightarrow{x^*} \|} \tag{1.17}$$

Эта формула несколько более трудоемка в сравнении с (1.16), так как дополнительно требуется оценить и $\frac{\| A^{-1}M\overrightarrow{x} \|}{\| \overrightarrow{x^*} \|}$. Однако это можно сделать аналогично оценке нормы

обращенной матрицы посредством решения системы $A\overrightarrow{z} = \overrightarrow{e}$ для нескольких случайных векторов $\overrightarrow{e}$.

Отметим, что приведенные выше оценки погрешности решения допускают следующую интерпретацию: «ошибка решения = неопределенность модели + неопределенность исходных данных + ошибки вычислений» В таком случае матрицу задачи можно интерпретировать как модель исследуемой системы, а правую часть — как вектор возбуждающих воздействий (исходные данные).

Интересно, что существуют ситуации, когда преднамеренное возмущение модели является хорошим способом получить правдоподобное решение. Если обусловленность матрицы задачи $\mu_2 \geq (\epsilon_M)^{-2/3}$, то надежда получить приемлемое решение крайне мала. Столь плохая обусловленность задачи говорит о значительной степени линейной зависимости строк (столбцов) матрицы системы. Однако внесение в нее малых возмущений порядка $n\sqrt{\epsilon_m}$ часто позволяет существенно улучшить обусловленность системы и, как правило, не приводит к заметному изменению исходной задачи. В практических целях, можно рекомендовать генерацию матрицы возмущения вида

$$M_d = n\sqrt{\epsilon_M} \parallel A \parallel \cdot E. \tag{1.18}$$

Задача (1.3) тем самым преобразуется в $(A + M_d)\overrightarrow{x} = \overrightarrow{b}$.

1.2.5 Итерационные методы

Итерационные методы решения СЛАУ (1.3) являются бесконечно-шаговыми и приводят лишь к приближенным решениям. Однако довольно часто применение прямых методов не эффективно. Например, для задач с сильно разряженными матрицами (они возникают при решении краевых задач для дифференциальных уравнений в частных производных, а также при анализе больших электронных цепей), итерационные алгоритмы требуют гораздо меньшей оперативной памяти, чем прямые методы. Для задач большой размерности трудоёмкость прямых алгоритмов становится чрезмерной. В этих обстоятельствах целесообразнее использовать итерационные алгоритмы.

Методы Якоби и Гаусса - Зейделя являются наиболее простыми из многочисленного семейства итерационных алгоритмов. Однако их свойства и особенности применения достаточно типичны для всего класса итерационных методов.

Представим матрицу A задачи $A\overrightarrow{x} = \overrightarrow{b}$ в виде трех слагаемых $A = A_L + D + A_U$. Здесь A_L — нижняя-треугольная матрица, у нее все элементы под главной диагональю и элементы главной диагонали равны нулю, D — диагональная, имеются только элементы главной диагонали, A_U — верхняя-треугольная матрица у которой все элементы под главной диагональю и элементы главной диагонали равны нулю.

$$A_L = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{pmatrix},\ D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix},\ A_U = \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}.$$

Пусть $a_{ii} \neq 0\ \forall i$, тогда матрица D^{-1} существует и решение системы (1.3) можно представить в виде:

$$\overrightarrow{x} = D^{-1}\overrightarrow{b} - D^{-1}(A_L + A_U)\overrightarrow{x}.$$

Вектор $\overrightarrow{x^*}$ является решением данной системы и должен удовлетворять этому уравнению. На базе этой формулы построим численный метод, генерирующий следующие последовательности векторов $\overrightarrow{x}_k$:

$$\overrightarrow{x}_{k+1} = D^{-1}\overrightarrow{b} - D^{-1}(A_L + A_U)\overrightarrow{x}_k. \tag{1.19}$$

Рассмотрим поведение такой последовательности.

$$D(\overrightarrow{x}_{k+1} - \overrightarrow{x}_k) + (A_L + D + A_U)\overrightarrow{x}_k - \overrightarrow{b} = 0,$$

$$D(\overrightarrow{x}_{k+1} - \overrightarrow{x}_k) = \overrightarrow{b} - A\overrightarrow{x}_k.$$

Из последнего уравнения хорошо видно, что если последовательность $\{\overrightarrow{x}_k, k = 0, 1, ...\}$ сходиться, то $\lim\limits_{k\to\infty} \overrightarrow{x}_k = \overrightarrow{x^*}$. Действительно, если $(\overrightarrow{x}_{k+1} - \overrightarrow{x}_k) \to 0$, то и $\overrightarrow{b} - A\overrightarrow{x}_k \to 0$, т.е. $\overrightarrow{x}_k \to \overrightarrow{x^*}$, при $k \to \infty$. Вычислительная схема (1.19) названа методом **Якоби**. Для этого метода характерно, что решение на $(k+1)$-ом шаге явно зависит от решения на k-ом шаге.

Попытка модифицировать схему (1.19), так чтобы компоненты решения на $(k+1)$ шаге сразу же вовлекались в решение, привела к появлению усовершенствованного метода — **метода Гаусса-Зейделя**. Его вычислительная схема имеет вид:

$$\overrightarrow{x}_{k+1} = D^{-1}\overrightarrow{b} - D^{-1}A_L\overrightarrow{x}_{k+1} - D^{-1}A_U\overrightarrow{x}_k \tag{1.20}$$

или

$$(D + A_L)(\overrightarrow{x}_{k+1} - \overrightarrow{x}_k) = \overrightarrow{b} - A\overrightarrow{x}_k.$$

Из последнего соотношения следует аналогичный вывод о сходимости $\{\overrightarrow{x}_k\}$ к решению $\overrightarrow{x^*}$.

Каноническая запись всякого итерационного метода имеет вид:

$$B_k\frac{\overrightarrow{x}_{k+1} - \overrightarrow{x}_k}{\tau_k} = \overrightarrow{b} - A\overrightarrow{x}_k \tag{1.21}$$

Здесь B_k — подпоследовательность невырожденных матриц и τ_k — итерационный параметр. Если матрица $B_k = B$ и параметр $\tau_k = \tau$, то метод стационарный, если $B_k = E$ —

метод явный.

1.2.6 Условие сходимости итерационного метода

Главной отличительной чертой всех итерационных процедур является сильная зависимость самой возможности получения решения от свойств итерационной матрицы, которая интегрирует свойства матрицы задачи и свойства метода. Пусть $\overrightarrow{x^*}$ — решение системы $A\overrightarrow{x} = \overrightarrow{b}$ и $\delta\overrightarrow{x}_k = \overrightarrow{x}_k - \overrightarrow{x^*}$ — погрешность k-го приближения. Тогда $\overrightarrow{x}_k = \delta\overrightarrow{x}_k + \overrightarrow{x^*}$ и $\overrightarrow{x}_{k+1} = \delta\overrightarrow{x}_{k+1} + \overrightarrow{x^*}$ подставив эти выражения в (1.21) и преобразовав получим выражение для погрешности

$$\delta\overrightarrow{x}_{k+1} = (E - \tau B^{-1}A)\delta\overrightarrow{x}_k.$$

Матрица $S = E - \tau B^{-1}A$ является матрицей перехода от k-ой к $(k+1)$-ой итерации. Она носит название итерационной матрицы. Для метода Якоби эта матрица имеет следующий вид $S = E - D^{-1}A$, для метода Гаусса-Зейделя $S = E - (D + A_L)^{-1}A$.

Итерационный метод (1.21) сходится при любом начальном приближении тогда и только тогда, когда все собственные значения матрицы S по модулю меньше единицы $| \lambda_{max}(S) | < 1$.

Иногда пытаются установить чисто качественные, «визуальные» признаки сходимости того или иного алгоритма. При этом стремятся найти связь между малостью собственных чисел итерационной матрицы и видом матрицы A СЛАУ . Например, для метода Якоби хорошую сходимость ожидают при хорошей аппроксимации матрицы системы A диагональной матрицей. Для метода Гаусса-Зейделя можно ожидать хорошую сходимость процедуры Гаусса-Зейделя для «почти» нижняя-треугольных матриц. Однако такие признаки всегда носят только качественный характер и не являются условиями сходимости.

1.3 Методы решения СЛАУ и оценки их погрешности в среде Wolfram Mathematica

1.3.1 Нахождение точного и численного решения СЛАУ

Пусть $A = \begin{pmatrix} 7 & 3 & 6 \\ 9 & 3 & 7 \\ 11 & 6 & 9 \end{pmatrix}$; $\overrightarrow{b} = \begin{pmatrix} 1 \\ 13 \\ 7 \end{pmatrix}$. Найти решение уравнения $A\overrightarrow{x} = \overrightarrow{b}$.

Решить систему линейных уравнений можно несколькими способами. Для всех этих способов нам потребуется ввести матрицу системы и столбец свободных членов. Для некоторых способов необходимо ввести вектор неизвестных.

```
In[•]:= (*Матрица системы*)
        A = {{7, 3, 6}, {9, 3, 7}, {11, 6, 9}};
        (*Вектор свободных членов*)
        b = {{1}, {13}, {7}};
        (*Создадим вектор неизвестных для точного решения*)
        X = Table[x_i, {i, 1, 3}];
        (*Создадим вектор неизвестных для численного решения*)
        Xn = Table[xn_i, {i, 1, 3}];
```

Встроенные методы.

1) Домножение на обратную матрицу.

```
In[•]:= (*Команда Inverse[ ] – создает обратную матрицу
            Для перемножения матриц используется символа точка . *)
        Inverse[A].b
```

$$\text{Out}[\bullet]= \left\{\left\{\frac{41}{3}\right\}, \left\{-\frac{8}{9}\right\}, \left\{-\frac{46}{3}\right\}\right\}$$

2) С использованием команды Solve[].

```
In[•]:= (*Команда Solve[expr,vars] – находит точное решение *)
        Solve[A.X == b, X]
```

$$\text{Out}[\bullet]= \left\{\left\{x_1 \to \frac{41}{3}, x_2 \to -\frac{8}{9}, x_3 \to -\frac{46}{3}\right\}\right\}$$

3) С использованием команды NSolve[].

```
In[•]:= (*Команда NSolve[expr,vars] – находит численное решение  *)
        NSolve[A.Xn == b, Xn]
```

$$\text{Out}[\bullet]= \{\{xn_1 \to 13.6667, xn_2 \to -0.888889, xn_3 \to -15.3333\}\}$$

4) С использованием команды LinearSolve[].

```
In[•]:= (*Команда LinearSolve[m,b] – находит точное решение  *)
        X = LinearSolve[A, b]
```

$$\text{Out}[\bullet]= \left\{\left\{\frac{41}{3}\right\}, \left\{-\frac{8}{9}\right\}, \left\{-\frac{46}{3}\right\}\right\}$$

1.3.2 Нахождение чисел обусловленности

Для матрицы СЛАУ из п. 1.3.1 определим естественное и стандартное числа обусловленности. В качестве нормы матрицы будем использовать первую норму.

```
In[•]:= (*Естественное число обусловленности*)
        (*Команда Norm[expr] производит вычисление нормы*)
        (*Команда N[expr] выдает численое значение выражения expr*)
        μ1 = N[Norm[Inverse[A], 1] * Norm[b, 1] / Norm[Xn, 1]]
```

$$\text{Out}[\bullet]= 3.12266$$

```
In[•]:= (*Стандартное число обусловленности*)
        μ2 = N[Norm[Inverse[A], 1] * Norm[A, 1]]
Out[•]= 120.
```

Сравним точное значение стандартного числа обусловленности с его оценкой. Для этого грубо оценим норму обратной матрицы A по формуле (1.13).

```
In[•]:= (*Грубая оценка нормы обратной матрицы*)
        (*Команда Max [x1,x2,...] находит максимальное значение из всех аргументов*)
        (*Команда Table [expr,n] создает список элеменов *)
        (*Команда RandomReal [xmax] создает случайное число от 0 до xmax*)
        GrubNormA = 1.5 * Max[Table[{yi = Table[RandomReal[10], {i, 1, 3}];
                    zi = LinearSolve[A, yi]}; N[Norm[zi, 1] / Norm[yi, 1]], {i, 1, 5}]]
Out[•]= 4.83345
```

Найдем оценку стандартного числа обусловленности.

```
In[•]:= (*Оценка стандартного числа обусловленности*)
        μ2 G = GrubNormA * Norm[A, 1]
Out[•]= 130.503
```

Как видно из полученных результатов, грубая оценка и точное значение стандартного числа обусловленности слабо отличаются.

1.3.3 Определение относительной погрешности решения и её оценка

Найдем относительную погрешность решения (1.4).

```
In[•]:= (*Относительная ошибка решения*)
        (*X – Точное решение СЛАУ, Xn – численное решение СЛАУ*)
        δx = Norm[X – Xn, 1] / Norm[Xn, 1]
Out[•]= 0.0000260222
```

Сравним найденную относительную погрешность решения с оценкой погрешности решения, полученной по формулам (1.9), (1.11). Для этого найдем вектор невязки $\overrightarrow{r}$ и его норму для численного решения СЛАУ из п. 1.3.1.

```
In[•]:= (*Вектор невязки*)
        r = b – A.Xn
Out[•]= {{-0.004}, {-0.005}, {-0.006}}

In[•]:= (*Норма вектора невязки*)
        Norm[r, 1]
Out[•]= 0.015

In[•]:= (*Оценка погрешности решения*)
        δxest = μ1 * Norm[r, 1] / Norm[b, 1]
Out[•]= 0.00223047
```

In[•]:= $\delta x_{std} = \mu_2 * \text{Norm}[r, 1] / \text{Norm}[b, 1]$

Out[•]= 0.0857143

Обе полученные погрешности — это всего лишь оценки возможной погрешности решения. Как уже говорилось, они часто дают завышенный результат, и очень редко заниженный.

1.3.4 Реализация метода Гаусса

Решать СЛАУ в среде Wolfram Mathematica можно и методами, не входящими в пакет. Например, метод Гаусса. Реализовать этот метод очень просто.

```
(*Метод Гаусса*)
gauss[n_?MatrixQ, q_?MatrixQ] := Block[
    {m = Join[n, q, 2], st, sb, a, b, s, e, k, r, x},
    (*Размерность расширенной матрицы,
    столбец свободных членов,матрица системы*)
    {st, sb} = Dimensions[m]; b = m[[All, sb]]; a = m[[1 ;;, 1 ;; st]];
    (*Проверяем, имеет ли система решение*)
    If[Det[a] == 0, Print["Определитель системы равен нулю"];
      Abort[]];
    (*Прямой ход метода Гаусса*)
    Do[
      (*Перестановка строк при нулевом ведущем элементе*)
      If[m[[j, j]] == 0, s = Take[m[[All, j]], {j + 1, st}];
          e = Last[Select[s, (# 6 != 0) &]];
          k = Last[Position[m[[All, j]], e] // Flatten];
          m = swp[m, j, k]];
    (*Обнуление поддиагональных элементов*)
        m[[j]] = m[[j]] / m[[j, j]] // N;
        Do[m[[i]] = m[[i]] - m[[j]] × m[[i, j]] // N, {i, 2 + j - 1, st}], {j, 1, st}]
    (*Обратный ход метода Гаусса*)
    Table[x_i = m[[i, sb]] - Sum[m[[i, j]] * x_j, {j, i+1, st}], {i, st, 1, -1}];

    (*Векторы решения и невязки*)
    x = .; x = Table[x_i, {i, 1, st}];
        r = b - a.x // Chop;
        Print["Решение x=", x, "с вектором невязки r=", r, "."]];
```

Опробуем этот метод для решения тестовой задачи.

In[•]:= gauss[A, b]

Решение x={13.6667, −0.888889, −15.3333}с вектором невязки r={0, 0, 0}.

Как можно увидеть, результат, полученный данным методом, хорошо согласуется с

результатом из раздела 1.3.1.

1.3.5 Реализация метода Якоби и Гаусса-Зейделя

Реализуем метод Якоби для решения СЛАУ.

```
(*Построим алгоритм решения СЛАУ методом Якоби*)
(*Аргументами будут Матрица системы – sysMatrix,
столбец свободных членов – freeCol, *)
(*погрешность решения – epsilon, начальное приближение – firstPribl*)
Jacobi[sysMatrix_, freeCol_, epsilon_, firstPribl_] := Module[
    {A = sysMatrix, b = freeCol, lastStep = firstPribl,  nextStep, Dm, ALU, S, R, i, n},
     (*Узнаем размерность матрицы СЛАУ*)
    n = Length[A];
    (*Строим диаганальную матрицу Dn*)
    Dm = Table[If[i == j, A[[i, j]], 0], {i, 1, n}, {j, 1, n}];
    (*Сумма нижне–треугольной и верхне–треугольной матрицы*)
    ALU = A – Dm;
    (*Создаем итерационную матрицу и вектор свободных членов метода*)
    S = N[Inverse[Dm].ALU];
    (*Матрица свободных членов в методе*)
    R = N[Inverse[Dm].b];
    (*Первое приближение*)
    nextStep = R – S.lastStep;
    (*Первое приближение*)
    nextStep = R – S.lastStep;
    i = 0;
    (*В этом цикле формируем решение*)
       (*Сравниваем решения на предыдущем и следующем шаге,
       пока погрешность не станет меньше наперед заданной*)
       While[Max[Abs[lastStep – nextStep]] > epsilon,
         (*Запоминаем решение на предыдущем шаге*)
         lastStep = nextStep;
         (*Получаем решение на следующем шаге*)
         nextStep = R – S.lastStep;
         (*Ограничение количества операций*)
         If[i > 150, Break[]];
    i++;
          ];
      (*Вывод на экран решения и количества шагов*)
      Print[" Решение x= ", lastStep, " найдено за ", i , " итераций" ]
      ]
```

Применим этот метод для решения следующей задачи: пусть дана матрица и вектор сво-

бодных членов

$$A = \begin{pmatrix} 20 & 2 & 1 & 1 \\ 2 & 12 & -3 & -4 \\ 5 & -3 & 13 & 0 \\ 6 & -2 & -1 & 15 \end{pmatrix}; \overrightarrow{b} = \begin{pmatrix} 5 \\ 4 \\ -3 \\ 7 \end{pmatrix}.$$

Найдем решение уравнения $A\overrightarrow{x} = \overrightarrow{b}$ с точностью до 0.00001 и Сравним полученный результат с решением, полученным встроенным методом.

In[•]:= **(∗Вводим матрицу A∗)**

$$\mathrm{A} = \begin{pmatrix} 20 & 2 & 1 & 1 \\ 2 & 12 & -3 & -4 \\ 5 & -3 & 13 & 0 \\ 6 & -2 & -1 & 15 \end{pmatrix};$$

(∗Вводим столюец свободных членов b∗)

$$\mathrm{b} = \begin{pmatrix} 5 \\ 4 \\ -3 \\ 7 \end{pmatrix};$$

(∗Вводим начальное приближение ∗)

$$\mathrm{Prib} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix};$$

In[•]:= **(∗Точное решение СЛАУ ∗)**

```
N[LinearSolve[A, b]]
(∗Найденное методом Якоби ∗)
Jacobi[A, b, 0.00001, Prib]
```

Out[•]= {{0.201195}, {0.386055}, {−0.219062}, {0.423059}}

Решение x= {{0.201199}, {0.386043}, {−0.219071}, {0.42305}} найдено за 12 итераций

Решения, найденные двумя методами, совпали.

Модифицируем метод Якоби в метод Гаусса-Зейделя для решения СЛАУ.

```
(∗Построим алгоритм решения СЛАУ методом Гауса−Зейделя∗)
(∗Аргументами будут Матрица системы − sysMatrix,
столбец свободных членов − freeCol, ∗)
(∗погрешность решения − epsilon, начальное приближение − firstPribl∗)
GaussZedel[sysMatrix_, freeCol_, epsilon_, firstPribl_] :=
  Module[{A = sysMatrix, b = freeCol,
    lastStep = firstPribl,  nextStep, Dm, AL, AU, S, R, M, i, n},
   (∗Узнаем размерность матрицы СЛАУ∗)
   n = Length[A];
```

```
(*Строим диаганальную матрицу Dm*)
Dm = Table[If[i == j, A[[i, j]], 0], {i, 1, n}, {j, 1, n}];
(*Строим нижнюю − треугольную матрицу*)
AL = Table[If[i >= j, 0, A[[i, j]]], {i, 1, n}, {j, 1, n}];
(*Строим верхнюю − треугольную матрицу*)
AU = Table[If[i <= j, 0, A[[i, j]]], {i, 1, n}, {j, 1, n}];
(*Создаем итерационную матрицу и вектор свободных членов*)
M = Inverse[IdentityMatrix[n] + Inverse[Dm].AL];
(*Итерационная матрица*)
S = N[M.Inverse[Dm].AU];
(*Матрица свободных членов в методе*)
R = N[M.Inverse[Dm].b];
i = 0;
(*Первое приближение*)
nextStep = R − S.lastStep;
(*В этом цикле формируем решение*)
(*Сравниваем решения на предыдущем и следующем шаге,
пока погрешность не станет меньше наперед заданной*)
While[Max[Abs[lastStep − nextStep]] > epsilon,
   (*Запоминаем решение на предыдущем шаге*)
   lastStep = nextStep;
   (*Получаем решение на следующем шаге*)
   nextStep = R − S.lastStep;
(*Ограничение количества операций*)
      If[i > 150, Break[]];
      i++;
      ];
  (*Вывод на экран решения и количества шагов*)
  Print["Решение x= ", lastStep, " найдено за ", i , " итераций"]
  ]
```

Сравним решения задачи полученным методом Якоби и методом Гаусса - Зейделя.

```
In[ ]:= (*Точное решение СЛАУ *)
        N[LinearSolve[A, b]]
        (*Найденное методом Гаусса−Зейделя *)
        GaussZedel[A, b, 0.00001, Prib]
Out[ ]= {{0.201195}, {0.386055}, {−0.219062}, {0.423059}}

        Решение x= {{0.201194}, {0.386059}, {−0.219056}, {0.423065}} найдено за 7 итераций
```

Два метода дали одинаковый результат.

1.4 Задание на лабораторную работу №1

Необходимо провести численные эксперименты в среде Wolfram Mathematica / MATLAB, на основании которых:

1) Сравнить между собой естественное и стандартное числа обусловленности матрицы, а также — точное значение стандартного числа обусловленности с его грубой оценкой, вычисленной по формуле (1.13).
2) Оценить точность решений, получаемых методом исключения Гаусса для систем с одинаково хорошо обусловленными матрицами порядка от 3 до 15:
   - провести анализ точности, как функции порядка матрицы;
   - сравнить фактически получаемую ошибку с ее оценками (1.9), (1.11).
3) Выполнить задания п.2 для систем с одинаково плохо обусловленными матрицами.
4) Для разных систем (от «очень хороших» до «очень плохих») и одного порядка ($n > 3$) оценить точность решений при внесении малого возмущения в задачу (выражения (1.14), (1.15)). Результаты анализа представить в виде зависимости относительной точности решения от числа обусловленности. Обратить внимание на величину нормы вектора невязки и проследить ее зависимость от обусловленности системы и связь с фактической ошибкой решения.
5) Исследовать возможность улучшения обусловленности задачи посредством внесения малого возмущения M_d (выражение (1.18)) в матрицу системы.
6) Для 2-3 задач с «хорошей» матрицей ($\mu_2 = 10^2 - 10^4$) посредством внесения в матрицу системы возмущений различной величины (выбрать одну из моделей возмущения (1.14), (1.15)) сделать заключение о приемлемой для получения требуемой (наперед заданной) точности решения степени неопределенности в задании исходных данных.
7) Повторить эксперимент п.6 для 2-3 задач с плохо обусловленной матрицей.
8) Выполняя п.п. 6 и 7 , исследовать работоспособность различных методов оценки ошибок решения (выражения (1.9), (1.11), (1.16) или (1.17)) при наличии возмущения левой части системы.
9) Применить для решения нескольких систем из пунктов 2-4 итерационные методы Якоби и Гаусса-Зейделя:
   - проверить реализацию задаваемого критерия точности. Исследованием спектра матрицы S проверить выполнение теоремы сходимости стационарного метода;
   - выявить взаимосвязь скорости сходимости итерационного процесса с величиной спектрального радиуса матрицы S.
10) Провести исследование влияния вида доминирования матрицы задачи на сходимость процедур Якоби и Гаусса-Зейделя.

2. Экспериментальное исследование методов построения приближающего полинома

2.1 Цель работы

В данной работе будут рассмотрены и опробованы методы построения приближающего полинома к заданной функции.

2.2 Интерполирование и приближение функций

Задача интерполирования состоит в том, чтобы по значениям функции $f(x)$ в нескольких точках отрезка восстановить ее значения в остальных точках этого отрезка. Такую задачу можно решить различными способами. Задач, в которых требуется замена исходной функции на приближающую, довольно много, например, когда известны результаты измерения $y_k = f(x_k)$ некоторой физической величины $f(x)$ в точках x_k, $k = 0, 1, ..., n$, и требуется определить ее значения в других точках. Интерполирование используется также при сгущении таблиц, когда вычисление значений $f(x)$ трудоемко. Иногда возникает необходимость приближенной замены или аппроксимации данной функции другими функциями, которые легче вычислить.

Общая постановка задачи приближения функции формулируется следующим образом:

- пусть нам дана функция $f(x) \in F$,
- существует класс функций $B = \{\phi_1(x), \phi_2(x), ...\}$,
- необходимо найти такую функцию $\phi^*(x)$, чтобы в заранее определенном смысле $f(x) \approx \phi^*(x)$.

Таким образом, функция $\phi^*(x)$ должна являться решением задачи $\| f(x) - \phi^*(x) \| \to \min\limits_{\phi \in B}$.

Ясно, что выбор класса B приближающих функций зависит от самой функции $f(x)$. Например, если $f(x)$ непрерывна на промежутке $[a, b]$ и достаточно гладкая, то в качестве функций $\phi^*(x)$ можно выбрать класс алгебраических полиномов:

$$\phi_n(x) = P_n(x) = \sum_{i=0}^{n} a_i x^i.$$

Эти функции легко дифференцируются и интегрируются.

Если приближаемая функция периодична и непрерывна, т.е. $f(x) = f(x + T)$, то

функцию $\phi^*(x)$ удобно строить как линейную комбинацию тригонометрических полиномов

$$\phi_n(x) = \sum_{i=0}^{n} [a_i \cos(\frac{2\pi}{T} kx) + b_i \sin(\frac{2\pi}{T} kx)].$$

Выбор критерия близости $f(x)$ и $\phi(x)$ позволяет разделить методы построения приближающей функции на два больших класса — интерполяция и аппроксимация. В данной работе в качестве класса функций B мы ограничимся лишь классом алгебраических полиномов.

2.2.1 Интерполирование функций

Пусть на отрезке $[a, b]$ задана функция $f(x)$. Решение задачи приближения функции методами интерполирования предполагает выполнение условия

$$f(x_i) = \phi(x_i), i = 0, ..., n$$

в узлах сетки $\{x_i\}$. Если в качестве приближающей функции выбран класс алгебраических полиномов, то это условие может быть записано в следующем виде:

$$f(x_i) = P_n(x_i) = \sum_{k=0}^{n} a_k x_i^k. \tag{2.1}$$

Задача 2.1 имеет решение, если степень полинома $n = m - 1$, где m — количество точек интервала [a,b], в которых задана функция $f(x)$. Таким образом, многочлен n-ой степени может обеспечить совпадение с приближаемой функцией $f(x)$ в $(n + 1)$ точке конечного интервала.

Метод неопределенных коэффициентов

Легко заметить, что коэффициенты a_k являются компонентами вектора решения $\overrightarrow{a}$ системы линейных уравнений:

$$\begin{cases} x_0^0 a_0 + x_0^1 a_1 + \cdots + x_0^n a_n = f(x_0) \\ x_1^0 a_0 + x_1^1 a_1 + \cdots + x_1^n a_n = f(x_1) \\ \cdots \\ x_n^0 a_0 + x_n^1 a_1 + \cdots + x_n^n a_n = f(x_n) \end{cases} \tag{2.2}$$

Или записав в другом виде:

$$W\overrightarrow{a} = \overrightarrow{f}, \tag{2.3}$$

где матрица W является матрицей Вандермонда:

$$W = \begin{pmatrix} x_0^0 & x_0^1 & \cdots & x_0^n \\ x_1^0 & x_1^1 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \cdots & x_n^n \end{pmatrix}, \tag{2.4}$$

а компоненты вектора $\overrightarrow{f}$ являются значениями функции в точках x_i,

$$\overrightarrow{f} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_n) \end{pmatrix}.$$

Отметим, что вычисление коэффициентов полинома посредством решения системы (2.3) в вычислительной практике используется крайне редко. Причиной этого является плохая обусловленность матрицы (2.4), приводящая к заметному росту погрешности в выполнении условий интерполирования уже при сравнительно невысоких порядках полинома. К этому следует добавить, что вычислительные затраты реализации метода пропорциональны n^3.

Метод Лагранжа

Более высокую устойчивость построения интерполяционного полинома демонстрирует метод Лагранжа. Интерполяционный полином, очевидно, можно построить в форме:

$$P_n(x) = \sum_{k=0}^{n} g_k(x) f(x_k), \tag{2.5}$$

где $g_k(x)$ — многочлены n-ой степени, обладающие следующими свойствами:

$$g_k(x) = \begin{cases} 1, & \text{при } x = x_i, i = k \\ 0, & \text{при } x = x_i, i \neq k \end{cases}$$

Таким свойством обладает, в частности, полином вида:

$$L_k = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)}.$$

Тогда

$$P_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k) = \sum_{k=0}^{n} B_k \prod_{i=0, i \neq k}^{n} (x - x_i), \tag{2.6}$$

где коэффициенты B_k определются выражением

$$B_k = \frac{f(x_k)}{\prod\limits_{i=0, i\neq k}^{n}(x_k - x_i)}. \tag{2.7}$$

Выражения (2.6) и (2.7) совместно образуют интерполяционную формулу Лагранжа. Отметим, что коэффициенты Лагранжа $L_k(x)$ не зависят от значений интерполируемой функции в узлах. Это существенно снижает вычислительные затраты при интерполировании системы функций на общей сетке.

Интерполяционный полином в форме Ньютона

Интерполяционный многочлен легко определяется, если его построить в виде:

$$P_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + ... + C_n(x - x_0)(x - x_1)...(x - x_{n-1}). \tag{2.8}$$

Исходя из условия интерполяции $f(x_i) = P_n(x_i)$ для коэффициентов C_i получим систему уравнений треугольного вида

$$\begin{cases} f(x_0) = C_0 \\ f(x_1) = C_0 + C_1(x_1 - x_0) \\ f(x_2) = C_0 + C_1(x_1 - x_0) + C_2(x_2 - x_0)(x_2 - x_1) \\ \dots \\ f(x_n) = C_0 + C_1(x_n - x_0) + C_2(x_n - x_0)(x_n - x_1) + ... + C_n(x_n - x_0)(x_n - x_1)...(x_n - x_{n-1}). \end{cases}$$

Из этой системы легко находятся коэффициенты C_n:

$$C_0 = f(x_0),$$

$$C_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

$$C_2 = \frac{\frac{f(x_2)-f(x_0)}{x_2-x_0} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}, ...$$

и так далее.

Величины, стоящие в правой части приведённых выше равенств, получили название разделённых разностей, соответственно, нулевого, первого и второго порядков. Для них приняты обозначения $f[x_i], f[x_i, x_{i-1}], f[x_i, x_{i-1}, x_{i-2}]$ и т.д. С учётом этих обозначений выражение (2.8) можно переписать в виде:

$$\begin{aligned} P_n(x) = f[x_0] + f[x_1, x_0](x - x_0) + f[x_2, x_1, x_0](x - x_0)(x - x_1) + ...+ \\ +f[x_n, x_{n-1}, ..., x_0](x - x_0)(x - x_1)...(x - x_{n-1}) \end{aligned} \tag{2.9}$$

Можно показать, что

$$f[x_k, x_{k-1}, ..., x_0] = \sum_{i=0}^{k} \frac{f(x_i)}{(x_i - x_0)(x_i - x_1)...(x_i - x_{i-1})(x_i - x_{i+1})...(x_i - x_k)}. \quad (2.10)$$

Выражения (2.9) и (2.10) определяют интерполяционный полином в форме Ньютона. Вычисление полинома в Ньютоновской форме удобно при последовательном дополнении сетки $(n+2)$-м узлом и наращивании порядка интерполяционного полинома. При этом необходимо вычислить лишь одно дополнительное слагаемое $f[x_{n+1}, x_n, ...x_0](x - x_0)(x - x_1)...(x - x_n)$ в выражении (2.9).

2.2.2 Погрешность интерполирования

Пусть $f(x)$ — непрерывная и достаточно гладкая на интервале $[a, b]$ функция. $P_n(x)$ — интерполяционный полином степени n, построенный на сетке $x_i \in [a, b]$, $i = 1, ..., n$. Очевидно, что при построении полинома было выполнено условие интерполяции $P_n(x_i) = f(x_i)$. Таким образом, остается открытым вопрос о качестве интерполяционного полинома, т.е. какова степень приближения полинома $P_n(x)$ к исходной функции $f(x)$ при $x \neq x_i, x \in [a, b]$.

Рассмотрим следущую функцию

$$r(x) = f(x) - P_n(x), x \in [a, b],$$

которая часто называется остаточным членом интерполяционной формулы. Ясно, что $r(x) = 0 \ \forall x_i$ и $r(x) \neq 0$ при $x \neq x_i$.

Оценку величины остаточного члена интерполяционной формулы можно определить по формуле

$$|r(x)| \leq \frac{M_{n+1}}{(n+1)!} \prod_{i=0}^{n} (x - x_i), \quad (2.11)$$

где $M_{n+1} = \max\limits_{x \in [a,b]} |f^{(n+1)}(x)|$ обозначает максимум абсолютного значения производной $(n+1)$-го порядка функции $f(x)$ на интервале интерполирования $[a, b]$.

Это выражение позволяет сделать следующие выводы:

- может быть получена сколь угодно малая погрешность приближения, так как M_{n+1} — ограниченная функция,
- при фиксированном порядке полинома величина $\max\limits_{x \in [a,b]} |r(x)|$ является функцией расположения узлов сетки.

2.2.3 Выбор узлов интерполирования

Не всегда выбор равномерной сетки, узлы которой расположены на равном расстоянии друг от друга, оптимален для интерполирования и аппроксимирования. Величина

уклонения $\|f(x)-P_n(x)\|$ зависит, в частности, от расположения узлов интерполяции $\{x_i\}$ на интервале $[a,b]$. Выбором узлов сетки возможно влиять на поведение функции уклонения. Близкой к оптимальной, в смысле минимума функции $\max\limits_{x\in[a,b]}\|f(x)-P_n(x)\|$, является, так называемая, чебышевская сетка. Узлами этой сетки выступают корни многочлена Чебышева.

Многочленом Чебышева называется функция $T_n(x)=\cos(n\arccos(x))$, где n — натуральное число или нуль, $x\in[a,b]$. Рекуррентная формула для многочленов имеет следующий вид

$$T_n(x)=2xT_{n-1}(x)-T_{n-2}(x).$$

Координаты узлов сетки Чебышева для произвольного отрезка $[a,b]$ определяются выражением

$$x_k=\frac{b+a}{2}+\frac{b-a}{2}\cos(\frac{2k+1}{2(n+1)}\pi). \tag{2.12}$$

Эта сетка обладает рядом замечательных свойств. Так, если функция $f(x)$ обладает непрерывной первой производной, то интерполяционный процесс на чебышевской сетке всегда сходится. Данная сетка также позволяет:

- построить интерполяционный полином близкий к полиному наилучшего равномерного приближения,
- обеспечивает, в сравнении с равномерной сеткой, существенно более высокую устойчивость полинома к погрешностям задания исходных данных,
- многочлен Лагранжа, построенный по специально подобранным узлам Чебышева, будет многочленом наилучшего приближения $f(x)$ на отрезке $[a,b]$.

Однако следует иметь в виду, что достоинства сетки Чебышева не предопределяют абсолютную целесообразность её использования. Это связано с тем, что:

- она может оказаться не всегда удобной для вычисления соответствующих значений функции $f(x)$,
- сходимость интерполяционного процесса может потребовать построения другой сетки.

В общем случае, для каждой функции существует своя система сеток, на которых интерполяционный процесс сходится.

2.2.4 Среднеквадратичное приближение функций

Задача приближения возникает во многих областях прикладных исследований, например, при статистической обработке данных эксперимента с использованием регрессивного анализа, при оценивании параметров моделей, в задачах фильтрации и т.п.

Когда уровень неопределенности в задании приближаемой функции $f(x_i)$, $i=1..m$, достаточно велик, что характерно для обработки экспериментальных данных, бессмысленно требовать выполнения условий интерполирования. Кроме того, число точек задания функции $f(x_i)$ часто весьма велико. Все это делает применение интерполирования

малоперспективным по причинам плохой обусловленности задачи, высокой размерности и проблем сходимости процесса интерполяции.

Имеет смысл вместо построения интерполяционной функции строить приближающую функцию на основе среднеквадратичного критерия близости

$$\frac{1}{b-a}\int_a^b [f(x)-\phi(x)]^2 dx \le \varepsilon, \tag{2.13}$$

где ε — приемлемая мера близости. Если в качестве функции $\phi(x)$ выбран полином $P_n(x)$, то критерий преобразуется в

$$\frac{1}{b-a}\int_a^b [f(x)-P_n(\overrightarrow{a},x)]^2 dx \le \varepsilon. \tag{2.14}$$

В случае дискретного набора точек $\{x_i\}, \quad i=0,1,...,m$, этот критерий приобретает вид

$$\frac{1}{m-1}\sum_{i=0}^{m} [f(x)-P_n(\overrightarrow{a},x)]^2 dx \le \varepsilon. \tag{2.15}$$

Задачи (2.14) и (2.15) могут быть развиты в задачи наилучшего приближения. В случае задания функции $f(x)$ на непрерывном отрезке $[a,b]$ задача построения полинома $P_n(\overrightarrow{a},x)$ сводится к нахождению вектора коэффициентов $\overrightarrow{a^*}$ из задачи минимизации функционала вида

$$\varphi(\overrightarrow{a}) = \frac{1}{b-a}\int_a^b [f(x)-P_n(\overrightarrow{a},x)]^2 dx.$$

Если же нам задана функция в дискретном виде на узлах $\{x_i\}$, то стоит задача минимизации функционала

$$\delta(\overrightarrow{a}) = \frac{1}{m-1}\sum_{i=0}^{m} [f(x)-P_n(\overrightarrow{a},x)]^2 dx.$$

Существуют различные подходы к решению задачи минимизации функционала $\delta(\overrightarrow{a})$. Простейший из них приводит к необходимости решения нормальной системы линейных алгебраических уравнений

$$\sum_{j=0}^{n} a_j \sum_{i=0}^{m-1} x_i^j x_i^k = \sum_{i=0}^{m-1} f(x_i) x_i^k, \qquad k=1,...,n. \tag{2.16}$$

Однако, зачастую уже при $n>5$ матрица такой системы оказывается настолько плохо обусловленной, что полученные из (2.16) значения $\overrightarrow{a}$ оказываются мало пригодными для вычисления $P_n(x)$. Причиной этому служит сильная линейная зависимость системы степенных функций. Для улучшения обусловленности системы необходимо выбрать ортогональные функции на множестве точек $\{x_i\}$. Примером таких функций могут служить полиномы Чебышева на отрезке $[-1,1]$.

Поэтому, при необходимости построения полиномов наилучшего среднеквадратичного приближения более высоких степеней применяют другие алгоритмы, например, метод сингулярного разложения [4].

Рассмотренный метод приближения минимизирует среднеквадратичное уклонение аппроксимирующего полинома от аппроксимируемой функции, но не гарантирует отсутствие значительных локальных ошибок. Для предотвращения подобной возможности используют полиномы наилучшего равномерного приближения.

2.2.5 Равномерное приближение функций

Близость полинома $P_n(\overrightarrow{a}, x)$ к таблично заданной функции $f(x_i)$, $x_i \in [a, b]$, $i = 0, 1, ..., m$, $m \geq n + 1$, можно оценить величиной модуля их разности

$$\max_k |P_n(\overrightarrow{a}, x_k) - f(x_k)| = \max_k |\delta(\overrightarrow{a}, x_k)|. \tag{2.17}$$

Если $m = n+1$, то можно построить полином, обеспечивающий выполнение условия $\delta(\overrightarrow{a}, x_k) = 0$, $k = 1, 2, ..., m$, т.е. решить задачу интерполирования.

При $m > n + 1$ интерполяция нереализуема, но возможно построение полинома $P_n(\overrightarrow{a}, x)$, который минимизирует модуль уклонения в узлах сетки, т.е. минимизирует функционал

$$\upsilon(\overrightarrow{a}) = \min_{\overrightarrow{a}} |\max_k |\delta(\overrightarrow{a}, x_k)||. \tag{2.18}$$

Такой полином называется полиномом наилучшего равномерного приближения табличной функции.

2.3 Методы интерполирования и аппроксимации в среде Wolfram Mathematica

2.3.1 Интерполяция методом наименьших коэффициентов

Реализуем метод интерполяции методом наименьших коэффициентов. Для определенности выберем модельную функцию $f(x) = |x|$. Интерполировать будем на промежутке $[-1, 1]$

```
In[•]:= (*Участок интерполирования*)
        a = -1;
        b = 1;

In[•]:= (*Укажем количество интервалов. Количество точек на интервале будет n+1*)
        n = 10;
```

```
(*Зададим модельную функцию, которую хотим интерполировать*)
F[x_] := N[Abs[x]];

(*Составим сетку для интерполирования. Равномерная сетка*)
UzlRav[i_] := N[a + (b - a) * i / n];

(*Составим сетку для интерполирования. Чебышевская сетка*)
UzlCheb[i_] := N[(b + a)/2 + (b - a)/2 Cos[(2 i + 1)/(2 (n + 1)) Pi]]

(*Сформируем два набора точек*)
(*Первый - набор значений функции в узлах равномерной сетки *)
Arav = Table[{UzlRav[i], F[UzlRav[i]]}, {i, 0, n}];

(*Второй - набор значений функции в узлах чебышевской сетки *)
ACheb = Table[{UzlCheb[i], F[UzlCheb[i]]}, {i, 0, n}];

(*Построим метод неопределенных коэффициентов*)
MNC[sysCoord_, argument_, Wander_] :=
  Module[{A = sysCoord,  arg = argument, wan = Wander,  W, i, j, f, n,  Obusl, X},
    (*Узнаем количество точек, входящих в набор*)
    n = Length[A];
    (*Строим матрицу Вандермонда*)
    W = Table[If[i == 0, 1, A[[j, 1]]^i], {j, 1, n}, {i, 0, n - 1}];
    (*Узнаем обусловленность матрицы Вандермонда*)
    Obusl = Norm[Inverse[W], 1] * Norm[W, 1];
    (*Создадим вектор свободных членов*)
    f = Table[A[[i, 2]], {i, 1, n}];
    (*Создадим вектор переменных*)
    X = Table[arg^i, {i, 0, n - 1}];
    If[wan == 1, Obusl, Expand[LinearSolve[W, f].X]]
  ]
```

```
In[•]:= (*Команда для интерполяции методом неопределенных коэффициентов будет
          иметь  вид MNC[data, argument, option]. Если вместо option поставить 1,
        то команда посчитает число обусловленности матрицы Вандермонда,
        если 0 – покажет полученный полином.*)
        (*Получим интерполяционный полином командой MNC на равномерной сетке*)
        Prav = MNC[Arav, x, 0]
        (*Число обусловлености матрицы Вандермонда*)
        MNC[Arav, x, 1]
```

$$\text{Out}[\bullet]= -4.24729\times10^{-15} - 2.2899\times10^{-14}\,x + 6.45635\,x^2 + 4.02456\times10^{-13}\,x^3 - 41.2809\,x^4 - 1.67415\times10^{-12}\,x^5 + 128.4\,x^6 + 2.44548\times10^{-12}\,x^7 - 167.927\,x^8 - 1.15089\times10^{-12}\,x^9 + 75.352\,x^{10}$$

$$\text{Out}[\bullet]= 40\,145.3$$

```
In[•]:= (*Сравним результат с полученным встроеным методом, равномерная сетка*)
        Expand[InterpolatingPolynomial[Arav, x]]
```

$$\text{Out}[\bullet]= 0. + 6.45635\,x^2 + 2.66454\times10^{-15}\,x^3 - 41.2809\,x^4 - 7.10543\times10^{-15}\,x^5 + 128.4\,x^6 - 167.927\,x^8 - 1.42109\times10^{-14}\,x^9 + 75.352\,x^{10}$$

```
In[•]:= (*Получим интерполяционный полином командой MNC на сетке Чебышева*)
        PCheb = MNC[ACheb, x, 0]
        (*Число обусловлености матрицы Вандермонда*)
        MNC[ACheb, x, 1]
```

$$\text{Out}[\bullet]= -2.76029\times10^{-15} + 5.48061\times10^{-15}\,x + 4.80651\,x^2 - 3.34561\times10^{-14}\,x^3 - 18.7389\,x^4 + 8.76719\times10^{-14}\,x^5 + 39.5072\,x^6 - 1.27486\times10^{-13}\,x^7 - 38.2289\,x^8 + 6.84129\times10^{-14}\,x^9 + 13.667\,x^{10}$$

$$\text{Out}[\bullet]= 9250.6$$

```
In[•]:= (*Сравним результат с полученным встроеным методом, сетка Чебышева*)
        Expand[InterpolatingPolynomial[ACheb, x]]
```

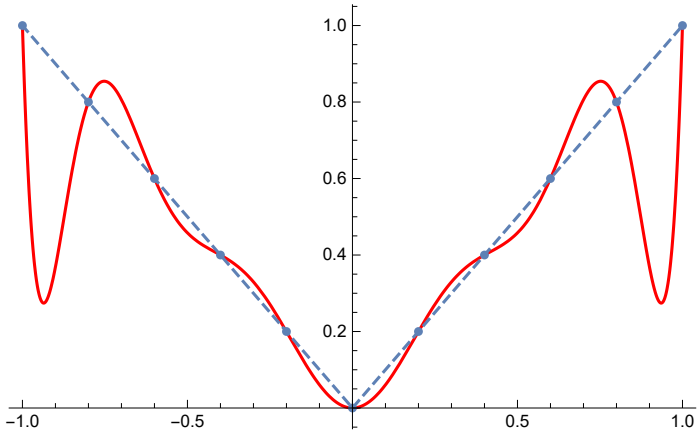
$$\text{Out}[\bullet]= 1.11022\times10^{-16} + 4.80651\,x^2 - 1.77636\times10^{-15}\,x^3 - 18.7389\,x^4 + 1.06581\times10^{-14}\,x^5 + 39.5072\,x^6 - 1.42109\times10^{-14}\,x^7 - 38.2289\,x^8 + 8.88178\times10^{-15}\,x^9 + 13.667\,x^{10}$$

```
In[•]:= (*Выведем на экран графики*)
        G1 = Plot[Prav, {x, a, b}, PlotStyle → {Red}];
        G2 = Plot[PCheb, {x, a, b}, PlotStyle → {Green}];
        G3 = Plot[{F[x]}, {x, a, b}, PlotStyle → Dashed];
        G4 = ListPlot[Arav];
        G5 = ListPlot[ACheb];
```

На рис. 2.1 показано окно вывода данных. После вывода на экран графиков можно сделать вывод, что наилучшее качество построения полинома для метода наименьших коэффициентов реализуется на сетке Чебышева (для модельной функции $|x|$).

In[•]:= Show[G1, G3, G4]

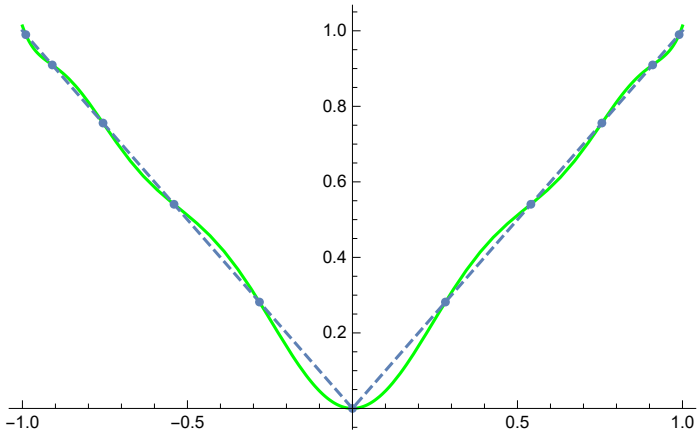
Out[•]=



а)

In[•]:= Show[G2, G3, G5]

Out[•]=



б)

Рис. 2.1. Интерполяция методом наименьших коэффициентов: а — полином, построенный на равномерной сетке; б — на сетке Чебышева, пунктирная линия — исходная функция.

In[•]:= (*Подсчитаем погрешность интерполяции*)
(*Среднеквадратичная погрешность *)
(*На равномерной сетке*)

$$\frac{1}{b-a}\int_a^b (F[x]-Prav)^2\,dx$$

Out[•]= 0.0406798

In[•]:= (*На сетке Чебышева*)

$$\frac{1}{b-a}\int_a^b (F[x]-PCheb)^2\,dx$$

Out[•]= 0.000581171

In[•]:= (*Модуль максимального отклонения*)
(*На равномерной сетке*)
FindMaximum[{Abs[F[x] − Prav], a ≤ x ≤ b}, {x, a}]

Out[•]= {0.663548, {x → −0.940113}}

```
In[•]:= (*На чебышевской сетке*)
        FindMaximum[{Abs[F[x] - PCheb], a ≤ x ≤ b}, {x, a}]
Out[•]= {0.0188327, {x → -0.648523}}
```

2.3.2 Интерполяция методом Лагранжа

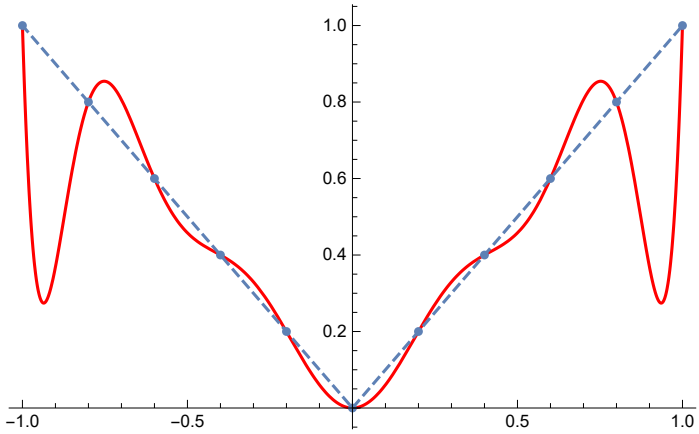
Многочлен Лагранжа в явном виде содержит значения функций в узлах интерполяции, поэтому он удобен, когда значения функций меняются, а узлы интерполяции неизменны. Построим метод, позволяющий интерполировать методом Лагранжа.

```
In[•]:= (*Построим метод Лагранжа*)
        Lagrange[sysCoord_, argument_] := Module[{A = sysCoord, arg = argument, i, j, n},
          (*узнаем количество точек, входящих в набор*)
          n = Length[A];
          (*Строим полином Лагранжа*)
          Expand[∑_{j=1}^{n} A[[j, 2]] × ∏_{i=1}^{n} If[i == j, 1, (arg - A[[i, 1]])/(A[[j, 1]] - A[[i, 1]])]]
        ]

In[•]:= (*Интерполяционный полином,
        полученный командой Lagrange[], на равномерной сетке*)
        Plrav = Lagrange[Arav, x]
Out[•]= 0. + 6.45635 x^2 + 7.10543 × 10^-15 x^3 - 41.2809 x^4 + 7.10543 × 10^-15 x^5 +
          128.4 x^6 - 1.20792 × 10^-13 x^7 - 167.927 x^8 + 1.77636 × 10^-14 x^9 + 75.352 x^10

In[•]:= (*Интерполяционный полином,
        полученный командой Lagrange[], на сетке Чебышева*)
        PlCheb = Lagrange[ACheb, x]
Out[•]= 0. - 1.66533 × 10^-16 x + 4.80651 x^2 - 2.13163 × 10^-14 x^3 - 18.7389 x^4 - 2.4869 × 10^-14 x^5 +
          39.5072 x^6 + 1.27898 × 10^-13 x^7 - 38.2289 x^8 - 2.84217 × 10^-14 x^9 + 13.667 x^10

In[•]:= (*Выведем на экран графики*)
        G6 = Plot[Plrav, {x, a, b}, PlotStyle → {Red}];
        G7 = Plot[PlCheb, {x, a, b}, PlotStyle → {Green}];
```

На рис. 2.2 показано окно вывода данных. Как и в случае построения полинома методом наименьших коэффициентов, наилучшее качество полинома на сетке Чебышева (для модельной функции $|x|$). При отсутствии погрешности исходных данных полиномы одного порядка, на одинаковой сетке, построенные методом Лагранжа и наименьших коэффициентов совпали.

In[•]:= Show[G6, G3, G4]

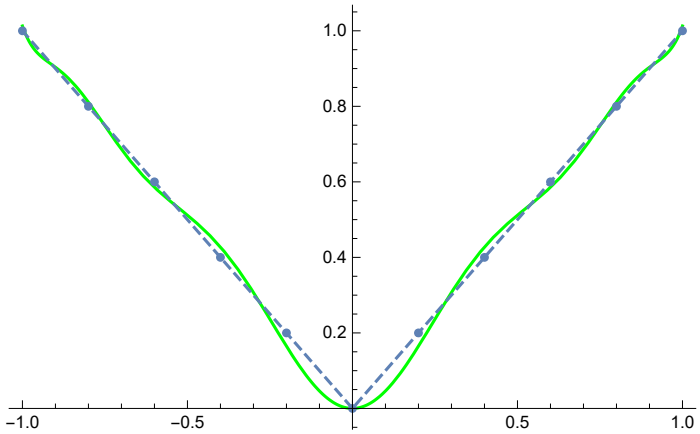
Out[•]=



а)

In[•]:= Show[G7, G3, G4]

Out[•]=



б)

Рис. 2.2. Интерполяция методом Лагранжа: а — полином, построенный на равномерной сетке; б — на сетке Чебышева, пунктирная линия — исходная функция.

2.3.3 Среднеквадратичное приближение функции

Рассмотрим случай, когда количество точек для интерполяции слишком велико. Как отмечалось, метод наименьших квадратов широко применяется для сглаживания экспериментальных кривых, полученных с некоторой погрешностью. Допустим у нас имеется набор экспериментальных точек, полученных в результате исследования затухающего процесса $f(x) = \cos(3x)e^{-x/2}$. Погрешность измерений смоделируем добавкой случайной величины. Рассматривать задачу приближения будем на промежутке $[0, 4]$

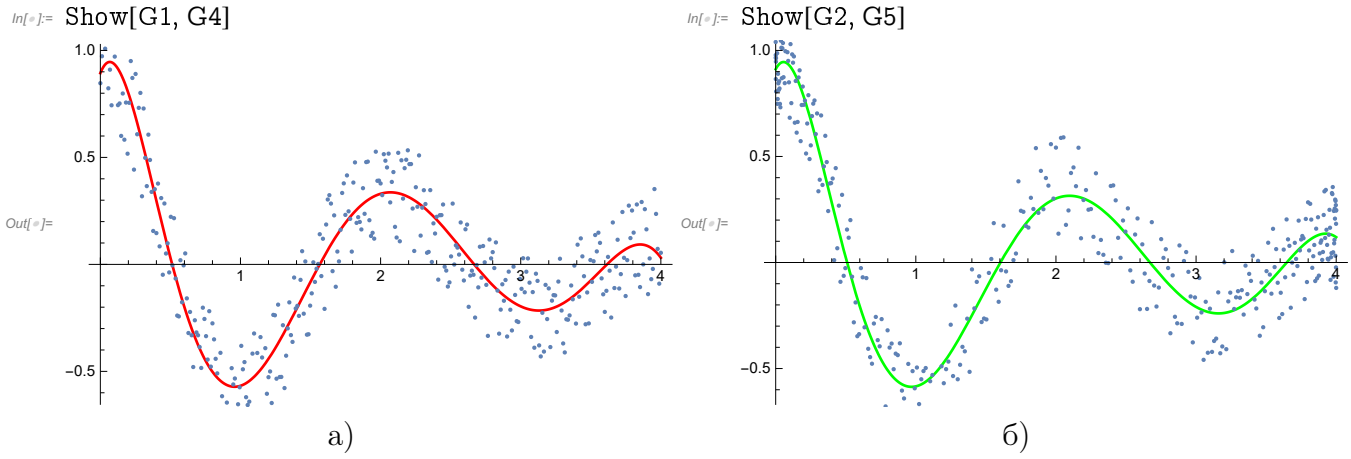
In[•]:=
```
(*Участок интерполирования*)
a = 0;
b = 4;
```

In[•]:=
```
(*Укажем количество интервалов. Количество точек будет n+1*)
n = 350;
```

```
In[ ]:= (*Зададим модельную функцию, которую хотим аппроксимировать*)
        (*Пусть в каждой токе присутствует погрешность *)
        F[x_] := N[Cos[3 * x]] * Exp[-x/2] + RandomReal[{-1, 1}] / 4

In[ ]:= (*Составим сетку для аппроксимации. Равномерная сетка*)
        UzlRav[i_] := N[a + (b - a) * i / n];

In[ ]:= (*Составим сетку для аппроксимации. Чебышевская сетка*)
        UzlCheb[i_] := N[(b + a)/2 + (b - a)/2 Cos[(2 i + 1)/(2 (n + 1)) Pi]]

In[ ]:= (*Сформируем два набора экспериментальных точек*)
        (*Первый - набор значений функции в узлах равномерной сетки *)
        Arav = Table[{UzlRav[i], F[UzlRav[i]]}, {i, 0, n}];

In[ ]:= (*Второй - набор значений функции в узлах чебышевской сетки *)
        ACheb = Table[{UzlCheb[i], F[UzlCheb[i]]}, {i, 0, n}];

In[ ]:= (*Среднеквадратичное приближение функций*)
        RMS[sysCoord_, stepen_, argument_] :=
          Module[{A = sysCoord, arg = argument, m = stepen, i, j, aA1},
            (*Узнаем количество точек, входящих в набор*)
            n = Length[A];
            (*Создаем набор базисных степенных функций*)
            X = Table[y^i, {i, 0, m}];
            (*Создаем набор коэффициентов *)
            aA = Table[aa_i, {i, 0, m}];
            argX = Table[arg^i, {i, 0, m}];
            P[y_] = X.aA;
            (*Ищем минимум суммы квадратов разностей функции и полинома*)
            aA1 = aA /. Last[NMinimize[Sum_{i=1}^{n} (P[A[[i, 1]]] - A[[i, 2]])^2, aA]];
            (*Вывод полинома*)
            aA1.argX
          ]

In[ ]:= (*С помощью команды RMS[ ] находим
          аппроксимационный полином 7 степени на равномерной сетке*)
        Prmsrav = RMS[Arav, 7, x]

Out[ ]= 0.885354 + 1.65863 x - 14.1088 x^2 + 19.7663 x^3 -
          11.7566 x^4 + 3.46642 x^5 - 0.4994 x^6 + 0.0280075 x^7
```

```
In[•]:= (*С помощью встроеного метода (команда Fit[])
          находим аппроксимационный полином*)
        Fit[Arav, X, y, "BestFit"]
Out[•]= 0.885354 + 1.65863 y – 14.1088 y^2 + 19.7663 y^3 –
          11.7566 y^4 + 3.46642 y^5 – 0.4994 y^6 + 0.0280075 y^7

In[•]:= (*С помощью команды RMS[ ] а находим
          аппроксимационный полином 7 степени на чебышевской сетке*)
        PrmsCheb = RMS[ACheb, 7, x]
Out[•]= 0.963679 + 0.914218 x – 12.3516 x^2 + 17.9177 x^3 –
          10.7747 x^4 + 3.19653 x^5 – 0.46373 x^6 + 0.0263014 x^7

In[•]:= (*С помощью встроеного метода (команда Fit[])
          находим аппроксимационный полином *)
        Fit[ACheb, X, y, "BestFit"]
Out[•]= 0.963679 + 0.914217 y – 12.3516 y^2 + 17.9177 y^3 –
          10.7747 y^4 + 3.19653 y^5 – 0.46373 y^6 + 0.0263014 y^7

In[•]:= (*Выведем на экран графики*)
        G1 = Plot[Prmsrav, {x, a, b}, PlotStyle → {Red}];
        G2 = Plot[PrmsCheb, {x, a, b}, PlotStyle → {Green}];
        G4 = ListPlot[Arav];
        G5 = ListPlot[ACheb];
```

На рис. 2.3 показано окно вывода данных.



а) б)

Рис. 2.3. Аппроксимация методом среднеквадратичного приближения. Порядок приближающего полинома — 7: а — полином, построенный на равномерной сетке, б — на сетке Чебышева, точками показаны эксперементальные данные.

Получений результат можно считать приемлемым.

2.3.4 Равномерное приближение функции

Построим метод равномерного приближения функции применительно к рассмотренной в разделе 2.3.3 задаче. Аппроксимировать данные будем полиномом 9-ой степени.

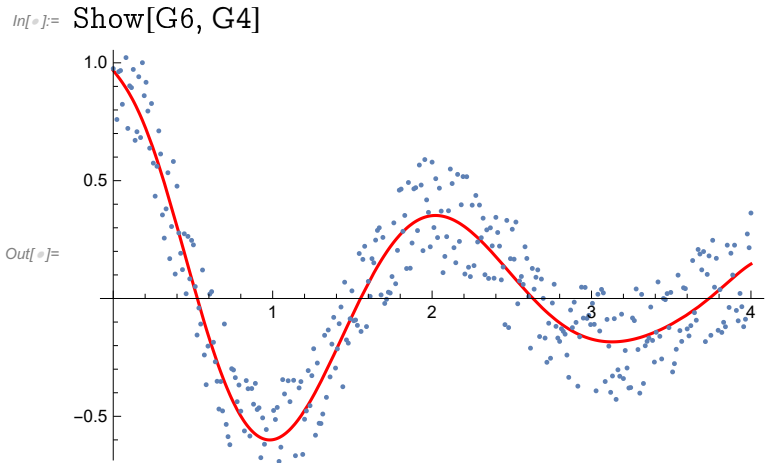
```
In[ ]:= (*Равномерное приближение функций*)
RavnomerPribl[sysCoord_, stepen_, argument_] :=
 Module[{A = sysCoord,  arg = argument,  m = stepen, i, j, aA1},
  (*Узнаем количество точек, входящих в набор*)
  n = Length[A];
  (*Создаем набор базисных степенных функций*)
  X = Table[y^i, {i, 0, m}];
  (*Создаем набор коэффициентов *)
  aA = Table[aa_i, {i, 0, m}];
  argX = Table[arg^i, {i, 0, m}];
  P[y_] = X.aA;
  (*Ищем минимум максимального уклонения в узлах аппроксимации*)
  aA1 =
   aA /. Last[NMinimize[Max[Table[Abs[A[[i, 2]] - P[A[[i, 1]]]], {i, 1, n}]],  aA]];
  (*Вывод полинома*)
  aA1.argX
 ]
```

```
In[ ]:= (*С помощью команды RavnomerPribl[ ] находим
    аппроксимационный полином 9 степени на равномерной сетке*)
 Pravpriblrav = RavnomerPribl[Arav, 9, x]
Out[ ]= 0.967805 - 0.851529 x - 0.37925 x^2 - 10.6982 x^3 + 23.0304 x^4 -
    19.2131 x^5 + 8.27716 x^6 - 1.9651 x^7 + 0.245199 x^8 - 0.0126062 x^9
```

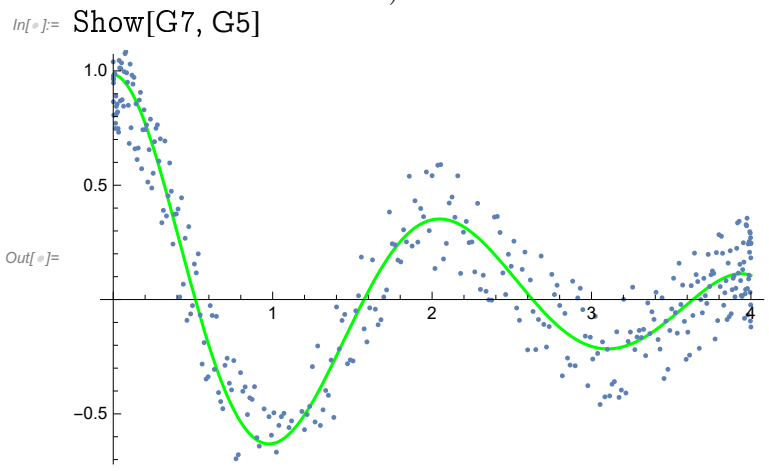
```
In[ ]:= (*С помощью команды RavnomerPribl[ ] находим аппроксимационный полином 9
    степени на чебышевской сетке*)PravpriblCheb = RavnomerPribl[ACheb, 9, x]
Out[ ]= 1.04134 - 1.4467 x + 1.15552 x^2 - 12.0349 x^3 + 22.773 x^4 -
    18.0849 x^5 + 7.52456 x^6 - 1.73229 x^7 + 0.209923 x^8 - 0.0104927 x^9
```

```
In[ ]:= (*Выведем на экран графики*)
G6 = Plot[Pravpriblrav, {x, a, b}, PlotStyle → {Red}];
G7 = Plot[PravpriblCheb, {x, a, b}, PlotStyle → {Green}];
```

На рис. 2.3 показано окно вывода данных.



а)



б)

Рис. 2.4. Аппроксимация методом равномерного приближения. Порядок приближающего полинома — 9: а — полином, построенный на равномерной сетке, б — на сетке Чебышева, точками показаны эксперементальные данные.

2.4 Задание на лабораторную работу №2

Составить план и провести вычислительные эксперименты в среде Mathematica / MATLAB, отвечающие задачам исследования, перечисленным ниже.

1) Провести сравнение качества построения интерполяционного полинома 3 - 4 различными методами. При неизменной исходной функции определить влияние порядка полинома на погрешность. Исследование провести на разных сетках — равномерной и чебышевской. При построении полинома методом неопределенных коэффициентов зафиксировать изменения значений числа обусловленности интерполяционной матрицы от порядка полинома).
2) Провести исследование погрешности интерполирования для 2-3 модельных функций, отличающихся свойствами гладкости и монотонности на интервале интерполирования. Выявить зависимость погрешности от:
   - порядка интерполяционного полинома,
   - величины интервала,
   - типа сетки.
3) Исследовать устойчивость решения задачи интерполирования к погрешности исходных данных (значений функции в узлах сетки):
   - зафиксировать значения уклонений в узлах, значения погрешности интерполирования (в равномерной метрике) на чебышевской и равномерной сетках,
   - сравнить эти данные с результатами аналогичных экспериментов при отсутствии возмущения исходных данных.
4) Для двух функций, монотонной и имеющей экстремум на интервале приближения, решить задачу приближения полиномами 2-й, 3-й, 4-й и максимально достижимой степени. При двух различных порядках приближающего полинома исследовать зависимость погрешности решения (равномерной и среднеквадратичной) от величины массива исходных данных.
5) Исследовать устойчивость решения задачи среднеквадратичного приближения к погрешности исходных данных. Критерием может служить отличие величин погрешности аппроксимации (среднеквадратичной и равномерной), полученных при наличии и отсутствии возмущений.
6) Решить задачу наилучшего равномерного приближения для функций, использованных в эксперименте по п. 4; установить возможность построения полинома максимально высокого (в пределах, допускаемых программой) порядка; сравнить точность решений задач приближения, полученных в п.4 и п.6, по равномерному и среднеквадратичному критериям.
7) Исследовать устойчивость решения задачи равномерного приближения к ошибкам исходных данных. Критерием может служить отличие величин погрешности аппроксимации (среднеквадратичной и равномерной), полученных при наличии и отсутствии возмущений.

3. Экспериментальное исследование численных методов решения обыкновенных дифференциальных уравнений

3.1 Цель работы

В настоящей работе будут рассмотрены методы решения задачи Коши для обыкновенных дифференциальных уравнений (ОДУ). Кроме того будет проведен анализ и сравнение вычислительных схем.

3.2 Численные методы решения обыкновенных дифференциальных уравнений

Задача Коши для обыкновенного дифференциального уравнения первого порядка формулируется в следующем виде:

$$\frac{dy}{dt} = f(t, y), \qquad t \in [t_0, t_n], \qquad y(t_0) = y_0. \tag{3.1}$$

3.2.1 Простейшие вычислительные схемы

Метод Эйлера

Наиболее простым способом построения решения в точке t_{n+1}, при известном решении в точке t_n, является способ, основанный на разложении в ряд Тейлора

$$y(t_{n+1}) = y(t_n) + hF(t_n, y_n, h),$$

где

$$F(t_n, y_n, h) = y'(t) + \frac{h}{2}y''(t) + \frac{h^2}{3!}y'''(t) + ...$$

Если этот ряд оборвать и заменить $y(t_n)$ приближенным значением y_n, то получим приближенную формулу:

$$y_{n+1} = y_n + \frac{h}{2}f'(t_n, y_n) + ... + \frac{h^k}{k!}f^{(k)}(t_n, y_n) \tag{3.2}$$

При $k = 1$ формула представляет собой вычислительную схему явного метода Эйлера.

$$y_{n+1} = y_n + \frac{h}{2} f'(t_n, y_n) \tag{3.3}$$

Хотя метод Эйлера и играет важную роль в теории численных методов решения ОДУ, однако он не часто используется в практических расчетах из-за его невысокой точности. Вывод основных расчетных соотношений этого метода может быть получен несколькими способами: с помощью геометрической интерпретации, с использованием разложения в ряд Тейлора, конечно разностным методом (с помощью разностной аппроксимации производной), квадратурным способом (использованием эквивалентного интегрального уравнения).

В предположении достаточной гладкости решения $y(t)$ можно легко получить выражения для оценки ошибки на каждом шаге интегрирования $\varepsilon_A = \frac{y''(\xi)}{2} h^2$, где $\xi \in [t_{n-1}, t_n]$. Погрешность решения на всей области ведет себя как $\varepsilon = Ch$ линейная функция в окрестности $h = 0$, и, именно поэтому явный метод Эйлера имеет первый порядок точности относительно шага интегрирования h.

Метод Рунге — Кутты

Первые методы данного класса были предложены около 1900 года немецкими математиками К. Рунге и М. В. Куттой, основанные на построении формулы для

$$y_{n+1} = y_n + \Phi(t_n, y_n, h)$$

в которой функция Φ близка к F, но не содержит производных от функции правой части уравнения. Было получено семейство явных и неявных методов, требующих s-кратного вычисления функции правой части на каждом шаге интегрирования (s — этапные методы).

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i,$$

гдеh — величина шага по t и вычисление нового значения проходит в s этапов:

$$\begin{aligned}
k_1 &= f(t_n, y_n), \\
k_2 &= f(t_n + c_2 h, y_n + a_{21} h k_1), \\
&\cdots \\
k_s &= f(t_n + c_s h, y_n + a_{s1} h k_1 + a_{s2} h k_2 + \cdots + a_{s,s-1} h k_{s-1})
\end{aligned}$$

Конкретный метод определяется числом s и коэффициентами b_i, a_{ij} и c_i. Эти коэффициенты часто упорядочивают в таблицу, называемую таблицей Бутчера (см.,таб. 3.1):

Для коэффициентов метода Рунге - Кутты должны быть выполнены условия

$$\sum_{j=1}^{i-1} a_{ij} = c_i$$

Таблица 3.1. Таблица Бутчера

0	0				
c_2	a_{21}	0			
c_3	a_{31}	a_{32}			
$\vdots$	$\vdots$	$\vdots$	$\ddots$	0	
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss-1}	0
	b_1	b_2	$\dots$	b_{s-1}	b_s

для $i = 2, \dots, s$.

Формулы этих методов идеально приспособлены для практических расчетов: они позволяют легко менять шаг интегрирования, являются одношаговыми, достаточно экономичны, по крайней мере, до формул четвертого порядка включительно. Наиболее часто используется и реализован в различных областях классический метод Рунге - Кутты, имеющий четвёртый порядок точности.

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{3.4}$$

Вычисление нового значения проходит в четыре стадии:

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1), \\ k_3 &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2), \\ k_4 &= f(t_n + h, y_n + hk_3), \end{aligned} \tag{3.5}$$

где h — величина шага сетки по t.

Этот метод имеет четвёртый порядок точности. Это значит, что ошибка на одном шаге имеет порядок $O(h^5)$, а суммарная ошибка на конечном интервале интегрирования имеет порядок $O(h^4)$.

Одна из проблем всех явных методов, в том числе и методов Эйлера и Рунге - Кутты, состоит в выборе шага интегрирования h, обеспечивающего устойчивость вычислительной схемы.

3.2.2 Неявные методы

За счет вычислительного усложнения формулы 3.2 был получен класс неявных методов, у которых отмеченная проблема в значительной степени снята. Неявные вычислительные схемы представляют собой алгебраические уравнения, в общем случае нелинейные,

относительно значений y_{n+1}.

Например:

- неявный метод Эйлера

$$y_{n+1} = hf(t_{n+1}, y_{n+1}). \tag{3.6}$$

- метод трапеции

$$y_{n+1} = y_n + \frac{h}{2}[f(t_{n+1}, y_{n+1}) + f(t_n, y_n)] \tag{3.7}$$

Подобно явным методам Рунге - Кутты возможно построить и неявные вычислительные схемы вида

$$y_{n+1} = y_n + h\Phi(t_{n+1}, y_{n+1}, h).$$

Неявные вычислительные схемы внешне похожи на явные того же порядка точности. Например, метод четвертого порядка имеет вид

$$\begin{gathered} y_{n+1} = y_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 = f(t_{n+1}, y_{n+1}), \\ k_2 = f(t_{n+1} + \frac{h}{2}, y_{n+1} - \frac{h}{2}k_1), \\ k_3 = f(t_{n+1} - \frac{h}{2}, y_{n+1} - \frac{h}{2}k_2), \\ k_4 = f(t_n, y_{n+1} - hk_3). \end{gathered} \tag{3.8}$$

Очевиден тот факт, что реализация такого метода предполагает решение нелинейных уравнений, что типично для любого неявного метода. Свойства устойчивости не накладывают каких-либо ограничений на шаг интегрирования, и он может выбираться исходя из соображений требуемой точности решения и сходимости метода решения нелинейных алгебраических уравнений.

Отметим некоторые проблемы реализации этих методов.

Все неявные вычислительные схемы, как правило, предполагают наличие какого - либо начального приближения $y_{n+1}^{(0)}$ к искомому решению y_{n+1}. Кажется очевидным желание выбрать как можно более «хорошее» начальное приближение $y_{n+1}^{(0)}$ посредством явной формулы того же порядка точности. В таком случае явная схема выполняет роль некого «прогноза» начального приближения, а неявная схема реализует коррекцию решения и весь комбинированный процесс становится методом прогноза-коррекции — П(ВК).

3.2.3 Устойчивость вычислительного метода

Минимальное требование, которое предъявляется к численному методу — это его сходимость. Методы не являющиеся сходящимися — бесполезны. Вместе с тем, не все

сходящиеся методы полезны для практического применения.

При решении задач возможно столкнуться с явлением, приводящим к непригодности использования полученного решения, а именно с явлением неустойчивости численного решения. Его источником могут стать как неустойчивость метода, так и неустойчивость задачи. Рассмотрим области устойчивости простейших методов интегрирования.

Рассмотрим следующую модельную задачу:

$$\frac{dy}{dt} = \lambda y, \qquad y(t_0) = y_0, \qquad Re(\lambda) < 0, \tag{3.9}$$

где параметр λ — комплексный.

Решение задачи (3.9) не представляет труда. Аналитическое решение этой задачи — функция $y(t) = y_0 e^{\lambda t}$, которая описывает затухающий процесс. Применяя к этой задаче вычислительные схемы, мы можем узнать параметры метода, при которых возможно получить заранее известное решение. Свидетельством неустойчивости численного метода будет являться факт неограниченного роста решения при каких то значениях параметров метода (например, шага).

Явный метод Эйлера

Вычислительная схема метода, применительно к решению (3.9) даёт

$$y_n = y_0(1 + \lambda h)^n.$$

Точное решение уравнения для $Re(\lambda) < 0$ затухает при $n \to \infty$. Такой же характер должна иметь и последовательность $\{y_n\}$ при $n \to \infty$, то есть должно выполняться

$$\lim_{n\to\infty} \frac{|y_n|}{|y_0|} = 0.$$

Такое становиться возможно, если

$$|1 + \lambda h| \leq 1. \tag{3.10}$$

Найдем область плоскости λh, где это условие выполняется. Положим $\lambda h = U + iV$, тогда

$$|1 + U + iV| \leq 1$$

и

$$(1 + U)^2 + V^2 \leq 1.$$

Последнему неравенству удовлетворяют точки комплексной плоскости, лежащие внутри и на границе окружности с центром (-1, 0). Таким образом, для точек λh, лежащих внутри заштрихованной на рис. 3.1 области и на ее границах явный метод Эйлера будет абсолютно устойчив. Значит, чем больше $|\lambda|$, тем меньшим должен быть шаг интегрирования h.

Неявный метод Эйлера

Проделав аналогичные операции с модельной задачей (3.9) вычислительная схема дает

$$y_{i+1} = y_0(1 - \lambda h)^{-(i+1)},$$

а значит должно выполняться неравенство $|1-\lambda h|^{-1} \leq 1$, что равносильно неравенству $(1-U)^2+V^2 \geq 1$. Такому неравенству удовлетворяют все точки, лежащие вне и на окружности, изображенной на рис. 3.2. При $Re(\lambda) < 0$ свойство устойчивости метода не накладывает ограничений на величину шага интегрирования.

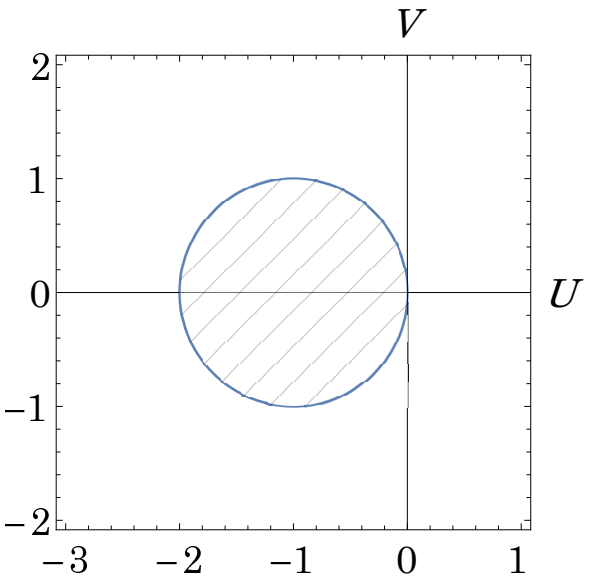


Рис. 3.1. Область абсолютной устойчивости явного метода Эйлера.

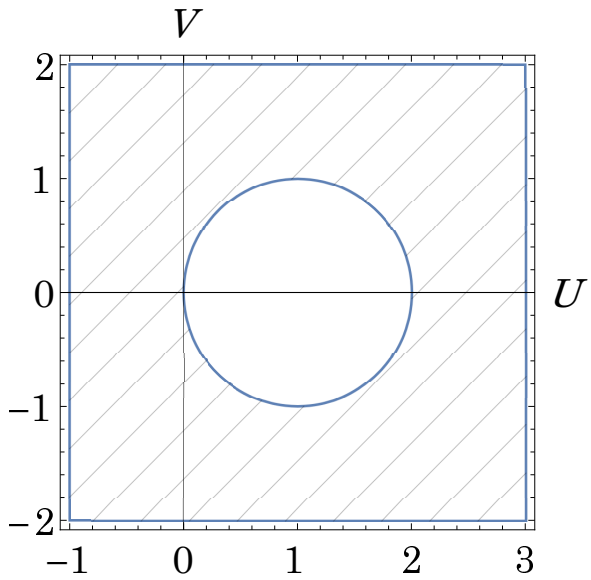


Рис. 3.2. Область абсолютной устойчивости неявного метода Эйлера.

Метод трапеции

Преобразования, аналогичные проведенным выше, дают

$$y_{i+1} = y_0 \left(\frac{2 + \lambda h}{2 - \lambda h} \right)^{i+1}.$$

Для $Re(\lambda) < 0$ последнее выражене должно генерировать равномерно невозрастающую последовательность, что возможно лишь при условии

$$\left| \frac{2 + \lambda h}{2 - \lambda h} \right| \leq 1.$$

Отсюда получаем

$$\frac{|2 + U + iV|}{|2 - U - iV|} = \frac{(2 + U)^2 + V^2}{(2 - U)^2 + V^2} \leq 1.$$

Это неравенство выполняется всегда при $U < 0$, что означает, что ограничений на величину шага интегрирования свойство устойчивости метода трапеции не накладывает.

Явные методы Рунге - Кутты

Применение рассмотренных методов Рунге - Кутты к модельной задаче 3.9 дает возможность определить области их устойчивости. Как видно из рис. 3.3, области устойчивости расширяются с ростом порядка аппроксимации метода.

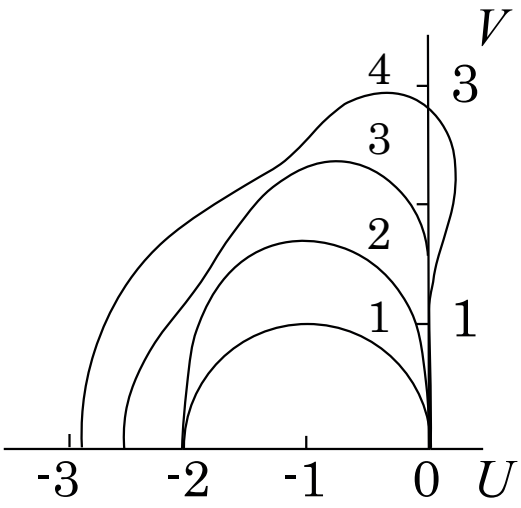


Рис. 3.3. Области абсолютной устойчивости явных методов Рунге - Кутты 1 – 4 порядков. Отображена только верхняя полуплоскость.

3.2.4 Решение задач Коши для систем ОДУ

Во многих случаях протекающие в реальных технических устройствах и системах процессы, описываются не одним обыкновенным дифференциальным уравнением, а системой ОДУ. Рассмотрение таких процессов приводит, в частности, к необходимости решения задачи Коши вида

$$\frac{d\vec{y}}{dt} = \overrightarrow{f}(\overrightarrow{y}, t), \qquad t \in [t_0, t_n], \qquad \overrightarrow{y}(t_0) = \overrightarrow{y}_0, \qquad \overrightarrow{y}(t) \in R^m, \qquad \overrightarrow{f}(t, \overrightarrow{y}) \in R^m, \quad (3.11)$$

где $\overrightarrow{y}(t)$ — некая векторная функция с компонентами $y_1(t), y_2(t), \ldots y_m(t)$ и $\overrightarrow{f}(t)$ — векторная функция $f_1(t, \overrightarrow{y}), f_2(t, \overrightarrow{y}), \ldots, f_m(t, \overrightarrow{y})$.

Формально, все рассматриваемые схемы решений скалярной задачи, переносятся на систему уравнений путём замены скаляра y_n на вектор $\overrightarrow{y}_n$ и скаляра $f(t, y_n)$ на векторную функцию $f(t_n, \overrightarrow{y}_n)$.

Вопрос устойчивости схем при решении систем ОДУ проще исследовать на примере линейной системы

$$\frac{d\overrightarrow{y}}{dt} = A\overrightarrow{y}, \qquad \overrightarrow{y}(t_0) = y_0, \qquad (3.12)$$

где A — матрица, содержащая m строк и m столбцов, и ее элементы не зависят от переменной t.

Положим, что матрица A имеет m различных собственных чисел $\{\lambda_i\}$, $\quad i = 1, ..., m$, а этим числам соответствует собственные векторы $\overrightarrow{e}_1, \overrightarrow{e}_2, \ldots, \overrightarrow{e}_m$. Их совокупность образует матрицу $P = [\overrightarrow{e}_1, \overrightarrow{e}_2, \ldots, \overrightarrow{e}_m]$, по средствам которой возможно приведение матрицы A к диагональной преобразованием вида

$$P^{-1}AP = \Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 & \\ 0 & \lambda_2 & 0 & \\ 0 & 0 & \ddots & 0 \\ & & 0 & \lambda_m \end{pmatrix}.$$

Воспользуемся свойством собственных векторов матрицы – собственный вектор матрицы A может служить базисом в пространстве R^m. Произвольный вектор пространства R^m может быть представлен в виде линейной комбинации базисных векторов

$$\overrightarrow{y} = \sum_{i=1}^{m} a_i \overrightarrow{e}_i = P\overrightarrow{a}, \qquad \overrightarrow{a} = P^{-1}\overrightarrow{y}.$$

С учетом этого, уравнение (3.12) может быть преобразовано к виду

$$P^{-1}\frac{d\overrightarrow{y}}{dt} = P^{-1}AP\overrightarrow{a}.$$

В силу линейности оператора дифференцирования, получим

$$\frac{d\overrightarrow{a}}{dt} = \Lambda\overrightarrow{a}. \tag{3.13}$$

Поскольку матрица Λ имеет диагональную структуру, то уравнение (3.13) представляет собой m независимых линейных уравнений

$$\frac{da}{dt} = \lambda a_i(t), \qquad i = 1, \ldots, m.$$

Решая каждое их них, получаем

$$a_i(t) = c_i e^{\lambda_i(t-t_0)}, c_i = a_i(t_0).$$

В отличие от обыкновенного дифференциального уравнения, для системы характерно наличие m постоянных во времени $\tau_i = 1/|Re\lambda_i|, \quad i = 1, \ldots, m$.

Если решение задачи (3.12) устойчиво по Ляпунову, то для устойчивости численного интегрирования необходимо потребовать, чтобы для любого i точка $h\lambda_i$ принадлежала области устойчивости применяемого метода интегрирования. Очевидно, что для этого достаточно, чтобы условия устойчивости удовлетворялось по отношению с максимальному собственному числу матрицы A. Здесь и возникают существенно новые явления. Важнейшее из них, — жесткость задачи.

3.2.5 Решение жестких задач

Часто, при решении задач радиотехники, химической кинетики, использование популярного метода Рунге — Кутты 4-го порядка и других хорошо отлаженных и протестированных программ, содержащих методы автоматического выбора величины шага интегрирования, приводит к неприемлемо большим трудозатратам. Оказалось, что в таких задачах присутствует две характерные области. Первая — очень малая начальная область, в которой происходит быстрое изменение решения, и малый шаг интегрирования необходим для адекватного описания процесса. Вторая — большая область медленного протекания процесса, в которой можно ожидать увеличения шага интегрирования. Однако программа интегрирования не всегда могла заметить изменение характера поведения решения и продолжает интегрирование с шагом, характерном для начального участка. Попытки увеличить шаг интегрирования приводили к резкому росту погрешности решения.

Анализ спектра матриц Якоби правой части таких систем уравнений послужил для определения понятия «жесткости». Так задачу Коши 3.12 называют жесткой, если спектр матрицы Якоби функции $f(t, \overrightarrow{y})$ делится на две части: жесткую, для которой выполняются условия

$$Re(\lambda \overrightarrow{y}) \leq -L, \quad |Im(\lambda_i(\overrightarrow{y}))| < |Re(\lambda_i(\overrightarrow{y}))|, \quad L > 0, i = 1, \ldots, I$$

и мягкую, где выполняются условие

$$|\lambda_j(\overrightarrow{y})| < l << L, \quad l > 0, j = 1, \ldots, J.$$

Отношение L/l называют показателем жесткости задачи. Систему уравнений можно назвать жесткой, если этот показатель больше 10, однако во многих прикладных и научных задачах это соотношение $10^6 \div 10^{15}$. На рис. 3.4 показано схематическое изображение расположения собственных чисел матрицы Якоби на комплексной плоскости.

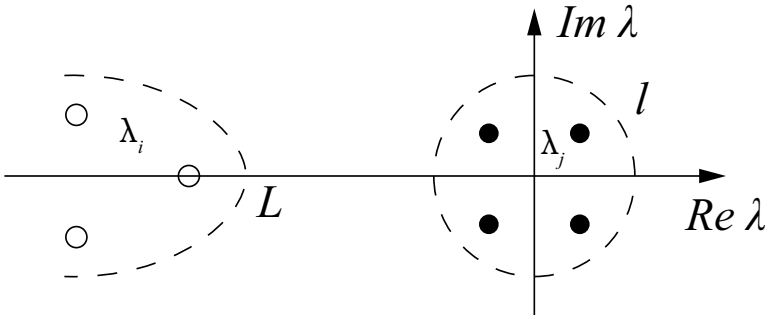


Рис. 3.4. Схематическое расположение собственных чисел матрицы Якоби функции правой части жесткой системы.

Понятие жесткой устойчивости метода позволило сконструировать группу методов, в которых величина шага интегрирования выбирается так, чтобы быстро затухающие и не

оказывающие существенного влияния компоненты решения аппроксимировались устойчиво, тогда как для компонент с большими постоянными времени гарантировалась точность аппроксимации. Наиболее распространенным классом линейных многошаговых методов для жестких задач являются «формулы дифференцирования назад» (более общее название — методы Гира). Их вычислительная схема базируется на интерполяционной формуле для правой части задачи Коши

$$\begin{gathered} f(t_{m+1}, \overrightarrow{y}_{m+1}) = -\frac{1}{h_{m+1}} \sum_{i=0}^{k} \alpha_i \overrightarrow{y}_{m+1-i} \\ a_i = \frac{t_m - t_{m-1}}{t_m - t_{m-i}} \prod_{p=1, p\neq i}^{k} \frac{t_m - t_{m-p}}{t_{m-i} - t_{m-p}}, \quad i = 1, \ldots, k \\ a_0 = -\sum_{j=1}^{k} a_j. \end{gathered} \tag{3.14}$$

Соотношения 3.14 представляют систему нелинейных уравнений относительно $\overrightarrow{y}_{n+1}$. Для вычисления начального приближения $\overrightarrow{y}_{n+1}^{(0)}$ можно использовать явную (предсказывающую) формулу Гира

$$\begin{gathered} y = \sum_{j=0}^{k+1} \cdot \chi_j y_{m+1-i} \\ \chi_j = \prod_{p=1 p\neq j}^{k+1} \frac{t_m - t_{m-p}}{t_{m-j} - t_{m-p}}, \quad i = 1, \ldots, k+1 \end{gathered} \tag{3.15}$$

3.3 Методы решения задачи Коши в среде Wolfram Mathematica

В данном разделе будут приведены методы решений обыкновенных дифференциальных уравнений и систем дифференциальных уравнений как аналитически, так и численно.

3.3.1 Аналитические методы решений задачи Коши

Процедуры пакета Mathematica для решения дифференциальных уравнений могут применяться ко многим различным классам дифференциальных уравнений, автоматически выбирая соответствующие алгоритмы, без необходимости их предварительной обработки.

Воспользуемся командой DSolve[ ] для решения дифференциального уравнения (3.9).

Для начала получим первообразную.

```
In[•]:= (*Команда DSolve[ ] позволяет аналитически
          решать дифференциальные уравнения*)
        (*Полученное решение будет первообразной
          с одной неизвестной константой  C*)
        DSolve[y'[x] == λ * y[x], y[x], x]
Out[•]= {{y[x] → ℯ^(x λ) C[1]}}
```

Полученная функция действительно является решением уравнения (3.9). Очень часто нужно не только получить первообразную, но и решить задачу Коши. Допустим, что $y(a) = b$.

```
In[•]:= (*Команда DSolve[ ] при задании начального
          условия позволяет аналитически решать задачу Коши*)
        (*Константа будет пересчитана из задачи Коши. Пусть y(a)=b*)
        DSolve[{y'[x] == λ * y[x], y[a] == b}, y[x], x]
Out[•]= {{y[x] → b ℯ^(-a λ+x λ)}}
```

Мы получили решение задачи Коши аналитически. Теперь решим следущую задачу Коши $y'(x) = 10\exp(-x)\cos(10x) - y(x), \quad y(0) = 0.$

```
In[•]:= (*Решим задачу Коши и запомним результат решения *)
        sol = DSolve[{y'[x] == 10 Cos[10 x] Exp[-x] - y[x], y[0] == 0}, y[x], x]
Out[•]= {{y[x] → ℯ^(-x) Sin[10 x]}}
```

Вывод полученного решения производим с помощью команды Plot[] рис. 3.5.

```
In[•]:= (*Вывод функции y[x] из решения уравнения sol на экран*)
        Plot[y[x] /. sol, {x, 0, 5},  PlotRange → All]
```

Out[•]=

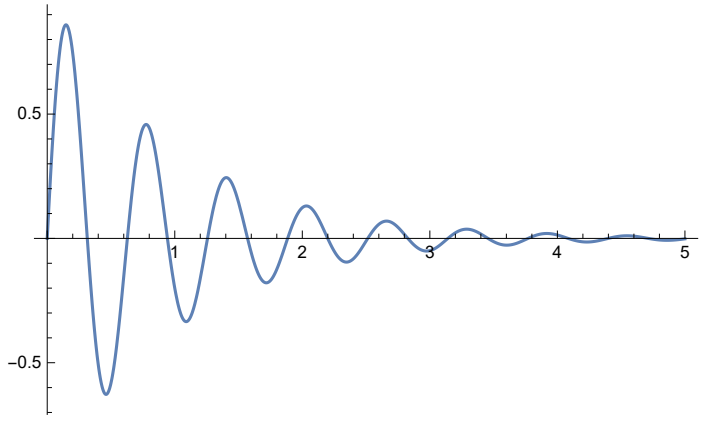


Рис. 3.5. Окно вывода данных.

Кроме решения обыкновенных дифференциальных уравнений команда DSolve[] поз-

воляет решать системы дифференциальных уравнений. Например, решим такую систему

$$\frac{d\overrightarrow{y}}{dt} = \begin{pmatrix} 1 & 3 \\ -4 & 2 \end{pmatrix} \overrightarrow{y}. \tag{3.16}$$

Для этого последовательно зададим неизвестные и матрицу системы.

```
In[ ]:= (*Решим систему обыкновенных дифференциальных уравнений*)
        (*Размерность системы*)
        n = 2;
        (*Создаем векторную функцию с компонентами yi(t)*)
        vecy = Table[yi[t], {i, 1, n}];
        (*Создаем производную векторной функции с компонентами yi'(t)*)
        vecdy = Table[yi'[t], {i, 1, n}];
        (*Задаем матрицу системы*)
        A = {{1, 3}, {-4, 2}};
```

```
In[ ]:= (*Запишем полученную систему дифференциальных уравнений – y'=Ay*)
        (*В принципе, это не обязательное действие,
        правую часть можно сразу подставить в команду Solve[]*)
        equ = vecdy == A.vecy
```

```
Out[ ]= {y1′[t], y2′[t]} == {y1[t] + 3 y2[t], -4 y1[t] + 2 y2[t]}
```

```
        (*Решим систему дифференциальных уравнений аналитически;
        получим решение с двумя константами*)
        DSolve[equ, vecy, t]
```

$$\text{Out[ ]= } \left\{\left\{y_1[t] \to \frac{6\, e^{3\,t/2}\, c_2 \operatorname{Sin}\left[\frac{\sqrt{47}\ t}{2}\right]}{\sqrt{47}} + \frac{1}{47} e^{3\,t/2} c_1 \left(47 \operatorname{Cos}\left[\frac{\sqrt{47}\ t}{2}\right] - \sqrt{47}\ \operatorname{Sin}\left[\frac{\sqrt{47}\ t}{2}\right]\right),\right.\right.$$
$$\left.\left. y_2[t] \to -\frac{8\, e^{3\,t/2}\, c_1 \operatorname{Sin}\left[\frac{\sqrt{47}\ t}{2}\right]}{\sqrt{47}} + \frac{1}{47} e^{3\,t/2} c_2 \left(47 \operatorname{Cos}\left[\frac{\sqrt{47}\ t}{2}\right] + \sqrt{47}\ \operatorname{Sin}\left[\frac{\sqrt{47}\ t}{2}\right]\right)\right\}\right\}$$

Поставим задачу Коши, пусть - $\overrightarrow{y}(0) = (0,1)^T$. Укажем это условие в команде следующей строкой.

```
In[•]:= (*Решим задачу Коши с заданной матрицей системы A*)
        (*Пусть функция y в момент t=0 имеет следущие компоненты y(0)=(0,1)*)
        (*Решим задачу Коши аналитически;
        получим решение с двумя компонентами*)
        solA = DSolve[{equ, {y1[0], y2[0]} == {0, 1}}, vecy, t]
```

$$\text{Out}[\bullet]= \left\{\left\{y_1[t] \to \frac{6\, e^{3\,t/2}\, \text{Sin}\left[\frac{\sqrt{47}\,t}{2}\right]}{\sqrt{47}},\ y_2[t] \to \frac{1}{47}\, e^{3\,t/2}\left(47\,\text{Cos}\left[\frac{\sqrt{47}\,t}{2}\right] + \sqrt{47}\,\text{Sin}\left[\frac{\sqrt{47}\,t}{2}\right]\right)\right\}\right\}$$

Вывод графиков на экран двух компонент решения показан на рис. 3.6.

```
In[•]:= (*Выведем решение задачи в виде графика на экран*)
        Plot[{y1[t] /. solA, y2[t] /. solA}, {t, 0, 5},
          PlotRange → All, PlotLegends → {"y1(t)", "y2(t)"}]
```

Out[•]=

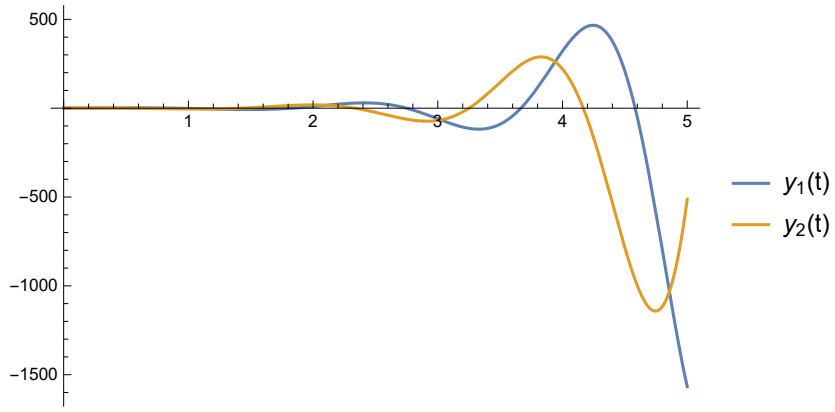


Рис. 3.6. Окно вывода данных. Приведены графики компонент решения $y_1(t)$ и $y_2(t)$.

3.3.2 Численные методы решений задачи Коши

Не всегда возможно получить решение дифференциального уравнения в аналитическом виде. Тогда, вместо использования DSolve[ ], можно воспользоваться функцией NDSolve[ ] для получения численного решения.

Теперь решим задачу Коши $y'(x) = 10\exp(-x)\cos(10x) - y(x), \quad y(0) = 0$, но уже численным методом. Выведем на экран график полученного решения.

```
In[ ]:= (*Численное решение можно получить,
        если воспользоваться командой NDSolve[ ]*)
        (*Если  указать в команде только уравнение и задачу Коши,
        то  метод решения и шаг интегрирования будут выбраны автоматически *)
        sol1 n = NDSolve[{y'[x] == 10 Cos[10 x] Exp[-x] - y[x], y[0] == 0}, y[x], {x, 0, 10}]

Out[ ]= {{y[x] → InterpolatingFunction[ Domain: {{0., 10.}}
                                        Output: scalar ][x]}}
```

На рис. 3.7 показан график полученного решения.

```
In[ ]:= (*Вывод графика на экран осуществляется аналогично,
        что  и в предыдущем разделе*)
        Plot[y[x] /. sol1 n, {x, 0, 10},  PlotRange → All]
```

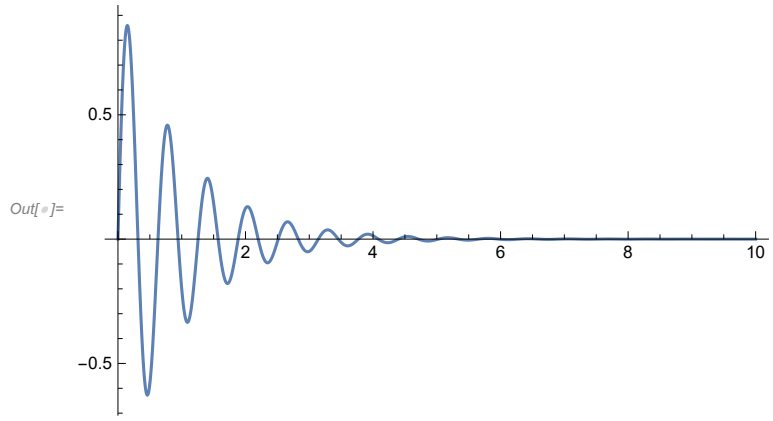


Рис. 3.7. Окно вывода данных. Показан вывод графика решения $y(x)$.

Команда NDSolve[ ] позволяет выбирать метод решения, шаг, регулировать изменение шага. Выбор параметров интегрирования осуществляется с помощью команды Method[]. В табл. 3.2 приведена часть методов, которые могут быть использованы командой NDSolve[ ] для решения ОДУ.

Имя метода	Описание
Adams	многошаговые методы Адамса 1 − 12-го порядка
BDF	формулы дифференцирования назад (методы Гира) 1 − 5-го порядка
ExplicitRungeKutta	явные методы Рунге-Кутты
ImplicitRungeKutta	семейство неявных методов Рунге - Кутты произвольного порядка
ExplicitEuler	явный метод Эйлера
LinearlyImplicitEuler	неявный метод Эйлера

Таблица 3.2. Основные методы решений систем ОДУ, применяемые в пакете Mathematica.

```
In[•]:= (*Можно решать уравнения принудительно указав метод решения,
        шаг, включить или выключить регулировку шага*)
        (*Например, следущая команда позволит решить
          уравнение явным методом Эйлера с шагом 0.01 *)
        sol1nEu001 = NDSolve[{y'[x] == 10 Cos[10 x] Exp[-x] - y[x], y[0] == 0},
              y[x], {x, 0, 10}, StartingStepSize → 0.01,
              Method → {"ExplicitEuler"}, Method → {"FixedStep"}];
        (*получим решение уравнение методом Эйлерас шагом
          0.3. Сравним полученные решения*)
        sol1nEu03 = NDSolve[{y'[x] == 10 Cos[10 x] Exp[-x] - y[x], y[0] == 0}, y[x], {x, 0, 10},
              StartingStepSize → 0.3, Method → {"ExplicitEuler"}, Method → {"FixedStep"}];
```

Графическое сравнение решения, полученного при разных величинах шага интегрирования, приводится на рис. 3.8.

```
In[•]:= (*Выводим полученные решения на график *)
        Plot[{Evaluate[y[x] /. sol1nEu001], Evaluate[y[x] /. sol1nEu03]},
          {x, 0, 10}, PlotLegends → {"0.01", "0.3"}]
```

Out[•]=

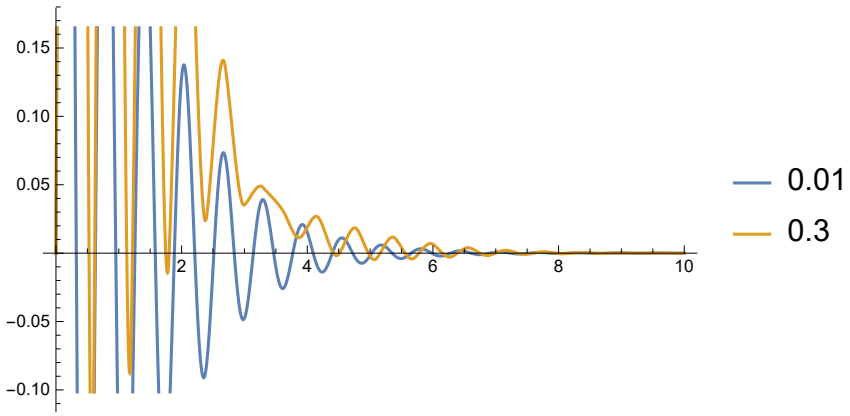


Рис. 3.8. Окно вывода данных. Сравнение двух решений при разном шаге интегрирования.

Как можно увидеть, при шаге интегрирования 0.3, полученное решение задачи явным методом Эйлера, некорректно. При таком большом шаге этот метод неустойчив. При значительном уменьшении шага до 0.01 устойчивость метода резко возрастает.

Решение систем ОДУ проводится аналогичным способом. Применим команду NDSolve[ ] для решения задачи 3.16. Сформулируем для этого задачу Коши $\overrightarrow{y}(0) = (0, 1)^T$. Команда для решения записывается аналогично команде DSolve[ ]. Полученные решения выведем на экран (рис. 3.9).

```
In[•]:= (*Аналогично решаются и системы дифференциальных уравнений. Например,
        решим задачу Коши  явным методом Рунге–Кутты*)
        soln = NDSolve[{equ, {y1[0], y2[0]} == {0, 1}}, vecy, {t, 0, 5}, StartingStepSize → 0.15,
            Method → {"ExplicitRungeKutta"}, Method → {"FixedStep"}];

In[•]:= Plot[{Evaluate[y1[t] /. soln], Evaluate[y2[t] /. soln]},
          {t, 0, 5}, PlotRange → All, PlotLegends → {"y1(t)", "y2(t)"}]
```

Out[•]=

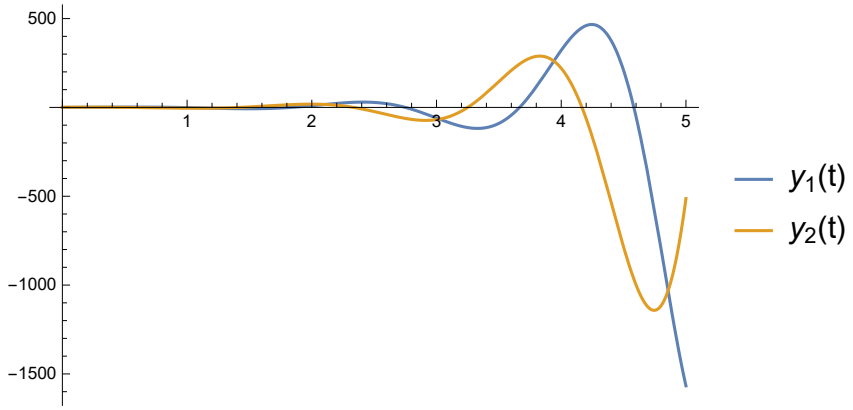


Рис. 3.9. Окно вывода данных. Показаны графики компонент решения $y_1(t)$ и $y_2(t)$.

3.4 Задание на лабораторную работу №3

Необходимо провести численные эксперименты в среде Wolfram Mathematica/ Matlab, на основании которых:

1) При интегрировании систем линейных уравнений на основе информации о значениях собственных чисел матрицы системы оценить максимальную величину шага устойчивого интегрирования и проверить эту оценку экспериментально.
2) Исследовать поведение полной ошибки численного решения при интегрировании с постоянным шагом методами различного порядка (Методы Эйлера, Рунге-Кутты 2 - 4-го порядков), т.е. получить зависимость максимальной погрешности решения задачи разными методами при условии постоянства шага интегрирования. Дать качественное описание поведения функции полной погрешности решения.
3) Сравнить оценки временных затрат при интегрировании на $[t_0, t_n]$ методами с регулированием шага и без регулирования. Сравнивать следует методы одинакового порядка, например, Рунге - Кутты 4-го порядка и Рунге-Кутты-Фельберга; трапеции, Гира.
4) При интегрировании жестких задач:
   - получить экспериментальное подтверждение низкой эффективности (временных затрат) явных методов (методы Рунге - Кутты 4-го порядка, Рунге - Кутты - Фельберга),
   - установить возможность и условия интегрирования задачи неявными методами с большим и постоянным шагом (метод трапеции),
   - сравнить эффективность применения метода Гира второго порядка с методом трапеции.

Литература

[1] Самарский, А.А. Численные методы: учебное пособие для вузов / А.А. Самарский, А.В. Гулин. — Москва: Наука. Гл. ред. физ-мат. лит., 1989. — С. 432.

[2] Воеводин, В.В. Вычислительные основы линейной алгебры / В.В. Воеводин. — Москва: Наука. Гл. ред. физ-мат. лит., 1977. — С. 304.

[3] Синепол, В.С. Информатика. Численные методы / В.С. Синепол, О.А. Смирнова. — Санкт-Петербург: Изд-во СПбГТУ СПб., 2001. — С. 128.

[4] Форсайт, Дж. Машинные методы математических вычислений / Дж. Форсайт, М. Малькольм, К. Моулер. — Москва: МИР, 1980. — С. 280.