Санкт-Петербургский государственный университет

Физический факультет
Кафедра вычислительной физики

# Практикум по численным методам для студентов второго курса Часть I-II

**Учебно-методическое пособие**

Санкт-Петербург
2005

*Утверждено на заседании кафедры*
*вычислительной физики*

Авторы: И.В. Андронов, В.Б. Курасов, В.В. Монахов, А.В. Кожедуб, П.А. Науменко, Т.В. Фролова, А.В. Цыганов, С.Л. Яковлев, Е.А. Яревский.

Рецензенты: докт. физ.-мат. наук, проф. Ю.М. Письмак.

Учебно-методическое пособие содержит в сжатом виде материал, необходимый для решения задач по применению численных методов на втором курсе дневного отделения физического факультета С.-Петербургского университета.

The manual contains the extract of the material which is necessary for the second year students of the Faculty of Physics of St. Petersburg State University to fulfill the program of the application of the numerical methods.

# Оглавление

# Работа № 1

## MATLAB в режиме командной строки

Система MATLAB создана таким образом, что любые (подчас весьма сложные) вычисления можно выполнять как в режиме прямых вычислений (то есть без подготовки программы), так и посредством создания программы (m-файла). Работа с системой в режиме прямых вычислений носит диалоговый характер и происходит по правилу "задал вопрос, получил ответ". Пользователь набирает на клавиатуре вычисляемое выражение, редактирует его (если нужно) в командной строке и завершает ввод нажатием клавиши `ENTER`. Ниже перечислены некоторые особенности работы в режиме командной строки:

- диалог происходит в стиле "задал вопрос - получил ответ"
- для указания ввода исходных данных используется символ
- данные вводятся с помощью простейшего строчного редактора
- для блокировки вывода результата вычислений некоторого выражения после него надо установить знак ; (точка с запятой)
- если не указана переменная для значения результата вычислений, то MATLAB назначает такую переменную с именем `ans`
- знаком присваивания является знак равенства =
- результат вычислений выводится в строках вывода (без знака )
- встроенные функции (например, `sin`) записываются строчными буквами, и их аргументы указываются в круглых скобках

- длинное выражение для удобства восприятия можно перенести на другую строку с помощью многоточия

## 1.1 Основные объекты MATLAB

### Переменные и присваивание им значений

Переменные - это имеющие имена объекты, способные хранить некоторые, обычно разные по значению, данные. В зависимости от этих данных переменные могут быть числовыми или символьными, векторными или матричными. В системе MATLAB можно задавать переменным определенные значения. Для этого используется операция присваивания, вводимая знаком равенства

```
Имя_переменной=Выражение
```

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Так, если это выражение - вектор или матрица, то переменная будет векторной или матричной. Исторически MATLAB создавалась как матричная лаборатория, что и отразилось в названии и, поэтому, MATLAB - система, специально предназначенная для проведения сложных вычислений с векторами, матрицами (массивами). При этом она по умолчанию предполагает, что каждая заданная переменная - это вектор или матрица. Все определяется конкретным значением переменной. Например, если задано $X = 1$, то это значит, что $X$ - это вектор с единственным элементом, имеющим значение 1 (а точнее MATLAB рассматривает все переменные как матрицы, так что в данном случае $X$ отвечает матрице размером $1 \times 1$).

Имя переменной (ее идентификатор) может содержать сколько угодно символов, но запоминается и идентифицируется только 31 начальный символ. в пределах этих 31 символов имя любой переменной должно быть уникальным, т.е. не должно совпадать с именами других переменных, функций и процедур системы.

Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания _. Недопустимо включать в имена переменных пробелы

и специальные знаки, например +, ., -, *, / и т. д., поскольку в этом случае правильная интерпретация выражений становится невозможной.

Как и большинство языков высокого уровня, система MATLAB различает регистры - т.е. имя A отличается от имени a.

### Математические выражения

Математическое выражение задает то, что должно быть вычислено в численном (реже символьном) виде. Математические выражения строятся на основе чисел, констант, переменных, операторов, функций и разных спецзнаков (например, 2.301*sin(x); 4+exp(3)/5; sqrt(y)/2; sin(pi/2); 2+3).

### Действительные и комплексные числа

Число - простейший объект языка MATLAB, представляющий количественные данные. Числа можно считать константами, имена которых совпадают с их значениями. Числа могут быть целыми, дробными, с фиксированной и плавающей точкой. Можно представлять числа в формате с указанием мантиссы и порядка числа (1.23e-02, что эквивалентно $1.23 * 10^{-2}$ в обычной записи). Пробелы между символами в числах не допускаются. Числа могут быть комплексными: z=Re(x)+Im(x)*i. Такие числа содержат действительную Re(z) и мнимую Im(z) части. Мнимая часть должна иметь множитель i (или j), (2+3i; -3.141j; -123.456+2.7e-3i). Функция real(z) возвращает действительную часть комплексного числа, а функция imag(z) - мнимую. Для получения модуля комплексного числа используется функция abs(z), а для вычисления фазы - angle(Z).

Операции над числами выполняются в формате, который принято считать форматом с двойной точностью. В зависимости от платформы, числа с плавающей точкой имеют приблизительно 16 значащих цифр и лежат примерно в диапазоне от $10^{-368}$ до $10^{368}$. Такой формат удовлетворяет подавляющему большинству требований к численным расчетам.

### Константы и системные переменные

Константа - это предварительно определенное числовое или символьное значение, представленное уникальным именем. Константы в MATLAB принято

назвать системными переменными, поскольку, с одной стороны, они задаются системой при ее загрузке, а с другой - могут переопределяться. Основные системные переменные, применяемые в системе MATLAB, указаны ниже:

- `i` или `j` - мнимая единица (корень квадратный из `-1`);
- `pi` - число `3.1415926...`;
- `eps` - погрешность операций над числами с плавающей точкой ($2.2204 \cdot 10^{-16}$);
- `realmin` - наименьшее число с плавающей точкой ($2.2251 \cdot 10^{-308}$);
- `realmax` - наибольшее число с плавающей точкой ($1.7977 \cdot 10^{308}$);
- `inf` - значение машинной бесконечности;
- `ans` - переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея;
- `NaN` - указание на нечисловой характер данных (Not-a-Number).

Системные переменные могут переопределяться. Можно задать системной переменной `eps` иное значение, например `eps=0.0001`. Однако важно то, что их значения по умолчанию задаются сразу после загрузки системы. Поэтому неопределенными, в отличие от обычных переменных, системные переменные не могут быть никогда. Символьная константа - это цепочка символов, заключенных в апострофы, например, `'Hello my friend!'`, `'2+3'`. Если в апострофы помещено математическое выражение, то оно не вычисляется и рассматривается просто как цепочка символов. Так что `'2+3'` не будет возвращать число `5`.

### Уничтожение определений переменных

В памяти компьютера переменные занимают определенное место, называемое рабочей областью (workspace). Для очистки рабочей области используется функция `clear` в разных формах, например:

- `clear` - уничтожение определений всех переменных пользователя;

- `clear x` - уничтожение определения переменной x;

- `clear a,b,c` - уничтожение определений нескольких переменных. Уничтоженная (стертая в рабочей области) переменная становится неопределенной. Использовать неопределенные переменные нельзя, и такие попытки будут сопровождаться выдачей сообщений об ошибке.

  Функция `clear` не влияет на системные переменные!

**Задание 1.1** Произведите присвоение переменным `a`, `b`, `c`, `d`, ... действительных или комплексных значений. Для комплексных переменных проверьте действия функций `real(z)`, `imag(z)`, `abs(z)`, `angle(z)`. Переопределите системную переменную `pi`. Уничтожьте определение некоторых из них, попытайтесь к ним обратиться, попытайтесь обратиться к не уничтоженным переменным. Уничтожьте определения всех переменных. Обратитесь к системной переменной `pi`.

### Матрицы и векторы

Двумерный массив чисел или математических выражений принято называть матрицей. Одномерный массив называют вектором. MATLAB допускает так же задание и использование многомерных массивов. Векторы и матрицы могут иметь имена, например `V` - вектор или `M` - матрица. Элементы векторов и матриц рассматриваются как индексированные переменные, `V(2)` - второй элемент вектора `V`; `M(2,3)` - третий элемент второй строки матрицы `M`. Как уже отмечалось, даже обычные числа и переменные в MATLAB рассматриваются как матрицы размера $1 \times 1$, что дает единообразные формы и методы проведения операций над обычными числами и массивами. Данная операция обычно называется векторизацией. Векторизация обеспечивает и упрощение записи операций, производимых одновременно над всеми элементами векторов и матриц, и существенное повышение скорости их выполнения. Это также означает, что большинство функций может работать с аргументами в виде векторов и матриц. При необходимости вектора и матрицы преобразуются в массивы, и значения вычисляются для каждого их элемента.

### Задания векторов и матриц

Для задания вектора, значения его элементов следует перечислить в квадратных скобках, разделяя пробелами или запятыми. Так, например, присваивание `V=[1 2 3]` (или `V=[1,2,3]`) задает вектор `V`, имеющий три элемента со значениями `1`, `2` и `3`. После ввода вектора, если строка не заканчивается символом `;`, система выводит его на экран дисплея. Задание матрицы требует указания нескольких строк. Для разграничения строк используется знак `;` (точка с запятой). Так, ввод

```
M=[1 2 3; 4 5 6; 7 8 9]
```

задает квадратную матрицу. Возможен ввод элементов матриц и векторов в виде арифметических выражений, содержащих любые доступные системе функции, например,

```
V=[2+2/(3+4) exp(5) sqrt(10)].
```

Для указания отдельного элемента вектора или матрицы используются выражения вида `V(k)` или `M(k,l)` (В тексте программ MATLAB лучше не использовать `i` и `j` как индексы, так как `i` и `j` - обозначение квадратного корня из `-1`, но можно использовать `I` и `J`). Если нужно присвоить элементу `M(k,l)` новое значение, следует использовать выражение `M(k,l)='новое значение'`. Для вывода значения переменной на экран необходимо набрать имя данной переменной и нажать клавишу Enter.

**Задание 1.2** Задайте вектор `V`. Задайте матрицу `M`. Выведите на экран какой-либо элемент вектора (затем матрицы). Присвойте этому элементу новое значение. Выведите на экран всю матрицу.

Выражение `M(k)` с одним индексом дает доступ к элементам матрицы, развернутым в один столбец. Такая матрица образуется из исходной, если подряд выписать ее столбцы.

**Задание 1.3** Для понимания последнего утверждения выведите на экран элемент `M(m*n -1)`, где `m`, `n` - размер матрицы.

Имеется также ряд особых функций для задания векторов и матриц. Например, функция `magic(n)` задает магическую матрицу размера $n \times n$, у которой сумма всех столбцов, всех строк и даже диагоналей равна одному и тому же числу, например, `M=magic(4)`.

### Объединение малых матриц в большую

Описанный способ задания матриц позволяет выполнить операцию конкатенации - объединения малых матриц в большую. Допустим, мы создали матрицу `A`, размером $4 \times 4$. Теперь можно построить матрицу, содержащую четыре матрицы `B=[A A+16; A+32 A+16]`. Полученная матрица имеет уже размер $8 \times 8$.

### Удаление столбцов и строк матриц

Для формирования матриц и выполнения ряда матричных операций возникает необходимость удаления отдельных столбцов и строк матрицы. Для этого используются пустые квадратные скобки `[ ]`. Проделаем это с матрицей `M=[1 2 3; 4 5 6; 7 8 9]`. Удалим второй столбец используя оператор `:` (двоеточие) `M(:,2)=[ ]`. Теперь удалим вторую строку `M(2,:)=[ ]`. Если удалить элемент матрицы `M(k)=[ ]`, матрица превратится в вектор.

**Задание 1.4** Задайте матрицу. Удалите строку. Удалите столбец. Удалите элемент матрицы.

### Операторы и функции

Оператор - это специальное обозначение для определенной операции над данными - операндами.

### Арифметические операторы

При работе с массивами чисел установлены следующие уровни приоритета арифметических операций:

1. транспонирование `(.')`, поэлементное возведение в степень `(.^)`, эрмитово сопряженное транспонирование матрицы `(')` , возведение матрицы в степень `(^)`
2. унарное сложение `(+)`, унарное вычитание `(-)`
3. поэлементное умножение массивов `(.*)`, поэлементное правое деление `(./)`, поэлементное левое деление массивов `(.\)`, умножение матриц `(*)`, решение систем линейных уравнений операция `(\)` и операция `(/)`

4. сложение (+), вычитание (-)

5. оператор формирования массивов (:)

Функция X=B\A находит решение системы уравнений вида AX=B, где A - прямоугольная матрица размера $m \times n$ и B - матрица размера $n \times k$. Функция X=B/A находит решение системы уравнений вида XA=B, где A - прямоугольная матрица размера $n \times m$ и B - матрица размера $m \times k$.

**Задание 1.5**

1. Поэлементное умножение и деление. Задайте две матрицы одинакового размера a и b. Умножьте одну матрицу на другую поэлементно a.*b. Разделите матрицы поэлементно справа (a./b)и слева (a.\b). Разделите матрицы поэлементно в другом порядке: b./a и b.\a. Сравните результаты.

2. Матричное умножение и деление. Задайте две квадратные матрицы a, b. Вычислите c=a*b и d=b*a. Убедитесь в том, что результаты не совпадает. Задайте не вырожденную матрицу a. Задайте вектор x. Вычислите f=a*x. Найдите x (зная a и f) x1=a\f. Убедитесь, что x совпадает с x1.

3. Возведение в степень. Задайте не единичную матрицу (например m=[2 2;0 1]) Осуществите поэлементное возведение в степень (m.^2). Возведите в степень матрицу w размера $5 \times 5$, состоящую из единиц над главной диагональю (остальные элементы 0). w^2

4. Транспонирование. Транспонируйте матрицу.

Обратите внимание на различные результаты действия операторов * и .*, / и ./, ^ и .^. Поэлементные операции важны при построении графиков.

**Операторы отношения**

В MATLAB определено 6 операторов отношения: меньше (<), меньше или равно (<=), больше (>), больше или равно (>=), равно тождественно (==), не равно(~=).

**Задание 1.6** Сравните две матрицы одинакового размера. Объясните результат. Сравните две матрицы разного размера (например, a>=b).

### Логические операторы

И (`&`), ИЛИ (`|`), НЕ (`~`). Полный список операторов можно получить, используя команду `help ops`.

### Функции

Функции - это имеющие уникальные имена объекты, выполняющие определенные преобразования своих аргументов и при этом возвращающие результаты этих преобразований. При этом результат вычисления функции с одним выходным параметром подставляется на место ее вызова, что позволяет использовать функции в математических выражениях. Функции в общем случае имеют список аргументов (параметров), заключенный в круглые скобки. Многие функции допускают ряд форм записи, отличающихся списком параметров. Если функция возвращает несколько значений, то она записывается в виде `[Y1, Y2....]=func(X1,X2,...)`, где `Y1`, `Y2`, `...` - список выходных параметров и `XI`, `X2`, `...` - список входных аргументов (параметров).

Со списком элементарных функций можно ознакомиться, выполнив команду `help elfun`, а со списком специальных функций - с помощью команды `help specfun`.

Функции могут быть встроенными (внутренними) и внешними (или m-функциями). Встроенными являются наиболее распространенные элементарные функции например, `sin(x)` и `exp(y)`, тогда как функция `sinh(x)` является внешней функцией. Внешние функции содержат свои определения в m-файлах. Встроенные функции хранятся в откомпилированном ядре системы MATLAB, в силу чего они выполняются предельно быстро. Скорость вычислений по внешним определениям несколько ниже, чем по встроенным функциям или операторам. При этом вычисления происходят следующим образом: вначале система быстро определяет, имеется ли введенное слово среди служебных слов системы. Если да, то нужные вычисления выполняются сразу, если нет, система ищет m-файл с соответствующим именем на диске. Если файла нет, то выдается сообщение об ошибке, и вычисления останавливаются. Если же файл найден, он загружается с жесткого диска в память машины и исполняется. Вычислим некоторые функции от матрицы. В боль-

шинстве математических систем вычисление, например, `sin(M)` и `exp(M)`, где `M` - матрица, сопровождалось бы выдачей ошибки, поскольку функции `sin` и `exp` должны иметь аргумент в виде скалярной величины. Однако MATLAB - матричная система. Поэтому в нашем случае результат вычислений будет матрицей того же размера, что и аргумент `M`, но элементы возвращаемого вектора будут синусами или экспонентами от элементов матрицы `M`. Чтобы в качестве аргумента функции выступала вся матрица целиком, следует использовать функцию от матрицы `funm(M,@sin)` или `funm(M,'sin')`, где `M` - матрица, `sin` - та функция, которую Вы хотите использовать. Однако, несколько функций от матриц имеются в списке стандартных функций MATLAB,а именно: `sqrtm(M)`, `expm(M)`, `logm(M)`.

**Задание 1.7** Вычислите `cos(M)`, `exp(M)`, `sqrt(M)`, `log(M)`. Вычислите те же функции от матриц с помощью функции `funm(M, @...)`. Сравните результаты действия данной функции с результатами известных функций `sqrtm(M)` и `expm(M)`, `logm(M)`.

### Некоторые функции формирования матриц и массивов

- `X=rand(n)` формирует массив размера $n \times n$, элементами которого являются случайные величины, распределенные по равномерному закону в интервале `(0, 1)`.
- `X=rand(m,n)` формирует массив размера $m \times n$, элементами которого являются случайные величины, распределенные по равномерному закону в интервале `(0, 1)`.
- `X=rand(size(A))` формирует массив соразмерный с матрицей `A`, элементами которог являются случайные величины, распределенные по равномерному закону в интервале `(0, 1)`.
- `rand` без аргументов формирует одно случайное число, подчиняющееся равномерному закону распределения в интервале `(0, 1)`, которое изменяется при каждом последующем вызове.
- `X=randn(n)` формирует массив размера $n \times n$, элементами которого являются случайные величины, распределенные по нормальному закону

с математическим ожиданием 0 и среднеквадратическим отклонением 1.

- Функции `randn(m, n)`, `randn(size(A))`, `randn` действуют аналогично.
- `L=tril(X)` сохраняет нижнюю треугольную часть матрицы `X`.
- `L=tril(X, k)` сохраняет нижнюю треугольную часть матрицы `X` начиная с диагонали с номером `k`. При `k>0` это номер `k`-й верхней диагонали, при `k<0` это номер `k`-й нижней диагонали.
- `U=triu(X)` сохраняет верхнюю треугольную часть матрицы (массива) `X`.
- `U=triu(X, k)` сохраняет верхнюю треугольную часть матрицы (массива) `X` начиная с диагонали с номером `k`. При `k>0` это номер `k`-й верхней диагонали, при `k<0` это номер `k`-й нижней диагонали.
- `X=diag(v)` формирует квадратную матрицу `X` с вектором `v` на главной диагонали.
- `X=diag(v,k)` формирует квадратную матрицу `X` порядка `length(v)`* `abs(k)`- с вектором `v` на `k`-й диагонали.
- `v=diag(X)` извлекает из матрицы `X` главную диагональ.
- `v=diag(X,k)` извлекает из матрицы `X` диагональ с номером `k`; при `k>0` это номер `k`-й верхней диагонали, при `k<0` это номер `k`-й нижней диагонали.

### Функции, возвращающие некоторые характеристики матриц

- `d=det(A)` вычисляет определитель квадратной матрицы; если матрица `A` целочисленная, то результатом является также целое число.
- `t=trace(A)` вычисляет след квадратной матрицы, равный сумме ее диагональных элементов.

**Задание 1.8** Ознакомьтесь с действием операций над матрицами и массивами: `rand`, `randn`, `tril`, `triu`, `diag`, `det`, `trace`.

**Применение оператора ':' (двоеточие)**

Очень часто необходимо произвести формирование упорядоченных числовых последовательностей. Такие последовательности нужны для создания векторов или значений абсциссы при построении графиков. Для этого в MATLAB используется оператор : (двоеточие):

```
Начальное_значение:Шаг:Конечное_значение
```

Данная конструкция порождает возрастающую последовательность чисел, которая начинается с начального значения, идет с заданным шагом и завершается конечным значением. Если `Шаг` не задан, то он принимает значение `1`. Если `конечное_значение` указано меньшим, чем `начальное_значение` - выдается сообщение об ошибке. Выражения с оператором : могут использоваться в качестве аргументов функций для получения множественных их значений. Таким образом, оператор : является весьма удобным средством задания регулярной последовательности чисел. Он широко используется при работе со средствами построения графиков.

## 1.2 Двумерные графики

Построение графика функций одной переменной MATLAB строит графики функций по ряду точек, соединяя их отрезками прямых, т. е. осуществляя линейную интерполяцию функции в интервале между смежными точками. Поскольку MATLAB - матричная система, совокупность точек `y(x)` задается векторами `X` и `Y` одинакового размера. Для построения графиков функций в декартовой системе координат служит команда `plot`. Эта команда имеет ряд параметров. `plot (X, Y)` - строит график функции `y(x)`, координаты точек `(x,y)` которой берутся из векторов одинакового размера `Y` и `X`. Если `X` или `Y` - матрица, то строится семейство графиков по данным, содержащимся в колонках матрицы. `plot(X,Y,'S')` - аналогична команде `plot(X,Y)`, но тип линии графика можно задавать с помощью строковой константы `S` (например `plot(x,y,'mx-')` ). С помощью строковой константы `S` можно изменять цвет линии, представлять узловые точки различными отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и

т. д.) и менять тип линии графика. Значениями константы `S` могут быть следующие символы.

| Цвет линии | Тип точки | Тип линии |
|---|---|---|
| Y желтый | . точка | - сплошная |
| M фиолетовый | o окружность | : двойной пунктир |
| C голубой | x крест | -. штрих-пунктир |
| R красный | + плюс | – штриховая |
| G зеленый | * звездочка | |
| B синий | S квадрат | |
| W белый | D ромб | |
| K черный | V треугольник (вниз) | |
| | A треугольник (вверх) | |

**Задание 1.9** Постройте какой-либо график (`sin(x)`, `cos(x)`, `x.^2`, ...). Для построения графика задайте вектор `x=Хнач:шаг:Хкон`, а затем использовать команду построения графиков `plot(x,sin(x))`. Сделайте вывод об оптимальном шаге.

Заметим, что функции из задания 1.9 могут быть обозначены переменными, не имеющими явного указания аргумента в виде `y(x): y1=sin(x); y2=cos(x); y3=:` Такая возможность обусловлена тем, что эти переменные являются векторами - как и переменная `x`.

Графики MATLAB строит в отдельных окнах, называемых графическими окнами. В главном меню окна появилась позиция `Edit` (Редактировать), которая позволяет вывести или скрыть панели управления параметрами рисунка (`Figure Properties`), параметрами осей (`Axes Properties`) и параметрами выделенного объекта (`Current Object Properties`), где в качестве такого объекта могут служить как весь рисунок или оси, так и часть рисунка. Позиция `Insert` (Вставить) позволяет подписать оси (`Xlabel`, `Ylabel`, `Zlabel`), вставить в указанное место рисунка текст (`Text`), стрелку (`Arrow`) или линию (`Line`), а также выполнить некоторые другие операции. Позиция `File` (Файл) окна графики содержит типовые файловые операции. Однако они относятся не к файлам документов, а к файлам графиков. В частности, можно присваивать имя записываемым на диск рисункам с графиками.

**Задание 1.10** Измените с помощью позиции `Edit` цвет графика и нанесите надписи на график, используя позицию `Insert`. Сохраните график в файл. Изучите другие возможности позиций `Edit` и `Insert`.

Для построения графиков функций, заданных параметрически служит та же команда `plot`. В качестве обоих аргументов выступают функции.

**Задание 1.11** Постройте параметрический график `plot(sin(x), cos(x))`.

**Задание 1.12** Фигуры Лиссажу Постройте график в полярных координатах. `t=0:pi/100:4*pi`; `plot(sin(w*t),cos(v*t))`.
Сначала возьмите соизмеримые (отношение `w/v` - рациональное число) значения частот `w` и `v`, например `w=2`, `v=3`. Затем возьмите не соизмеримые частоты, например `w=2`, `v=pi`. Убедитесь, что независимо от диапазона изменения аргумента `t`, получится незамкнутая кривая.

### Построение в одном окне графиков нескольких функций

Команда
`plot(X1,Y1,'S1',X2,Y2,'S2',X3,Y3,'S3',...)`
строит на одном графике ряд линий, представленных данными вида `(X,Y,'S')`, где `X` и `Y` - векторы или матрицы, а `S` - строки. С помощью такой конструкции возможно как построение графиков нескольких функций, так и графика одной функции линией, цвет которой отличается от цвета узловых точек. Так, если надо построить график функции линией синего цвета с красными точками, то вначале надо задать построение графика с точками красного цвета (без линии), а затем графика только линии синего цвета (без точек).

**Задание 1.13** Построить в одном окне графики трех функций: `q./(x.^2+q.^2)`, при различных значениях `q=1`, `q=0.5`, `q=0.1`, причем цвет одного из графиков должен отличаться от цвета его узловых точек.

Разумеется, MATLAB имеет средства для построения графиков и таких функций, как `sin(x)/x`, которые имеют устранимые неопределенности, с помощью другой графической команды
`fplot`: `fplot('f(x)', [xmin,xmax])`

Она позволяет строить функцию, заданную в символьном виде, в интервале изменения аргумента `x` от `xmin` до `xmax` без фиксированного шага изменения `x`.

**Задание 1.14** Постройте функцию `sin(x)/x` с помощью функции `fplot`.

### Графики в логарифмическом масштабе

Для построения графиков функций со значениями `x` и `y`, изменяющимися в широких пределах, нередко используются логарифмические масштабы. Рассмотрим команды, которые используются в таких случаях. `loglog(...)` - синтаксис команды аналогичен ранее рассмотренному для функции `plot(...)`. Логарифмический масштаб используется для координатных осей `X` и `Y`. Ниже дан пример применения данной команды:

```
x=logspace(-1,1);
loglog(exp(x)./x)
grid on
```

**Задание 1.15** Постройте график функции `exp(x)./x` в логарифмическом масштабе. Вектор `X` задается следующим образом `x=logspace(-1,3)`. Поясните, что это значит. Нанесите масштабную сетку с помощью команды `grid on`.

Обратите внимание на то, что командой `grid on` строится координатная сетка. Неравномерное расположение линий координатной сетки указывает на логарифмический масштаб осей.

### Графики в полулогарифмическом масштабе

В некоторых случаях предпочтителен полулогарифмический масштаб графиков, когда по одной оси задается логарифмический масштаб, а по другой - линейный. Для построения графиков функций в полулогарифмическом масштабе используются следующие команды: `semilogx(...)` - строит график функции в логарифмическом масштабе (основание 10) по оси `X` и линейном по оси `Y`; `semilogy(...)` - строит график функции в логарифмическом масштабе по оси `Y` и линейном по оси `X`. Запись параметров выполняется по аналогии с функцией `plot(...)`.

**Задание 1.16** Постройте график экспоненциальной функции в логарифмическом масштабе по оси `Y`. Вектор `x` задайте самостоятельно.

Нетрудно заметить, что при таком масштабе график экспоненциальной функции выродился в прямую линию.

### Вывод заголовка

После того как график уже построен, MATLAB позволяет выполнить его форматирование или оформление в нужном виде. Так, для установки над графиком заголовка используется следующая команда: `title('string')` - установка на двумерных и трехмерных графиках титульной надписи, заданной строковой константой `'string'`.

### Подпись осей

Для подписи осей `x`, `y` и `z` используются следующие команды:
`xlabel('String'),`
`ylabel('String'),`
`zlabel('String')`
Соответствующая надпись задается символьной константой или переменной `'String'`.

**Задание 1.17** Подпишите график с помощью команды `title`. Подпишите оси.

### Ввод текста в любое место графика

Часто возникает необходимость добавления текста в определенное место графика, например для обозначения той или иной кривой графика. Для этого используется команда `text`: `text(X,Y,'string')` - добавляет в двумерный график текст, заданный строковой константой `'string'`, так что начало текста расположено в точке с координатами `(X, Y)`. Если `X` и `Y` заданы как одномерные массивы, то надпись помещается во все позиции `[x(i),y(i)]`;

### Вывод пояснений

Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется легендой. Для созда-

ния легенды используются различные варианты команды `legend`. Приведем две из них: `legend(string1,string2,string3,...)` - добавляет к текущему графику легенду в виде строк, указанных в списке параметров; `legend (string1,string2,string3,...,Pos)` - помещает легенду в точно определенное место, специфицированное параметром `Pos`: `Pos=0` - лучшее место, выбираемое автоматически; `Pos=1` - верхний правый угол; `Pos=2` - верхний левый угол; `Pos=3` - нижний левый угол; `Pos=4` - нижний правый угол; `Pos=-1` - справа от графика. Чтобы перенести легенду, установите на нее курсор, нажмите левую кнопку мыши и перетащите легенду в необходимую позицию. Двойным щелчком можно вывести легенду на редактирование.

**Задание 1.18** постройте график трех функций с легендой, размещенной в поле графика.

### Наложение графиков друг на друга

Во многих случаях желательно построение многих наложенных друг на друга графиков в одном и том же окне. Для этого служит команда продолжения графических построений `hold`. Она используется в следующих формах: `hold on` - обеспечивает продолжение вывода графиков в текущее окно, что позволяет добавлять последующие графики к уже существующим; `hold off` - отменяет режим продолжения графических построений; `hold` - работает как переключатель, последовательно включая режим продолжения графических построений и отменяя его.

**Задание 1.19** Постройте график функции `0.5*x+0.1*sin(25*x)`. С помощью команды `hold on` на этом же графике постройте график `abs(x)`.

Для построения нескольких графических объектов также используется команда `drawnow`.

**Задание 1.20** В качестве примера рассмотрите построение графика расширяющегося облака газа - модель броуновского движения, которое реализуется с помощью генератора случайных чисел:

```
shg
```

```
clf
set(gcf,'doublebuffer','on')
delta = .002;
x = zeros(100,2);
h = plot(x(:,1),x(:,2),'.');
axis([-1 1 -1 1])
axis square
stop =uicontrol('style','toggle','string','stop');
while get(stop,'value') == 0
   x = x + delta*randn(size(x));
   set(h,'xdata',x(:,1),'ydata',x(:,2))
   drawnow
end
```

### Разбиение графического окна

Бывает, что в одном окне надо расположить несколько координатных осей с различными графиками без наложения их друг на друга. Для этого используются команда `subplot`, применяемые перед построением графиков: `subplot(m,n,p)` или `subplot(m n p)` - разбивает графическое окно на $m \times n$ подокон, при этом `m` - число подокон по горизонтали, `n` - число подокон по вертикали, а `p`- номер подокна, в которое будет выводиться текущий график (подокна отсчитываются последовательно по строкам).

**Задание 1.21** Постройте в одном графическом окне 4 графика (2 по горизонтали, 2 по вертикали) `sin(x)./x` в интервале `-3*pi:3*pi`, `sin(5*x)`, `cos(2*x+0.2)=`, `cos.^2(x)`.

Для всех графиков возможна индивидуальная установка дополнительных объектов, например титульных надписей, надписей по осям и т. д.

В отличие от двумерных (2D) графиков форматирование трехмерных графиков содержит ряд дополнительных возможностей. Покажем их на простом примере построения 3D-графики с помощью следующих простых команд:

```
Z=peaks(40):
mesh(Z);
```

## 1.3 Движение точки

Для отображения движения точки по траектории используется команда `comet`. При этом движущаяся точка напоминает ядро кометы с хвостом. Используются следующие формы представления этой команды:

- `comet (Y)` - отображает движение <кометы> по траектории, заданной вектором Y;
- `comet (X.Y)` - отображает движение <кометы> по траектории, заданной парой векторов Y и X;
- `comet (X.Y.p)` - аналогична предшествующей команде, но позволяет задавать длину хвоста кометы (отрезка траектории, выделенного цветом) как p*1ength(Y), где length(Y) - размер вектора Y. а p<1. По умолчанию p = 0.1

Обратите внимание, что если Вы используете лупу, как-то иначе пытаетесь изменить размер Вашего рисунка или используете вкладку Copy Figure меню Edit, то график, полученный при использовании `comet` или `comet3`, исчезает. Следующий пример иллюстрирует применение команды `comet`:

```
X=0:0.01:15;
comet(X,sin(X) )
```

Есть еще одна команда, которая позволяет наблюдать движение точки, но уже в трехмерном пространстве. Это команда `comet3`:

- `comet3(Z)` - отображает движение точки с цветным "хвостом" по трехмерной кривой, определенной массивом Z;
- `comet3(X,Y,Z)` - отображает движение точки "кометы" по кривой в пространстве, заданной точками [X(i),Y(i),Z(i)];
- `comet3(X,Y,Z,p)` - аналогична предшествующей команде с заданием длины "хвоста кометы" как `p*length(Z)`. По умолчанию параметр p равен 0.1.

Ниже представлен пример применения команды `comet3`:

```
W=0:pi/500:10*pi;
comet3(cos(W),sin(W)+W/10.W)
```

## 1.4 Построение графиков поверхностей - справка

Команда `plot3(...)` является аналогом команды `plot(...)`, но относится к функции двух переменных `z(x, y)`. Она строит аксонометрическое изображение трехмерных поверхностей и представлена следующими формами:

- `plot3(x,y,z)` - строит массив точек, представленных векторами x, y и z, соединяя их отрезками прямых. Эта команда имеет ограниченное применение;
- `plot3(X,Y,Z)` где X, Y и Z - три матрицы одинакового размера, строит точки с координатами X(i.:), Y(i,:) и Z(i,:) и соединяет их отрезками прямых.

Ниже дан пример построения трехмерной поверхности, описываемой функцией $z(x, y) = x^2 + y^2$

```
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z)
```

### График поверхности, построенный линиями

`plot3 (X, Y, Z, S)` - обеспечивает построения, аналогичные рассмотренным ранее, но со спецификацией стиля линий и точек, соответствующей спецификации команды `plot`. Ниже дан пример применения этой команды для построения поверхности кружками:

```
[X.Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
plot3(X,Y,Z.'o')
```

`plot3(xl ,yl,zl,sl,x2,y2,z2,s2,x3,y3,z3,s3,...)`- строит на одном рисунке графики нескольких функций zl(xl ,yl), z2(x2,y2) и т. д. со спецификацией линий и маркеров каждой из них.

Пример применения последней команды дан ниже:

```
[X.Y]=meshgrid([-3:0.15:3]);
```

```
Z=X.^2+Y.^2;
plot3(X,Y,Z,'-k',Y,X,Z,'-k')
```

### Построение поверхности с окраской

Особенно наглядное представление о поверхностях дают сетчатые графики, использующие функциональную закраску ячеек. Например, цвет окраски поверхности z(x, y) может быть поставлен в соответствие с высотой z поверхности с выбором для малых высот темных тонов, а для больших - светлых. Для построения таких поверхностей используются команды класса `surf (...)`:

- `surf (X, Y, Z, C)` - строит цветную параметрическую поверхность по данным матриц X, Y и Z с цветом, задаваемым массивом C;
- `surf(X,Y,Z)` - аналогична предшествующей команде, где C=Z, так что цвет задается высотой той или иной ячейки поверхности;
- `surf(x,y,Z)` и `surf(x,y,Z,C)` с двумя векторными аргументами x и y - векторы x и y заменяют первых два матричных аргумента и должны иметь длины length(x)=n и length(y)=m, где [m.n]=size(Z). В этом случае вершины областей поверхности представлены тройками координат (x(j), yd), Z(1,j)). Заметим, что x соответствует столбцам Z, а y соответствует строкам;
- `surf(Z)` и `surf(Z.C)` используют x = 1:n и y = 1:m. В этом случае высота Z - однозначно определенная функция, заданная геометрически прямоугольной сеткой;
- `h=surf (...)` -строит поверхность и возвращает дескриптор объекта класса surface.

Команды `axis, caxis, color-map, hold, shading и view` задают координатные оси и свойства поверхности, которые могут использоваться для большей эффектности показа поверхности или фигуры.

Ниже приведен простой пример построения поверхности - параболоида:

```
[X,Y]=meshgrid([-3:0.15:3]);
Z=X.^2+Y.^2;
surf(X,Y,Z)
```

### Построение освещенной поверхности

Пожалуй, наиболее реалистичный вид имеют графики поверхностей, в которых имитируется освещение от точечного источника света, расположенного в заданном месте координатной системы. Графики имитируют оптические эффекты рассеивания, отражения и зеркального отражения света. Для получения таких графиков используется команда `surfl`:

- `surfl(...)` - аналогична команде `surf (...)`, но строит график поверхности с подсветкой от источника света;
- `surfl (Z.S)` или`surfl(X,Y,Z,S)` - строит графики поверхности с подсветкой от источника света, положение которого в системе декартовых координат задается вектором S=[Sx,Sy.Sz], а в системе сферических координат - вектором S=[AZ.EL];
- `surfl (..., 'light')` - позволяет при построении задать цвет подсветки с помощью объекта Light;
- `surfl (..., 'cdata')` - при построении имитирует эффект отражения;
- `surfl(X,Y,Z,S,K)` - задает построение поверхности с параметрами, заданными вектором K = [ka, kd, ks, spread], где ka - коэффициент фоновой подсветки, kd - коэффициент диффузного отражения, ks - коэффициент зеркального отражения и spread - коэффициент глянцевитости;
- `H=surf 1 (...)` - строит поверхность и возвращает дескрипторы поверхности и источников света.

По умолчанию вектор S задает углы азимута и возвышения в 45°. Используя команды `cla`, `hold on`, `view(AZ,EL)`, `surfl (...)` и `hold off`, можно получить дополнительные возможности управления освещением. Надо полагаться на упорядочение точек в X, Y, и Z матрицах, чтобы определить внутреннюю и внешнюю стороны параметрических поверхностей. Попробуйте транспонировать матрицы и использовать `surf(X',Y',Z')`, если вам не понравился результат работы этой команды. Для вычисления векторов нормалей поверхности `surfl` требует в качестве аргументов матрицы с размером по крайней мере 3x3.

Ниже представлен пример применения команды `surfl`:

```
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
surfl(X,Y,Z)
colormap(gray)
shading interp
colorbar
```

### Построение сечений функций трех переменных

Графики сечений функций трех переменных строит команда `slice` (в переводе - "ломтик"). Она используется в следующих формах:

- `slice(X,Y,Z,V,Sx,Sy,Sz)` - строит плоские сечения объемной фигуры V в направлении осей x,y,z с позициями, задаваемыми векторами Sx, Sy, Sz. Массивы X, Y, Z задают координаты для V и должны быть монотонными и трехмерными (как возвращаемые функцией `meshgrid`) с размером MxNxP. Цвет точек сечений определяется трехмерной интерполяцией в объемной фигуре V;
- `slice(X,Y,Z,V,XI,YI,ZI)` - строит сечения объемной фигуры V по поверхности, определенной массивами XI, YI, ZI;
- `slice (... 'method')` - при построении задается метод интерполяции, который может быть одним из следующих: 'linear', 'cubic' или 'nearest'. По умолчанию используется линейная интерполяция - 'linear';
- `slice(V,Sx,Sy,Sz)` или `slice(V,XI,YI,ZI)` - подразумевается X=1:N, Y=1:M, Z=1:P;
- `H=slice(...)` - строит сечение и возвращает дескриптор объекта класса
- `surface`.

Пример

```
[x,y,z]= meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
v = sin(x) .* exp(-x.*2 - y.^2 - z.^2);
slice(x,y,z,v,[-1.2 .8 2],2,[-2 -.2])
```

# Работа № 2

## Программирование в MATLAB

До сих пор мы в основном использовали систему MATLAB в командном режиме. Однако при решении серьезных задач возникает необходимость сохранения последовательностей вычислений, а также их дальнейшей модификации. Иными словами, существует необходимость программирования решения задач. Система MATLAB имеет входной язык, напоминающий Бейсик (с примесью Фортрана и Паскаля). Запись программ в системе традиционна. Система дает возможность редактировать программы с помощью любого привычного для пользователя текстового редактора. Имеет она и собственный редактор с отладчиком. Язык системы MATLAB в части программирования математических вычислений намного богаче любого универсального языка программирования высокого уровня. Он реализует почти все известные средства программирования, в том числе объектно-ориентированное и (средствами Simulink) визуальное программирование. Это дает опытным программистам необъятные возможности для самовыражения. Большинство объектов языка программирования MATLAB, в частности все команды, операторы и функции, одновременно являются объектами входного языка общения с системой в командном режиме работы. Программы на языке программирования MATLAB сохраняются в виде текстовых m-файлов (файл с расширением `.m`). При этом могут сохраняться как целые программы в виде файлов-сценариев, так и отдельные программные модули - функции.

## 2.1 Основные средства программирования

Язык программирования системы MATLAB имеет следующие средства: данные различного типа; константы и переменные; операторы, включая, операторы математических выражений; встроенные команды и функции; функции пользователя; управляющие структуры; системные операторы и функции; средства расширения языка. Язык программирования MATLAB является типичным интерпретатором. Это означает, что каждая инструкция программы распознается и тут же исполняется, что облегчает обеспечение диалогового режима общения с системой. Этап компиляции всех инструкций, т. е. полной программы, отсутствует. Интерпретация означает, что MATLAB не создает исполняемых конечных программ. Они существуют лишь в виде m-файлов. Для выполнения программ необходима среда MATLAB.

## 2.2 Текстовые комментарии

Поскольку MATLAB используется для достаточно сложных вычислений, важное значение имеет наглядность их описания. Она достигается, в частности, с помощью текстовых комментариев. Текстовые комментарии вводятся с помощью символа %, например, так:

```
% This is my first \textsf{MATLAB} program
```

Обычно первые строки m-файлов служат для описания их назначения. Основным комментарием является первая строка текстовых комментариев, а дополнительным - последующие строки. Основной комментарий выводится при выполнении команд `lookfor` и `help имя_каталога`. Полный комментарий выводится при выполнении команды `help имя_файла`. Знак % в комментариях должен начинаться с первой позиции строки. В противном случае команда `help имя_файла` не будет воспринимать комментарий (иногда это может понадобиться) и возвратит сообщение вида `No help comments found in name.m`. Считается правилом хорошего тона вводить в m-файлы достаточно подробные текстовые комментарии. Без таких комментариев даже разработчик программных модулей быстро забывает о сути собственных решений. В текстовых комментариях и в символьных константах могут использоваться

буквы русского алфавита - при условии, что установлены содержащие эти буквы наборы шрифтов. Команда `help имя_файла` обеспечивает чтение первой строки с текстовым комментарием и тех строк с комментариями, которые следуют непосредственно за первой строкой до первой пустой стоки. Комментарий, расположенный за пределами этой области, не выводится. Это позволяет создавать не выводимый программный комментарий. Пустая строка прерывает вывод комментария при исполнении команды `help имя_файла`. Команда `type имя_файла` выводит текст программы со всеми комментариями, в том числе и следующими после пустых строк. Команда `help имя_каталога`, где `имя_каталога` - имя каталога с m-файлами, позволяет вывести комментарий, общий для всего каталога. Такой комментарий содержится в файле `contents.m`, который пользователь может создать самостоятельно с помощью редактора m-файлов. Если такого файла нет, то будет выведен список первых строк комментариев для всех m-файлов каталога.

### M-файлы сценариев и функций

Для создания m-файлов может использоваться как встроенный редактор, так и любой текстовый редактор, поддерживающий формат ASCII. Подготовленный и записанный на диск m-файл становится частью системы, и его можно вызывать как из командной строки, так и из другого m-файла. Записывать m-файл можно в любой каталог, однако с помощью команды path следует проследить, чтобы этот каталог содержался в путях доступа MATLAB. Если вашего каталога в списке путей доступа нет, следует использовать команду:

```
path (path,'полный путь к каталогу')
```

Однако данная команда действительна только на одну сессию MATLAB (в следующий раз придется давать её заново, либо обратиться к администратору класса, чтобы ваш каталог внесли в список путей доступа). Есть два типа m-файлов: файлы-сценарии и файлы-функции. Важно, что в процессе своего создания (если используется редактор-отладчик MATLAB'а) они проходят синтаксический контроль с помощью встроенного в систему MATLAB редактора/отладчика m-файлов.

### Структура и свойства файлов сценариев

Файл-сценарий, именуемый также Script-файлом, является просто записью серии команд без входных и выходных параметров. Файлы-сценарии имеют следующие свойства:

- они не имеют входных и выходных аргументов;
- работают с данными из рабочей области;
- в процессе выполнения не компилируются;
- представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется в сессии.

Рассмотрим следующий файл-сценарий:

```
%Plot with color red
%Строит график синусоиды линией красного цвета
%с выведенной масштабной сеткой в интервале [xmin,xmax]
 x=xmin:0.1:xmax;
 plot(x,sin(x),'r')
 grid on
```

Первые три строки здесь - это комментарий, остальные - тело файла. Обратите внимание на возможность задания комментария на русском языке (предупреждение: в MATLAB 6.x нельзя использовать русскую букву "с", следует заменять ее латинской "c"). Обратите внимание на то, что такой файл нельзя запустить без предварительной подготовки, сводящейся к заданию значений переменным `xmin` и `xmax`, использованным в теле файла. Это следствие первого свойства файлов-сценариев - они работают с данными из рабочей области. Переменные, используемые в файлах-сценариях, являются глобальными, т. е. они действуют одинаково в командах сессии и внутри программного блока, которым является файл-сценарий. Поэтому заданные в сессии значения переменных используются и в теле файла. Имена файлов-сценариев нельзя использовать в качестве параметров функций, поскольку файлы-сценарии не возвращают значений. Можно сказать,

что файл-сценарий - это простейшая программа на языке программирования MATLAB.

**Задание 2.1** Откройте текстовый редактор. Составьте файл-сценарий из тех команд, которые изучались в работе №1 (например, постройте график какой-либо функции). Используйте комментарии. Сохраните его в домашний каталог с именем `my_file.m`. Запустите его. Дайте команду `help my_file`.

### Структура М-файла-функции

М-файл-функция имеет следующие свойства:

- он начинается с объявления `function`, после которого указывается имя переменной `var` - выходного параметра, имя самой функции и список ее входных параметров;
- функция возвращает свое значение и может использоваться в виде `name (Список_параметров)` в математических выражениях;
- все переменные, имеющиеся в теле файла-функции, являются локальными, т. е. действуют только в пределах тела функции;
- файл-функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;
- правила вывода комментариев те же, что у файлов-сценариев;
- файл-функция служит средством расширения системы MATLAB;
- при обнаружении файла-функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы MATLAB.

М-файл-функция является типичным объектом языка программирования системы MATLAB. Одновременно он является полноценным модулем с точки зрения структурного программирования, поскольку содержит входные и выходные параметры и использует аппарат локальных переменных. Структура такого модуля с одним выходным параметром выглядит следующим образом:

```
function var=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
var=выражение
```

Последняя конструкция `var=выражение` вводится, если требуется, чтобы функция возвращала результат вычислений. Переменные, указанные в списке параметров функции, являются локальными и служат для переноса значений, которые подставляются на их место при вызовах функций. Возврат из функции производится после обработки всего тела функции, т. е. при достижении конца файла функции. При использовании в теле функции условных операторов, циклов или переключателей иногда возникает необходимость осуществить возврат функции раньше, чем будет достигнут конец файла. Для этого служит команда `return`. В любом случае, результатом, возвращаемым функцией, являются значения выходных параметров, присвоенные им на момент возврата. Приведенная выше форма файла-функции характерна для функции с одним выходным параметром. Если выходных параметров больше, то они указываются в квадратных скобках после слова `function`. При этом структура модуля имеет следующий вид:

```
 function [var1,var2....]=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
var1=выражение
var2=выражение
```

Такая функция во многом напоминает процедуру. Ее нельзя использовать непосредственно в математических выражениях, поскольку она возвращает не единственный результат, а множество результатов - по числу выходных параметров. Если функция используется как имеющая единственный выходной параметр, но имеет ряд выходных параметров, то для возврата значения будет использоваться первый из них. Это зачастую ведет к ошибкам в математических вычислениях. Поэтому, как отмечалось, данная функция используется как отдельный элемент программ вида:

`[var1,var2,...]=f_name(Список_параметров)`

После его применения переменные выхода `var1`, `var2`,... становятся определенными и их можно использовать в последующих математических выражениях и иных сегментах программы. Если функция используется в виде `name(Список_параметров)`, то возвращается значение только первого выходного параметра - переменной `var1`.

### Особенности выполнения m-файлов функций

M-файлы-функции могут использоваться как в командном режиме, так и вызываться из других m-файлов (наравне со встроенными функциями MATLAB, такими как `sin(x)`, `exp(x)`, ...). При этом необходимо указывать все входные и выходные параметры. Исключением является случай, когда выходной параметр единственный - в этом варианте функция возвращает единственный результат и может использоваться в математических выражениях. При использовании глобальных переменных они должны быть объявлены во всех m-файлах, используемых в решении заданной задачи, и во всех входящих в них встроенных подфункциях. Имена функций должны быть уникальными. Это связано с тем, что при обнаружении каждого нового имени MATLAB проверяет, относится ли это имя к переменной, подфункции в данном m-файле, частной функции в каталогах `PRIVATE` или функции в одном из каталогов пути доступа. Если последняя встречается, то будет исполнена именно эта функция. Если аргумент функции используется только для вычислений, и его значения не меняются, то аргумент передается ссылкой, что уменьшает затраты памяти. В других случаях аргумент передается значением.

**Задание 2.2** Составьте m-файл-функцию с одним выходным параметром. Вычислите функцию

$$y(x) = 1 - \frac{x^2}{2}.$$

**Задание 2.3** Составьте m-файл-функцию с несколькими выходными параметрами. Вычислите

$$x_1(t) = \cos(2\pi t), \qquad x_2(t) = \sin(2\pi t).$$

## 2.3 Управляющие структуры языка MATLAB

Поскольку язык программирования системы MATLAB ориентирован на структурное программирование, в нем нет номеров строк и программных операторов безусловного перехода `GO TO`.

В MATLAB имеются лишь управляющие структуры следующих типов: условных выражений `if...else...elseif...end`, циклы `for...end` и `while...end`. С позиций теории структурного программирования этих средств достаточно для решения любых задач. В MATLAB имеются также операторы-переключатели типа `case`. Однако в MATLAB исключены те средства, возможности которых можно реализовать уже имеющимися средствами. Зато резко увеличен набор средств программирования для решения математических задач, прежде всего сводящихся к матричным вычислениям и реализации современных численных методов. Помимо программ с линейной структурой, инструкции которых исполняются строго по порядку, существует множество программ, структура которых нелинейная. При этом ветви программ могут выполняться в зависимости от определенных условий, иногда с конечным числом повторений - циклов, иногда в виде циклов, завершаемых при выполнении заданного условия. Практически любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры.

### Функции ввода-вывода

Для ввода пользователем значений переменных служит функция `input`. При использовании данной функции в следующем виде:

```
d=input('Введите значение переменной d=');
```

переменной `d` присваивается значение, введенное пользователем после запроса `'введите значение переменной'`. Функция `input` может использоваться и для ввода произвольных строковых выражений. При этом она задается в следующем виде:

```
s=input('Введите выражение','s')
```

При выполнении этой функции она останавливает вычисления и ожидает ввода строкового комментария. После ввода возвращается набранная строка. В результате действия данной команды строковой переменной `s` будет присвоено значение введенной пользователем строки символов. Чтобы вычислить выражение, заданное (полученное от функции `input`) в символьном виде существует функция `eval(s)`.

**Задание 2.4** Проверьте действие команды `input` в режиме командной строки. В качестве строковой переменной введите какое-либо выражение, например $sin(2 * x)$, $x$ при этом должен быть определен. Вычислите значение заданной функции с помощью функции `eval`.

Для вывода на экран текста и значений переменных служит функция `disp`:

```
disp('pi='); disp(pi)
```

**Задание 2.5** В текстовом редакторе создайте программу вычисления площади круга с заданием радиуса. Сохраните этот файл. Вызовите из командной строки написанную программу.

### Условный оператор

Условный оператор `if` в общем виде записывается следующим образом:

```
 if Условие,
Инструкции_1, elseif Условие, Инструкции_2, else Инструкции_3, end
```

Эта конструкция допускает несколько частных вариантов. Простейший:

```
 if Условие Инструкции end
```

Пока Условие возвращает логическое значение `1` (то есть "истина"), выполняются `Инструкции`, составляющие тело структуры `if...end`. При этом оператор `end` указывает на конец перечня инструкций. Инструкции в списке разделяются оператором `,` (запятая) или `;` (точка с запятой). Если Условие не выполняется (дает логическое значение `0`, "ложь"), то Инструкции также не выполняются. Еще одна конструкция

```
 if Условие, Инструкции_1, else
Инструкции_2, end
```

выполняет Инструкции_1, если выполняется Условие, или Инструкции_2 в противном случае. Условия записываются в виде:

Выражение_1 Оператор_отношения Выражение_2,

причем в качестве Операторов_отношения используются следующие операторы: ==, <, >, <=, >= или ~=. Все эти операторы представляют собой пары символов без пробелов между ними.

**Пример 1:** Простейшая программа на языке MATLAB с использованием условного оператора. Вычисление корней квадратного уравнения.

```
% Решение квадратного уравнения

a=input('Input coefficient a=');
b=input('Input coefficient b=');
c=input('Input coefficient c=');

% Вычисление дискриминанта
d= b^2-4*a*c; if d>0
    disp('Корни вещественны')
    disp('x1='), disp((-b+sqrt(d))/(2*a))
    disp('x2='), disp((-b-sqrt(d))/(2*a))
elseif d==0
    disp('Кратные корни')
    disp('x1=x2='), disp(-b/(2*a))
else
    disp('Корни комплексные')
    disp('x1='), disp((-b+sqrt(d))/(2*a))
    disp('x2='), disp((-b-sqrt(d))/(2*a))
end
%Конец программы
```

**Задание 2.6** Проверить работоспособность программы из примера 1.

### Циклы типа for...end

Циклы типа for...end обычно используются для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет

следующий вид:

```
for var=Выражение,
Инструкция,
...,
Инструкция,
end
```

Выражение чаще всего записывается в виде `s:d:e`, где `s` - начальное значение переменной цикла `var`, `d` - приращение этой переменной и `e` - конечное значение управляющей переменной, при достижении которого цикл завершается. Возможна и запись в виде `s:e` (в этом случае `d=1`). Список выполняемых в цикле инструкций завершается оператором `end`. Оператор `continue` передает управление в следующую итерацию цикла, пропуская операторы, которые записаны за ним, причем во вложенном цикле он передает управление на следующую итерацию основного цикла. Оператор `break` может использоваться для досрочного прерывания выполнения цикла. Как только он встречается в программе, цикл прерывается. Возможны вложенные циклы. MATLAB допускает использование в качестве переменной цикла массива `A` размера $m \times n$. При этом цикл выполняется столько раз, сколько столбцов в массиве `A`, и на каждом шаге переменная `var` представляет собой вектор, соответствующий текущему столбцу массива `A`.

**Пример 2:** функция на языке MATLAB с использованием цикла типа `for...end`. Вычисление факториала.

```
% Факториал
function f=factorial(n)
 if n==0
  v=1;
 else
  v=1,
  for i=1:n, v= v*I, end,
 end
f=v
```

**Задание 2.7** Проверить работоспособность программы из примера 2.

### Циклы типа while...end

Цикл типа `while` выполняется до тех пор, пока выполняется `Условие`:

```
while Условие,
Инструкции
end
```

Досрочное завершение циклов реализуется с помощью операторов `break` или `continue`.

**Пример 3:** Программа на языке MATLAB с использованием цикла типа `while...end`. Приближенное вычисление бесконечной суммы.

```
% Вычисление числа \pi=4*(1-1/3+1/5-1/7+:)
% Из-за медленной сходимости ряда точность eps не рекомендуется
% брать меньше 0.0001

eps=input('задайте точность')
s=0;
k=0;
while 1/(2*k+1)>eps
    s=s+(-1)^k/(2*k+1);
    k=k+1;
end
disp('pi='),
disp(s*4)
```

**Задание 2.8** Проверить работоспособность программы из примера 3.

### Конструкция переключателя

Для осуществления множественного выбора (или ветвления) используется конструкция с переключателем типа `switch`:

```
switch switch_Выражение
case case_Выражение
Список_инструкций
case{case_Выражение1, case_выражение2, case_Выражение3,...}
Список_инструкций
Otherwise
Список_инструкций
end
```

Если выражение после заголовка `switch` имеет значение одного из выражений `case_Выражение`, то выполняется блок операторов `case`, в противном случае - список инструкций после оператора `otherwise`. При выполнении блока `case` исполняются те списки инструкций, для которых `case_Выражение` совпадает со `switch_Выражением`. Обратите внимание на то, что `case_Выражение` может быть числом, константой, переменной, вектором ячеек или даже строчной переменной. В последнем случае оператор `case` истинен, если функция `strcmp` (значение, выражение) возвращает логическое значение "истина".

Создание паузы в вычислениях Для остановки программы используется оператор `pause`. Он используется в следующих формах: `pause` - останавливает вычисления до нажатия любой клавиши; `pause(N)` - останавливает вычисления на N секунд; `pause on` - включает режим отработки пауз; `pause off` - выключает режим отработки пауз.

**Задание 2.9** Для заданной функции (задается преподавателем) написать m-файл-функцию и сохранить его в домашнем каталоге.

**Задание 2.10** Напишите m-файл-сценарий (программу) для вычисления максимума и минимума запрограммированной Вами функции на заданном (преподавателем) интервале `(a,b)`, вычислив значения заданной функции на интервале `(a,b)` с шагом `(b-a)/n` (`n` задается преподавателем) и найдя из них максимальное и минимальное значение. Использовать операторы цикла `for:end`, `if:end`.

# Работа № 3

## Полиномиальная интерполяция

## 3.1 Теоретическая справка

### Задача интерполяции

Пусть задана таблица чисел $\{x_i,\, f_i\}_{i=0}^{N},\ \ i = 0\,,\, 1\,,\, ...\,,\, N\ ;\ \ x_0 < x_1 < ... < x_N$ .

**Определение**. Всякая функция $f(x)$ такая, что $f(x_i) = f_i,\ i = 0\,,\, 1\,,\, ...\,,\, N,$ называется *интерполирующей (интерполяцией)* для таблицы $\{x_i\,,\, f_i\,\}_{i=0}^{N}$ .

Задача интерполяции состоит в построении интерполирующей функции (т.е. принимающей в заданных узлах интерполяции $x_i$ заданные значения $f_i$ ), принадлежащей заданному классу функций. Один из способов интерполяции состоит в том, что интерполирующая функция ищется в виде линейной комбинации некоторых конкретных функций. Такая интерполяция называется линейной.

### Интерполяционный полином в форме Лагранжа

$$p(x) = \sum_{j=0}^{N} f_j \prod_{k \neq j} \frac{x - x_k}{x_j - x_k}\ .$$

### Интерполяционный полином в форме Ньютона

$$p(x) = \sum_{k=0}^{N} f_{012...k} \prod_{i=0}^{k-1} (x - x_i)\ .$$

где разделенные разности $f_{012\ldots k}$ определяются рекурентно

$1^{\text{го}}$ порядка — $f_{ij} = f(x_i, x_j) = \frac{f_i - f_j}{x_i - x_j}$ ;

$2^{\text{го}}$ порядка — $f_{ijk} = f(x_i, x_j, x_k) = \frac{f_{ij} - f_{jk}}{x_i - x_k}$ ;

. . . . . .

$k^{\text{го}}$ порядка — $f_{\beta_0,\beta_1\ldots\beta_k} = \frac{f_{\beta_0\beta_1\ldots\beta_{k-1}} - f_{\beta_1\beta_2\ldots\beta_k}}{x_0 - x_k}$ .

## 3.2 Использование встроенных функций MATLAB

В MATLAB имеется встроенная функция `polyfit`, которая строит полином заданной степени аппроксимирующий таблицу в смысле наименьшей суммы квадратов отклонений. При числе узлов $N$ и степени полинома $N-1$, существует полином, интерполирующий таблицу. Функция `polyfit` возвращает коэффициенты такого полинома. Для вычисления значения полинома, заданного своими коэффициентами служит встроенная функция `polyval`.

Построим интерполяцию функции $f(x) = \exp(x)$ на интервале $-1 \le x \le 1$. Для этого сначала зададим массив `X`, содержащий 5 узлов $x_k$ с равным шагом.

```
X=[-1,-.5,0,.5,1]"
```

Затем вычислим значения функции $f(x)$ и поместим их в массив `F`

```
F = exp(X)
```

Теперь найдем интерполяционный полином

```
P = polyfit(X,F,4)
```

Проверим выполнение условий интерполяции

```
polyval(P,X)-F
```

Заметим, что погрешность наших вычислений имеет порядок $10^{-15}$.

Построим график функции $f(x)$ и интерполяционного полинома. Для этого зададим большой массив `x` значений аргумента на интервале интерполяции

```
x=-1:-.1:1;
```

и выполним функцию построения графика

```
plot(x,polyval(P,x),'r',x,exp(x),'g')
```

Некоторое отклонение заметно лишь в области значений $x$ близких к нулю. Построим погрешность интерполяции

```
plot(x,polyval(P,x)-exp(x))
```

Заметим, что погрешность не превышает $1.5 \cdot 10^{-3}$.

**Задание 3.1** Произведите интерполяцию на промежутке $-5 \leq x \leq 5$ по пяти, по 10 и по 15 точкам.

**Выводы:**

- при построении интерполяционного полинома 14 степени MATLAB выдал предупреждение о плохой обусловленности.

**Задание 3.2** Произведите интерполяцию функции $f(x) = \exp(x - x_0)$ на промежутке $x_0 - 1 \leq x \leq x_0 + 1$ по пяти точкам для $x_0 = 100$, 1000 и 10000.

**Выводы:**

- предупреждение о плохой обусловленности
- увеличение погрешности, с которой выполняются условия интерполяции, с ростом $x_0$
- увеличение погрешности интерполяции с ростом $x_0$
- при $x_0 = 10000$ погрешность превышает значение функции и носит случайный характер. Это связано с неудачным способом задания полинома его коэффициентами.

## 3.3 Интерполяция в форме Лагранжа

Решим задачу интерполяции, воспользовавшись записью интерполяционного полинома в форме Лагранжа. Напишем функцию `ILagrange`

```
function p = ILagrange(x,X,F)
N=length(X); p=zeros(size(x)); for k=1:N,
  Lagrange=ones(size(x)).*F(k);
  for m=1:N,
    if m ~= k
      Lagrange=Lagrange.*(x-X(m))./(X(k)-X(m));
    end
   end
p=p+Lagrange;
end
```

Решим задачу интерполяции функции $f(x) = \exp(x - x_0)$ на промежутке $x_0 - 1 \leq x \leq x_0 + 1$ по пяти узлам. Для этого сначала сформируем массивы `X` и `F`, содержащие данные интерполяции и вызовем функцию `ILagrange`

```
ILagrange(X,X,F)
```

для проверки выполнения условий интерполяции.

**Задание 3.3** Выполните задание 3.2 при помощи интерполяции по Лагранжу. Сравните с результатами, полученными ранее.

**Выводы:**

- Результат не зависит от интервала интерполяции

## 3.4 Интерполяция в форме Ньютона

Для построения интерполяционного полинома в форме Ньютона, необходимо вычислить разделенные разности. Поместим их в массив `f`

```
f=zeros(N,N); f(1,:)=F;
  for m=2:N,
```

```
   for k=1:N-m+1,
    f(m,k)=(f(m-1,k)-f(m-1,k+1))/(X(k)-X(k+m-1));
   end
  end
```

Теперь можно вычислить интерполяционный полином

```
p=f(1,1);
 for k=2:N,
  Newton=1;
  for m=1:k-1,
    Newton=Newton.*(x-X(m));
  end
  p=p+Newton.*f(k,1);
 end
```

**Задание 3.4** Напишите функцию `INewton` с теми-же аргументами, что и у `ILagrange`, которая вычисляет интерполяционный полином в форме Ньютона. Учтите, что для обеспечения возможности вычисления интерполяционного полинома в массиве точек, необходимо модифицировать приведенный выше фрагмент программы.

**Задание 3.5** Сравните результаты интерполяции $\exp(x)$ на интервале $-5 \leq x \leq 5$, проводя вычисления по Лагранжу и по Ньютону для различных значений $N$. Составьте таблицу максимальных значений погрешности от числа $N$ (в качестве максимальных значений погрешности можно взять длину вертикальной оси на графике погрешности).

**Выводы:**

- Таблица погрешностей

| степень | Ньютон | Лагранж | MATLAB |
|---|---|---|---|
| 4 | 14 | | |
| 5 | 6 | | |
| 6 | 2.5 | | |
| 8 | 0.3 | | |
| 10 | 0.025 | | |
| 13 | 4.0e-4 | | * 4.0e-4 |
| 15 | 1.8e-5 | | * 1.8e-5 |
| 20 | 4.0e-9 | | * 4.0e-9 |
| 22 | 1.0e-10 | 1.0e-10 | * 0.8e-11 |
| 25 | 8.0e-12 | 2.0e-10 | * 6.0e-11 |
| 30 | 4.0e-10 | 4.0e-9 | * 1.2e-9 |

Звездочкой помечены предупреждения о плохой обусловленности

- Погрешность сначала уменьшается (до $N \approx 22$), а затем увеличивается
- Форма Ньютона оказалась предпочтительней с точки зрения погрешности
- Наибольшие погрешности наблюдаются вблизи концов интервала интерполяции.

## 3.5 Неравномерная сетка

Для того чтобы погрешность интерполяции была более равномерна на всем промежутке, выберем неравномерное распределение узлов. Будем, например, выбирать узлы в точках

$$x_k = 5\sin\left(\pi\left(\frac{k-1}{N-1} - \frac{1}{2}\right)\right), \qquad k = 1, 2, \ldots, N \tag{3.1}$$

**Задание 3.6** Выполнить задание 3.4 при выборе узлов согласно (3.1).

**Выводы:**

- погрешности распределены по интервалу более равномерно

- максимальная погрешность ниже, чем в случае равноотстоящих узлов

| степень | равноотстоящие узлы | сетка (3.1) |
|---|---|---|
| 5 | 6.0 | 3.0 |
| 10 | 0.025 | 0.005 |
| 15 | 1.8e-5 | 8.0e-7 |
| 20 | 4.0e-9 | 2.5e-11 |
| 25 | 8.0e-12 | 1.5e-13 |
| 30 | 4.0e-10 | 1.5e-13 |

- при больших степенях встроенный алгоритм дает меньшую погрешность, что связано с применением вариационных методов.

**Задание 3.7** Выполнить интерполяцию функции $\sin(x)$, используя равномерную сетку, сетку (3.1) и сетку, связанную с корнями полиномов Чебышева $T_N(x)$. Корни — следующие

$$x_k = \cos\left(\frac{\pi}{2}\frac{2k-1}{N}\right), \qquad k = 1, 2, \ldots, N. \tag{3.2}$$

**Выводы:**

- Узлы Чебышева дают более высокую точность

| степень | равноотстоящие узлы | сетка (3.1) | Чебышев |
|---|---|---|---|
| 5 | 0.5 | 0.25 | 0.2 |
| 10 | 8.0e-3 | 1.5e-3 | 0.8e-3 |
| 15 | 2.5e-6 | 5.0e-8 | 5.0e-8 |
| 20 | 2.0e-9 | 1.5e-11 | 0.8e-11 |
| 25 | 5.0e-12 | 5.0e-13 | 4.0e-13 |
| 30 | 5.0e-12 | 8.0e-13 | 5.0e-13 |
| 35 | 1.5e-9 | 1.0e-12 | 0.8e-12 |

**Задание 3.8** Постройте полиномы Ньютона для сетки (3.2). Проверьте, что

$$|\mathcal{N}_k(x)| \leq 1, \qquad |x| \leq 1.$$

## 3.6 Двумерная табличная интерполяция

Двумерная интерполяция существенно сложнее, чем одномерная, рассмотренная выше, хотя смысл ее тот же - найти промежуточные точки некоторой зависимости z(x, y) вблизи расположенных в пространстве узловых точек. Для двумерной табличной интерполяции используется функция `interp2`:

- `ZI = interp2(X,Y,Z,XI,YI)` - возвращает матрицу ZI, содержащую значения функции в точках, заданных аргументами XI и YI, полученные путем интерполяции двумерной зависимости, заданной матрицами X, Y и Z. При этом X и Y должны быть монотонными и иметь тот же формат, как если бы они были получены с помощью функции `meshgrid` (строки матрицы X являются идентичными; то же можно сказать о столбцах массива Y). Матрицы X и Y определяют точки, в которых задано значение Z.
  Параметры XI и YI могут быть матрицами, в этом случае `interp2` возвращает значения Z, соответствующие точкам (XI(i,j),YI(i.j)). В качестве альтернативы можно передать в качестве параметров вектор-строку xi и вектор-столбец yi. В этом случае interp2 представляет эти векторы так, как если бы использовалась команда `meshgrid(xi,yi)`;

- `ZI = interp2(Z,X,Y)` - подразумевает, что X=1:n и Y=l:m, где [m.n]=size(Z);

- `ZI = interp2(Z,ntimes)` - осуществляет интерполяцию рекурсивным методом с числом шагов `ntimes`;

- `ZI = interp2(X,Y,Z,XI,YI,method)` - позволяет с помощью опции method задать метод интерполяции: 'nearest' - интерполяция по соседним точкам;

  1. 'linear' - линейная интерполяция;
  2. 'cubic' - кубическая интерполяция (полиномами Эрмита);
  3. 'spline' - интерполяция сплайнами.

Все методы интерполяции требуют, чтобы X и Y изменялись монотонно и имели такой же формат, как если бы они были получены с помощью функции

meshgrid. Когда X и Y - векторы равномерно распределенных точек, для более быстрой интерполяции лучше использовать методы 'linear', 'cubic', или 'nearest'.

Пример:

```
[X,Y]=meshgrid(-3:0.25:3);
Z=peaks(X/2,Y^2):
[Xl,Yl]=meshgrid(-3:0.1:3);
Zl=interp2(X,Y,Z,Xl,Yl):
mesh(X,Y,Z),hold on,mesh(Xl,Yl,Zl+15),hold off
```

# Работа № 4

## Интерполяционные сплайны

Само слово сплайн возникло для обозначения разметочной веревки, кривизна которой регулируется подвешенными грузиками. Использовалась для строительства рангоута деревянных кораблей и натягивалась главным корабельным мастером. Перерубившему эту веревку рабу отрубали голову и, поэтому, к сплайнам в те времена относились крайне почтительно.

Современное математическое определение: сплайны — это кусочно-полиномиальные функции. Сплайн-интерполяция используется для представления данных отрезками полиномов невысокой степени - чаще всего третьей. При этом кубическая интерполяция обеспечивает непрерывность первой и второй производных результата интерполяции в узловых точках. Из этого вытекают следующие свойства кубической сплайн-интерполяции:

- график кусочно-полиномиальной аппроксимирующей функции проходит точно через узловые точки;
- в узловых точках нет разрывов и резких перегибов функции;
- благодаря низкой степени полиномов погрешность между узловыми точками обычно достаточно мала;
- связь между числом узловых точек и степенью полинома отсутствует;
- поскольку используется множество полиномов, появляется возможность аппроксимации функций с множеством пиков и впадин.

Простейшая реализация сплайн-интерполяции это следующая функция:
`y1 = spline(x,y,x1)` - использует векторы $x$ и $y$, содержащие аргументы

функции и ее значения, и вектор $x1$, задающий новые точки; для нахождения элементов вектора $y1$ используется кубическая сплайн-интерполяция.

Пример:

```
x=0:10; y=3*cos(x);
x1=0:0.1:11;
y1=spline(x,y,x1);
plot(x,y,'o',x1,y1,'--')
```

Сплайн-интерполяция дает неплохие результаты для функций, не имеющих разрывов и резких перегибов. Особенно хорошие результаты получаются для монотонных функций.

Решение большинства задач интерполяции и аппроксимации функций и табличных данных обычно сопровождается их визуализацией. Она, как правило, заключается в построении узловых точек функции (или табличных данных) и в построении функции аппроксимации или интерполяции. Для простых видов аппроксимации, например полиномиальной, желательно нанесение на график формулы, полученной для аппроксимации.

## 4.1 PP-форма задания функций в MATLAB

В работе № 3 мы использовали встроенную функцию `polyfit`, которая строит полином заданной степени аппроксимирующий таблицу в смысле наименьшей суммы квадратов отклонений. Эта функция возвращает коэффициенты полинома. Задание полинома при помощи коэффициентов называется в MATLAB $p$-формой представления функции.

В MATLAB сплайны могут быть заданы в pp-форме. PP-форма объединяет список узлов (breaks) $x_1$, $x_2$, $\ldots x_n$ и матрицу коэффициентов полиномов (coefs) $a_{ij}$, $i = 1, \ldots n$, $j = 0, \ldots, N$ (здесь $N$ — степень полиномов) и задает

функцию

$$f(x) = \begin{cases} a_{10}(x-x_1)^N + a_{11}(x-x_1)^{N-1} + \dots \\ \qquad + a_{1\,N-1}(x-x_1) + a_{1N}, & x < x_2 \\ a_{20}(x-x_2)^N + a_{21}(x-x_2)^{N-1} + \dots \\ \qquad + a_{2\,N-1}(x-x_2) + a_{2N}, & x_2 < x < x_3 \\ \qquad \dots & \dots \\ a_{k0}(x-x_k)^N + a_{k1}(x-x_k)^{N-1} + \dots \\ \qquad + a_{k\,N-1}(x-x_k) + a_{kN}, & x_k < x < x_{k+1} \\ \qquad \dots & \dots \\ a_{n-1\,0}(x-x_{n-1})^N + a_{n-1\,1}(x-x_{n-1})^{N-1} + \dots \\ \qquad + a_{n-1\,N-1}(x-x_{n-1}) + a_{n-1\,N}, & x_{n-1} < x \end{cases}$$

Для работы с функциями, заданными в pp-форме имеется несколько функций:

`ppval(F,x)` вычисляет значение функции `F` от аргумента `x` (подобно `polyval` для полинома).

`fnder(F)` вычмсляет производную функции `F` (результат также задается в PP-форме).

`fnint(F)` вычисляет первообразную функции `F`, обращающуюся в нуль в узле $x_0$.

`fncmb` выполняет различные арифметические операции с функциями, например:

`fncmb(F, c)`, где `c` — скаляр, умножает функцию `F` на множитель `c`

**Внимание:** в некоторых версиях пакета эта функция работает неверно

`fncmb(F, G)` вычисляет сумму двух функций `F` и `G`.

## 4.2 Построение интерполяционного сплайна

Для построения интерполяционного сплайна $S_3^1$ необходимо задать интерполяционную таблицу $\{x_j, y_j\}$ и два краевых условия. В MATLAB существует несколько функций, которые строят сплайны. Рассмотрим лишь одну — `csape`.

Функция `csape` строит кубический интерполяционный сплайн $S_3^1$ по заданной интерполяционной таблице `X`= $\{x_j\}$, `Y`= $\{y_j\}$ и может быть вызвана лишь с этими двумя аргументами

```
S =csape(X,Y)
```

Возвращаемый функцией `csape` результат является сплайном $S_3^1$, интерполирующим таблицу $\{x_j, y_j\}$ и удовлетворяющим некоторым краевым условиям, которые приняты по умолчанию. Сплайн `S` задается в pp-форме.

Проинтерполируем функцию sin на промежутке $[0, \pi]$ по пяти равно отстоящим узлам.

Зададим вектор `X`, содержащий узлы

```
X = linspace(0,pi,5);
```

Вычислим значения функции sin в этих узлах и поместим результат в вектор `Y`

```
Y = sin(X);
```

Теперь построим сплайн

```
S = csape(X,Y)
```

Построим график сплайна `S` и функции sin на промежутке $[-\pi/2, 3\pi/2]$

```
x=[-pi/2:0.01:3*pi/2];
plot(x,ppval(S,x),x,sin(x),X,Y,'+')
```

Выводы:

- Значение сплайна в узлах совпадает со значениями функции.
- Отклонения сплайна от функции внутри промежутка интерполяции сравнительно невелики.
- Вне промежутка интерполяции сплайн сильно отклоняется от интерполируемой функции.

**Задание 4.1** Сравните погрешность интерполяции для построенного выше сплайна $S_3^1$ и для интерполяционного полинома, построенного по тем же узлам.

**Задание 4.2** Постройте графики всех производных сплайна.

Выводы:

- Производная 3 порядка представляет собой ступенчатую (кусочно-постоянную) функцию.
- Производные более высокого порядка равны нулю.
- Построенные выше графики показывают, что сплайн `S` не удовлетворяет ни одним из стандартных краевых условий.

В функции `csape` по умолчанию в качестве краевых условий задаются значения первой производной сплайна в точках $x_0$ и $x_N$ такими же как значения первой производной интерполяционных полиномов третьей степени, построенных по 4 крайним левым и крайним правым узлам.

**Задание 4.3** Постройте интерполяционный полином $P_3$ третьей степени по узлам $x_0$, $x_1$, $x_2$ и $x_3$ и проверьте выполнение краевого условия

$$\frac{dP_3(x_0)}{dx} = \frac{dS(x_0)}{dx}.$$

Если нужно построить сплайн, удовлетворяющий иным краевым условиям чем принятые в функции `csape` по умолчанию, функция `csape` вызывается с 3 или 4 аргументами. В качестве третьего аргумента можно указать `'not-a-knot'`, `'periodic'`, `'complete'`, `'second'`, `'variational'`.

**Задание 4.4** При помощи команды `help csape` выясните смысл перечисленных выше краевых условий и проверьте выполнение соответсвующих условий, для чего постройте графики производных сплайнов.

## 4.3 Использование фундаментальных сплайнов

При необходимости производить интерполяцию большого числа функций на одной и той-же сетке узлов может оказаться предпочтительным использовать фундаментальные сплайны $F_j$. Фундаментальные сплайны интерполируют таблицы $\delta_{ij}$, то есть, сплайн $F_0$ удовлетворяет условиям интерполяции

$$F_0(x_0) = 1, \quad F_0(x_1) = 0, \quad \ldots, \quad F_0(x_N) = 0.$$

Фундаментальный сплайн $F_k$ удовлетворяет условиям интерполяции

$$F_k(x_0) = 0, \quad \ldots, \quad F_k(x_{k-1}) = 0, \quad F_k(x_k) = 1, \quad F_k(x_{k+1}) = 0,$$

$$\ldots, \quad F_0(x_N) = 0.$$

Если фундаментальные сплайны построены, задача интерполяции произвольной функции $f(x)$ сводится к вычислению суммы

$$S(x) = \sum_{j=0}^{N} f(x_j) F_j(x). \tag{4.1}$$

О выполнении арифметических операций с функциями, заданными в pp-форме см. параграф 1.

**Задание 4.5** Напишите функцию MATLAB, реализующую построение системы фундаментальных сплайнов на заданной сетке. Для сетки из задания 4.1 постройте графики всех фундаментальных сплайнов.

**Задание 4.6** Напишите функцию MATLAB, реализующую вычисление интерполяционного сплайна по формуле (4.1). Обратите внимание, что `fncmb` неправильно работает при умножении функции в pp-форме на скаляр. Проверьте совпадение получаемого по формуле (4.1) интерполяционного сплайна со сплайном из задания 4.1.

**Задание 4.7** Исследуйте сплайн-интерполяцию функций

$$f(x) = \cos(5x)\exp(-0.3x^2) \qquad \text{и} \qquad f(x) = \sin(5x)\exp(-0.3x^2)$$

на отрезке [0,5] используя сплацны с периодическими граничными условиями.

Данные функции описывают затухающие колебания маятника с нулевой и ненулевой начальной скоростью. Для какой из этих функций более подходят сплайны с периодическими граничными условиями?

## 4.4 Обработка данных

В MATLAB совмещение функций аппроксимации с графической визуализацией доведено до логического конца - предусмотрена аппроксимация рядом

методов точек функции, график которой построен. И все это выполняется прямо в окне редактора графики Property Editor. Для этого в позиции Tools графического окна имеются две команды:

Basic Fitting - основные виды аппроксимации (регрессии);

Data Statistics - статистические параметры данных.

Команда Basic Fitting открывает окно, дающее доступ к ряду видов аппроксимации и регрессии: сплайновой, эрмитовой и полиномиальной со степенями от 1 (линейная аппроксимация) до 10. В том числе со степенью 2 (квадратичная аппроксимация) и 3 (кубическая аппроксимация).

Команда Data Statistics открывает окно с результатами простейшей статистической обработки данных.

**Пример:** При калибровке термопары получены следующие значения:

| $t$, $^0$C | 0 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| $U$, мВ | -0.67 | -0.254 | 0.171 | 0.609 | 1.057 | 1.517 |

Исследуем полученные данные:

```
T=[0,20,40,60,80,100];
U=[-0.67,   -0.254,   0.171,  0.609,   1.057,  1.517];
plot(T,U.'o')
```

Напомним, что последняя команда строит график узловых точек кружками (без соединения их отрезками прямых).

Исполнив команду Tools > Basic Fitting, можно получить окно интерполяции. В этом окне надо отметить одну или несколько желаемых интерполяций. Стоит отметить какой-либо вид интерполяции, как соответствующая кривая интерполирующей функции или аппроксимации появится в графическом окне.

Установив птичку у параметра Show equations (Показать уравнения), можно получить в графическом окне запись уравнений интерполяции или аппроксимации.

Наконец, исполнив команду Tools > Data Statistics, можно получить окно с рядом статистических параметров данных, представленных векторами X и Y. Отметив птичкой тот или иной параметр в этом окне, можно наблюдать соответствующие построения на графике, например вертикалей с минимальным, средним и максимальных значением у и горизонталей с минимальным,

средним и максимальным значением x.

**Задание 4.8** Дисперсия скорости звука в сероуглероде при $25\ ^0$C задается таблицей:

| $f$, МГц | 10 | 30 | 50 | 70 | 150 | 250 |
|---|---|---|---|---|---|---|
| $v$, м/с | 1142 | 1153 | 1171 | 1187 | 1219 | 1228 |

Исследуйте различные возможности аппроксимации и интерполяции этих данных.

**Задание 4.9** Давление насыщенных паров воды от температуры задается так:

| $t$, $^0$C | 0 | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|---|
| $p$, Па | 6.108(2) | 2.337(3) | 7.374(3) | 1.991(4) | 4.735(4) | 1.013(5) |

Исследуйте различные возможности аппроксимации и интерполяции этих данных.

# Работа № 5

## Решение одномерных нелинейных уравнений

Задача нахождения решений уравнений может быть сформулирована различными способами. Две наиболее распространенные формулировки – это нахождение корней $f(x) = 0$ и нахождение неподвижной точки $x = F(x)$. В данной работе мы рассмотрим основные методы решения таких задач в одномерном случае. Необходимо отметить, что для применения того или иного метода может потребоваться преобразование исходного уравнения к другой форме, эквивалентной исходной.

## 5.1 Метод деления отрезка

Начнем с вычисления $\sqrt{2}$. Нам известно, что искомое число лежит между 1 и 2. Попытаемся использовать среднее значение $x = 1\frac{1}{2}$ в качестве первого приближения. Так как $x^2$ больше 2, то это значение $x$ слишком велико. В качестве второго приближения возьмем меньшее число $x = 1\frac{1}{4}$, опять поделив отрезок пополам. Так как $x^2$ меньше 2, это значение $x$ слишком мало и его надо увеличить. Продолжая деление пополам мы получим последовательность

$$1\frac{1}{2}; \qquad 1\frac{1}{4}; \qquad 1\frac{3}{8}; \qquad 1\frac{5}{16}; \qquad 1\frac{13}{32}; \qquad 1\frac{27}{64}; \qquad \dots$$

Данный алгоритм вычислений можно реализовать с помощью следующей программы в среде MATLAB

```
a = 1 b = 2 k = 0;
while (b-a > eps) & (k<100)
  x = (a + b)/2;
```

```
  if x^2 > 2
      b=x
    else
      a=x
  end
k = k + 1;
end
```

Метод деления отрезка пополам всегда сходиться при разумных значениях погрешности $\epsilon$. Тем не менее в программы подобного сорта всегда принято вставлять *счетчик* и задавать его граничное значение в качестве защиты от не профессионального использования программ.

Наличие погрешностей округления приводит к тому, что для каждого корня уравнения $f(x) = 0$ будет существовать область неопределенности $[a, b]$, внутри которой лежит вычисляемый корень.

**Задание 5.1** Используя приведенную выше программу проверьте, что после 52 итераций мы получим следующие выражение для области неопределенности корня в двоичном формате (для вывода в двоичном формате используйте команду `format hex`)

```
a = 3ff6a09e667f3bcc
b = 3ff6a09e667f3bcd
```

В этом случае область неопределенности равна одному биту.

Для нахождения корней уравнения $f(x) = 0$ MATLAB программу необходимо немного изменить

```
function bisect(f,a0,b0,eps,max)
% Входные параметры:
% - 'fcn' - встроенная функция или функция из файла 'fcn.m'
% - a0,b0 - отрезок внутри которого находится искомый корень
% - eps - задаваемая точность
% - max - максимально допустимое число итераций
% Выходные значения:
% - k - число выполненных итераций
```

```
% - root - полученное приближенное решение
% - Error_bound - ошибка полученного решения
% Пример использования
%    root = bisect('sin',3,4,1.0E-12,100)
k=0; if a0>b0
  'a0 < b0 is not true.  Stop!'
  return
end
&
format short e a=a0; b=b0; fa=feval(f,a); fb=feval(f,b);
&
if sign(fa)*sign(fb)> 0
  'f(a0) and f(b0) of same sign.  Stop!'
  return
end
%
x=(a+b)/2;
while (b-x > eps) & (k<max)
  fc=feval(f,x);
  if sign(fb)*sign(fc) <= 0
    a=x;
  else
    b=x;
  end
  x=(a+b)/2;
  k=k+1;
 end
%
format long
root=x
format short e
Error_bound=b-x
Iterations=k
format short
```

Метод деления отрезка является достаточно медленным методом, основным достоинством которого является его безусловная сходимость.

**Задание 5.2** Найдите корень уравнения $f(x) = 0$ на отрезке $[a, b]$ для следующих функций

1. $f = x^3 - 2x - 5$, $[0, 3]$,
2. $f = x^3 - 0.001$, $[-1, 1]$,
3. $f = \ln(x + 2/3)$, $[0, 1]$,
4. $f = \mathsf{sign}(x-2)\sqrt{|x-2|}$, $[1, 4]$,
5. $f = \arctan(x) - \frac{\pi}{3}$, $[0, 5]$,
6. $f = 1/(x - \pi)$, $[0, 5]$.

## 5.2 Метод итераций

Метод итераций применяется для решения уравнений вида

$$x = F(x), \tag{5.1}$$

для функции $F : [a, b] \to [a, b]$. Если функция $F(x)$ дифференцируема и

$$q = \sup_{x \in [a,b]} |F'(x)| < 1, \tag{5.2}$$

то решение $x^*$ уравнения (1) может быть найдено как предел простой итерационной процедуры:

$$x^* = \lim_{n \to \infty} x_n, \quad x_{n+1} = F(x_n), \tag{5.3}$$

где начальное приближение $x_0$ – произвольная точка интервала $[a, b]$. Метод итераций обладает линейной сходимостью, и может быть легко обобщен на многомерные линейные и нелинейные системы уравнений.

Рассмотрим функцию в среде MATLAB, осуществляющую решение уравнения (5.1) методом простых итераций:

```
 function [k,p, err,P] =fixpt (F,p0,tol,max1)
% Входные параметры:
% - F - функция (1), заданная в виде строковом виде
% - p0 - начальная точка для итераций
% - tol - задаваемая точность
% - max1 - максимально допустимое число итераций
% Выходные значения:
```

```
% - k - число выполненных итераций
% - p - полученное приближенное решение
% - err - ошибка полученного решения
% - P - массив значений итераций
 P(1)=p0;
 for k=2:max1
   P(k)=feval(F,P(k-1));
   err=abs(P(k)-P(k-1));
   relerr=err/(abs(P(k))+eps);
   p=P(k);
   if(err<tol) | (relerr<tol),break; end
 end
 if k == max1
   disp( 'maximum number of iterations exceeded')
 end
 P=P';
```

Команда

```
[k,p, err,P]=fixpt('cos',0.5,1e-5,40)
```

иллюстрирует использование функции `fixpt` для уравнения $x = \cos(x)$.

**Задание 5.3** Проверьте, что решение $x^* = 0.73908$ с точностью $10^{-5}$ получено за $k = 29$ итераций.

Рассмотрим задачу об отыскании корней функции $f(x) = 2x - 3 - \ln(x)$, т.е. задачу об отыскании корней уравнения $f(x) = 0$. Если построить график этой функции, то на графике наблюдаются два корня уравнения $f(x) = 0$, приближенно в точках x $= 0.05$ и x $= 1.8$. (Постройте график самостоятельно!)

Чтобы найти эти корни с большей точностью мы применим простой итерационный метод, который состоит из следующих шагов:

- перепишем уравнение $f(x) = 0$ в виде $x = F(x)$ (5.1), где в нашем случае $F(x) = (\ln(x) + 3)/2$.

- убедимся, что условие $|F'(x)| < 1$ (5.2), достаточное для сходимости метода итераций, выполнено в окрестности корня $x = 1.8$. Для этого постройте график функции $|F'(x)|$.

- найдем решение уравнений $f(x) = 0$ и $x = F(x)$ с заданной точностью $10^{-8}$, используя функцию `fixpt`.

Легко убедиться, что условие (5.2) не выполняется в окрестности второго корня $x = 0.05$. Поэтому, для того, чтобы найти второй корень, необходимо использовать другую функцию $\tilde{F}(x)$ в уравнении (5.1) для того же самого исходного уравнения $f(x) = 0$. В нашем случае это $\tilde{F}(x) = \exp(2x - 3)$. Проверьте условие сходимости для этой функции и найдите корень с заданной точностью $10^{-8}$.

**Задание 5.4** Определите с точностью $10^{-8}$ все решения уравнения (5.1) для следующих функций:

1. $F(x) = \cos(\sin(x))$;
2. $F(x) = x^2 - \sin(x + 0.15)$;
3. $F(x) = x^5 - 3x^3 - 2x^2 + 2$;
4. $F(x) = \tan(x)$.

В последнем случае найдите три наименьших положительных корня.

**Задание 5.5** Если условие (5.2) нарушается, то последовательность $x_n$, как правило, не сходится. В случае $q = 1$, однако, сходимость может присутствовать. Исследуйте уравнение $x = F(x) = 2\sqrt{x - 1}$ в окрестности точки $x = 2$, в том числе скорость сходимости итераций к решению. Можно ли использовать разность $|x_n - x_{n+1}|$ для оценки близости $x_n$ к решению $x^*$?

## 5.3 Метод Ньютона

Для решения уравнения $f(x) = 0$, $f \in C^1$, можно применить метод простых итераций (1) для функции $F(x) = x - f(x)/f'(x)$. Последовательные итерации записываются в виде:

$$x_{j+1} = x_j - \frac{f(x_j)}{f'(x_j)}. \tag{5.4}$$

Можно убедиться, что уравнение (4) определяет корень касательной к функции $f(x)$, проведенной из точки $x_j$. В соответствии с этой интерпретацией,

метод (4) называют еще методом касательных. Если функция $f \in C^2$, то существует окрестность корня, в которой $|F'(x)| < 1$, и метод Ньютона гарантированно сходится в этой окрестности.

Наличие погрешностей округления приводит к тому, что для каждого корня уравнения $f(x) = 0$ будет существовать область неопределенности, внутри которой лежит вычисляемый корень. Если $x^*$ точное значение корня, то $\tau_k = |x_{k+1} - x^*|$ определяется величиной

$$\tau_k \approx \frac{\delta f(x_k)}{f'(x^*)},$$

где $\delta f(x_k)$ - погрешность сделанная при вычислении функции $f$ в точке $x_k$. Таким образом, область неопределенности корня в основном определяется погрешностью вычислений и числом обусловленности корня. Чем больше кратность корня, тем обширнее область неопределенности.

Когда $x_k$ попадает в область неопределенности корня, то при дальнейших итерациях не будет происходить уточнения приближенного значения, а будут происходить изменения согласно капризам погрешностей округления.

Для простых корней сходимость метода Ньютона будет квадратичной, а для кратных корней – линейной, т.е.для кратных корней теряется свойство сверхсходимости. Это свойство можно сохранить, если нам априори известно что корень $x = x^*$ имеет кратность $s$. В этом случае следует итерации вести по формуле

$$x_{k+1} = x_k - s\frac{f(x_k)}{f'(x_k)}.$$

Программа, реализующая метод Ньютона, может быть написана аналогично программе `fixpt`. Отличия заключаются в необходимости использовать производную функции `DF` в заголовке

```
function [k,p, err,P] =fixpt (F,DF,p0,tol,max1)
```

и изменении расчетной формулы на

```
P(k)=P(k-1)-feval(F,P(k-1))/feval(DF,P(k-1));
```

**Задание 5.6** Напишите программу для вычисления корней методом Ньютона. Вычислите решения для уравнения 1 из Задания 3.1. Сравните количество

итераций, необходимых для достижения одинаковой точности в методе деления отрезка пополам и методе Ньютона. Заметим, что именно этот полином использовался в качестве примера в докладе Уоллеса (Wallis) во Французской Академии наук, посвященном открытию метода Ньютона.

**Задание 5.7** Напишите программу для вычисления корней методом Ньютона. Вычислите решения для уравнений (3-4) из Задания 3.4. Сравните количество итераций, необходимых для достижения одинаковой точности в методе простых итераций и методе Ньютона.

**Задание 5.8** Обычно метод Ньютона либо сходится достаточно быстро, либо не сходится вообще. В качестве примера изучите поведение метода Ньютона для уравнения 4 из Задания 3.1. Корень этого уравнения $x^* = 2$, однако метод Ньютона для этого уравнения не сходится при любом начальном приближении. В силу этого в программе необходимо установить счетчик и максимальное количество итераций, например 100.

## 5.4 Метод секущих

Метод Ньютона сходится достаточно быстро, однако он требует вычисления производной функции, что зачастую представляет дополнительную проблему. Чтобы избежать вычисления производной, можно заменить ее на разностную, вычисленную по двум предыдущим итерациям. При этом итерационный процесс приобретает вид

$$x_{j+1} = x_j - f_j \frac{x_j - x_{j-1}}{f_j - f_{j-1}}, \tag{5.5}$$

где $f_j = f(x_j)$. Порядок сходимости метода секущих ниже, чем у метода Ньютона, и равен в случае однократного корня $(\sqrt{5}+1)/2 \approx 1.62$.

С точки зрения вычислительной сложности, однако, метод секущих оказывается эффективнее метода Ньютона. За единицу сложности алгоритма примем сложность вычисления значения функции или ее производной, так как вычисление производной обычно имеет сложность не меньшую, чем вычисление функции. Ясно, что при таком условии один шаг метода Ньютона надо сравнивать с двумя шагами метода секущих, так как на каждом шаге

метода Ньютона происходит 2 вычисления функции, а в методе секущих – только одно. Итак, при одном шаге в методе Ньютона имеем квадратичную сходимость

$$|x_{k+1} - x^*| \approx N(x_k - x^*)^2,$$

а для двух шагом метода секущих имеем

$$|x_{k+2} - x^*| \approx N^{1.62}(x_k - x^*)^{2.618}.$$

Тем самым метод секущих дает асимптотически более высокую скорость сходимости. Если вычисление производной - более сложная процедура, чем вычисление функций, то контраст между этими методами будет еще более резким.

**Задание 5.9** Напишите программу для вычисления корней методом секущих. Вычислите решения для нескольких уравнений из Задания 3.1 и Заданий 3.3 (по выбору преподавателя).

**Задание 5.10** Постройте в полулогарифмическом масштабе на одном графике зависимости текущей погрешности от номера итерации для методов простой итерации, Ньютона и секущих, т.е. зависимости $\log(|x_j - x^*|)$ от $j$ для одного уравнения из Задания 3.1 или Заданий 3.3 (по выбору преподавателя) Объясните полученные результаты.

**Задание 5.11** Изучите поведение метода секущих на примере уравнения 4 из Задания 3.1, для которого метод Ньютона не работает.

## 5.5 Метод парабол

Метод парабол является трехшаговым итерационным методом, в котором значение нового приближения $x_{j+1}$ определяется по трем предыдущим точкам $x_j$, $x_{j-1}$ и $x_{j-2}$. Функция $f(x)$ заменяется параболой, проходящей через точки $(x_j, f_j)$, $(x_{j-1}, f_{j-1})$ и $(x_{j-1}, f_{j-1})$. Такая парабола может быть записана в форме Ньютона:

$$p_2(x) = f_j + f_{j-1,j}(x - x_j) + f_{j-2,j-1,j}(x - x_j)(x - x_{j-1}). \qquad (5.6)$$

Точка $x_{j+1}$ определяется как тот из корней этого полинома, который ближе по модулю к точке $x_j$. Порядок сходимости метода выше, чем у метода секущих, но ниже, чем у метода Ньютона.

Следующий кусок MATLAB кода иллюстрирует данный метод

```
k =0;
while (abs(c-b)> eps*abs(c)) & (k<100)
  x =polyinterp([f(a),f(b),f(c)],[a,b,c],0)
  a = b;
  b = c;
  c = x;
  k = k +1;
end
```

Однако данный метод работает только если $f(a)$, $f(b)$ и $f(c)$ различны, что не всегда верно. Например, вычисляя по методу парабол $\sqrt{2}$, когда $f(x) = x^2$, при начальных значениях $a = -2$; $b = 0$; $c = 2$ мы получим $f(a) = f(c)$ и метод парабол не будет работать. Более того, если $a = -2.001$; $b = 0$; $c = 1.999$ то по методу парабол на втором шаге мы получим $x_2 = 500$. В общем можно сказать, что чем быстрее сходимость метода, тем аккуратнее надо подходить к выбору начального приближения.

**Задание 5.12** Напишите программу для вычисления корней методом парабол. Вычислите решения для нескольких уравнений из Задания 3.2 (по выбору преподавателя). Повторите Задание 3.9, включив результаты для метода парабол.

## 5.6 Вычисление комплексных корней

Среда MATLAB имеет встроенные функции для поиска корней. Функция `FZERO(FUN,X0)` ищет корень вещественной функции `FUN` в окрестности точки `X0`. В данной встроенной программе используется комбинация метода секущих, метода параболической интерполяции и метода деления отрезка пополам. Это позволяет сделать данную процедуру и быстрой, и безошибочной.

**Задание 5.13** Проверьте, что следующий пример использования этой функции:

```
fzero(inline('x-cos(x)'),1)
```

дает значение $0.7391$.

**Задание 5.14** Найти все вещественные корни полинома

$$f(x) = 816x^3 - 3835x^2 + 6000x - 3125\,.$$

с помощью встроенной программы и одного из рассмотренных выше методов. Сравните результаты. Корни расположены очень близко друг к другу, учтите это при выборе оптимального метода вычислений.

**Задание 5.15** Уравнение Кеплера, которое описывает орбиты планет, имеет вид

$$E - e\sin E = M\,.$$

Здесь $e$ - эксцентриситет орбиты (эллипса), $E$ - эксцентрическая аномалия и $M$ - средняя аномалия. Решение этого уравнение относительно $E$ известно

$$E = M + 2\sum_{m=1}^{\infty}\frac{1}{m}J_m(me)\sin(mM)$$

где $J_m(x)$ - функции Бесселя первого рода порядка $m$, которые в среде MATLAB обозначаются `besselj(m,x)`.

Найдите значение $E$ для $M = 24.851090$ и $e = 0.1$ с помощью процедуры `fzero` и с помощью "точной" формулы. Сравните полученные результаты. Сколько слагаемых ряда пришлось использовать в "точной" формуле?

Встроенная функция `ROOTS(C)` вычисляет вещественные и комплексные корни полинома

$$C(1) * X^N + ... + C(N) * X + C(N+1),$$

если вектор $C$ имеет $N+1$ компоненту. Пример использования функции `roots([1,0,1])` дает два значения $0 + 1.0000i$ и $0 - 1.0000i$.

Важным достоинством итерационных методов является возможность вычисления комплексных корней функций, вещественных на вещественной оси. В качестве примера рассмотрим возможность использования метода Ньютона для нахождения корней уравнения $z^3 - 1 = 0$.

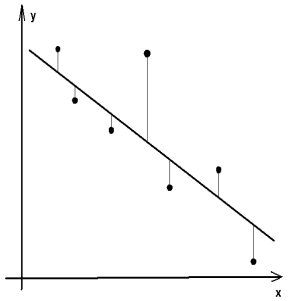
```
function newton3(x0,y0,n)
%   Метод Ньютона для нахождения корней уравнения z^3=1
%   x0,y0 --  начальное приближение z0=x0+iy0
%   n     --  # число итераций;
%   если   n = 0 то происходит 10 итераций итерактивно
%
clf;
colors = 'kbgrcmy';
ncolors = size(colors,2);
%
axis([-1.5 1.5 -1.5 1.5]);axis manual; hold on;
solx =[1-.5-.5];soly = [0 -.8666 .8666];
plot(solx,soly,'rx');
%
nn = n; if n <= 0 nn = 10; end
%
v = [x0;y0];
    fprintf('\n  n          point\n\n')
%
for i=0:nn
   plot(v(1),v(2),[colors(mod(i,ncolors)+1),'o'])
    fprintf(1,'%3d     ',i)
    fprintf(1,'%5.4f     ',v')
    fprintf(1,'\n')
    if n <= 0 disp('Press any key to continue. . .'), pause;end
   x = v(1); y = v(2);
   w = [x^3 - 3*x*y^2 - 1; 3*x^2*y - y^3];
   A = [3*x^2 - 3*y^2, -6*x*y;6*x*y, 3*x^2 - 3*y^2];
   v = v - A\w;
end
```

**Задание 5.16** Модифицируйте программу так, чтобы исследовать зависимость скорости сходимости метода Ньютона от выбора начального приближения.

# Работа № 6

## Метод наименьших квадратов

Рассмотрим следующий эксперимент: тележка движется равноускоренно, через каждые несколько секунд измеряют скорость тележки, ожидается, что скорость тележки будет линейно уменьшаться. Результаты измерений показаны на рис. 1.




Заметим, что точки не лежат на одной линии. Это не так уж неожиданно. Измерительные приборы могут быть не совсем точными, может оказаться невозможным точно интерпретировать измерения, а скорость может изменяться не совсем так, как предсказано. Чтобы определить скорость с которой движется тележка, экспериментатору следовало бы аппроксимировать данные прямой линией, которая в некотором смысле "наилучшим образом" аппроксимирует данные. На рис. 1. изображена одна из таких линий.

Ситуация такого типа встречается весьма часто. Когда данные зашумлены, т.е. полны случайных ошибок, тогда аппроксимация позволяет нам изучать тренды (тенденции изменения) в данных; это называется *сглаживанием.*

Теперь попробуем сформулировать задачу приближения данных методом наименьших квадратов более четко.

Пусть нам задан набор точек

$$(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$$

($x_i \neq x_j$ при $i \neq j$), причем значения $y_i$ считаются содержащими ошибки. Требуется найти функцию $F(x)$, для которой

$$y_i = F(x_i) - \eta_i \tag{6.1}$$

при $i = 1, \ldots, m$, причем погрешности $\eta_i$ малы. Такая функция ищется среди функций определенного класса, а именно, среди линейных комбинаций заранее выбранных *базисных функций* $f_j$

$$F(x) = \sum_{j=1}^{m} c_j f_j(x) \tag{6.2}$$

где $c_j$ - искомые коэффициенты. Уравнения (6.2) можно выразить в матричной форме. Пусть $A$ - $m \times n$-матрица значений базисных функций в заданных точках:

$$A = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \ldots & f_n(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ f_1(x_m) & f_2(x_m) & \ldots & f_n(x_m) \end{pmatrix}$$

а $c$ - вектор из $n$ искомых коэффициентов. Тогда

$$Ac = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \ldots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \ldots & f_n(x_2) \\ \vdots & \vdots & \ldots & \vdots \\ f_1(x_m) & f_2(x_m) & \ldots & f_n(x_m) \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_n \end{pmatrix} = \begin{pmatrix} F(x_1) \\ F(x_2) \\ \ldots \\ F(x_n) \end{pmatrix} \tag{6.3}$$

будет вектор из $m$ значений, через которые проходит искомая кривая. Подставив (6.1) в (6.3) получим

$$\eta = Ac - y \tag{6.4}$$

где $\eta$ - вектор *невязки*. Необходимо решить уравнение (6.4) таким образом, чтобы вектор невязки был как можно меньше (короче). В классическом

варианте решается следующая задача:

$$\min ||\eta||_2^2 = ||Ac - y||_2^2 = \sum_{i=1}^{n} (\sum_{j=1}^{n} a_{ij}c - y_i)^2 \tag{6.5}$$

т.е. минимизируется сумма квадратов отклонений искомой прямой от заданных точек, этот способ называется методом наименьших квадратов. Продифференцируем (6.5) по всем переменным $c_k$ и приравняем производные нулю:

$$\frac{\delta||\eta||_2^2}{\delta c_k} = \sum_{i=1}^{n} 2(\sum_{j=1}^{n} a_{ij}c - y_i)a_{ik}$$

Эту систему из $n$ уравнений $(k = 1, 2, \ldots, n)$ можно записать как матричное уравнение

$$(Ac - y)^T A = 0,$$

которое эквивалентно уравнению

$$A^T(Ac - y) = 0.$$

Раскрыв скобки, приходим к уравнению

$$A^T Ac = A^T y, \tag{6.6}$$

Называемому в математической статистике *нормальным уравнением.* Матрица $A^T A$ будем симметрической и, если $A$ имеет полных столбцовый ранг, положительно определенной. В этом случае существует обратная матрица $(A^T A)^{-1}$, и решение системы (6.6) единственно:

$$c = (A^T A)^{-1} A^T y = A^+ y.$$

Здесь $A^+$ - псевдообратная матрица к матрице $A$, которая обладает следующими свойствами:

$$\begin{aligned} &AA^+A = A, \\ &A^+AA^+ = A^+, \\ &(AA^+)^T = AA^+. \end{aligned} \tag{6.7}$$

В MATLAB псевдообратную матрицу можно найти с помощью команды `pinv(A)` (в случае, если $A$ квадратная матрица полного ранга, $A^+ = A^{-1}$).

**Задание 6.1** Проверить условие $A^+ = A^{-1}$, для этого выполнить последовательно следующие команды:

```
C = fix(10*rand(3,3))
X =pinv(C)
Q = X*C
P = C*X
```

**Задание 6.2** Проверить условие (6.7) для прямоугольной матрицы $m \times n$. Матрицу $A$ можно задать командой:

```
A = fix(10*rand(n,m))
```

## 6.1 Пример полиномиальной аппроксимации

Рассмотрим в качестве примера решение следующей задачи. Даны 5 точек $(-1, 2)$, $(1, 1)$, $(2, 1)$, $(3, 0)$, $(5, 3)$ проведем рядом с ними график функции $F(x) = c_1 + c_2 x + c_3 x^2$.
Зададим векторы $x$, $y$ и матрицу $A$

```
x = [-1,1,2,3,5]
y =[2,1,1,0,3]
A = zeros(5,3
```

Заполним матрицу $A$ значениями базисных функций (из условия задачи $f_1 = 1$, $f_2 = x$, $f_3 = x_2$).

```
A(:,1)=1 A(:,2)=x A(:,3)=x.^2
```

Найдем решение

```
c=pinv(A)*y
```

Построим график найденной функции

```
x2=-2:0.1:6
y2=1.2-0.7571*x+0.2143*x.^2
plot(x,y,'p',x,y2,'pr',x2,1.2-0.7571*x2+0.2143*x2.^2)
```

Нарисуем на графике линии, показывающие отклонения от функции

```
 for i=1:5
   t=[x(i),x(i)]
   t1=[y(i),y2(i)]
   line(t,t1)
end
```

**Задание 6.3** Даны точки (1,1), (2,1), (3,3), (4,8). Методом наименьших квадратов построить приближение вида

$$f(x) = c_1 + c_2 x \log x + c_3 e^x$$

## 6.2 Встроенные функции

Встроенная MATLAB функции `polyfit` и `polyval` предназначены для полиномиальной аппроксимации по методу наименьших квадратов. Эти же функции встроены в графический интерфейс. Для примера нарисуем произвольный набор данных с помощью команды `plot`

```
x = [-1,1,2,3,5,7,6,10]
y =[2,1,1,0,3,-1,-3,-8]
plot(x,y,'ro')
```

Если выберем следующий пункт в меню `Tools>Basic Fitting`, то появиться выпадающее меню, которое позволит выбрать семейство кривых, которые будут использованы для аппроксимации по методу наименьших квадратов.

**Задание 6.4** Какой тип кривых из встроенных наилучшим образом подходит к введенным вами данным?

В данном графическом интерфейсе реализованы возможности встроенных функций `fminsearch`, `fmincon`, `fminunc`, `lsqnonlin`, `lsqcurvefit` и некоторых других из пакета `Optimization Toolbox`, который должен быть установлен.

Для того, чтобы показать, как использовать подобные подпрограммы, мы рассмотрим пример аппроксимации в классе кривых следующего вида

$$y = \beta_1 \exp \lambda_1 t + \beta_2 \exp(\lambda_2 t).$$

Чтобы найти искомую аппроксимацию для данных нам данных, нам необходимо найти по методу наименьших квадратов два линейных параметра $\beta_{1,2}$ и два нелинейных параметра $\lambda_{1,2}$.

Итак, наберите следующую программу, которая иллюстрирует возможности встроенной подпрограммы `fminsearch`

```
function expfitdemo
t = (0:.1:2)';
y = [5.8955 3.5639 2.5173 1.9790 1.8990 1.3938 1.1359 ...
1.0096 1.0343 0.8435 0.6856 0.6100 0.5392 0.3946 ...
0.3903 0.5474 0.3459 0.1370 0.2211 0.1704 0.2636]';
clf
shg
set(gcf,'doublebuffer','on')
h = plot(t,y,'o',t,0*t,'-');
h(3) = title('');
axis([0 2 0 6.5])
lambda0 = [3 6]';
lambda = fminsearch(@expfitfun,lambda0,[],t,y,h)
set(h(2),'color','black')
end
%----------------------
function res = expfitfun(lambda,t,y,h)
m = length(t);
n = length(lambda);
X = zeros(m,n);
 for j = 1:n
   X(:,j) = exp(-lambda(j)*t);
 end
beta = X\y; z = X*beta; res = norm(z-y);
set(h(2),'ydata',z);
set(h(3),'string',sprintf('%8.4f %8.4f',lambda))
pause(.1)
end
```

Сохраните эту программу в m-файл и запустите его командой `expfitdemo`.

Программа будет выполняться до тех пор пока вы не нажмете любую из клавиш для остановки.

**Задание 6.5** Постройте аппроксимацию с помощью "expfitdemo" для данных

```
t = 1:25
 y = [ 5.0291 6.5099 5.3666 4.1272 4.2948 ...
 6.1261 12.5140 10.0502 9.1614 7.5677 ...
 7.2920 10.0357 11.0708 13.4045 12.8415 ...
 11.9666 11.0765 11.7774 14.5701 17.0440 ...
 17.0398 15.9069 15.4850 15.5112 17.6572]
```

Сравните с полиномиальной аппроксимацией, полученной с помощью графического интерфейса.

**Задание 6.6** Выражение $z = ax^2 + bxy + cy^2 + dx + ey + f$ определяет квадратичную форму двух переменных $x, y$. Множество точек $(x, y)$ таких, что $z = 0$ называется коническим сечением, которое может быть эллипсом, параболой или гиперболой в зависимости от значения дискриминанта $b^2 - 4ac$.

Постройте график произвольного конического сечения с помощью команд

```
[X,Y] = meshgrid(xmin:deltax:xmax,ymin:deltay:ymax);
Z = a*X.^2 + b*X.*Y + c*Y.^2 + d*X + e*Y + f;
contour(X,Y,Z,[0 0])
```

Не забудьте вместо букв подставить понравившиеся вам цифры! Изменяя коэффициенты квадратичной формы постройте эллипс, параболу и гиперболу.

Известно, что согласно законам Кеплера планетарные орбиты являются коническими сечениями. Предположим, что астрономы наблюдая за движением небесного тела получили следующий набор координат в $(x, y)$ плоскости:

```
x = [1.02 .95 .87 .77 .67 .56 .44 .30. 16 .01]';
y = [0.39 .32 .27 .22 .18 .15 .13 .12 .13 .15]';
```

Определите по методу наименьших квадратов коэффициенты квадратичной формы и постройте график орбиты.

Используя генератор случайных чисел `randn` добавьте к каждому измерению шум в интервале [-.005,.005]. Вычислите новые коэффициенты квадратичной формы и постройте на одном графике старую орбиту и возмущенную новую орбиту. Объясните полученный результат.

# Работа № 7

## Численное дифференцирование

### Как работать с программой

Программа для проведения лабораторного исследования работает в среде BARSIC, документацию по которой можно найти в
*http://www.niif.spbu.ru/ monakhov/index-r.html*
и обладает следующими возможностями:

- Участок любого графика можно растянуть на весь экран, выделив соответствующую область с помощью "мыши". Вернуть предыдущие (или начальные) границы просмотра можно выбором пункта "назад" (или "восстановить") во всплывающем меню, появляющемся по щелчку правой кнопки "мыши" в области графика.

- Числовые значения границ и другие параметры просмотра можно задать в пункте "Параметры" этого меню.

- Задание кусочно-определенной функции осуществляется с помощью встроенной функции `case`. Например, задать функцию

$$f(x) = \begin{cases} \sin(x)/x, & x \neq 0 \\ 1, & x = 0 \end{cases}$$

можно так:

```
f(x)=case(x<>0=>sin(x)/x)else(1)
```

либо

```
f(x)=case(x<>0=>sin(x)/x|x=0=>1)
```

(При этом сравнение с нулём происходит не на точное равенство, а на попадание в область на уровне нескольких "машинных эпсилон") Функцию

$$f(x) = \begin{cases} 1, & x \geq 1 \\ x, & -1 \leq x < 1 \\ -1, & x < -1 \end{cases}$$

можно задать как

```
f(x)=case(x>=1 => 1 | x in [-1..1)=> x | x<-1 => -1)
```

- Задание псевдослучайного шума осуществляется с помощью функции `noise(a)`, где значение a задаёт амплитуду шума. При этом псевдослучайные величины распределены равномерно на интервале от `0` до `a`. Так, `f(x)=sin(x)+ noise(0.1)` даст шум амплитудой `0.1`, добавляемый к функции `sin(x)`. Функция `f(x)=(1+noise(0.1))*sin(x)` даст случайную модуляцию амплитуды. "Функция" `f(x)=sin(x+noise(0.1))` даст имитацию измерений со случайной аддитивной составляющей при "измерении" аргумента `x`. "Функция" `f(x)=sin((1+noise(0.01))*x)` даст имитацию относительной погрешности "измерения" аргумента `x`, равную 1%.

## 7.1 Общие представления о численном дифференцировании

Рассмотрим выполнение дифференцирования функций, выполняемое численно на компьютере с помощью замены отношения дифференциала функции $df$ к дифференциалу аргумента $dx$ отношением приращения функции $\Delta f$ на интервале к величине этого интервала $\Delta x$. Выполнение данной процедуры отличают следующие особенности:

- Вычисления можно проводить только в конечном числе точек. О ходе функции между точками без дополнительных сведений судить нельзя.

Обычно предполагают, что ход функции между соседними точками мало отличается от хода интерполяционного полинома малой степени (линейного, квадратичного или кубического). Если же функция является быстро осциллирующей, по вычисленным значениям вообще невозможно восстановить форму функции и вычислить производную. Соответствующие погрешности становятся катастрофически большими. Мы будем называть их ошибками частоты дискретизации данных.

- Формулы численного дифференцирования носят приближенный характер из-за пренебрежения остаточными членами в ряде Тейлора. Будем называть соответствующие погрешности аналитическими.

- Компьютер может оперировать лишь с конечным числом чисел - подмножеством рациональных чисел. Вещественные числа хранятся в компьютере с конечной точностью (называемой "машинным эпсилон"), зависящей от формата хранения. Все арифметические действия с такими числами вносят дополнительные погрешности, связанные с особенностями такого формата. Будем называть связанные с этим погрешности вычислений погрешностями компьютерного представления вещественных чисел.

- Вычисления по формулам численного дифференцирования характеризуются резким возрастанием вычислительной ошибки как при увеличении $\Delta x$ по сравнению с оптимальным значением $\Delta x_{opt}$, так и при уменьшении $\Delta x$ по сравнению с этим значением.

- Часто данные, которые необходимо продифференцировать, представлены в дискретном виде, не допускающем аналитического дифференцирования. Практически всегда такие данные имеют погрешности, на порядки превышающие "машинное эпсилон".

Ниже мы сделаем попытку исследовать данные особенности.

Узнайте у преподавателя и запишите в рабочую тетрадь, на примере каких функций и на каких интервалах изменения аргумента будут выполняться исследования. Продифференцируйте функции аналитически (на бумаге или при помощи программ аналитических вычислений). Мы будем рассматривать только функции, производные которых с ростом номера производной

растут не слишком быстро, так что погрешность ряда Тейлора с ростом числа учитываемых членов стремится к нулю. О значении таких функций будет рассказано на третьем курсе.

В качестве примера можно взять функции

$$f(x) = \frac{\sin x}{x}, \quad f(x) = x^{\alpha} \sin x, \quad f(x) = \frac{a_1 x^2 + b_1 x + c_1}{a_2 x^2 + b_2 x + c_2}$$

и т.п. на промежутке $[-10, 10]$. Традиционное определение производной предполагает вычисление предела

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Мы не можем на компьютере обеспечить бесконечную последовательность точек, нужную для вычисления предела. Поэтому выполним вычисления просто при достаточно малом $h$, заменяя $df/dx$ в точке $x$ отношением $\Delta f / \Delta x$. Простейший вариант - несимметричная относительно точки $x$ формула

$$f'(x) = \frac{f(x+h) - f(x)}{h} \tag{7.1}$$

## 7.2 Ошибки частоты дискретизации данных

**Задание 7.1** Функция Дирихле равна единице при рациональных значениях аргумента и нулю при всех остальных. Чему равна ее производная? Существует ли она? А что даст вычисление на компьютере?

Конечно, функция Дирихле - "очень плохая". А вот "хорошая" функция, имеющая гладкие производные всех порядков:

$$y(t) = \sin 2\pi F t$$

где $t$ - время [с], а $F$ - частота колебаний [Гц]. Такая функция уже встречалась в колебаниях на первом курсе.

- Что даст построение графиков функции $y(t)$ и вычисление численной производной этой функции по 1000 точкам на интервале от 0 до 1с для следующих частот из характерного диапазона частот механических колебаний - 1 Гц, 1000 Гц, 999 Гц, 998 Гц, 997 Гц? Постройте график по

1001 точке для частоты 1000 Гц. Можно ли на основании этих графиков судить о форме функции? Объясните полученные результаты. Если это не получится, попробуйте увеличить число точек для расчёта до 1100, 1111, 2000, 2111.

- Что даст построение графиков функции $y(t)$ и вычисление численной производной этой функции по 1000 точкам на интервале от 0 до 1с для диапазона радиочастот (100кГц 1гГц)? Оптических частот (порядка $10^{15}$ Гц)?

- Оцените, насколько надо увеличить число точек для получения графиков разумной формы в диапазоне радиочастот и в оптическом диапазоне частот. Реалистично ли это? Постройте в одном и том же окне графики функции $y(t)$ для одного значения оптической частоты при отличающихся значениях числа точек на графике. Можно ли на основании этих графиков судить о форме функции?

- Сделайте выводы о том, как правильно выбирать масштабы при вычислениях в физических задачах, и запишите их в рабочую тетрадь.

**Задание 7.2** На интервале -1..1 постройте график функции

$$f(x) = \begin{cases} x\sin(1/x), & x \neq 0 \\ 0, & x = 0 \end{cases}$$

Найдите её аналитическую и численную производную. Как будет зависеть погрешность численного вычисления производной от значения $x$ и числа точек, используемых для расчета? Занесите результаты в рабочую тетрадь.

## 7.3 Аналитические погрешности численного дифференцирования

**Задание 7.3** Сравните графики аналитически вычисленной производной и производной, вычисленной по формуле (7.1), при нескольких достаточно больших значениях $h$. Постройте графики абсолютной и относительной погрешности численного вычисления производной. Объясните формулу графика абсолютной погрешности при больших $h$. Занесите результаты в рабочую тетрадь.

**Задание 7.4**

- Проанализируйте погрешность, связанную с приближенным характером формулы (7.1). Проанализируйте ее связь с остаточным членом ряда Тейлора для этой формулы:

$$\frac{f(x+h)-f(x)}{h} = f'(x) + \frac{1}{2!}f''(x)h + \frac{1}{3!}f'''(x)h^2 + \dots \qquad (7.2)$$

- Попробуйте устранить вклад первого и, если необходимо, второго поправочного членов в погрешность численного дифференцирования, введя вычитание этих членов в формуле для нахождения $\frac{\Delta f}{\Delta x}$. Выясните и запишите в отчет значения $h$, при котором их вклад в погрешность становится явно заметным на графике погрешности при заданном типе чисел.

- Укажите, вклад какого члена остаточного ряда является определяющим. Попробуйте улучшить точность формулы (7.1) путём вычитания соответствующего члена ряда в пункте экранной формы, где задаётся формула для нахождения численной производной.

- Попробуйте привести пример функции, для которой в некоторой области вкладом второго и последующих членов ряда (7.2) нельзя пренебречь ни при каких значениях $h$.

- Занесите результаты в рабочую тетрадь.

**Задание 7.5** Наиболее очевидным способом увеличения точности при заданном достаточно большом значении шага $h$ является использование симметричной разностной формулы

$$\frac{\Delta f}{\Delta x} = \frac{f(x+\frac{h}{2})-f(x-\frac{h}{2})}{h} \qquad (7.3)$$

- Определите порядок по $h$ погрешности метода формулы (7.3), записав для неё в рабочей тетради разложение, подобное разложению (7.2) для формулы (7.1). Почему этот порядок выше, чем у формулы (7.2)?

- Занесите результаты в рабочую тетрадь.

**Задание 7.6** Повторите задания 5.3-5.4 для симметричной формулы (7.3). Сравните погрешности метода формулы (7.1) и формулы (7.3). Постройте графики для двух-трех значений $h$.
Сравните результаты с полученными при вычитании главного члена остаточного ряда для формулы (7.1).
Занесите результаты в рабочую тетрадь.

## 7.4 Погрешности компьютерного представления чисел

Следует отличать погрешности вычислений, возникающие при выполнении арифметических действий из-за способа компьютерного представления чисел, от аналитических погрешностей приближения производной той или иной формулой. Теоретически оценить их вклад оказывается затруднительно. Наиболее простым способом увидеть вклад таких ошибок арифметических действий можно, если изменить типы переменных в программе. Все другие ошибки (не связанные с формой представления вещественных чисел в компьютере) назовем погрешностями метода.

**Задание 7.7**

- Повторите Задание 5.3, уменьшая значение $h$ в $10^1$, $10^2$ и т.д. раз. Сравните результаты с полученными в Задании 3 и сделайте вывод о том, при каких значениях $h$ велики ошибки, связанные с погрешностями компьютерного представления чисел, при использовании формулы (7.1).
- Оцените оптимальное значение $h$, при котором погрешность численного дифференцирования минимальна.
- Занесите результаты в рабочую тетрадь.

**Задание 7.8**

- Измените разрядность вычислений и повторите Задание 5.7. Объясните, как можно качественно оценить, какая часть погрешности связана погрешностями внутреннего представления чисел, а какая - аналитическая.

- Занесите результаты в рабочую тетрадь.

**Задание 7.9**

- Повторите задания 5.7 и 5.8 для формулы (7.3) при различных разрядностях вычислений.
- Сравните оптимальные значения $h$ для формул (7.1) и (7.3) и соответствующие значения погрешностей.
- Занесите результаты в рабочую тетрадь.

## 7.5 Вторые производные

Численное нахождение второй производной $d^2f(x)/dx^2$ можно осуществить использованием формулы (7.3) для производной $df(x)/dx$, вычисленной по той же формуле:

$$\frac{d^2f(x)}{dx^2} \approx \frac{f'(x+h) - f'(x)}{h} = \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \qquad (7.4)$$

Чтобы формула была симметричной относительно концов интервала, вычисленное значение обычно относят не к точке $x$, а точке $x + h$. Тогда для точки $x$ получим

$$\frac{d^2f(x)}{dx^2} \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \qquad (7.5)$$

**Задание 7.10**

- Вычислите вторую производную по формуле (7.5) при изменении $h$ в области относительно больших значений, и сравните с результатами аналитических вычислений. Сравните абсолютные и относительные погрешности вычисления первой и второй производной при одном и том же значении $h$ и характер убывания погрешности с уменьшением $h$.
- Влияет ли на точность результатов тип представления вещественных чисел?
- Занесите результаты в рабочую тетрадь.

**Задание 7.11**

- Проведите аналогичное исследование в области малых $h$.
- Занесите результаты в рабочую тетрадь.

**Задание 7.12**

- Выясните оптимальные значения $h$ для разных типов представления вещественных типов и соответствующие им величины погрешностей.
- Сравните значения $h$ и абсолютные погрешности с соответствующими "машинными эпсилон ".
- Оцените возможности численного нахождения третьей и более высоких производных.
- Занесите результаты в рабочую тетрадь.

**Задание 7.13**

- Используйте для вычислений производной формулу (7.4) вместо (7.5). Выясните, какая из них даёт лучшую точность. Почему? Есть ли заметное отличие в области малых $h$? Почему? В области больших $h$? Почему?
- Занесите результаты в рабочую тетрадь.

## 7.6 Численное дифференцирование таблично заданных данных

До сих пор нами исследовались аналитически заданные функции, известные совершенно точно. Для нахождения их производных, естественно, лучше использовать системы аналитических вычислений. Однако в подавляющем большинстве случаев приходится работать с дискретными массивами данных, и к тому же полученных с погрешностью, заметно превышающей величину "машинного эпсилон". Эти данные могут быть получены либо в результате экспериментальных измерений, либо в результате численного расчета с

большими случайными ошибками. Встает вопрос о способах нахождения производной по этим данным.

Пусть у нас имеется массив значений $f_i$ измерения величины $f(x)$, и соответствующий массив значений аргумента $x_i$, и при этом $x_i$ расположены с монотонным возрастанием. Самым простым способом вычисления зависимости $df(x)/dx$ является построение зависимости $\Delta f_i/\Delta x_i$ от $x_i$, где $\Delta f_i = f_{i+1} - f_i$, $\Delta x = x_{i+1} - x_i$, то есть

$$\frac{df(x)}{dx} \approx \frac{\Delta f_i}{\Delta x_i} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} = \frac{f(x+h) - f(x)}{h} \tag{7.6}$$

Аналогичным образом может быть записана симметричная формула (подумайте, в каких точках ее вычислять). Для имитации погрешностей экспериментальных данных будем использовать числовую генерацию псевдослучайного шума заданной величины, добавляемого в соответствующем пункте программы к аналитически задаваемой функции.

### Задание 7.14

- На равномерной сетке вычислите значения $f_i$ по 100 точкам при разных амплитудах шума, добавляемого к $f(x_i)$, и найдите численные производные по формуле (7.6). Выясните, как зависит погрешность вычисления $df(x)/dx$ от амплитуды шума, и при какой амплитуде шума погрешность измерения производной не слишком велика.

- Занесите результаты в рабочую тетрадь.

### Задание 7.15

- Увеличьте в несколько раз число "экспериментальных " точек при неизменных интервале $[a, b]$ и амплитуде шума. Как изменилась погрешность? Почему?

- Занесите результаты в рабочую тетрадь.

### Задание 7.16

- Сгладьте входные данные с помощью усреднения по нескольким точкам и вычислите производную. Выясните, как зависит погрешность от величины входного шума при заданном числе точек усреднения, и от числа

точек усреднения при заданном шуме "малой", "средней" и "большой" величины.

- Занесите результаты в рабочую тетрадь.

**Задание 7.17**

- Повторите то же, используя сглаживание с помощью аппроксимирующих B1- и B2- сплайнов. Сравните полученные результаты.

- Занесите результаты в рабочую тетрадь.

**Задание 7.18**

- Вычислите вторую производную с помощью дифференцирования по точкам, найденным в результате первого дифференцирования. Как ведет себя погрешность в зависимости от амплитуды шума и числа точек усреднения?

- Надо ли перед взятием второй производной проводить повторное сглаживание данных?

- Занесите результаты в рабочую тетрадь.

**Задание 7.19**

- Задайте в качестве входных данных функцию $f(x) = \sin(\omega(x)x)$, где функцию $\omega(x)$ выберите по своему усмотрению.

- Выясните, сколько точек надо задать на интервале [-10,10] при амплитуде шума 0,01 (т.е. 1%), чтобы можно было получить при использовании сглаживания погрешность в вычислении а) первой производной 1%; б) второй производной 10%.

- Занесите результаты в рабочую тетрадь.

# Работа № 8

## Численное интегрирование

## 8.1 Формулы Ньютона-Котеса

Пусть нам надо сосчитать интеграл

$$I = \int_a^b f(x)dx.$$

Выполним следующие действия:

- На интервале $[a, b]$ зададим узлы $x_k$, $k = 1, \ldots, N$
- произведем интерполяцию функции $f(x)$ по узлам $x_k$. Для этого воспользуемся встроенной функцией MATLAB, которая вычисляет коэффициенты полинома

  ```
  P=polyfit(X,F(X),N-1)
  ```

  Здесь предполагается, что узлы интерполяции записаны в вектор `X`, длина которого `N`, а для вычисления функции $f(x)$ имеется m.-функция `F`, которая допускает векторный аргумент.

- Произведем аналитическое интегрирование полинома при помощи стандартной функции `polyint`, которая возвращает вектор коэффициентов полинома, являющегося первообразной подинтегрального выражения

  ```
  Q = polyint(P)
  ```

- Теперь получим приближенное значение интеграла $I$ по формуле

$$I = Q(b) - Q(a).$$

Для вычисления значения полинома, заданного своими коэффициентами, используем стандартную функцию `polyval`

```
I = polyval(Q,b) - polyval(Q,a)
```

Перечисленные выше действия удобно оформить в виде m-файла, что облегчит выполнение дальнейших заданий.

**Задание 8.1** Вычислите интеграл

$$\int_0^{\pi} \sin(x)dx,$$

используя равно отстоящие узлы. Сравните результаты для различных `N`. Постройте график абсолютной величины погрешности от числа узлов `N`$= 2, \ldots, 50$ в полу логарифмическом масштабе (воспользуйтесь функцией `semilogy`). Объясните, почему при большом числе узлов погрешность нарастает.

**Задание 8.2** Вычислите интеграл, предложенный преподавателем, с заданной точностью (Оцените погрешность по разности приближенных значений, полученных для разных `N`).

Вместо того, чтобы для каждой интегрируемой функции строить интерполяционный полином, удобнее получить квадратурную формулу в виде

$$\int_a^b f(x)dx \approx \sum_{k=1}^{N} \lambda_k f(x_k),$$

то есть, заранее вычислить весовые коэффициенты $\lambda_k$. Как известно,

$$\lambda_k = \int_a^b \mathcal{L}_{N-1}^k(x)dx, \tag{8.1}$$

где $\mathcal{L}_{N-1}^k$ — $k$-тый полином Лагранжа на сетке $\{x_j\}_{j=1}^N$.

Для вычисления весового коэффициента $\lambda_k$ выполним перечисленные в начале параграфа действия, но в качестве значений функции в узлах сетки возьмем $f(x_j) = \delta_k^j$.

**Задание 8.3** Построить квадратурную формулу для заданного в задании 2 интеграла. Проверить, что

$$\sum_{k=1}^{N} = b - a.$$

Вычисление весовых коэффициентов оформить в виде m-файла.

## 8.2 Формулы Гаусса-Кристофеля

Как известно формулы наивысшей алгебраической степени точности строятся при помощи ортогональных полиномов. В качестве узлов квадратурной формулы выбираются нули ортогонального полинома, а весовые коэффициенты вычисляются по формуле (8.1).

Будем искать ортогональный полином $P_N(x)$ в виде разложения по степеням $x$

$$P_N(x) = \sum_{k=0}^{N} c_k x^k.$$

Условия ортогональности $P_N$ степеням $x^m$, при $m = 0, \ldots, N-1$ приводит к системе уравнений на коэффициенты $c_k$

$$\sum_{k=0}^{N} c_k \int_a^b x^{k+m} dx \equiv \sum_{k=0}^{N} c_k \frac{b^{k+m+1} - a^{k+m+1}}{k+m+1} = 0, \quad m = 0, 1, \ldots, N-1.$$

Поскольку ортогональные полиномы достаточно знать с точностью до нормировочного множителя, будем считать $c_N = 1$. Тогда получим для остальных коэффициентов систему с квадратной матрицей

$$A\vec{c} = \vec{b},$$

где

$$\vec{c} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{N-1} \end{pmatrix}, \qquad \vec{b} = - \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{pmatrix}, \qquad A = \begin{pmatrix} a_1 & a_2 & \cdots & a_N \\ a_2 & a_3 & \cdots & a_{N+1} \\ \vdots & \vdots & \ddots & \vdots \\ a_N & a_{N+1} & \cdots & a_{2N-1} \end{pmatrix},$$

$$a_\ell = \frac{b^\ell - a^\ell}{\ell}.$$

Реализуем вычисление коэффициентов ортогональных полиномов в виде функции

```
function P=ortpoly(a,b,N) for
k=1:N
    for m=1:N
        l=k+m-1;
        A(k,m) = (b^l-a^l)/l;
    end
    l=k+N;
    B(k)=-(b^l-a^l)/l;
end
C = B/A;
P(1)=1;
for k=1:N
    P(k+1)=C(N+1-k);
end
```

Функция получает в качестве аргументов интервал интегрирования и степень ортогонального полинома, который надо вычислить. Результатом является вектор `P`, содержащий коэффициенты полинома в виде пригодном для функций `polyval`, `polyint` и других функций MATLAB, предназначенных для работы с полиномами.

**Задание 8.4** Напишите указанную выше функцию `ortpoly`. Используя стандартные функции `conv` (перемножение полиномов), `polyint` (интегрирование полиномов), проверьте выполнение условий ортогональности. Для этого сгенерируйте полином степени $N-1$ со случайными коэффициентами (`rand` генерирует случайный вектор) и проверьте его ортогональность $P_N(x)$. Постройте графики нескольких ортогональных полиномов.

**Задание 8.5** Модифицируйте функцию `ortpoly` так, чтобы ортогональные полиномы были нормированы. Постройте графики нескольких ортогональных полиномов.

Узлы квадратурной формулы наивысшей алгебраической степени точности совпадают с нулями ортогонального полинома

$$P_N(x_k) = 0.$$

Для поиска нулей полинома воспользуемся встроенной функцией `roots`

```
P=ortpoly(a,b,N);roots(P)
```

**Задание 8.6** Найти узлы и весовые коэффициенты квадратурной формулы наивысшей алгебраической степени точности для заданного $N$. Проверьте, что все узлы лежат на интервале $[a, b]$, являются простыми и располагаются симметрично. Проверьте, что сумма весов равна $b - a$ и симметрию $\lambda_k = \lambda_{N-k}$.

**Задание 8.7** Найдите алгебраическую степень точности квадратурной формулы, построенной в задании 6. Для этого вычислите интегралы от полиномов различной степени и сравните результаты с аналитическим интегрированием (`polyint`).

**Задание 8.8** Выполните задание 6.1 используя интегрирование по квадратурной формуле наивысшей алгебраической степени точности. Сравните погрешности для разных $N$, получаемые при использовании равноотстоящих узлов и узлов в корнях ортогональных полиномов. Объясните рост погрешности при больших $N$.

**Задание 8.9** Выполните задание 6.2 используя интегрирование по квадратурной формуле наивысшей алгебраической степени точности.

## 8.3 Составные формулы

Как показали численные примеры, безграничное увеличение степени интерполирующего полинома не приводит к снижению погрешности. Поэтому для вычисления интегралов используется метод разбиения интервала интегрирования на короткие участки, на каждом из которых применяется квадратурная формула с некоторым не слишком большим $N$. Такие квадратурные формулы называются составными.

Составная формула левых прямоугольников отвечает

$$x_k = a + \frac{k-1}{N-1}(b-a), \quad k = 1, \ldots, N,$$

$$\lambda_k = \begin{cases} \frac{b-a}{N-1}, & k = 1, \ldots, N-1 \\ 0, & k = N \end{cases}$$

Составная формула правых прямоугольников отвечает тем же узлам, но

$$\lambda_k = \begin{cases} 0, & k = 1, \\ \frac{b-a}{N-1}, & k = 2, \ldots, N \end{cases}$$

Формула средних

$$x_k = a + \frac{k - 1/2}{N}(b-a), \quad \lambda_k = \frac{b-a}{N}, \qquad k = 1, \ldots, N$$

Формула трапеций

$$x_k = a + \frac{k-1}{N-1}(b-a), \quad k = 1, \ldots, N,$$

$$\lambda_k = \begin{cases} \frac{b-a}{N-1}, & k = 2, \ldots, N-1 \\ \frac{b-a}{2N-2}, & k = 1, N \end{cases}$$

**Задание 8.10** Выполните задания 6.1 и 6.2, используя перечисленные выше составные формулы.

# Работа № 9

## Решение систем линейных уравнений в MATLAB

## 9.1 Обращение матриц

Обращение матриц - одна из наиболее распространенных операций матричного анализа. Обратной называют матрицу, получаемую в результате деления единичной матрицы $E$ на исходную матрицу $X$. Таким образом, $X^{-1} = \frac{E}{X}$. Следующая функция обеспечивает реализацию данной операции:

`inv(X)` - возвращает матрицу, обратную квадратной матрице $X$. Предупреждающее сообщение выдается, если $X$ плохо масштабирована или близка к вырожденной.

Пример:

```
inv(rand(4,4))
```

**Вычисление псевдообратных матриц**

Команда `B = pinv(A)` - возвращает матрицу, псевдообратную матрице $A$ (псевдообращение матрицы по Муру-Пенроузу). Напомним, что результатом псевдообращения матрицы по Муру-Пенроузу является матрица $B$ того же размера, что и $A$, и удовлетворяющая условиям $A*B*A = A$ и $B*A*B = B$. Вычисление основано на использовании функции `svd(A)` и приравнивании к нулю всех сингулярных чисел, меньших величины `tol`.

Команда `B = pinv (A. tol)` - возвращает псевдообратную матрицу и отменяет заданный по умолчанию порог, равный `tol=max(size(A))*norm(A)*eps`.

Пример:

```
pinv(rand(4,3))
```

На практике вычисление явной обратной матрицы не так уж необходимо. Чаще операцию обращения применяют при решении системы линейных уравнений вида $Ax = b$. Один из путей решения этой системы - вычисление $x = inv(A) * b$. Но лучшим с точки зрения минимизации времени расчета и повышения точности вычислений является использование оператора матричного деления `x=A\b`. Эта операция использует метод исключения Гаусса без явного формирования обратной матрицы.

## 9.2 Нахождение общего решения системы линейных алгебраических уравнений.

Рассмотрим систему линейных уравнений:

$$Ax = B$$

где $A$ - матрица $m \times n$, $B$ - $m$-мерный вектор (столбец свободных членов), $x$ - искомый $n$-мерный вектор. При решении такой системы могут быть следующие варианты:

**1) m=n**

**a)** Ранг матрицы $A$ равен $n$. В этом случае существует единственное решение системы линейных уравнений, оно же и является ее общим решением. MATLAB позволяет найти его в одно действие которое рассматривалось нами в первой работе: `x=A\b`

**b)** Ранг матрицы $A$ меньше $n$ . В этом случае, вообще говоря, решений нет. Но существуют такие столбцы свободных членов, при которых решение есть (например, если два уравнения в системе одинаковы). Независимо от этого MATLAB при выполнении операции `x=A\b` выдаст сообщение о том, что матрица вырождена, либо близка к вырожденной, и правильного решения может не выдать.

**2) m<n**

**a)** Ранг матрицы $A$ равен $m$ . В этом случае система называется недоопределенной и имеет бесконечно много решений. Все множество решений системы называется *общим решением.* Для оно представляет собой $k$-мерное линейное многообразие в $n$-мерном линейном векторном пространстве. Его можно записать в следующем виде:

$$x = x_0 + c_1x_1 + c_2x2 + \ldots + ckxk$$

Здесь $x_0$ - какое-либо частное решение, а остальная часть суммы - общее решение однородной системы уравнений, т. е. системы с той же матрицей и нулевым столбцом свободных членов. Общее решение однородной системы уравнений представляет собой $k$-мерное подпространство, натянутое на вектора $x_1$ ,$x_2$, ..., $x_k$, называемое ядром матрицы $A$. Числа $c_1$, $c_2$, $c_3$ и т. д. - произвольные константы.

Чтобы найти общее решение системы с помощью MATLAB надо найти частное решение уже известным нам способом:

$$x0 = A\ b$$

и общее решение однородной системы, которое можно искать в двух вариантах: 1) `z = null(A)` 2) `z = null(A,'r')` Функция `null` возвращает матрицу, столбцами которой являются векторы $x_1$, $x_2$ ,..., $x_k$. Эти векторы, как и вектор $x_0$, вообще говоря, определены неоднозначно. MATLAB всегда находит вектор $x_0$, имеющий не менее $k$ нулевых компонент в разложении по исходному базису. Векторы $x_1$, $x_2$, ..., $x_k$ в варианте (1) представляют собой один из ортонормированных базисов ядра, в варианте (2) - рациональный базис ядра. Вектора рационального базиса имеют $k-1$ нулевых компонентов и 1 единичный компонент в разложении по исходному базису (это последние $k$ компонентов).

**b)** Ранг матрицы $A$ меньше $m$. В этом случае, вообще говоря, решений нет, как и в случае 1(b). Тем не менее, MATLAB позволяет выполнить операцию `x = A\b` и получить в качестве результата вектор , для которого значение минимально. При этом будет выдано предупреждение о недостаточности ранга матрицы.

**3) m>n**

**a)** Ранг матрицы $A$ равен $n$. В этом случае система, как правило, не имеет решений. Однако MATLAB позволяет выполнить знакомую нам операцию `x = A\b` и получить вектор $x$, для которого значение $|Ax - b|^2$ минимально. Никаких предупреждений при этом не выдается, поскольку считается что вы решаете именно эту задачу минимизации.

**b)** Ранг матрицы $A$ меньше $n$. В этом случае система, конечно (как правило), тоже не имеет решений. MATLAB же поступает как в случае 2(b) - выполняет операцию `x=A\b`, выдает в качестве результата вектор $x$, для которого значение $|Ax - b|^2$ минимально, и выводит предупреждение о недостаточности ранга матрицы $A$.

Напомним, что для создания случайной матрицы следует использовать команду `X=rand(m,n)`, которая формирует массив размера m x n, элементами которого являются случайные величины, распределенные по равномерному закону в интервале (0, 1). Для создания вырожденной матрицы m x n можно создать случайную матрицу меньшего размера, например m-1 x n, и добавить к ней уже существующий столбец. Аналогичным образом создаются матрицы заданного ранга.

**Задание 9.1**

1. Задайте случайную квадратную матрицу, случайный столбец свободных членов и решите систему линейных уравнений. Найдите длину вектора, чтобы оценить погрешность.

2. Задайте вырожденную квадратную матрицу , случайный столбец свободных членов и попытайтесь решить систему линейных уравнений. Попытайтесь решить систему с двумя одинаковыми уравнениями. Попытайтесь решить систему, в которой первое уравнение является линейной комбинацией второго и третьего. Если MATLAB находит решение - проверьте его.

3. Задайте случайную матрицу, случайный столбец свободных членов и найдите общее решение системы линейных уравнений с ортонормированным

базисом в качестве решения однородной системы. Проверьте ортогональность и нормированность найденных базисных векторов. Найдите длины векторов для нескольких частных решений.

4. Найдите общее решение системы линейных уравнений из задания 7.3 с рациональным базисом в качестве решения однородной системы. Проверьте, являются ли найденные базисные вектора ортогональными или нормированными.

5. Задайте матрицу ранга 4 и найдите решение задачи минимизации величины. Единственно ли это решение? Найдите длину вектора.

6. Найдите общее решение однородной системы линейных уравнений с матрицей из предыдущего задания. Добавьте к вектору из предыдущего задания произвольное частное решение однородной системы и снова найдите длину вектора . Изменилась ли она? Почему?

7. Задайте случайную матрицу с числом строк превышающим число столбцов и случайный столбец свободных членов. Решите задачу минимизации величины . Единственно ли это решение?

8. Задайте матрицу с числом строк превышающим число столбцов и рангом меньшим, чем число столбцов. Задайте случайный столбец свободных членов. Решите задачу минимизации величины . Единственно ли это решение? Если решение не единственно, найдите общее решение.

9. Ответьте на вопрос: почему MATLAB выдает предупреждения о недостаточном ранге в случаях 2(b) и 3(b)?

## 9.3 $LU$- и $QR$-разложения

Так называемые $LU$- и $QR$-разложения реализуются следующими матричными функциями:

Функция `lu` выражает любую квадратную матрицу $X$ как произведение двух треугольных матриц, одна из которых (возможно, с перестановками) - нижняя треугольная матрица, а другая - верхняя треугольная матрица. Иногда эту операцию называют $LR$-разложением. В MATLAB, начиная с версии

6, аргументом (входным аргументом) функции `lu` может быть и полная прямоугольная матрица. Для выполнения этой операции служит следующие формы функции `lu`:

- `[L,U] = lu(X)` - которая возвращает верхнюю треугольную матрицу $U$ и нижнюю треугольную матрицу $L$ (т. е. произведение нижней треугольной матрицы и матрицы перестановок), так что $X = L * U$;

- `[L,U,P.] = lu(X)` - возвращает верхнюю треугольную матрицу $U$, нижнюю треугольную матрицу $L$ и сопряженную (эрмитову) матрицу матрицы перестановок $P$, так что $L * U = P * X$;

- `lu(X)` - вызванная с одним выходным параметром функция возвращает результат из подпрограмм DGETRF (для действительных матриц) или ZGETRF (для комплексных) известного пакета программ линейной алгебры LAPACK.

- `lu(X, thresh)` - где thresh в диапазоне [0...1] управляет центрированием в разреженных матрицах. Отдельная форма предыдущего случая. Центрирование происходит, если элемент столбца на диагонали меньше, чем произведение thresh и любого поддиагонального элемента. Thresh=1 - значение по умолчанию. Thresh=0 задает центрирование по диагонали. Если матрица полная (не разреженная), выводится сообщение об ошибке.

Функция `qr` выполняет $QR$-разложение матрицы. Эта операция полезна для квадратных и треугольных матриц. Она выполняет $QR$-разложение, вычисляя произведение унитарной матрицы $Q$ и верхней треугольной матрицы $R$. Функция используется в следующих формах:

- `[Q.R] = qr(X)` - вычисляет верхнюю треугольную матрицу $R$ того же размера, как и у $X$, и унитарную матрицу $Q$, так что $X = Q * R$;

- `[Q.R.E] = qr(X)` - вычисляет матрицу перестановок $E$, верхнюю треугольную матрицу $R$ с убывающими по модулю диагональными элементами и унитарную матрицу $Q$, так что $X * E = Q * R$. Матрица перестановок $E$ выбрана так, что $abs(diag(R))$ уменьшается;

- `[Q.R] = qr(X.0)` и `[Q.R.E] = qr(X,0)` - вычисляют экономное разложение, в котором $E$ - вектор перестановок, так что $Q * R = X(:, E)$. Матрица $E$ выбрана так, что невязка $abs(diag(R))$ уменьшается;
- `A = qr(X)` - возвращает результат из LAPACK.

Пример:
```
C=rand(5.4)
    [Q.R]=qr(C)
```

**Задание 9.2** Проверьте что в $QR$ разложении матрица $Q$ унитарна. Напомним определение унитарной матрицы - квадратная матрица с комплексными элементами, обладающая тем свойством, произведение ее на комплексно сопряженную матрицу равно единице т. е. $Q^*Q = QQ^* = 1$.

**Задание 9.3** Проверьте что унитарное преобразование $x' = Qx$, где $Q$ унитарная матрица, сохраняет скалярное произведение $(x', y') = (Qx, Qy) = (x, y)$, для любых векторов $x, y$.

Кроме этого команда `[Q,R] = qrdelete(Q,R, j)` - изменяет $Q$ и $R$ таким образом, чтобы пересчитать $QR$-разложение матрицы $X$ для случая, когда в ней удален j-й столбец. Входные значения $Q$ и $R$ представляют $QR$-разложение матрицы $X$ как результат действия `[Q.R]=qr(X)`. Аргумент $j$ определяет столбец, который должен быть удален из матрицы $X$.
Пример:
```
C=rand(3.3)
[Q.R]=qr(C)
[Q1.R1]=qrdelete(Q.R.2)
```
Другая команда `[Q.R] = qrinsert(Q,R,j,x)` - изменяет $Q$ и $R$ таким образом, чтобы пересчитать разложение матрицы $X$ для случая, когда в матрице $X$ перед $j$-м столбцом вставлен столбец $x$. Входные значения $Q$ и $R$ представляют $QR$-разложение матрицы $X$ как результат действия `[Q,R]=qr(X)`. Аргумент $x$ - вектор-столбец, который нужно вставить в матрицу $X$. Аргумент $j$ определяет столбец, перед которым будет вставлен вектор $x$.
Пример:
```
C=rand(3.3)
[Q,R]-qr(c)
x=[0.5.-0.3.0.2];[Q2.R2]=qrinsert(Q.R.2,x')
```

# Работа № 10

## Решение дифференциальных уравнений I.

В данной работе рассматриваются несколько простейших методов решения задачи Коши для дифференциального уравнения первого порядка

$$\frac{dy(t)}{d}t = f(t, y(t))$$

на интервале $t \in [a, b]$ с граничным условием $y(a) = y_0$.

Выберем фиксированное приращение $\Delta t = h$ и введем следующие обозначения

$$\begin{aligned} &t_k = t_{k-1} + h = a + kh, \\ &y_k \simeq y(t_k) = y(a + kh), \qquad k = 0, 1, \ldots, N, \\ &f_k = f(t_k, y_k) \simeq y'(x_k). \end{aligned}$$

Разность $y_{k+1} - y(x_{k+1})$ между вычисленным и точным значениями решения называют *ошибкой* усечения. Если в формулах численного интегрирования, которые следуют далее, заменить все точные значения $y(x_0), \ldots, y(x_k)$ на приближенные величины $y_0, \ldots, y_k$, то разность $y_{k+1} - y(x_{k+1})$ даст *локальную ошибку* усечения. Тем самым полная ошибка вызывается не только локальной ошибкой, но и накапливанием локальных ошибок на более ранних шагах интегрирования.

## 10.1 Методы Эйлера, Гойна и Рунге-Кутта

В методе Эйлера производная в дифференциальном уравнении заменяется приближенным значением

$$y'(x_{k+1}) \simeq \frac{y_{k+1} - y_k}{h}$$

откуда следует, что

$$y_{k+1} = y_k + h\, f(t_k, y_k), \qquad k = 0, 1, \ldots, N.$$

Метод Эйлера дает хорошее приближение решения только при достаточно малом шаге $h$ и только для нескольких первых точек.

В качестве примера рассмотрим решение уравнения

$$y' = 3 + t - y\,, \qquad t \in [0, 1], \qquad y(0) = 1\,.$$

Как и всюду далее, прежде чем преступить к реализации алгоритма решения, нам будет необходимо задать само уравнение, т.е. составить $m$-файл-подпрограмму для функции `f(t,y)`. Этот файл имеет очень простой вид

```
function f=f(t,y)
f=3+t-y
```

Эту подпрограмму необходимо *сохранить* в $m$-файле, прежде чем идти дальше. Теперь необходимо задать начальные условия, определить шаг $h$ или количество шагов $N = (b-a)/h$ и запрограммировать алгоритм интегрирования

```
clear t; clear y;
a=0; b=1;
y0=1;
N=10;
h=(b-a)/N;
t(1)=a; y(1)=y0;
for n=1:N
  t(n+1)=t(n)+h;
  y(n+1)=y(n)+h*f(t(n),y(n));
end
```

Ответ можно нарисовать, используя команду

```
plot(t,y)
title(['Euler method with N=',num2str(N),'steps'])
```

Для того, чтобы сравнить полученный результат с точным решением определим это точное решение с помощью $m$-файла-подпрограммы

```
function ys=ys(t)
ys=2*ones(size(t))+t-exp(-t)
```

Сохраним этот файл и построим график

```
plot(t,y,'-',t,ys(t),'-.')
title(['Euler method withN=',num2str(N),'steps'])
```

На этом графике видно как точное решение отличается от численного решения и как накапливается ошибка с ростом числа шагов.

Подставляя разные численные приближения производной $y'(t)$, которые изучались в одной из предыдущих работ, в дифференциальное уравнение мы получим разные одношаговые методы решения задачи Коши. Среди них выделим

- модифицированный метод Эйлера

$$y_{k+1} = y_k + \frac{h}{2}\Big(f(t_k, y_k) + f\left(t_k + h, y_k + h\, f(t_k, y_k)\right)\Big);$$

- метод средней точки

$$y_{k+1} = y_k + h\, f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}\, f(t_k, y_k)\right);$$

- метод Гойна

$$y_{k+1} = y_k + \frac{h}{4}\left(f(t_k, y_k) + 3f\left(t_k + \frac{2}{3}h, y_k + \frac{2}{3}hf(t_k, y_k)\right)\right);$$

- метод Рунге-Кутта третьего порядка - стандартное обозначение rk23

$$y_{k+1} = y_k + \frac{h}{6}\left(k_1 + 4k_2 + k_3\right),$$

где

$$k_1 = f(t_k, y_k), \qquad k_2 = f\left(t_k + \frac{h}{2}, y_k + \frac{k_1}{2}\right),$$
$$k_3 = f\left(t_k + h, y_k + 2k_2 - k_1\right).$$

**Задание 10.1** Перепишите приведенный выше методу Эйлера в виде подпрограммы `Euler`, которая зависит от подпрограммы функции `F`, вектора `tspan`, определяющего интервал интегрирования $[t_0, t_{final}]$, граничного условия `y0` и числа шагов `N`. Проверьте работоспособность вашей подпрограммы.

**Задание 10.2** Напишите подпрограммы реализующие еще два-три метода интегрирования - по выбору преподавателя из перечисленных выше одношаговых методов.

**Задание 10.3** С помощью созданного вами программного обеспечения постройте решения задачи Коши

$$\frac{dy}{dt} = y - t^2 + 1, \qquad t \in [0, 4], \qquad y(0) = 0.5$$

с шагом $h = 1$ и $h = 0.1$ различными методами. Сравните полученные решения с точным решением на графиках. Какой метод и при каком шаге дает более точное решение?

**Задание 10.4** Постройте решения задачи Коши

$$\frac{dy}{dt} = sin(t), \qquad t \in [1, 5], \qquad y(1) = 0$$

с шагом $h = 1$ и $h = 0.1$ различными методами. Сравните полученные решения с точным решением. Какой метод и при каком шаге дает более точное решение?

**Задание 10.5** Постройте решения задачи Коши

$$\frac{dy}{dt} = \frac{2y}{t} + t^2 \exp(t); \qquad t \in [1, 3], \qquad y(1) = 0$$

с шагом $h = 1$ и $h = 0.1$ различными методами. Сравните полученные решения с точным решением. Какой метод и при каком шаге дает более точное решение?

**Задание 10.6** Сравните поведение полученных вами погрешностей вычислений с теоретическими значениями погрешностей.

## 10.2 Встроенные процедуры решения дифференциальных уравнений

Для решения систем обыкновенных дифференциальных уравнений (ОДУ) в системе MATLAB реализованы различные численные методы. Их реализации названы решателями, от английского слова solver.

Краткое описание решателей

- `ode45` - одношаговые явные методы Рунге-Кутта 4-го и 5-го порядка. Это классический метод, рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты.
- `ode23` - одношаговые явные методы Рунге-Кутта 2-го и 4-го порядка. При умеренной жесткости системы ОДУ и низких требованиях к точности этот метод может дать выигрыш в скорости решения.
- `ode113` - многошаговый метод Адамса-Башворта-Мултона переменного порядка. Это адаптивный метод, который может обеспечить высокую точность решения.
- `ode23tb` - неявный метод Рунге-Кутта в начале решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем. Несмотря на сравнительно низкую точность, этот метод может оказаться более эффективным, чем `ode15s`.
- `ode15s` - многошаговый метод переменного порядка (от 1 до 5, по умолчанию 5), использующий формулы численного дифференцирования. Это адаптивный метод, его стоит применять, если решатель `ode45` не обеспечивает решения.
- `ode23s` - одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Может обеспечить высокую скорость вычислений при низкой точности решения жесткой системы дифференциальных уравнений.
- `ode23t` - метод трапеций с интерполяцией. Этот метод дает хорошие результаты при решении задач, описывающих колебательные системы с почти гармоническим выходным сигналом.

- `bvp4c` служит для решения краевых задач, т.е. решения систем дифференциальных уравнений вида $y' = f(t, y)$, при условии $F(y(a), y(b), p) = 0$.

- `pdepe` нужен для решения систем параболических и эллиптических дифференциальных уравнений в частных производных, введен в ядро системы для поддержки новых графических функций Open GL, пакет расширения Partial Differential Equations Toolbox содержит более мощные средства.

Все решатели могут решать системы уравнений явного вида

$$y' = F(t, y).$$

Кроме этого, решатели `ode15s,ode23t` способны найти корни дифференциально-алгебраических уравнений

$$M(t)y' = F(t, y),$$

где $M$ называется матрицей массы. Решатели `ode15s`, `ode23s`, `ode23t` и `ode23tb` могут решать уравнения неявного вида

$$M(t, y)y' = F(t, y).$$

И наконец, все решатели, за исключением `ode23s`, который требует постоянства матрицы массы, и `bvp4c`, могут находить корни матричного уравнения вида

$$M(t, y)y' - F(t, y).$$

Для решения жестких систем уравнений рекомендуется использовать только специальные решатели `ode15s`, `ode23s`, `ode23t`, `ode23tb`. Более подробно `ode23tb` и `ode23s` служат для решения жестких дифференциальных уравнений, `ode15s` – жестких дифференциальных и дифференциально-алгебраических уравнений, `ode23t` – умеренно жестких дифференциальных и дифференциально-алгебраических уравнений.

## 10.3 Использование решателей систем ОДУ - справка.

В описанных далее функциях для решения систем дифференциальных уравнений приняты следующие обозначения и правила:

- *options* - аргумент, создаваемый функцией *odeset* - задает самое главное - погрешность вычислений;
- *tspan* - вектор, определяющий интервал интегрирования $[t_0, t_{final}]$. Для получения решений в конкретные моменты времени $t_0, t_l, \ldots, t_{final}$ (расположенные в порядке уменьшения или увеличения) нужно использовать $tspan = [t_0, t_l, \ldots, t_{final}]$;
- $y_0$ - вектор начальных условий;
- $p_1, p_2, \ldots$ - произвольные параметры, передаваемые в функцию $F$;
- $T, Y$ - матрица решений $Y$, где каждая строка соответствует времени, возвращенном в векторе-столбце $T$;

Перейдем к описанию синтаксиса трех наиболее простых способов вызова функций для решения систем дифференциальных уравнений:

- `[T,Y] = solver(@F,tspan,y_0)` - где вместо `solver` подставляем имя конкретного решателя - интегрирует систему дифференциальных уравнений вида $y' = F(t, y)$ на интервале $tspan$ с начальными условиями $y0$. Здесь $@F$ - дескриптор ODE-функции (подпрограмма из $m$ файла). Каждая строка в массиве решений $Y$ соответствует значению времени, возвращаемому в векторе-столбце $T$;
- `[T,Y] = solver(@F, tspan, y_0, options)` - дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента `options`, созданного функцией `odeset`. Обычно используемые параметры включают допустимое значение относительной погрешности `RelTol` (по умолчанию $\mathbf{1e-3}$) и вектор допустимых значений абсолютной погрешности `AbsTol` (все компоненты по умолчанию равны $\mathbf{1e-6}$);

- `[T,Y]=solver(@F,tspan,y_0,options,p_1,p_2...)` - дает решение, подобное описанному выше, передавая дополнительные параметры $p_1, p_2, \ldots$ в m-файл $F$ всякий раз, когда он вызывается. Используйте $options = []$, если никакие параметры не задаются;

Параметры интегрирования (`options`) могут быть определены и в $m$-файле, и в командной строке с помощью команды `odeset`. Если параметр определен в обоих местах, определение в командной строке имеет приоритет. Кратко остановимся на особо важных параметрах

- `NormControl` - управление ошибкой в зависимости от нормы вектора решения, по умолчанию '$off$'. Установите '$on$', чтобы

$$||\epsilon|| <= \max(RelTol||y||, AbsTol).$$

  По умолчанию все решатели используют более жесткое управление по каждой из составляющих вектора решения;

- `RelTol` - относительный порог отбора - положительный скаляр. По умолчанию $10^{-3}$ и оценка ошибки на каждом шаге интеграции по правилу

$$\epsilon(i) <= \max(RelTol * |y(i|, AbsTol(i));$$

- `AbsTol` - абсолютная точность, по умолчанию положительный скаляр или вектор с компонентами $10^{-6}$ во всех решателях. Скаляр вводится для всех составляющих вектора решения, а вектор указывает на компоненты вектора решения.

- `Refine` - фактор уточнения вывода - положительное целое число - умножает число точек вывода на этот множитель. По умолчанию всегда равен 1, кроме `ode45`, где он равен 4. Не применяется, если `tspan>2`;

- `OutputFcn` - дескриптор функции вывода [`function`] - имеет значение в том случае, если решатель вызывается без явного указания функции вывода, `OutputFcn` по умолчанию вызывает функцию `odeplot`. В качестве альтернативы можно, например, установить свойство `OutputFcn` в

значение `odephas2` или `odephas3` для построения двумерных или трехмерных фазовых плоскостей;

- `Stats` - выводит статистику стоимости вычислений, по умолчанию равен $off$;

- `Mass` - матрица массы. Для задач $My' = f(t, y)$ установите имя постоянной матрицы. Для задач с переменной $M$ введите дескриптор функции, описывающей матрицу массы;

- `MstateDependence` - зависимость матрицы массы от $y$, может равняться $[none \mid weak \mid strong]$. Установите $none$ для уравнений $M(t)y' = F(t, y)$. И слабая ($weak$), и сильная ($strong$) зависимости означают что $M(t, y)$ зависит от $y$, но $weak$ приводит к неявным алгоритмам решения, использующим аппроксимации при решении алгебраических уравнений;

- `Initial Step` - предлагаемый начальный размер шага, по умолчанию каждый решатель определяет его автоматически по своему алгоритму;

- `MaxStep` - максимальный шаг, по умолчанию во всех решателях равен одной десятой интервала $tspan$;

Для того, чтобы узнать все параметры, их свойства и их допустимые значения можно просто набрать команду `odeset` без параметров.

## 10.4 Краткое сравнение возможностей различных решателей.

Рассмотрим в качестве примера гармонический осциллятор. Задача Коши состоит из дифференциального уравнения

$$\ddot{x}(t) = -x(t) \qquad t \in [0, 10\pi]$$

и начальных условий $x(0) = 1$, $\dot{x}(0) = 0$. Данное уравнение запишем и сохраним в файле `harmonic.m`

```
function ydot = harmonic(t,y)
```

```
ydot = [y(2); -y(1)];
```

Так как всем известное аналитическое решение данного уравнения является периодической функцией, то полная ошибка численного решения будет просто разность между начальным и конечным значением $|x(0)-x(10\pi)|$. Более того, решение ограничено и, поэтому, полная погрешность вычислений будет, как мы увидим, практически пропорционально локальной погрешности.

Следующий код позволяет вычислить полную погрешность, количество шагов и время вычислений для данной полной погрешности, возникающей при использовании решателя `ode23`:

```
function proba
y0 = [1 0];
for k = 1:5
 tol = 10^(-k);
 opts = odeset('reltol',tol,'abstol',tol,'refine',1);
 tic;
 [t,y] = ode23(@harmonic,[0 10*pi],y0',opts);
 time = toc;
 steps = length(t)-1;
 err = max(abs(y(end,:)-y0));
 X(k)=tol;  Y(k)=steps;  Z(k)=time;  W(k)=err;
end
shg
clf reset
%----------------- subplot(3,1,1); loglog(X,Y,'b*-')
xlabel('tol')
ylabel('Steps')
%----------------- subplot(3,1,2); loglog(X,Z,'m*-')
xlabel('tol')
ylabel('Time')
%----------------- subplot(3,1,3); loglog(X,W,'g*-')
xlabel('tol')
ylabel('Error')
```

**Задание 10.7** Модифицируйте программу таким образом, чтобы на всех трех графиках изображались локальная погрешность, количество шагов и время вычислений сразу для трех решателей `ode23`, `ode45`, и `ode113`. Объясните поведение графиков. Какой из трех решателей следует выбрать для данной задачи?

## 10.5 Жесткость.

Как можно заметить, что большинство из имен решателей в системе MATLAB встречаются дважды, с окончанием `s` и без этого окончания. Эти различные решатели используются для жестких (`stiff`) и не жестких систем. Предлагается много различных определений понятия жесткости, но ни одно из них нельзя признать полностью удовлетворительным. Например, в задачах химической кинетики для системы уравнений вида

$$\frac{dy_i}{dt} = f_i(y), \qquad i = 1, \ldots, n, \tag{10.1}$$

вводится матрица $F$ с элементами $F_{ij} = df_i/dy_j$. Если спектр матрицы $F$ имеет следующий характер: имеется несколько собственных значений с большими отрицательными вещественными частями, а остальная часть спектра состоит из собственных значений $|\lambda_i| \asymp 1$, то система уравнений (10.1) называется *жесткой*. Решение жестких задач Коши с заданным шагом $h$ требует как правило огромных затрат времени и памяти.

Существует много других формальных определения понятия жесткости, но ни одно из них не является общепринятым и полностью удовлетворительным. Смысл данного понятия мы поясним на примере уравнения

$$\dot{y} = y^2 - y^3, \qquad y(0) = \delta, \qquad t \in [0, 2/\delta]\,. \tag{10.2}$$

С физической точки зрения эта модель описывает поведение пламени в воздухе (шарика плазмы).

В зависимости от значения параметра $\delta$ эта задача Коши может быть и жесткой, и не жесткой. Так, если параметр $\delta$ не очень мал, то задача Коши не очень жесткая - для понимания этого факта посмотрим на график, который решатель `ode45` построит автоматически при $\delta = .01$ и максимальном

значении локальной ошибки $RelTol = 10^{-4}$.

```
delta = 0.01;
f = inline('y^2 - y^3','t','y');
opts = odeset('RelTol',1.e-4);
ode45(f,[0 2/delta],delta,opts);
```

До момента времени $1/\delta$, который в данном случае равен 100, функция растет слабо, затем резко возрастает и далее почти не меняется, так как наступает фаза насыщения.

При уменьшении параметра $\delta$ жесткость возрастает, что видно на следующем графике

```
delta = 0.0001;
ode45(f,[0 2/delta],delta,opts);
```

Время работы данной программы значительно, поэтому эту работу можно остановить, нажав на появившуюся снизу кнопку `Stop`.

Далее, используя увеличительное стекло `Zoom` рассмотрите как ведет себя функция на "гладком" участке при $t > 1/\delta$. Убедитесь, что решатель `ode45` борясь с жесткостью задачи вынужден уменьшить размер шага для того, чтобы сохранить локальную ошибку в пределах заданной величины $10^{-4}$.

Почти вертикальный участок траектории и его малая окрестность, где траектория резко меняет направление, называются релаксационным участком, на котором интегрирование лучше проводить с малым шагом. На участках траектории, которые не являются релаксационными интегрирование можно проводить с большим шагом. Именно эта стратегия решения и реализована в решателях, имена которых заканчиваются на `s`.

Для примера возьмем теперь более "слабый" решатель `ode23`, но с окончанием `s`, то есть решатель `ode23s` специально предназначенный для работы с жесткими системами:

```
delta = 0.0001;
ode23s(f,[0 2/delta],delta,opts);
```

Используя увеличительное стекло `Zoom` рассмотрите как ведет себя функция на "гладком" участке при $t > 1/\delta$. Даже на глаз видно, что качество численного решения практически не ухудшилось, а время работы значительно сократилось.

Фактически, для решения поставленной задачи решателю `ode23s` потре-

бовалось 99 шагов и 412 вычислений значения искомой функции, в то время как на первый взгляд более мощному решателю `ode45` требуется 3040 шагов и 20179 вычислений функции.

Заметим, что данная модель горения пламени может быть описана и аналитически с помощью функции Ламберта. Действительно, разделяя переменные и интегрируя уравнение (10.2) получим

$$\frac{1}{y} + \log\left(\frac{1}{y} - 1\right) = \frac{1}{\delta} + \log\left(\frac{1}{\delta} - 1\right) - t\,.$$

решая это уравнение относительно $y$ получим

$$y(t) = \frac{1}{W(ae^{a-t}) + 1}, \qquad a = \frac{1}{\delta} - 1, \tag{10.3}$$

где $W$ - функция Ламберта. Конечно мы можем получить это решения явно с помощью системы MATLAB и построить его график, хотя это и потребует некоторого машинного времени:

```
y = dsolve('Dy = y^2 - y^3','y(0) = 1/100');
y = simplify(y);
pretty(y)
ezplot(y,0,200)
```

Функция Ламберта $W$ названа в честь J. H. Lambert (1728-1777), который был коллегой Эйлера и Лагранжа по Берлинской Академии наук и который наиболее известен своим доказательством того, что число $\pi$ иррационально.

**Задание 10.8** Используя явное решение (10.3) попробуйте построить матрицу $F$ и найти ее собственные значения.

**Задание 10.9** Используя графические возможности системы MATLAB представьте и исследуйте процесс горения пламени с помощью программы `flame` при $\delta = .01$:

```
function flame
shg
set(gcf,'double','on','color','black')
set(gca,'color','black')
m = 30;
```

```
[rho,theta] = ndgrid((0:m)'/m, pi*(-m:m)/m);
x = rho.*sin(theta);
y = rho.*cos(theta);
r0 = .02;
t = (0:.002:2)/r0;
[t,r] = ode23s(inline('r.^2-r.^3','t','r'),t,r0,odeset('reltol',1.e-5));
z = 1-rho/r0;
p = pcolor(x,y,z);
shading interp
colormap(hot)
axis square
axis off
s = title(sprintf('t = %8.2f   r = %10.6f',0,r0),'fontsize',16);
set(s,'color','white');
for k = 1:length(t);
   z = max(0,1-rho/r(k));
   set(p,'cdata',z)
   set(s,'string',sprintf('t = %8.1f   r = %10.6f',t(k),r(k)));
   drawnow
end
uicontrol('string','close','callback','close(gcf)')
```

Измените значение параметра $\delta = r_0$ и исследуйте жесткий и не жесткий режимы.

## 10.6 Уравнение Ван-дер-Поля.

Покажем применение решателя ОДУ на ставшем классическом примере - решении уравнения Ван-дер-Поля, записанного в виде системы из двух дифференциальных уравнений первого порядка:

$$y_1' = y_2, \qquad y_2' = \varepsilon(1 - y_1)^2 y_2 - y_1, \qquad \varepsilon = 100,$$

при начальных условиях $y_1(0) = 0$ и $y_2(0) = 1$. Перед решением нужно записать систему дифференциальных уравнений в виде ode-функции. Для этого в главном меню выберем File → New → M-File и введем

```
function dydt = vdp100(t.y)
dydt= zeros(2.1); - начальные условия
dydt(l) = y(2);
dydt(2)=100*(1 -y(1))^2*y(2)-y(1);
```

Сохраним $m$-файл-функцию. Тогда решение решателем `ode15s` и сопровождающий его график можно получить, используя следующие команды:

```
[T,Y]=ode15s(@vdp100.[030].[20]);
plot(T.Y)
holdon:gtext('yl').gtext('y2')
```

Последние команды позволяют с помощью мыши нанести на графики решений $y_1 = y(1)$ и $y_2 = y(2)$ помечающие их надписи.

**Задание 10.10** Определите при каких значениях параметра $\varepsilon$ система уравнений Ван-дер-Поля будет жесткой и не жесткой.

## 10.7 Краевая задача

Рассмотрим теперь краевую задачу

$$y'''|y| = 0, \qquad y(0) = 0, \qquad y(4) = -2.$$

Для решения в пределах отрезка $[0, 4]$ с помощью решателя `bvp4c` достаточно привести эту систему к виду:

$$y' = -|y|, \qquad y(0) = 0, \qquad y(4) + 2 = 0.$$

Итак создаем две `ode`-функции: `twoode`

```
function dydx = twoode(x,y)
dydx = [ y(2) -abs(yd))]
```

и `twobc` в разных $m$-файлах:

```
function res = twobc(ya.yb)
res= [ ya(l) yb(l) + 2]
```

Теперь наберите в командной строке `type twobvp` и посмотрите само решение

уравнения, которое содержится в уже имеющемся в системе файле `twobvp`. А исполнив команду `twodvp`, можно наблюдать результат решения в виде графиков.

Рекомендуется просмотреть также дополнительные примеры решения систем дифференциальных уравнений, приведенные в файле `odedemo` системы MATLAB. Во многих случаях решение задач, сводящихся к решению систем дифференциальных уравнений, удобнее осуществлять с помощью пакета `Simulink`, входящего в дистрибутив системы MATLAB.

# Работа № 11

## Решение дифференциальных уравнений II.

Численные методы и методики проведения компьютерного моделирования и расчетов, реализованные в различных компьютерных системах, являются только инструментами, которые, как и в экспериментальной физике, имеет вполне определенную область применимости. Как для экспериментов, так и для расчетов нам необходимо, во-первых, выбирать и использовать правильный инструмент, обладающий достаточной точностью, и, во-вторых, проверять и обосновывать полученные результаты.

Напомним, что для решения систем дифференциальных уравнений существует несколько встроенных процедур, описание которых можно найти в предыдущей работе. Для примера напомним применение процедуры `ode45`, которая может решить систему уравнений следующего вида:

$$\frac{d}{dt}F(t) = G(t, x_1, x_2, \ldots, x_n)$$

где $F(t)$ - есть векторная функция $(\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n)$.

Один из возможных форматов вызова этой процедуры

`[t,r]=ode45(@Function,[Tstart,Tfinish], StartVector)`.

Здесь `Function` - вектор-функция $G$, `[Tstart,Tfinish]` - интервал, на котором ищется решение данной задачи Коши, `StartVector` - необходимые для решения задачи Коши начальные условия в точке `Tstart`.

В рассматриваемых ниже задачах необходимо доказать, что полученные результаты соответствуют поставленной физической задаче. Тем самым основные параметры - допустимое значение относительной погрешности `RelTol` (по умолчанию $\mathbf{1e-3}$) и вектор допустимых значений абсолютной погрешности `AbsTol` (все компоненты по умолчанию равны $\mathbf{1e-6}$).

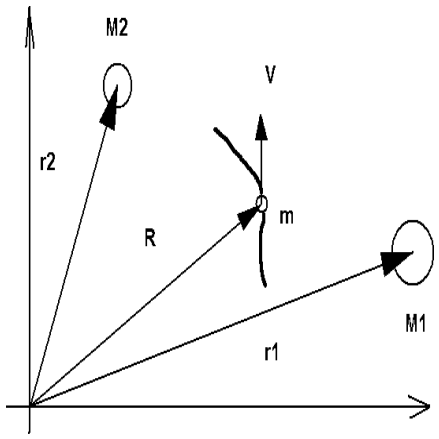
Изменяя эти параметры и выбирая различные решатели необходимо добиться оптимального решения поставленной задачи

- выбрать метод математического моделирования и свести физическую задачу к математической;
- решить численно поставленную математическую задачу "правильно", используя допустимый уровень погрешности который следует из физической задачи;
- затратить при этом минимум ресурсов - времени и памяти компьютера;
- оптимально использовать возможности компьютерного моделирования для представления полученных результатов - графики, анимация.
- обосновать выбор стратегии решения, по возможности "проверив" решение и оценив затраты на решение.

## 11.1 Задача трех тел

Рассмотрим пример, иллюстрирующий создание исходной функции пользователя `Function` для дальнейшего ее вызова встроенной процедурой `ode45`.

Пусть некоторая точка массы $m$ движется в гравитационном поле неподвижных точек с массами $M_1$ и $M_2$ см. рисунок




Уравнение Ньютона, описывающее движение в гравитационном поле точек $M_1$ и $M_2$, будет следующим:

$$m\ddot{\vec{R}} = -\gamma\frac{mM_1}{|\vec{R}-\vec{r}_1|^3}(\vec{R}-\vec{r}_1) - \gamma\frac{mM_2}{|\vec{R}-\vec{r}_2|^3}(\vec{R}-\vec{r}_2)$$

Как видим, данное дифференциальное уравнение имеет второй порядок. Но его можно свести к системе дифференциальных уравнений первого порядка:

$$\begin{cases} \dot{\vec{R}} = \vec{V} \\ \dot{\vec{V}} = -\gamma\frac{M_1}{|\vec{R}-\vec{r}_1|^3}(\vec{R}-\vec{r}_1) - \gamma\frac{M_2}{|\vec{R}-\vec{r}_2|^3}(\vec{R}-\vec{r}_2) \end{cases}$$

Будем решать эту систему уравнений на плоскости, т.е. в двумерном мире. Введем следующие обозначения: $\vec{R} = (x_1, x_2)$, $\vec{r}_1 = (C_x^1, C_y^1)$, $\vec{r}_2 = (C_x^2, C_y^2)$ и $\vec{V} = (x_3, x_4)$. В этих обозначениях систему дифференциальных уравнений движения точки в гравитационном поле можно представить следующим

образом:

$$\begin{cases} \dot{x}_1 = x_3 \\ \dot{x}_2 = x_4 \\ \dot{x}_3 = -\frac{M_1(x_1-C_x^1)}{\left((x_1-C_x^1)^2+(x_2-C_y^1)^2\right)^{3/2}} - \frac{M_2(x_2-C_x^2)}{\left((x_1-C_x^2)^2+(x_2-C_y^2)^2\right)^{3/2}} \\ \dot{x}_4 = -\frac{M_1(x_1-C_y^1)}{\left((x_1-C_x^1)^2+(x_2-C_y^1)^2\right)^{3/2}} - \frac{M_2(x_2-C_y^2)}{\left((x_1-C_x^2)^2+(x_2-C_y^2)^2\right)^{3/2}} \end{cases}$$

Положим значение гравитационной постоянной равной 1, и зададим значения остальных параметров следующим образом $M_1 = 50$, $M_2 = 0$, $C^1 = (5, 0)$, $C^2 = (0, 10)$.

Данную систему уравнений запишем как MATLAB-овскую файл-функцию по имени `threebody(t,x)`.

```
function f=threebody(t,x)
   M1=50; M2=0; C1x=5; C1y=0; C2x=0; C2y=10;
   f=[x(3);x(4);...
    -M1*(x(1)-C1x)/(sqrt((x(1)-C1x)^2+(x(2)-C1y)^2))^3-...
     M2*(x(1)-C2x)/(sqrt((x(1)-C2x)^2+(x(2)-C2y)^2))^3;...
    -M1*(x(2)-C1y)/(sqrt((x(1)-C1x)^2+(x(2)-C1y)^2))^3-...
     M2*(x(2)-C2y)/(sqrt((x(1)-C2x)^2+(x(2)-C2y)^2))^3];
```

Обязательно сохраните этот m-файл!

Для решения этой системы дифференциальных уравнений используем процедуру `ode45` , а для вывода результата напишем файл-функцию `orbit.m`:

```
function orbit()
[t,h]=ode45(@threebody,[0,1000],[0,0,0,4.3]);
x=h(:,1);  y=h(:,2);
x1=5; y1=0; x2=0; y2=100;
plot(x,y,'b-',x1,y1,'r+',x2,y2,'r*');
```

При таких начальных параметрах ($M_2 = 0$) наша точка движется в поле одного объекта. Для построения графика (орбиты движения) надо вызвать процедуру `orbit`. Крестиком и звездочкой на графике отмечены положения масс $M_1$ и $M_2$.

**Задание 11.1** Изучить, как меняется характер движения при увеличении массы второго тела и при изменении начальных условий движения. Если MATLAB

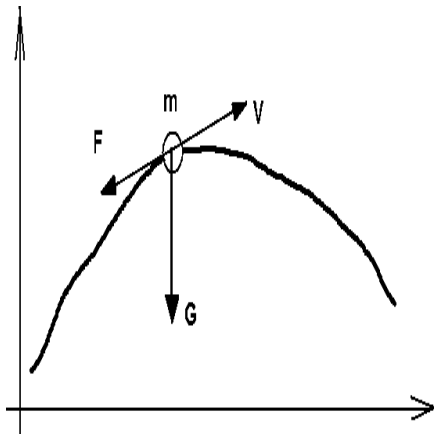
будет жаловаться на большую погрешность интегрирования, то замените процедуру ode45 на более точную.

**Задание 11.2** Почему даже в задаче двух тел траектория не является эллипсом? Как получить эллипс?

**Задание 11.3** Найдите в демо разделе системы MATLAB программу, описывающую ограниченную задачу трех тел. Сравните полученные вами результаты с результатами демо-программы.

## 11.2 Движение тела под действием силы тяжести.

Рассмотрим траекторию движения пули под действием силы тяжести (см. рисунок). При отсутствии сопротивления воздуха, как известно из курса средней школы, это будет парабола. Мы же рассмотрим случай, когда сила сопротивления воздуха пропорциональна квадрату скорости и противоположна направлению движения.




Как говорит школьный курс физики, уравнение баланса сил будет следующим: $m\vec{a} = \vec{G} + \vec{F}$. Учитывая то, что ускорение - производная скорости по времени, распишем это уравнение в векторном виде:

$$m\dot{\vec{V}} = m\vec{g} - \frac{\rho|\vec{V}|^2}{2} S \frac{\vec{V}}{|\vec{V}|}.$$

Здесь $\rho$ - плотность воздуха, $m$ - масса пули, $S$ - площадь поперечного сечения. Распишем это уравнение покоординатно:

$$\begin{cases} V_x = -\frac{\rho\sqrt{V_x^2+V_y^2}}{2m}SV_x \\ V_y = -g - \frac{\rho\sqrt{V_x^2+V_y^2}}{2m}SV_y \end{cases}$$

В таком виде система дифференциальных уравнений готова для того, чтобы попытаться решить ее при помощи процедуры `ode45`. Пусть масса пули - 10 грамм, поперечник - 1 см, плотность воздуха - 1 килограмм на кубический метр. Заметим, что на Земле такой плотности воздуха добиться довольно трудно. Мы выбрали такие данные просто для упрощения набора.

С такими данными мы составим файл-функцию `airsys2.m`.

```
function u=airsys2(t,v)
   g=10; ro=1; s=0.0001; m=0.01; k=ro*s/2/m;
   u=[-k*sqrt(v(1)^2+v(2)^2)*v(1);
     -g-k*sqrt(v(1)^2+v(2)^2)*v(2)];
```

В этой системе уравнений аргументом являются скорость и время. Траектория движения определяется стандартным образом

$$\begin{cases} x(t) = x_0 + \int\limits_{t_0}^{t} V_x(\tau)d\tau \\ y(t) = y_0 + \int\limits_{t_0}^{t} V_y(\tau)d\tau \end{cases}$$

Для краткости мы интегрирование заменим суммированием считая, что в начальный момент времени $t = 0$, пуля находилась в начале координат.:

$$\begin{cases} x_i = \sum\limits_{k=2}^{i} V_x^k(t_k - t_{k-1}) \\ y_i = \sum\limits_{k=2}^{i} V_y^k(t_k - t_{k-1}) \end{cases}$$

Помимо процедуры `ode45` существует еще ряд встроенных процедур решения систем дифференциальных уравнений. Для сравнения мы создадим файл-функцию `pulya`, в которой для решения данной системы уравнений мы используем процедуру `ode45` и более точную процедуру `ode113`, в которой используется метод Адамса - т.е. метод "прогноз-коррекции". Графики будут соответственно синего и красного цвета.

Пусть в начальный момент времени скорость пули по горизонтали - 800, по вертикали - 100.

```
function pulya()
   [t,h]=ode45(@airsys2,[0,5.6],[800,100]);
   vx=h(:,1); vy=h(:,2);
   m=length(t);
   x0=0; y0=0;
   x(1)=x0; y(1)=0;
   for i=2:m
      x(i)=x(i-1)+vx(i-1)*(t(i)-t(i-1));
      y(i)=y(i-1)+vy(i-1)*(t(i)-t(i-1));
   end;
   [ t1,h1]=ode113(@airsys2,[0,5.6],[800,100]);
    vx1=h1(:,1); vy1=h1(:,2);
    m1=length(t1);
    x01=0; y01=0;
    x1(1)=x0; y1(1)=0;
    for i=2:m1
        x1(i)=x1(i-1)+vx1(i-1)*(t1(i)-t1(i-1));
        y1(i)=y1(i-1)+vy1(i-1)*(t1(i)-t1(i-1));
    end;
    figure
    plot(x,y,'r-',x1,y1,'b-')
    grid on
```

Разница между результатами применения различных численных методов видна невооруженным глазом. Погрешность вычисления накапливается с ростом времени $t$.

Заметим однако, что мы решали систему дифференциальных уравнений

$$\begin{cases} V_x = -\frac{\rho\sqrt{V_x^2+V_y^2}}{2m}SV_x \\ V_y = -g - \frac{\rho\sqrt{V_x^2+V_y^2}}{2m}SV_y \end{cases}$$

И только затем находили траекторию пули, используя приближенное инте-

грирование. Вместо этого мы можем решать сразу всю систему уравнений

$$\begin{cases} \dot{x} = V_x \\ \dot{y} = V_y \\ \dot{V}_x = -\frac{\rho\sqrt{V_x^2+V_y^2}}{2m} S V_x \\ \dot{V}_y = -g - \frac{\rho\sqrt{V_x^2+V_y^2}}{2m} S V_y \end{cases}$$

Произведем небольшие изменения и напишем файл-функцию `airsys4.m`.

```
function u=airsys4(t,x)
  g=10; ro=1; s=0.0001; m=0.01; k=ro*s/2/m;
  u=[x(3);x(4);...
       -k*sqrt(x(3)^2+x(4)^2)*x(3);...
     -g-k*sqrt(x(3)^2+x(4)^2)*x(4)];
```

Для решения системы четырех дифференциальных уравнений, которая записана файл-функцией `airsys4.m` создадим файл-функцию `pulya2.m`.

```
function pulya2()
  [t,h]=ode45(@airsys4,[0,5.6],[0,0,800,100]);
  x=h(:,1); y=h(:,2);
  [t1,h1]=ode113(@airsys4,[0,5.6],[0,0,800,100]);
  x1=h1(:,1); y1=h1(:,2);
  figure
  plot(x,y,'r-',x1,y1,'b-')
  grid on
```

Видно, что в этом случае практически нет различий между решениями с помощью встроенных процедур `ode45` и `ode113`.

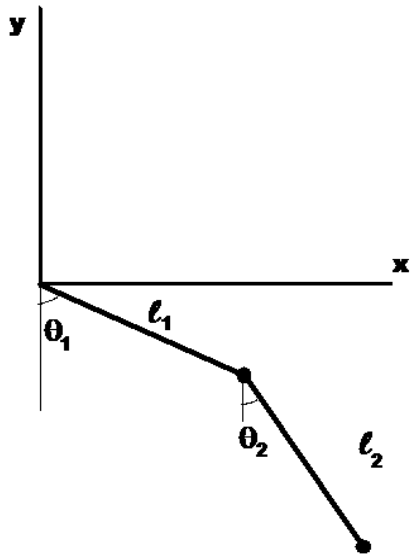
Следует отметить, что диапазон времени $[0;5.6]$ был эмпирическим путем подобран так, что траектория движения отображена от момента вылета до падения. Уровень "земли" соответствует значению ординаты 0.

**Задание 11.4** Исследуйте зависимость дальности полета пули в зависимости от массы пули и ее начальной горизонтальной и вертикальной скорости. Ответ представляет собой три графика.

## 11.3 Двойной маятник

В данном разделе мы покажем, как можно оформить полученное численное решение системы дифференциальных уравнений с помощью графических возможностей MATLAB.

Рассмотрим движение двух связанных друг с другом двумерных маятников. Массы маятников $m_1, m_2$, длины нитей $\ell_1, \ell_2$ - смотри рисунок:




Координаты маятников $(x_1, y_1)$ и $(x_2, y_2)$ выражаются через два угла $\theta_1, \theta_2$

$$x_1 = \ell_1 \sin\theta_1; \qquad y_1 = -\ell_1 \cos\theta_1;$$

$$x_2 = \ell_1 \sin\theta_1 + \ell_2 \sin\theta_2; \qquad y_2 = -\ell_1 \cos\theta_1 - \ell_2 \cos\theta_2$$

Соответствующие уравнения движения имеют вид

$$(m_1 + m_2)\ell_1\ddot{\theta}_1 + m_2\ell_2\ddot{\theta}_2 \cos(\theta_1 - \theta_2) = \; -g(m_1 + m_2)\sin\theta_1 - m_2\ell_2\dot{\theta}_2^2 \sin(\theta_1 - \theta_2)$$

$$m_2\ell_1\ddot{\theta}_1 \cos(\theta_1 - \theta_2) + m_2\ell_2\ddot{\theta}_2 = -gm_2 \sin\theta_2 + m_2\ell_1\dot{\theta}_2^2 \sin(\theta_1 - \theta_2) \qquad (11.1)$$

Положим $m_1 = m_2 = 1$, $\ell_1 = \ell_2 = 1$ и перепишем эту систему из 2 уравнений второго порядка в виде систему из 4 уравнений первого порядка. Если ввести обозначения

$$u(t) = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T, \qquad c = \cos(\theta_1 - \theta_2, \qquad s = \sin(\theta_1 - \theta_2),$$

то искомая система уравнений имеет вид

$$M\dot{u} = f, \qquad \text{где} \qquad M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & c \\ 0 & 0 & c & 1 \end{pmatrix}, \quad f = \begin{pmatrix} u_3 \\ u_4 \\ -g\sin u_1 - su_3^2 \\ -g\sin u_2 + su_4^2 \end{pmatrix}.$$

Матрицу $M$ иногда называют матрицей масс, а вектор $f$ - силовой функцией.

В нашем случае матрица масс не является постоянной матрицей, поэтому для решения такой системы уравнений необходимо использовать соответствующие решатели. Далее следует текст программы, которая позволяет решить систему уравнений (11.1) с использованием графических средств системы MATLAB:

```
function doss(x,y)
theta0 = dossinit(x,y);
set(gcf,'userdata',theta0)
delete(findobj('string','close'))
if isempty(findobj('string','stop'))
   uicontrol('style','toggle','string','stop','value',0, ...
      'units','normalized','position',[.02 .02 .09 .05]);
end
u0 = [theta0; 0; 0];
tspan = [0 1.e6];
opts = odeset('mass',@dossmass,'outputfcn',@dossplot,'maxstep',0.05);
ode23(@dossrhs,tspan,u0,opts);
% ------------------------
function M = dossmass(t,u)
c = cos(u(1)-u(2));
M = [1 0 0 0; 0 1 0 0; 0 0 2 c; 0 0 c 1];
% ------------------------
function f = dossrhs(t,u)
g = 1;
s = sin(u(1)-u(2));
f = [u(3); u(4); -2*g*sin(u(1))-s*u(4)^2; -g*sin(u(2))+s*u(3)^2];
% ------------------------
function theta = dossinit(x,y)
```

```
r = norm([x,y]);
if r > 2
   alpha = 0;
else
   alpha = acos(r/2);
end
beta = atan2(y,x) + pi/2;
theta = [beta+alpha; beta-alpha];
% ------------------------
function status = dossplot(t,u,task)
switch task
   case 'init'
      theta = u(1:2);
      x = cumsum(sin(theta));
      y = cumsum(-cos(theta));
      orbit = 0;
      if orbit
         h = plot([x(2) x(2)],[y(2) y(2)],'-','erasemode','none');
      else
         h = plot([0; x],[0; y],'o-');
      end
      set(h,'userdata',orbit);
      axis(2.25*[-1 1 -1 1])
      axis square
      ttl = title(sprintf('t = %8.1f',t));
      set(gca,'userdata',ttl)
      xlabel('Click to reinitialize');
   case ''
      h = get(gca,'child');
      theta = u(1:2);
      x = cumsum(sin(theta));
      y = cumsum(-cos(theta));
      orbit = get(h,'userdata');
      if orbit
```

```
        xo = get(h,'xdata');
        yo = get(h,'ydata');
        set(h,'xdata',[xo(2) x(2)],'ydata',[yo(2) y(2)])
     else
        set(h,'xdata',[0; x],'ydata',[0; y])
     end
     ttl = get(gca,'userdata');
     set(ttl,'string',sprintf('t = %8.1f',t))
     drawnow
     stop = findobj('string','stop');
     status = isempty(stop) || get(stop,'value');
  case 'done'
     set(findobj('string','stop'),'style','push', ...
        'string','close','callback','close(gcf)');
     delete(findobj('string','orbit'));
end
```

Данную программу лучше просто скопировать из соответствующего pdf файла и сохранить в m-файле `doss.m`. В некоторых операционных системах после этого потребуется символ ‘ красного цвета заменить на символ ’ фиолетового цвета. Для запуска программы наберите командную строку `doss(x,y)` - где $x, y$ начальные координаты.

**Задание 11.5** Измените программу так, чтобы можно было решать уравнения (11.1) при произвольных $m_{1,2}$ и $l_{1,2}$. Подберите длины нитей и массы маятников таким образом, чтобы наблюдать явление резонанса.

## 11.4 Решение задачи Коши на больших отрезках

Условимся независимую переменную отождествлять со временем $t$.

Выбор метода решения дифференциального уравнения требует некоторого компромисса между учетом локальной ошибки усечения, устойчивостью и временем расчета. Все эти характеристики зависят длины отрезка интегрирования, так как они либо прямо либо обратно пропорциональны величине

шага $\Delta t$ в некоторой степени, а $\Delta t$ грубо говоря равна длине отрезка интегрирования деленной на число шагов.

Когда длина отрезка велика, то и локальная и полная ошибка любого метода интегрирования велики, устойчивость мала, а время расчетов велико. Поэтому встает вопрос - возможно ли вообще решать задачу Коши на большом интервале и как проверить полученное решение?

Ответ на второй вопрос достаточно прост. Задача решается на интервале $[0, T]$, затем время обращается и задача решается в обратном направлении. Если при этом в момент $t = T$ было получено "плохое" решение с большой погрешностью, то, возвратившись в начальный момент $t = 0$, мы получим большое отличие от данного нам начального значения.

В ряде задач начальное значение воспроизводится с достаточно большой точностью. Как правило это объясняется свойствами самой системы уравнений, не методом дискретизации и интегрирования. Однако, если у дифференциального уравнения траектории имеют тенденцию сильно расходиться, то интегрирование таких систем на больших отрезках невозможно. Для таких систем, даже если мы будем интегрировать их с очень маленьким шагом, ошибки округления, возникающие на каждом шаге, в конце интервала приведут к очень большой погрешности в решении.

Классическим примером системы уравнений, все решения которой ограничены, но обладают экспоненциальной неустойчивостью, служит система уравнений описывающая движение по геодезическим на компактной поверхности постоянной кривизны.

Существует класс задач для которых численное интегрирование на больших отрезках осмысленно и приносит большую пользу при исследовании этих задач. Рассмотрим систему дифференциальных уравнений в пространстве $\mathbf{R}^n$

$$\frac{dx(t)}{dt} = f(x), \qquad x = (x_1, x_2, \ldots, x_n)\,.$$

Определим инвариантное множество $M$, которое состоит из начальных значений $x_0 \in M$ таких, что $x(t, x_0) \subset M$ - то есть в любой момент времени траектория, выпущенная из любой точки $x_0$ множества $M$ остается внутри множества $M$.

Простейшие инвариантные множества состоят из неподвижных точек и периодических решений. Если инвариантное множество притягивающее, то

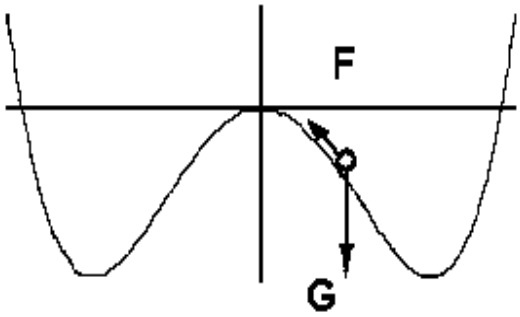
его называют *аттрактором*. Если система уравнений имеет *аттрактор* и ее траектории не уходят на бесконечность, то решение задачи Коши с начальными данными $x_0, x_0'$ будет себя вести так, что точка $x(t, x_0)$ будет стремиться к аттрактору при $t \to \infty$. Таким образом, при численном решении задачи Коши на большом интервале мы будем получать информацию об аттракторе, несмотря на то, что ошибка в расчете конкретной траектории будет по прежнему велика. Таким образом, на основании численных расчетов можно составить представление о структуре аттракторов и о качественном поведении модели. Появлению аттрактора предшествует целый ряд бифуркаций в системе или, грубо говоря, колебаний.

**Пример 1.**

Рассмотрим движение металлического шарика по поверхности

$$U(t) = \frac{x^4}{4} + \frac{(q(t)-1)}{2}x^2,$$

форма которой зависти от параметра $q(t)$.




На шарик действует сила тяжести $G$ и сила трения

$$F = -\alpha\frac{dx(t)}{dt}$$

с показателем трения $\alpha$. Мы будем исследовать характер движения в зависимости от величины силы трения. Выберем числовые значения остальных параметров в задаче таким образом, чтобы подчеркнуть особенности этой

зависимости, при этом уравнения движения имеют вид

$$\frac{d^2}{dt}x(t) + 0.7\alpha\frac{d}{dt}x(t) + x(t)^3 + \Big(q(t) - 1\Big)\,x(t) = 0\,.$$

$$\frac{d}{dt}\,q(t) = \alpha\Big(-0.16q(t) + x(t)^2\Big)\,.$$

Мы специально подобрали параметры системы так, чтобы изучить стохастическое поведение системы. Эти уравнения необходимо переписать в виде систему уравнений первого порядка и занести их в m-файл:

```
  function u=sys3(t,x)
alpha=1;
u=[x(2);-.7*alpha*x(2)-x(1)^3-(x(3)-1)*x(1);
alpha*(-.16*x(3)+x(1)^2)];
```

После этого мы можем исследовать характер движения, используя график, на котором отложена параметрическая зависимость координаты x(t) и скорости v(t) от времени

```
  function attr()
[t,h]=ode113(@sys3,[0,300],[-0.3,-0.1,0.5]);
x=h(:,1); y=h(:,2); z=h(:,3)
plot3(x,y,z);
```

Если вы хотите увидеть движение по этой траектории, то замените команду `plot3(x,y,z)` на команду `comet3(x,y,z,0.01)`.

Итак, мы получили график траектории в фазовом пространстве. Мы видим, что решение задачи Коши обнаруживает сложное поведение, которое приближается по своим свойствам к стохастическому (так называемый странный аттрактор). Отчетливо видно разделение траектории на две ветви, отвечающие значениям q(t)<1 и q(t)>1.

**Задание 11.6** Измените силу трения $\alpha = 0.1$ и $\alpha = 10$. На основании изменений в форме аттрактора опишите изменения в качественном поведении системы при уменьшении и увеличении силы трения.

Теперь посмотрим, как изменяется со временем потенциальная яма U(t). Для этого надо построить трехмерный график

```
function attr()
[t,h]=ode113(@sys3,[0,50],[-0.3,-0.1,0.5]);
surf(x,y);
```

**Задание 11.7** Покрутите график с помощью инструмента Camera. Измените силу трения $\alpha = 0.1$ и $\alpha = 10$. На основании изменений в данном графике опишите изменения в качественном поведении системы при уменьшении и увеличении силы трения.

**Пример 2**

Рассмотрим систему уравнений Лоренца в пространстве $\mathbf{R}^3$

$$\frac{dx_1}{dt} = -\beta x_1 + x2x_3, \qquad \frac{dx_2}{dt} = -\sigma x_2 + \sigma x_3, \qquad \frac{dx_3}{dt} = -x_1 x_2 + \rho x_2 - x_1.$$

Первая компонента $x_1(t)$ описывает конвекцию атмосферного потока, вторая и третья описывают горизонтальную и вертикальную составляющие температурного распределения. Величина $\sigma$ - число Прандтля, $\rho$ - нормированная постоянная Релея, а $\beta$ зависит от геометрии области.

Решение данной системы уравнений рассмотрено в демо примерах системы MATLAB. Мы приведем текст подобной процедуры явно - вы можете скопировать эту процедуру из pdf файла.В некоторых операционных системах после этого дополнительно потребуется символ ‘ красного цвета заменить на символ ’ фиолетового цвета.

Начнем с того, что перепишем уравнения в матричном виде

$$\dot{x}(t) = Ax, \qquad A = \begin{pmatrix} -\beta & 0 & x_2 \\ 0 & -\sigma & \sigma \\ -x_2 & \rho & -1 \end{pmatrix}$$

Если $\eta = y_2(t)$ то эта матрица сингулярна при $\eta = \pm\sqrt{\beta(\rho - 1)}$. Соответствующие собственные вектора задают две стационарных точки. Т.е. если начальное значение равно

$$x_0 = \begin{pmatrix} \rho - 1 \\ \eta \\ \eta \end{pmatrix}, \qquad \text{то} \qquad x(t, x_0) = 0, \quad \forall t\,.$$

Это нестабильные точки, которые принадлежат аттрактору. Для того, чтобы построить весь аттрактор мы будем выбирать начальные приближения в окрестности этих стационарных точек.

Итак, положим $\beta = 8/3$, $\sigma = 10$ и исследуем поведение системы при различных $\rho$ с помощью программы `lorenzMy`:

```
   function lorenzMy
if isequal(get(gcf,'name'),'Lorenz')
   rhos = [28 99.65 100.5 160 350];
   shg
   clf reset
   p = get(gcf,'pos');
   set(gcf,'color','black','doublebuff','on','name','Lorenz', ...
      'menu','none','numbertitle','off', ...
      'pos',[p(1) p(2)-(p(3)-p(4))/2 p(3) p(3)])
   klear=['set(gcf,''pos'',get(gcf,''pos'')+[0 0 0 1]),drawnow,'...
         'set(gcf,''pos'',get(gcf,''pos'')-[0 0 0 1]),drawnow'];
   paws = uicontrol('style','toggle','string','start', ...
      'units','norm','pos',[.02 .02 .10 .04],'value',0, ...
      'callback','lorenzMy');
   stop = uicontrol('style','toggle','string','close', ...
      'units','norm','pos',[.14 .02 .10 .04],'value',0, ...
      'callback','cameratoolbar(''close''), close(gcf)');
   clear = uicontrol('style','push','string','clear', ...
      'units','norm','pos',[.26 .02 .10 .04], ...
      'callback',klear);
   rhostr = sprintf('%6.2f|',rhos);
   rhopick = uicontrol('style','listbox','tag','rhopick', ...
      'units','norm','pos',[.82 .02 .14 .14], ...
      'string',rhostr(1:end-1),'userdata',rhos,'value',1);
else
   rhopick = findobj('tag','rhopick');
   rhos = get(rhopick,'userdata');
   rho = rhos(get(rhopick,'value'));
   sigma = 10;
```

```
   beta = 8/3;
   eta = sqrt(beta*(rho-1));
   A = [ -beta    0     eta
            0  -sigma   sigma
         -eta   rho    -1  ];
   xc = [rho-1; eta; eta];
   x0 = xc + [0; 0; 3];
   tspan = [0 Inf];
   opts=odeset('reltol',1.e-6,'outputfcn',@lorenzplot,'refine',4);
   ode45(@lorenzeqn, tspan, x0, opts, A);
end
```

Уравнения и графическую подпрограмму можно сохранить в том же m-файле:

```
function xdot = lorenzeqn(t,x,A)
A(1,3) = x(2);
A(3,1) = -x(2);
xdot = A*x;
 %------------------------
function fin = lorenzplot(t,x,job,A)
persistent X
if isequal(job,'init')
   rho = A(3,2);
   switch rho
      case 28,    R = [  5  45  -20  20  -25  25];  L = 100;
      case 99.65, R = [ 50 150  -35  35  -60  60];  L = 240;
      case 100.5, R = [ 50 150  -35  35  -60  60];  L = 120;
      case 160,   R = [100 220  -40  40  -75  75];  L = 165;
      case 350,   R = [285 435  -55  55 -105 105];  L =  80;
      otherwise,  R = [100 250  -50  50 -100 100];  L = 150;
   end
   set(gcf,'pos',get(gcf,'pos')+[0 0 0 1])
   drawnow
   set(gcf,'pos',get(gcf,'pos')-[0 0 0 1])
   drawnow
```

```
if get(gca,'userdata') ~= rho, delete(gca), end
set(gca,'color','black','pos',[.03 .05 .93 .95],'userdata',rho)
axis(R);
axis off
comet(1)=line(x(1),x(2),x(3),'linestyle','none','marker','.', ...
   'erasemode','xor','markersize',25);
comet(2) = line(NaN,NaN,NaN,'color','y','erasemode','none');
comet(3) = line(NaN,NaN,NaN,'color','y','erasemode','none');
X = x(:,ones(L,1));
uics = flipud(get(gcf,'children'));
paws = uics(1);
stop = uics(2);
set(paws,'string','pause','callback','','value',0);
set(stop,'string','stop','callback','','value',0);
beta = -A(1,1);
eta = sqrt(beta*(rho-1));
xc = [rho-1; eta; eta];
line(xc(1),xc(2),xc(3),'linestyle','none','marker','o','color','g')
line(xc(1),-xc(2),-xc(3),'linestyle','none','marker','o','color','g')
ax = [R(2) R(1) R(1) R(1) R(1)];
ay = [R(3) R(3) R(4) R(3) R(3)];
az = [R(5) R(5) R(5) R(5) R(6)];
p = .9;
q = 1-p;
grey = [.4 .4 .4];
line(ax,ay,az,'color',grey);
text(p*R(1)+q*R(2),R(3),p*R(5),sprintf('%3.0f',R(1)),'color',grey)
text(q*R(1)+p*R(2),R(3),p*R(5),sprintf('%3.0f',R(2)),'color',grey)
text(R(1),p*R(3)+q*R(4),p*R(5),sprintf('%3.0f',R(3)),'color',grey)
text(R(1),q*R(3)+p*R(4),p*R(5),sprintf('%3.0f',R(4)),'color',grey)
text(R(1),R(3),p*R(5)+q*R(6),sprintf('%3.0f',R(5)),'color',grey)
text(R(1),R(3),q*R(5)+p*R(6),sprintf('%3.0f',R(6)),'color',grey)
fin = 0;
cameratoolbar('setmode','orbit')
```

```
   uicontrol('style','text','units','norm','pos',[.38 .02 .34 .04], ...
      'foreground','white','background','black','fontangle','italic', ...
      'string','Click on axis to rotate view')
elseif isequal(job,'done')
   fin = 1;
else
   L = size(x,2);
   X(:,end+1:end+L) = x;
   comet = flipud(get(gca,'children'));
   set(comet(1),'xdata',X(1,end),'ydata',X(2,end),'zdata',X(3,end));
   set(comet(2),'xdata',X(1,2:end),'ydata',X(2,2:end),'zdata',X(3,2:end))
   set(comet(3),'xdata',X(1,1:2),'ydata',X(2,1:2),'zdata',X(3,1:2))
   X(:,1:L) = [];
   drawnow;
   uics = flipud(get(gcf,'children'));
   paws = uics(1);
   stop = uics(2);
   rhopick = uics(4);
   rho = A(3,2);
   while get(paws,'value')==1 & get(stop,'value')==0
      set(paws,'string','resume');
      drawnow;
   end
   set(paws,'string','pause')
   fin = get(stop,'value') | get(rhopick,'value')==rho;
   if fin
      set(paws,'value',0,'string','restart','callback','lorenzMy')
      set(stop,'value',0,'string','close', ...
         'callback','cameratoolbar(''close''), close(gcf)')
   end
end
```

**Задание 11.8** Исследуйте аттрактор Лоренца при различных значениях параметра $\rho$. При каком значении $\rho$ появлению аттрактора предшествуют наиболее значительные бифуркации?

**Задание 11.9** Измените текст программы и исследуйте так называемую систему или аттрактор Росселера:

$$\dot{x}_1 = -(x_2 + x_3), \qquad \dot{x}_2 = x_1 + ax_2, \qquad \dot{x}_3 = b + x_1x_3 - cx_3$$

при $a = 0.17$, $b = 0.4$, $c = 8.5$ и других значениях параметров.

## 11.5 Контрольные задачи

### Математические задачи

1. Решить систему уравнений:

$$\begin{cases} 3\dot{x} + \dot{y} + x = 1 \\ \dot{x} + 4\dot{y} + 3x = 0 \end{cases} \qquad x(0) = 0, \quad y(0) = 0$$

2. Решить систему уравнений:

$$\begin{cases} \dot{x} - x - 2y = t \\ \dot{y} - 2x - y = t \end{cases} \qquad x(0) = 2, \quad y(0) = 4$$

3. Решить систему уравнений:

$$\begin{cases} 2\ddot{x} - \dot{x} + 9x - \ddot{y} - \dot{y} - 3y = 0, & x(0) = 1, \quad \dot{x}(0) = 1 \\ 2\ddot{x} + \dot{x} + 7x - \ddot{y} + \dot{y} - 5y = 0, & y(0) = 0, \quad \dot{y}(0) = 0 \end{cases}$$

4. Решить уравнение:

$$\ddot{y} + \dot{y} - 2y = e^t, \qquad y(0) = -1, \quad \dot{y}(0) = 0$$

5. Решить систему уравнений:

$$\begin{cases} \ddot{x} - x + y + z = 0, & x(0) = 1, \quad \dot{x}(0) = 0 \\ \ddot{y} - y + x + z = 0, & y(0) = 0, \quad \dot{y}(0) = 0 \\ \ddot{z} - z + x + y = 0, & z(0) = 0, \quad \dot{z}(0) = 0 \end{cases}$$

6. Решить уравнение:

$$y^{(4)} + y^{(3)} = \cos t, \qquad y(0) = 0, \quad \dot{y}(0) = 0, \quad \ddot{y}(0) = 0, \quad y^{(3)}(0) = 2$$

7. Найти огибающую семейства решений уравнения $\dot{y} + y^2 = x^2$.

8. Решить уравнение:

$$y^{(4)} + 4y = t^2, \qquad y(0) = 0, \;\; \dot{y}(0) = 1, \;\; \ddot{y}(0) = 2, \;\; y^{(3)}(0) = 3$$

9. Решить уравнение:

$$4y^{(3)} - 8y^{(2)} - 2\dot{y} = 8e^t, \qquad y(0) = 1, \;\; \dot{y}(0) = 1, \;\; \ddot{y}(0) = 1$$

10. Решить уравнение:

$$y^{(3)} - 6y^{(2)} + 11\dot{y} - 6y = 0, \qquad y(0) = 0, \;\; \dot{y}(0) = 0, \;\; \ddot{y}(0) = 10$$

11. Решить уравнение:

$$y^{(4)} + 2y^{(2)} = t \sin t, \qquad y(0) = 0, \;\; \dot{y}(0) = 10, \;\; \ddot{y}(0) = 0.1, \;\; y^{(3)}(0) = 0.01$$

12. Решить систему уравнений:

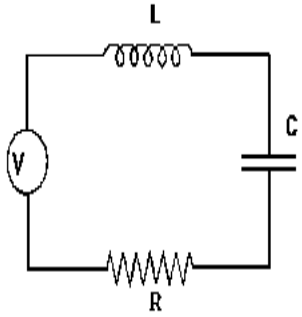
$$\begin{cases} \ddot{x} - x + 2y = 0, & x(0) = 0, \qquad \dot{x}(0) = -1 \\ \ddot{x} - 2y = 0, & y(0) = 1/2 \end{cases}$$

## Физические задачи

13. Привязанный у дна шарик с воздухом отпускают и он начинает всплывать. Объем шарика увеличивается по мере того, как уменьшается окружающее давление. Первоначальный диаметр шарика 0.5 метра. Закон изменения объема в зависимости от давления, считать таким:$V(P) = V_0(1 + 2\lambda(P))$ . Где $\lambda(P) = \frac{P_H - P}{P_H - P_{atm}}$ . Шарик находится на глубине $H = 500m$. $P_H$ - давление на глубине $H$. $P_{atm}$ - атмосферное давление. Считать, что плотность воды от глубины не изменяется.

    Оценить время всплытия для шарика массой 500 грамм. При расчете движения учитывать только силу тяжести, силу Архимеда и силу сопротивления водной среды. Сила сопротивления водной среды $f = \frac{\rho v^2}{2} S$, где $\rho$ - плотность воды, $v$ - скорость движения шарика, $S$ - площадь поперечного сечения.

14. Исследовать заряд конденсатора в сети




Заряд $q(t)$ удовлетворяет уравнению

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} = V(t)$$

которое получено из закона Ома

$$V(t) = L\frac{dI}{dt} + RI + \frac{q}{C}$$

в который мы подставили ток $I = \frac{dq}{dt}$.

Исследовать это уравнение при разных значениях параметров, с источником типа батарейка $V(t) = 3$ и с источником типа аккумулятор $V(t) = 3 - a\exp(t)$, где $a$ - малый коэффициент.

15. Парашютист массой 80 килограмм прыгает с высоты 700 метров. Парашют открывается через $t_0 = 10$ секунд после прыжка. Уравнение движения имеет вид

$$\ddot{y} = -g + \frac{\alpha(t)}{m}, \qquad y(0) = 700, \quad \dot{y}(0) = 0\,.$$

Здесь $g = 9.81\,m/s^2$ - ускорение свободного падения, а $\alpha(t)$ - коэффициент трения, который имеет вид $\alpha(t) = k_1\dot{y}(t)^2$ при $t < t_0$ и $\alpha(t) = k_2\dot{y}(t)^2$. Оценить время полета парашютиста и силу удара о землю при $k_1 = 1/150$ и $k_2 = 4/150$.

16. Уровень воды в сосуде составляет 10 метров (относительно дна сосуда). Свинцовый шарик диаметра 1 см падает с высоты 5 см над уровнем воды. Оценить время, за которое шарик упадет на дно сосуда.

17. Составить и решить уравнение движения бруска на пружине. Массу бруска, коэффициент трения о поверхность и жесткость пружины, считать заданными.

18. Исследовать стационарные решения уравнение Кортевега де Вриса, описывающее движение волны на мелководье,

$$(1-v)y' + a\frac{2}{3}yy' + b\frac{1}{6}y''' = 0,$$

при разных значениях параметров.

19. Уравнение движения тела, пролетающего через массивный шар постоянной плотности

$$y'' + G\frac{4}{3}\pi\rho r^3(y)y^{-2} = 0,$$

$$r(y) = \begin{cases} |y|, & |y| < R, \\ R, & |y| > R. \end{cases}$$

Особенностью данной задачи является неаналитический характер коэффициентов в уравнении. Найти начальные условия при которых тело вылетает из шара, а затем возвращается.

20. Рассмотрим траекторию движения пули под действием силы тяжести, когда сила сопротивления воздуха пропорциональна квадрату скорости. Пусть масса пули - 10 грамм, поперечник - 1 см, плотность воздуха - 1 килограмм на кубический метр. Методом стрельбы решить уравнения движения так, чтобы $x(0) = y(0) = 0$, и $x(6) = 20, y(6) = 0$. С какой погрешностью решена данная граничная задача?

21. Падение тела в вязкой жидкости под действием силы тяжести. Сила тяжести определяется плотностями тела $\rho$ и жидкости $\rho_0$

$$F = (\rho - \rho_0)\frac{4}{3}\pi r^3 g.$$

Здесь $r$ — радиус тела, $g$ — ускорение свободного падения. В случае сферического тела в стационарном приближении действующая на него сила сопротивления имеет вид

$$F_c = -6\pi\eta r v$$

Здесь $\eta$ — коэффициент вязкости, $v$ — скорость тела. Второй закон Ньютона в квазистационарном приближении приобретает вид

$$y'' = Ay' + B.$$

Пусть в начальный момент задано положение тела и его скорость

$$y(0) = y_0, \qquad y'(0) = y'_0$$

Требуется написать программу для решения указанной задачи Коши и исследовать характер движения в зависимости от соотношения параметров системы.

22. Исследовать явления резонанса и биений в системе

$$\frac{d^2x}{dt^2} + w^2x = F_0 sin(\gamma\, t), \quad x(0) = 0, \quad x'(0) = 0$$

Собственная частота системы $\frac{w}{2\pi}$,частота внешней силы $\frac{\gamma}{2\pi}$. (Например при частотах $\nu = \frac{11}{2\pi}$ и $\nu = \frac{13}{2\pi}$, $F_0 = 48$).