МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

Г. Л. Коткин, В. С. Черкасский

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ MATLAB

Учебное пособие

Новосибирск
2001

ББК 32.97:53
УДК 53.072

Коткин Г. Л., Черкасский В. С. Компьютерное моделирование физических процессов с использованием MATLAB: Учеб. пособие / Новосиб. ун-т. Новосибирск, 2001. 173 с.

Данное учебное пособие является руководством для компьютерного практикума.

Предназначено для студентов физического факультета НГУ, но может быть руководством по использованию пакета **MATLAB** студентами и исследователями других специальностей.

Подготовлено при содействии Федеральной целевой программы "Государственная поддержка интеграции высшего образования и фундаментальной науки на 1997-2000 годы", проект N274.

Рецензент
кандидат технических наук Ю.М.Прокопьев
Печатается по решению методической комиссии физического факультета НГУ.

Предисловие

Практикум «Компьютерное моделирование физических процессов» под тем или иным названием действовал на физического факультета НГУ более 20 лет. В настоящее время в связи с существенным изменением компьютеров произошел практически полный переход на новую операционную среду с графическим интерфейсом (типа Windows-95). Это предопределило изменения, которые были внесены в практикум. Перед нами стояла задача, не разрушая того полезного, что было наработано в практикуме за годы его эксплуатации, перейти на работу в новой операционной среде и расширить круг задач, которые решаются в практикуме. Кроме того, практикум предназначен студентам, уже имевшим дело с компьютером и знакомым с основными понятиями программирования.

Данное пособие представляет собой существенно переработанное и дополненное пособие, ранее изданное в НГУ [1], которое, в свою очередь, являлось объединением двух работ: Израйлев Ф.М.,Коткин Г.Л., Фрумин Л.Л., Эйдельман С.И. Моделирование физических процессов и явлений. Новосибирск, НГУ, 1986; Коткин Г.Л., Фрумин Л.Л. Моделирование физических явлений: Практикум. Новосибирск, НГУ, 1992.

Существенным отличием данного варианта практикума является использование не универсального языка программирования (как **Фортран** или **Паскаль**), а специальной системы **MATLAB** фирмы **MathWorks**, созданной для облегчения решения инженерных и научных задач. Поскольку данная система является относительно новой и литература по ней весьма ограничена [2-6], в настоящем пособии она будет описана отдельно. Следует сказать, что эта система является Windows-ориентированной [1], интерактивной, допускающей режим непосредственных вычислений как численного, так и аналитического характера, а также режим программирования на специальном языке программирования, работающем в режиме интерпретации. Для решения основных задач курса студентам предлагаются заготовки, которые решают простейшие вопросы и могут служить основой для доработки и усложнения соответствующих моделей. Эти заготовки доступны в виде исходных текстов — так называемых m-файлов.

В предлагаемых задачах затронуты разные подходы к моделированию. Это исследование моделей, движение в которых определяется обыкновенными дифференциальными уравнениями (задачи «МАЯТНИК», «ПЛАНЕТА», «ДИОД»), метод Монте-Карло («СЛУЧАЙНЫЕ БЛУЖДАНИЯ», «БРОУНОВСКОЕ ДВИЖЕНИЕ», «ПОТЕРИ ПУЧКА»), молекулярная динамика («ШАРЫ»).

[1] **Есть версии MATLAB и в операционной системе UNIX.**

Из важных и популярных объектов моделирования опущены задачи, связанные с уравнениями в частных производных, и задачи о фрактальных структурах: практикум рассчитан на ограниченное время.

При выполнении каждого из заданий за основу удобно брать соответствующие простейшие программы, входящие в пакет **MPP(Modeling of Physics Phenomena)**, специально подготовленный для этого практикума. Объединять программы, реализующие разные задания одной задачи, в одну большую программу не стоит.

Отчет студента — это действующая программа, которая демонстрируется преподавателю на месте (плюс ответ на вопросы по физике, решаемые с ее помощью). Письменный отчет при этом не требуется.

1. Введение

1.1. Зачем нужен такой практикум?

Основные применения компьютеров в физических исследованиях — это управление экспериментом (данного вопроса мы не касаемся) и моделирование [2]. Цель практикума — ознакомить студентов с некоторыми методами создания и исследования моделей физических явлений. Одновременно происходит изучение языка программирования **MATLAB** (не в полном объеме, но сразу же на уровне «разговорного»).

Разумеется, работа с моделями не может привести к открытию совершенно нового явления, скажем, элементарной частицы с неожиданными свойствами. Однако именно компьютерное моделирование привело, например, к возникновению нового взгляда на интересное и сложное явление — турбулентность. Кстати, и в работах, приводящих к открытию новых элементарных частиц и исследованию их свойств, моделирование не только используется на этапе проектирования экспериментальных установок, но и является непременной составной частью обработки экспериментальных данных. Расширяется применение компьютерного моделирования в технике. Наконец, моделирование может оказать заметную помощь студенту в изучении физики.

1.2. О чем сказано далее

В пп. 1.3, 1.4 «Введения» кратко сказано о системе **MATLAB**. Возможно, вы уже имели дело с программированием или использовали какую-либо систему научных

[2] **Разумеется, существует также необходимая «организационная» работа — поиск в компьютерных сетях полезной информации, редактирование текстов и т.п.**

расчетов и моделирования (например, **MathCad**) и не нуждаетесь в «популярных» объяснениях. Тогда достаточно только просмотреть эти пункты. С приложениями следует знакомиться при необходимости.

Физическую постановку задач, методы их решения и рекомендуемые к выполнению задания можно найти в разделах, относящихся к соответствующим задачам.

1.3. О системе MATLAB

Система **MATLAB** (MATrix LABoratory) давно и успешно разрабатывается фирмой MathWorks. Эта система создана для работы в среде Windows 3.1 (версия 4 и 4.2) и в среде Windows-95(98) (версии начиная с 5.0). Система представляет собой интерактивную среду для вычислений и моделирования, причем она может работать как в режиме непосредственных вычислений, так и в режиме интерпретации написанных программ. Если вы находитесь в системе **MATLAB**, то, набрав в ответ на приглашение текст

```
>>  y=sin(0.125)
```

и завершив его нажатием клавиши **ENTER** , получите в ответ

```
y=
  0.1247
>>
```

После ввода команды непосредственного вычисления система «интерпретирует» введенные инструкции и осуществляет вычисление. Результат сразу выводится на экран. Помимо обычных алгебраических вычислений система имеет большой набор встроенных функций (см. Приложение E), а также имеется возможность создавать свои собственные функции. Библиотеки функций (кроме встроенных) представляют собой специальные директории, в которых хранятся файлы с текстами функций. Эти тексты интерпретируются системой при обращении к ним и могут использоваться как образцы для написания своих функций.

Имеется также целый набор библиотек, позволяющих строить на экране 2- и 3-мерные изображения. Именно графическое представление результатов делает наши исследования чрезвычайно эффективными. Кроме того, имеется библиотека, которая обеспечивает удобное управление исполнением программ. Краткое описание этих и некоторых других библиотек приведено в Приложении E.

1.4. Немного о работе с системой MATLAB

После того как вы кликнули на иконке **MATLAB**, перед вами появится экран, в верхней части которого имеется строка с выпадающими меню, инструментальная панель с кнопками, реализующими наиболее часто выполняемые действия (рис. 1), и в самом окне - строка запроса в виде двух знаков >>. Это *командное окно* **MATLAB**

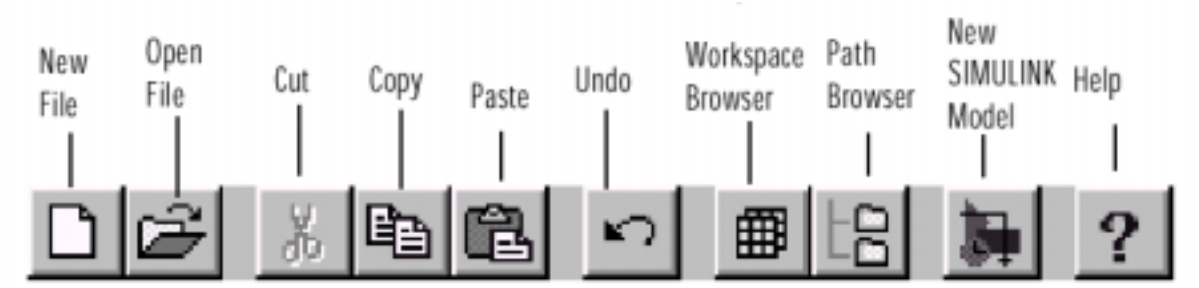


Рис. 1. Инструментальная панель командного окна

Стандартное выпадающее меню **File** содержит такие пункты, как **New** для создания новых файлов, **Open M-file** - открытие существующего файла-программы или файла-функции для редактирования, проверки текста или отладки. При использовании этого пункта вам предлагается стандартное окно выбора файлов, а после выбора необходимого файла открывается окно редактора/отладчика m-файлов. Подробнее об m-файлах будет сказано далее, сейчас же достаточно знать, что так называются текстовые файлы с расширением .m, содержащие тексты программ-сценариев или тексты функций из стандартных или собственных библиотек. В редакторе их можно исправлять, устанавливать точки остановки для отладки, но следует помнить, что для того, чтобы новый, исправленный вариант функции или программы вступил в силу, необходимо стандартным образом (через меню редактора **File** или с помощью соответствующей кнопки на панели инструментов редактора/отладчика) сохранить измененный файл.

Инструментальная панель (см. рис. 1) командного окна позволяет выполнять требуемые действия простым нажатием на соответствующую кнопку. Большинство кнопок имеют стандартный вид и выполняют стандартные, подобные другим программам действия - это копирование (**Copy**), открытие файла (**Open**), печать (**Print**) и т.д. Следует обратить внимание на кнопку **Path Browser**, которая позволяет прокладывать пути к разным директориям и делать необходимую директорию текущей, а также на кнопку **Workspace Browser**, позволяющую просматривать и редактировать переменные в рабочей области.

Команда **help**, набранная в ответ на запрос, завершаемая нажатием клавиши

Enter[3], или кнопка инструментальной панели со знаком вопроса позволяет получить список функций, для которых доступна оперативная помощь. Команда **help** <**имя_функции**> позволяет получить на экране справку по конкретной функции. Например, команда **help eig** позволяет получить оперативную справку по функции **eig** - функции вычисления собственных значений матрицы. С некоторыми возможностями системы **MATLAB** можно познакомиться с помощью команды **demo**.

В этом кратком введении следует отметить, что основные объекты – переменные, с которыми работает **MATLAB**, – это прямоугольные матрицы. Это дает возможность записывать программы очень кратко, делает прораммы легко обозримыми. Предусмотрено множество операций, выполняемых над матрицами. Разумеется, запись таких операций, как умножение и сложение матриц, следует запомнить. Изучать же и запоминать все возможности «впрок», до того, как они понадобятся, бессмысленно.

Если необходимо прервать работу, но сохранить все созданные в рабочей области переменные, то проще всего это сделать с помощью команды **save** <**имя_файла**>. Все переменные в двоичном виде сохраняются в файле <**имя_файла**>**.mat.** Впоследствии, при повторной загрузке системы можно загрузить всю рабочую область с помощью команды **load** <**имя_файла**> и продолжить вычисления с того же места. Для очистки рабочей области используется команда **clear** без аргументов, и в этом случае очищается вся область от всех переменных. Если команда **clear** сопровождается списком переменных, разделенных пробелами, то удаляются только перечисленные переменные.

Для завершения работы в системе используется команда **quit** или пункт меню **File/exit**.

2. Первые задачи

2.1. Фигуры Лиссажу

Начнем с очень простой задачи – построить на экране график функции, заданной параметрически:

$$\begin{cases} x = a_1 \cdot \cos(\omega_1 t) \\ y = a_2 \cdot \cos(\omega_2 t) \end{cases}$$

[3] **Далее везде, где это не будет специально оговорено, предполагается, что любая команда, вводимая с клавиатуры в режиме непосредственного вычисления, завершается нажатием клавиши Enter .**

Если отношение ω_1/ω_2 - число рациональное, то такая кривая называется фигурой Лиссажу. Скорее всего, такие кривые вы неоднократно видели на экране осциллографа.

Приведенный далее текст представляет собой протокол работы с системой **MATLAB**, полученный при решении поставленной задачи в режиме непосредственного вычисления. Для получения протокола и сохранения его в виде файла используется команда **diary** **<имя_файла>**. Такой протокол может быть полезен для последующего анализа решения, а также при небольших доработках может быть использован как основа для написания программы-сценария для последующего решения подобных задач. В системе **MATLAB** часть строки, следующая за знаком %, является комментарием , т.е.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Изображение кривой, заданной параметрически     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Задание амплитуд и частот
>> a1=1.2;   % Точка с запятой после оператора
>> a2=1.0;   % обеспечивает отсутствие вывода
>> w1=1.5;   % результата действия оператора
>> w2=1.0;

% Оператор whos позволяет в любой момент получить
% полную информацию о всех активных переменных.
% Все переменные введенные в данном сеансе, остаются
% активными, если их не удалили оператором clear.
>> whos
 Name    Size   Elements  Bytes  Density  Complex
   a1     1 by 1    1       8      Full     No
   a2     1 by 1    1       8      Full     No
   w1     1 by 1    1       8      Full     No
   w2     1 by 1    1       8      Full     No
 Grand total is 4 elements using 32 bytes
% Часть программы, относящаяся к оператору whos,
% приведена здесь только для ознакомления с его действием.
% Обратите внимание, что постоянные трактуются как матрицы
% размерности 1x1.

% Оператор задания вектора значений t
```

```
>> t=0:0.1:3.2;
% Вычисление векторов x, y
>> x=a1*cos(w1*t);
>> y=a2*cos(w2*t);
% Построение графика y(x)
>> plot(x,y);
```

В принципе на этом можно было бы и закончить, потому что по команде **plot(x,y)** будет нарисована искомая кривая. Точки с координатами **(x(i), y(i)), (x(i+1), y(i+1))** соединяются отрезками прямых. При этом происходит автоматический подбор удобного масштаба. Масштабные метки и числа при них изображаются на границах рамки, а изображение осей координат не предусмотрено.

Следует сделать несколько замечаний. Обратите внимание на способ задания вектора **t**. Дело в том, что знак **:** (подробнее см. Дополнение, п. 2.2) является одним из важнейших в синтаксисе языка **MATLAB**. Поставленный между двумя числами, он задает вектор, компоненты которого принимают значения от меньшего числа до большего с шагом 1. Например, оператор **x=4:15** задает целочисленный вектор **x=[4 5 6 7 8 9 10 11 12 13 14 15]**. Кстати, такое явное задание вектора (с помощью квадратных скобок) тоже допускается. Если значения компонент вектора отличаются не на 1, то их можно задать с помощью указания шага, например, так: **t=0:0.1:3.2**, что и было сделано в программе.

Написанные далее два оператора (присвоение соответствующих значений **x** и **y**) демонстрируют еще одну особенность языка **MATLAB**: основными объектами, с которыми он оперирует, являются матрицы и векторы (многомерные). Записанные таким образом функции (в данном примере - **cos**) вычисляют значение функции для каждого элемента вектора (или матрицы) аргумента и присваивают их соответствующим элементам вновь создаваемого вектора.

Эта и ряд подобных особенностей позволяют записывать алгоритмы на языке **MATLAB** в очень компактном виде. Подробнее возможности работы с векторами и матрицами описаны в Дополнении, п. 2.

Получив соответствующий рисунок, мы обнаруживаем, что нарисована лишь часть кривой, во всяком случае она не имеет привычный вид фигуры Лиссажу. Попробуем расширить диапазон вычисления и вывода кривой. Можно заново набрать написанный ниже текст, а можно воспользоваться буфером команд, который имеется в системе. Нажав необходимое число раз клавишу ↑, вы получите в строке ввода одну из предыдущих команд, которую можно редактировать с помощью стрелок и клавиш **Del, BkSp**, а также вставлять необходимые символы. После редактирования и соответствующего вычисления наш протокол будет выглядеть следующим

образом.

```
>> t=0:0.1:10;
>> x=a1*cos(w1*t);
>> y=a2*cos(w2*t);
>> plot(x,y);
```

Полученный в этом примере график будет иметь вид, показанный на рис. 2.

Представляет интерес получить различные графики при различных значениях параметров. Если вам необходимо просто нарисовать еще одну кривую с другими параметрами, то проще всего в том же сеансе присвоить им новые значения и снова повторить последовательность вычислений и вывод графика. Такой метод работы удобен при необходимости разовых вычислений, но если вы хотите исследовать зависимость получаемой кривой от параметров, то представляет интерес научиться делать это систематически.

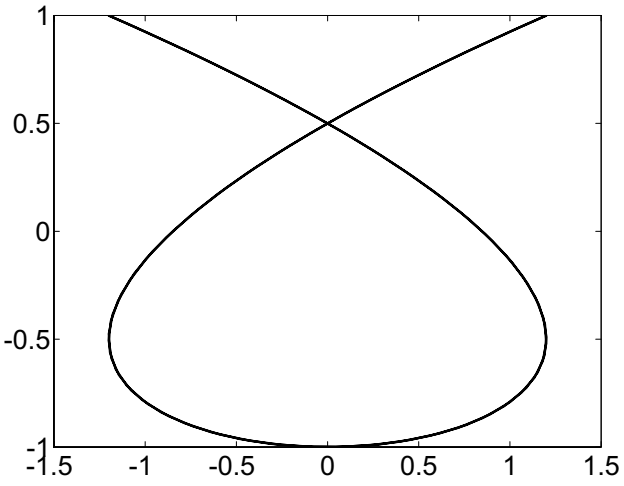


Рис. 2. Пример кривой Лиссажу

Задание 1. Используя возможность повторять выполнение операторов без их повторного набора (клавиша ↑), выполните вычисление и построение фигур Лиссажу для разных отношений ω_1/ω_2 и для разных значений амплитуд a_1 и a_2.

Для дальнейшей работы удобнее перейти к программе-сценарию. Так называется файл с расширением **.m**, который содержит последовательность операторов **MATLAB**. Наберите приведенный далее текст или модифицируйте протокол своей работы (который вы можете получить с помощью команды **diary**).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Второй вариант расчета фигур Лиссажу     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Задание коэффициентов
 a1=1.2; a2=1.0; w2=1.0;
% Задание значений вектора t
 t=0:0.1:15; x=a1*cos(w2*t);
% Задание разных частот w1
 w1=1.25:0.25:2.0;
% Цикл вывода разных графиков на одном листе
 for k=1:4,
    y=a2*cos(w1(k)*t);
  % Задание вектора-строки вывода надписей
   s=['w1/w2=' num2str(w1(k))];
  % Вывод разных графиков в разных местах листа (экрана)
  % Оператор subplot задает место на листе, где
  % график будет нарисован последующей командой
   subplot(2,2,k); plot(x,y); title(s);
end;
```

Написанная выше программа-сценарий имеет несколько новых операторов. Оператор **s=['w1/w2=' num2str(w1(k))];** можно для большей ясности представить в виде нескольких операторов **s1='w1/w2'** — формирование вектор-строки, каждый элемент которой является символом, т.е. формирование строки текста. Оператор **s2=num2str(w1(k))** — формирование строки текста с помощью функции **num2str**, которая превращает числовую переменнную **w1(k)** в ее строковое представление. Пусть, например, **w1(1)=2.34**, тогда оператор **s2=num2str(w1(1))** эквивалентен присвоению **s2='2.34'**. И последнее действие — объединение двух векторов **s1** и **s2** в один с помощью конструкции вида **s=[s1 s2]**. Таким образом можно явно задавать вектора и матрицы (см. Дополнение, п. 2), причем элементами, стоящими внутри квадратных скобок, могут быть как числа или буквы, так и целые вектора и даже матрицы.

Созданная описанным образом строковая переменная **s** используется впоследствии в операторе **title** в качестве аргумента, в результате чего у каждого подрисунка появляется свой заголовок.

Новым в приведенной программе является также оператор цикла **for...end** и оператор **subplot(m,n,k)**, который позволяет на одном рисунке создать матрицу из **mxn** отдельных рисунков, причем параметр **k** определяет номер рисунка по

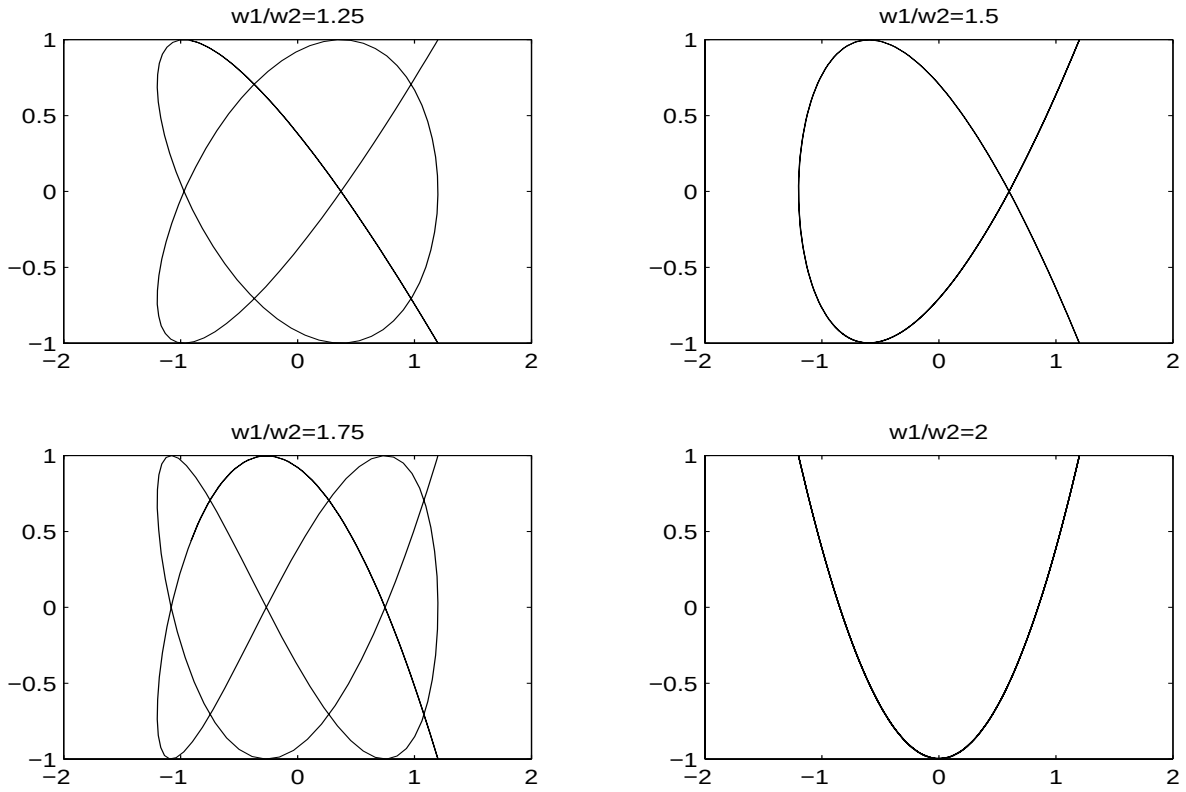


Рис. 3. Одновременный вывод 4 фигур Лиссажу.

порядку слева направо и сверху вниз. Результат использования программы показан на рис. 3. Заметим, что в исходном тексте программы представлены фигуры Лиссажу не самого общего вида.

Задание 2. Включите начальные фазы φ_1 и φ_2 в аргументы косинусов, т.е. $\cos(\omega_1 t)$ замените на $\cos(\omega_1 t + \varphi_1)$ и $\cos(\omega_2 t)$ на $\cos(\omega_2 t + \varphi_2)$.

Задание 3. Еще одна модификация программы — замена процедуры **plot** на процедуру **comet** — позволит изображать «бегущий» участок кривой (см. Дополнение, п. 8.1) и анализировать, сколько раз точка пробегает кривую при изменении времени t в заданном интервале.

2.2. Биения

Вторая программа для первого занятия изображает кривую

$$y(t) = a_1 \cos \omega_1 t + a_2 \cos \omega_2 t.$$

Несмотря на простоту формулировки, эта задача очень содержательна. В процессе исполнения программы **Beats** на экран выводится участок кривой $y(t)$ для интервалов времени t от t_0 до t_m. Исходный текст программы:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Учебная программа расчета биений                  %
% Это вариант с выводом полной кривой биений         %
% и перемещением по ней с помощью команды AXIS      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
a1=1.0;          % Амплитуды гармонических
a2=1.0;          %     колебаний
w1=1.0;          % Частоты гармонических
w2=1.2;          %     колебаний
t0=0;            % Начальный момент времени
tm=20;           % Конечный момент времени
N=600;           % Число точек вывода/расчета
T=tm-t0;         % Время вывода биений
dt=T/N;          % Шаг по времени
t=t0:dt:tm;      % Вектор времени
y=a1*cos(w1*t)+a2*cos(w2*t);   % Функция биений
plot(t,y);       % Вывод графика
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

После вывода результата расчета на экран можно изменить масштаб осей с повторным выводом соответствующего участка кривой. Так, в приведенном выше примере переменная t на графике будет изменяться от 0 до 20. Если мы хотим рассмотреть подробности графика в другом диапазоне (например, по t от 1 до 2), то необходимо ввести в командном окне команду **axis([xmin xmax ymin ymax])**, где **xmin xmax** - диапазон вывода по оси **x**, а **ymin ymax** - диапазон вывода по оси **y**. В результате выполнения этого оператора график будет перестроен в указанном масштабе[4].

Задание 1. Частота биений при сложении двух гармонических колебаний равна разности их частот. Повторится ли через период биений после наибольшего максимума в одном всплеске биений наибольший максимум в следующем?

[4]**Следует иметь в виду, что построение нового графика идет на основе все тех же насчитанных массивов t и y, поэтому если в выбранный диапазон попадет мало точек, то и качество графика будет невысокое.**

Задание 2. Будет ли синусоидой «огибающая максимумов» при неравных друг другу амплитудах складываемых колебаний?

Задание 3. Другого вида биения можно наблюдать при $\omega_1 \ll \omega_2$. Для того чтобы увидеть, как действительно выглядят колебания в этом случае, желательно увеличить время вывода биений (т.е. диапазон изменения переменной t) примерно в 10-20 раз.

2.3. Графический интерфейс в задаче «Биения»

Для решения многих учебных и научных задач очень часто бывает удобно не менять (даже так удобно, как в **MATLAB**) данные внутри программы с последующим ее запуском, а менять каким-либо образом исходные данные, не прекращая при этом наблюдать за тем, как изменяется график соответствующего решения. Для реализации такого способа взаимодействия с программой (*интерактивного интерфейса*) удобно иметь на экране одновременно окно вывода графической информации, «кнопки управления» и окошки «редактирования» данных (рис. 4)[5].

При выполнении задачи о биениях вы получаете готовое решение, в котором уже создано окно интерфейса (функция **Interface_Window**) и частично решение задач организации связи с вычислительной программой. При этом две функции (**Drive_Beats** и **Run_Problem**) при выполнении заданий вам придется видоизменять.

При запуске задачи мы будем вызывать программу **Drive_Beats**, а из нее будут вызываться остальные функции. В этой программе определяются начальные значения параметров (амплитуд, частот и т.п.). Они будут далее передаваться в функцию, содержащую основной, расчетный алгоритм задачи — **Run_Problem**. Параметры, которые не меняются в процессе исследования, можно передавать из этой программы с помощью глобальных переменных (а можно просто задать их в **Run_Problem**). Те же параметры, которые необходимо менять при исследовании режимов расчета («редактируемые» переменные), передаются специальным образом, описанным далее.

Для исходного варианта задачи «Биения» передаваемыми параметрами являются: амплитуды a_1, a_2, частоты ω_1 и ω_2, начальный момент времени вывода t_1 и конечный момент вывода t_2.

[5] **Для разработки такого интерфейса в MATLAB имеется специальный пакет, который называется GUI — Graphics User Interface (графический интерфейс пользователя). О работе с этим пакетом рассказывается в Дополнении (п. 9).**

При выполнении данной задачи обращаться к нему не потребуется.

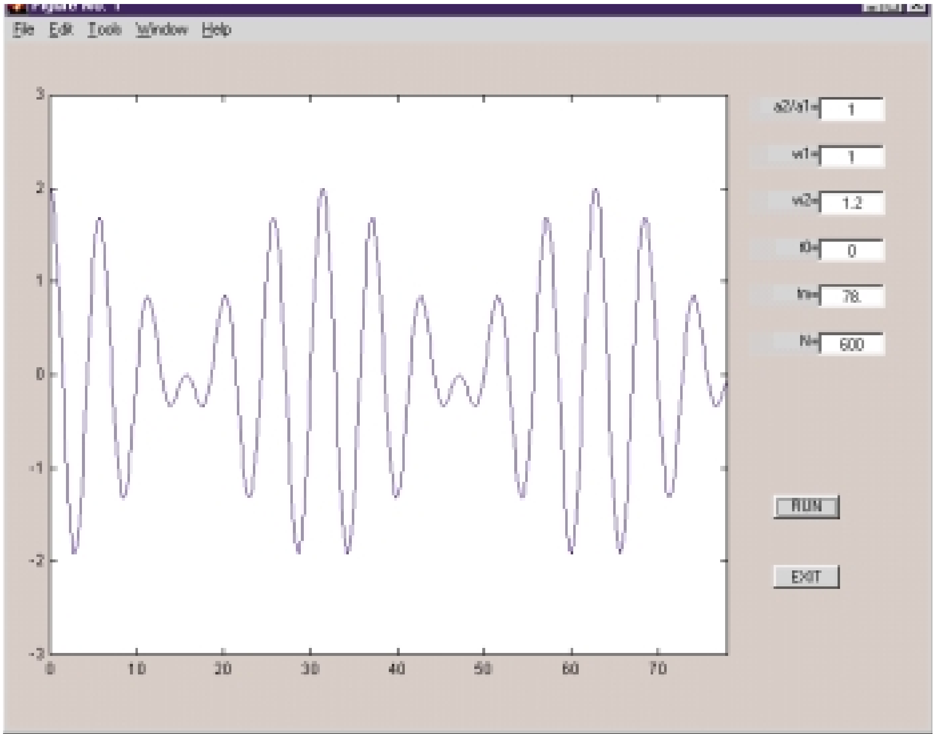


Рис. 4. Вид экрана при использовании разработанного графического интерфейса в задаче о биениях.

Рассмотрим примерную программу **Drive_Beats**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Это процедура для запуска BEATS               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 clear;% Первоначальная очистка рабочего пространства
% Задание начальных значений рабочих переменных
 a1=1; a2=1; w1=1; w2=1.2; t1=0; t2=20; N=600;
% Переменную N объявляем глобальной
 global N;
% Занесение начальных значений в транспортный массив

 VALUE(1)=a1;
 VALUE(2)=a2;
 VALUE(3)=w1;
 VALUE(4)=w2;
 VALUE(5)=t1;
 VALUE(6)=t2;
% Занесение имен передаваемых переменных
```

```
% в транспортный массив

 NAME(1,:)='     a1';
 NAME(2,:)='     a2';
 NAME(3,:)='     w1';
 NAME(4,:)='     w2';
 NAME(5,:)='     t1';
 NAME(6,:)='     t2';
% Вызов процедуры графического интерфейса и передача
% транспортных массивов
 Interface_Window(VALUE, NAME, 'Run_Beats')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

При разработке подобной программы для любой задачи необходимо иметь в виду:

1. Имена массивов (в примере — это массивы **VALUE** и **NAME**) могут быть любыми;
2. Размер (число строк) у массивов **VALUE** и **NAME** должны совпадать;
3. При задании строк-имен в массиве **NAME** длина строк (включая пробелы) должна быть одинаковой для всех переменных, желательно не более 8 символов;
4. Имя программы (m-файла), которое стоит в обращении, может быть любым — это имя основной, исполнительной программы. В нашем примере — это **Run_Beats**

Программа **Drive_Beats** вызывает функцию **Interface_Window** — основную функцию графического интерфейса. Эта функция строит изображение окна интерфейса на экране и помещает значения параметров, заданных в вызвавшей ее программе (в данном случае — в **Drive_Beats**), в «буферный объект» по имени **'UserData'**. Далее ее действия зависят от нажатия мыши на кнопках интерфейсного (графического) экрана или — при редактировании передаваемых данных — в специальных окнах редактирования.

Для редактирования (изменения) параметров задачи необходимо щелкнуть мышью в окошке численного значения данных, после чего эти численные значения можно изменять. Новые значения заносятся в буфер **'UserData'**.

При нажатии кнопки **Run** запускается программа **Run_Beats**. При этом параметры она получает из **'UserData'**. Графики, создаваемые программой **Run_Beats**,

строятся в графическом окне, созданном для этого программой **Interface_Window**. При завершении построения графика или работы программы **Run_Beats** программа **Interface_Window** проверяет, не было ли сигнала от мыши [6], и если был, то переходит к соответствующим действиям.

При нажатии кнопки **Exit** программа завершает свою работу, а рисунок удаляется с экрана.

Run_Beats- функция, к которой происходит обращение, при нажатии кнопки **RUN**. Как правило - это функция, содержащая основной алгоритм расчета задачи. Имя этой m-функции (совпадающее с именем файла) должно быть передано функции **Interface_Window** в качестве третьего параметра в виде текстовой строки.

Рассмотрим примерную программу **Run_Beats**.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Функция, выполняющая действия по нажатию кнопки RUN  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function r=Run_Beats()
% Задание параметров, не передающихся через меню,
% или их передача через механизм GLOBAL из головной
% программы
 global N;
% Идентификация текущего графика и определение
% его дескриптора
 h0=gcf;
% Извлечение совокупности передаваемых данных,
% хранящихся в UserData
 info=get(h0,'Userdata');
% Запись извлеченных данных в локальные переменные
 a1=info.VALUE(1);
 w2=info.VALUE(2);
 w1=info.VALUE(3);
 w2=info.VALUE(4);
 t1=info.VALUE(5);
 t2=info.VALUE(6);

% Подготовительные вычисления для реализации
```

[6]**По правде говоря, программа Interface_Window работает при этом совместно с другими, но это не существенно для ее использования.**

```
% основного алгоритма
 tm=t2-t1;            % Время вывода биений
 dt=tm/N;             % Шаг по времени
% Расчет биений
 t=t1:dt:t2;          % Вектор времени
 y=a1*cos(w1*t)+a2*cos(w2*t);  % Функция биений
% Организация графического вывода результатов
 cla;                 % Очистка области вывода
 axis([t1 t2 -2 2]);  % Задание осей
 hl=line(t,y);        % Вывод графика
 set(hl,'color','b'); % Задание цвета линии
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Заметим, что при исполнении программы **Interface_Window** в объект **'UserData'** был занесен массив **VALUE** с помощью записи **info.VALUE**, который и извлекается в функции **Run_Beats** с помощью оператора **info=get(gcf,'Userdata')**.

Для работы интерактивного графического интерфейса необходимо включить в свою рабочую директорию (в которой вы разрабатываете свои программы) еще две заранее приготовленные функции **Edit_Value.m** и **Exit_Problem.m**.

Задание 4. Изучите влияние второй и третьей гармоник на форму колебания. Для этого удобно выбрать функцию $y(t)$ в форме

$$y(t) = a_1 \cdot \cos(\omega t) + a_2 \cdot \cos(2\omega t + \varphi_2) + a_3 \cdot \cos(3\omega t + \varphi_3)$$

и контролировать амплитуды a_2, a_3 и фазы φ_2, φ_3.

2.4. Волны

Речь может идти о самых разных волнах — волнах на поверхности воды, звуке, радиоволнах и т.п. Если амплитуды волн не слишком велики, то для них справедлив принцип суперпозиции, что мы и будем предполагать.

Функция

$$y(x,t) = a\cos(kx - \omega t)$$

задает бегущую волну с амплитудой a, волновым числом k и угловой частотой ω. Длина волны $\lambda = 2\pi/k$, а ее период $T = 2\pi/\omega$. Скорость волны (скорость максимума, определяемого условием $kx - \omega t = 0$) равна $v = x/t = \omega/k$.

Суперпозиция двух волн

$$y(x,t) = a_1\cos(k_1 x - \omega_1 t) + a_2\cos(k_2 x - \omega_2 t) \quad (1)$$

в любой момент представляет картину биений в зависимости от x.

При $a_2 = a_1$

$$y(x,t) = a(x,t)\,\cos(kx - \omega t),$$

где

$$a(x,t) = 2a_1 \cos(dk\,x - d\omega\,t),$$

$$k = \frac{1}{2}(k_1 + k_2),\ \omega = \frac{1}{2}(\omega_1 + \omega_2),\ dk = \frac{1}{2}(k_1 - k_2),\ d\omega = \frac{1}{2}(\omega_1 - \omega_2).$$

Функция $|a(x,t)|$ определяет модуляцию суммарной волны; она перемещается со скоростью $u = d\omega/dk$, называемой групповой скоростью.

Программа **Wavepak** изображает сумму волн, причем изображение на экране сменяется с определенным шагом по времени т.е. осуществляется анимация изображения. Прежде чем подробно анализировать приведенную далее программу вывода бегущей волны, сделаем несколько общих замечаний относительно графики в системе **MATLAB**. Подробнее с особенностями графики в системе **MATLAB** можно познакомиться в Дополнении (п. 8) и в [1, 2, 3, 5].

2.4.1. Основные графические объекты и их использование

Дело в том, что функции, реализующие те или иные действия по созданию графических объектов, а также сами эти объекты (такие как линии, оси, надписи и т.д.) фактически являются объектами (подробнее см. Дополнение, п. 8.4) в смысле объектно-ориентированного программирования. В то же самое время большинство из них можно использовать как обычные функции, ничего не зная об их особенной природе. При реализации же анимации приходится использовать эти особенности, поэтому мы должны вкратце познакомиться с ними.

Существует иерархия объектов: рисунок (**figure**), оси координат (**axes**), линия (**line**). Обычно старшие объекты называют «родителями», а младшие — их «детьми». Существуют и другие объекты, но нам пока понадобятся только эти. Всякий объект имеет *дескриптор*, которым он однозначно определяется [7], и набор *свойств*, таких как цвет, размер, способ вывода и т.д. Полный *список свойств* любого объекта можно посмотреть с помощью системы помощи. Многие свойства, без которых объекты не могут существовать, имеют свои значения по умолчанию. Поэтому можно вызывать объекты, не указывая никаких свойств, кроме обязательных. Например, для функции **line** обязательно указать два массива **x** и **y** — координаты соединяемых точек. Для задания и/или изменения каких-либо свойств ис-

[7] **Лучше подошло бы отечественное слово *ярлык*, имеющее кроме смысла «метка» еще и исторический смысл — «ярлык на управление», созвучный с английским handle.**

пользуется функция **set(handle, 'PropertyName', 'PropertyValue', ...)**. Необходимо только знать имя дескриптора соответствующего объекта **handle**, имя соответствующего свойства **'PropertyName'** и какие значения оно (свойство) может принимать. Например, в приведенном далее примере, цвет (зеленый - 'g') отрисовываемого объекта **line** с дескриптором **h** задается оператором **set(h, 'color', 'g')**. Для получения значения какого-либо свойства или его присвоения другой переменной используется уже знакомый нам оператор **get**, использовавшийся в предыдущем параграфе.

Полезно будет заметить, что свойства родительских объектов наследуются объектами-детьми. Поэтому при неизбежно поверхностном знакомстве с системой, внося изменения в свойства каких-либо объектов, мы можем получить иногда непредвиденные изменения в работе программ. (Фактически это свойство унаследовано системой MATLAB от языка C++, на котором она написана.)

2.4.2. Отрисовка движущихся кривых

MATLAB предоставляет различные способы создания движущихся графиков или анимации. Использование свойства графических объектов **'EraseMode'** (режим стирания) удобно для длинных последовательностей простых графиков, у которых изменение от кадра к кадру минимально. Рассмотрим это на примере динамического вывода волны (программа WAVEPAK).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Программа WAVEPAK предназначенная для вывода      %
% волнового пакета с анимацией изображения           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 t=0:0.1:100; % Задание вектора времени
 x=0:0.3:30;  % Задание вектора координат
 k=1.3; w=0.9; n=length(t);
% Вычисление формы волны для момента времени t(1)
% во всех точках x
 y=cos(k*x-w*t(1))+cos(x-t(1));
 h=line(x,y);  % Подготовка графика волны и присвоение
% дескриптора этой линии переменной h
% Задание цвета объекта (линии) с дескриптором h
% через свойство 'color'
 set(h,'color','g');
  axis([0 30 -3 3]);% Выбор осей координат и их масштаб
 axis manual;
```

```
% Сохранение дескриптора линии волны и выбор значения
% свойства 'EraseMode' равным 'xor'. Это сообщает
% графической системе MATLAB, чтобы она не
% перерисовывала весь график (включая оси, цвет фона
% и те точки, которые не изменились), а перерисовала
% только те точки, которые изменили свои координаты.
 set(h,'EraseMode','xor');
 pause;     % Пауза перед запуском движения волны.
% Она гарантирует обязательный вывод на экран
% накопленного в буфере графики. Дальнейшее
% продвижение осуществляется нажатием на любую клавишу.
% Можно задать pause(0), тогда нажимать клавишу не надо
% Основной цикл вычисления и вывода движущейся волны
 for i=2:n;
% Вычисление формы волны для момента времени t(i)
   y=cos(k*x-w*t(i))+cos(x-t(i));
% Обновление координат линии, изображающей амплитуду
% волны с помощью обновления y объекта (линии)
% с дескриптором h свойства 'XData' (замена
% x-координат на новые значения) и аналогичные
% действия с y-координатами. Это стандартный
% способ обновления координат точек для анимации.
   set(h,'XData', x,'YData',y);
 end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Аналогичным образом можно создавать любые движущиеся графики. Проблема возникает тогда, когда скорость компьютера такова, что лимитирующим фактором плавного вывода на экран является скорость расчета новых точек. В таких случаях либо улучшают (ускоряют) алгоритм расчета, либо готовят анимацию путем последовательной покадровой записи результатов вывода на экран с последующим прокручиванием получаемого файла в режиме анимации (мультфильма).

Задание 5. Попробуйте, с помощью описанного выше алгоритма соединить анимацию движущейся волны и графический интерфейс пользователя, описанный в п. 2.3. При этом следует иметь в виду, что необходимо разбить описанную выше программу **WAVEPAK** на две части, первая из которых войдет в основную программу, вызывающую все остальные (типа **Drive_Beats**), а

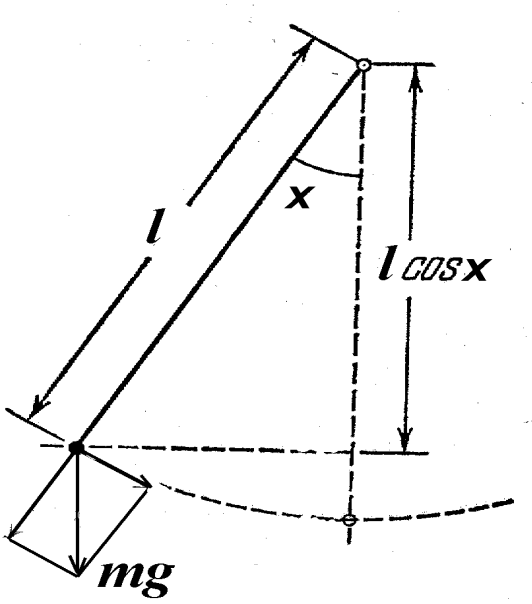


Рис. 5. Силы, действующие на математический маятник в поле тяжести

вторая часть войдет в функцию типа **Run_Beats**. Если у Вас не получится выполнить это задание самостоятельно, ознакомьтесь с текстами функций в директории **WAVEPAK**.

3. Об использовании безразмерных переменных

Во многих задачах бывает полезно вводить безразмерные переменные. (Это относится, разумеется, не только к работам с применением компьютера.) Показать, как это можно сделать и чем это удобно, лучше всего на конкретном примере.

Рассмотрим задачу о движении математического маятника — грузика массы m, подвешенного в поле тяжести на невесомом стержне длины l (рис. 5). Будем считать также, что маятник движется в одной плоскости.

Пусть на маятник действуют сила трения, пропорциональная скорости грузика, $\mathbf{F}_{\text{тр}} = -A\mathbf{v}$, и внешняя переменная сила $F(t) = F\cos\Omega t$, направленная горизонтально. (Можно представлять, что маятник помещен в большой плоский конденсатор с вертикальными пластинами, к которым приложено переменное напряжение, а на грузике находится постоянный заряд). Для удобства сопоставления с текстом программы мы обозначаем угол отклонения маятника от вертикального направления x.

Для угла отклонения нити от вертикали можно записать уравнение

$$ml\frac{d^2x}{dt^2} = -mg\,\sin x - Al\frac{dx}{dt} + F\cos x\,\cos\Omega t. \qquad (1)$$

В отсутствие силы трения и внешней переменной силы период малых колебаний маятника равен, как известно, $2\pi\sqrt{l/g}$.

Введем вместо времени t новую переменную τ согласно соотношению $t = \sqrt{l/g}\tau$; переменная τ оказывается, очевидно, безразмерной. Тогда уравнение (1) приводится к виду

$$\frac{d^2x}{d\tau^2} = -\sin x - a\frac{dx}{d\tau} + f\ \cos x \cos\omega\tau, \tag{2}$$

где

$$a = \frac{A}{m}\sqrt{\frac{l}{g}}\ ,\ f = \frac{F}{mg}\ ,\ \omega = \Omega\sqrt{\frac{l}{g}}\ - \tag{3}$$

безразмерные величины.

Такое преобразование обнаруживает некоторые законы подобия: зависимость $x(\tau)$ (при заданных $x(0), dx/d\tau(0)$) оказывается одной и той же при разных значениях m, l, g, A, F, Ω, если одинаковы составленные из них безразмерные комбинации a, f, ω. Этот факт позволяет существенно сократить объем полного исследования задачи, так как достаточно рассматривать различные значения трех параметров вместо шести. Иначе говоря, результаты исследования одного маятника можно перенести на другие простым изменением масштабов.

Кроме того, при численном определении решения уравнения (2) мы не будем, как правило, иметь дело с величинами, отличающимися друг от друга на много порядков, в то время как для уравнения (1) это вполне могло бы получиться при неудачном выборе единиц измерения. И хотя наш компьютер работает с «размахом» от -308 порядка (**realmin**) до 308 (**realmax**), лучше эту возможность не использовать без необходимости. Это позволит, в частности, не осложнять грубые оценки, выполняемые без компьютера.

Переход от уравнения (1) к (2) можно оформить и несколько иначе. Вместо обычных единиц измерения длины, массы и времени выберем «естественные» для данного маятника. За единицу длины примем его длину, за единицу массы — его массу, а единицу времени выберем такой, чтобы было $g = 1$ (т.е. равной $\sqrt{l/g}$). Подставив $l = m = g = 1$ в (1), мы получим уравнение вида (2), в котором величины t, A, F и Ω подразумеваются выраженными в этих «естественных» единицах. Принимая для этих величин обозначения τ, a, f, ω, приходим к (2). Чтобы получить на этом пути соотношения (3), нужно построить из l, m, g множители необходимой размерности. Скажем, сила имеет размерность mg, поэтому запишем $F = mgf$. Это равенство справедливо в «естественных» единицах и справедливо при переходе к любым другим единицам, если считать f безразмерной величиной, поскольку размерности его левой и правой частей тогда одинаковы. Аналогично могут быть получены и остальные соотношения (3). Именно выбор естественных для

задачи единиц измерения обеспечивает отмеченный выше «умеренный» характер изменения используемых безразмерных переменных. Описанный подход применяется весьма часто.

В задаче о движении частицы массы m в поле $U = -\alpha/R$ речь может идти как о движении планеты вокруг Солнца, так и о движении электрона вокруг атомного ядра. (В первом случае $\alpha = \gamma\, m\, M$, где γ – гравитационная постоянная, m – масса планеты, M – масса Солнца; во втором m – масса электрона, $\alpha = |qQ|$, где q – заряд электрона, $|Q|$ – заряд ядра[8].) Уравнение движения

$$m\frac{d^2\mathbf{R}}{dt^2} = -\frac{\alpha\mathbf{R}}{R^3}.$$

Введем в качестве единицы длины характерную длину R_0 (для астрономической задачи, например, $R_0 = 10^8$ км, для атомной $R_0 = 10^{-8}$ см). Тогда естественно в качестве единицы времени выбрать $t_0 = \sqrt{R_0^3 m/\alpha}$ (для движения планеты t_0 окажется порядка года, для движения электрона – порядка периода обращения электрона в атоме). Безразмерные длина r и время τ определяются равенствами

$$t = t_0\tau\,, \quad R = R_0 r,$$

а уравнение движения приобретает вид[9]

$$\frac{d^2\mathbf{r}}{d\tau^2} = -\frac{\mathbf{r}}{r^3}.$$

Еще один пример – задачи релятивистской физики частиц, где обычно полагают скорость света $c = 1$. При этом скорость частицы становится безразмерной величиной, равной v/c. Масса же остается размерной величиной, но ее размерность не отличается от размерности энергии. Например, масса электрона $m = 511$ кэВ.

4. Маятник

Задачи о колебаниях встречаются во всех областях физики. Во многом колебания совершенно различных физических объектов сходны друг с другом. Простейшие примеры – малые колебания маятника и электрические колебания в цепи, составленной из конденсатора и катушки.

[8] **Хотя движение электронов в атоме подчиняется не законам движения планет, а законам квантовой механики, необходимо знать также, что дали бы для такого движения законы классической механики.**

[9] **В работе «ПЛАНЕТА» сохранен коэффициент α=1, поскольку предполагается модификация поля U.**

Такое движение маятника хорошо известно — это гармонические колебания. Закон движения можно записать в виде $x = a\cos(\omega_0 t + \varphi_0)$, где ω_0 — частота колебаний, a — амплитуда, φ_0 — начальная фаза. (Угол отклонения маятника мы обозначили здесь x. Далее будем использовать безразмерные переменные, о которых говорилось в гл. 3, т.е. примем массу грузика m и длину маятника l равными единице, а также будем считать $\omega_0 = 1$.) Малые колебания описываются уравнением

$$\ddot{x} + x = 0,$$

линейным относительно функции x, поэтому их обычно называют линейными.

В этой работе мы будем исследовать движение математического маятника при больших углах отклонения, специально обращая внимание на отличие от законов малых колебаний. Подобные отклонения называют нелинейными эффектами.

4.1. Свободные колебания

Результаты исследования движения маятника удобно представить в виде набора кривых на плоскости (x, p), где $p = \dot{x}$ — скорость изменения угла. Плоскость (x, p) называется фазовой плоскостью, переменная p — импульсом[10], а кривые, определяемые параметрически законом движения как $x = x(t),\ p = p(t)$, — фазовыми траекториями.

Фазовая траектория определяется, например, начальными значениями координаты $x(0)$ и импульса $p(0)$.

Фазовые траектории линейного осциллятора представляют собой эллипсы, задаваемые законом сохранения энергии. Для математического маятника это справедливо при малых углах отклонения, в общем же случае, при больших значениях углов отклонения движение математического маятника будет более сложным. Кроме колебаний возможно вращение маятника в ту или другую сторону.

Угол отклонения маятника достаточно задавать в некоторых конечных пределах, например, принимая $-\pi \le x < \pi$. При этом следует представлять, что точки фазовой плоскости $(-\pi, p)$ и (π, p) отождествлены, иначе говоря, прямые $x = \pi$ и $x = -\pi$ склеены друг с другом так, что из полосы

$$-\pi \le x < \pi,$$

$$-\infty < p < \infty$$

[10] **Точнее говоря, каноническим импульсом в случае математического маятника называется величина $ml^2\dot{x}$, однако мы учитываем условия $m = l = 1$.**

получился цилиндр. Если маятник делает один или несколько оборотов, то точка, изображающая его состояние, движется по кривой, обвивающей этот цилиндр.

Отличие колебаний с большой амплитудой от малых колебаний сводится к тому, что закон изменения угла со временем отличен от гармонического, а частота их зависит от амплитуды.

При некоторой энергии колебания сменяются вращением.

Фазовая траектория, разделяющая на фазовой плоскости области, отвечающие колебаниям и вращению, называется *сепаратриссой*. (Точнее, это совокупность трех фазовых траекторий, из которых одна — просто точка.)

Мы будем сразу же использовать уравнения движения маятника, приведенные к безразмерному виду (3.1),(3.2). Свободное движение математического маятника без трения описывается дифференциальным уравнением

$$\ddot{x} + \sin x = 0. \tag{1}$$

Имеется аналитическое решение этого уравнения (довольно сложное), однако мы будем исследовать движение маятника численно.

Запишем это дифференциальное уравнение в виде системы уравнений первого порядка

$$\begin{aligned} \dot{x} &= p, \\ \dot{p} &= -\sin x. \end{aligned} \tag{2}$$

Основная идея численного расчета чрезвычайно проста: зная значения координаты и скорости в момент времени t, можно приближенно найти их значения через малый промежуток Δt

$$\begin{aligned} x(t+\Delta t) &= x(t) + \dot{x}(t)\Delta t \\ p(t+\Delta t) &= p(t) + \dot{p}(t)\Delta t, \end{aligned}$$

взяв значения величин $\dot{x}$ и $\dot{p}$ из уравнений (2). Многократно повторяя такие вычисления, мы найдѐм зависимости $x(t)$ и $p(t)$. Бо̀льшей точности можно достигнуть, практически не усложняя расчеты, если использовать так называемую вычислительную схему с перешагиванием. В этой схеме вычисляются значения координат в моменты времени

$$t - \frac{1}{2}\Delta t, \quad t + \frac{1}{2}\Delta t, \quad t + \frac{3}{2}\Delta t, \quad t + \frac{5}{2}\Delta t, ...,$$

а значения скоростей в моменты времени

$$t, \qquad t + \Delta t, \quad t + 2\Delta t, \quad t + 3\Delta t, ...$$

(подробнее об этом сказано в Приложении B). Данная вычислительная схема и использована в предлагаемой ниже программе.

Чтобы отклонение от точного решения уравнения (1) было небольшим, должен быть достаточно малым шаг по времени Δt. В любом случае необходим контроль правильности счета (т.е. того, что вычисленные зависимости $x(t)$ и $p(t)$ не слишком сильно отличаются от точных). Одним из методов контроля счета является контроль постоянства полной энергии системы (разумеется, если есть основания считать, что энергия сохраняется). Могут быть и другие сохраняющиеся величины, но часто вообще нет никаких интегралов движения (например, при наличии силы трения). В подобных случаях надежность счета можно проверить, применяя метод повторного счета с уменьшенным шагом Δt. Если при этом решение остается прежним, значит, шаг был выбран достаточно малым и можно быть уверенным в правильности результата.

Далее приводится текст простейшей программы на языке **MATLAB**, которая является исходной для начала работы над задачей МАЯТНИК.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Фазовая траектория математического маятника      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 clear;                       % Очистка рабочей области
% Задание начальных значений
 x1=2.2;                      % Начальная координата
 p1=0.0;                      % Начальный импульс
 dt=0.025;                    % Шаг по времени
 axis([-pi pi -pi pi]);       % Задание диапазона осей
 hl=line(x1,p1);              % Задание дескриптора линии
% Задание параметров выводимой линии
% 'EraseMode','none' - это режим вывода без стирания
% 'LineStyle',':'   - вывод линии в виде пунктира
% по умолчанию выводится сплошная линия
% 'Color','r'       - задание цвета линии
 set(hl,'EraseMode','none','LineStyle',':','Color','r');
 grid on;                     % Задание вывода координатной сетки
 pause;                       % Пауза, обеспечивает немедленный вывод
                              % рисунка на экран для продолжения программы
                              % необходимо нажать любую клавишу
 while 1                      % Бесконечный цикл
```

```
    % Основной алгоритм расчета
    x2=x1+ρ1*dt;
    ρ2=ρ1-sin(x2)*dt;
    % Склейка граничных условий
    if x2> ρi
      x2=x2-2*ρi;
    end;
    if x2< -ρi
      x2=x2+2*ρi;
    end;
   % Вывод очередного участка фазовой траектории
    set(hl,'XData',x2,'YData',ρ2);
   % Переприсвоение начальных значений
    x1=x2;ρ1=ρ2;
 end;
```

(Здесь шаг по времени Δt обозначен dt, pi=π.) Строго говоря, следовало бы перед началом основного цикла вставить операцию **x=x-p*dt/2**, имея в виду, что угол отклонения маятника **x** и скорость его изменения **p** в вычислительной схеме с перешагиванием относятся к разным моментам времени. Можно не делать этого, но следует учитывать, что, начальные значения x и p относятся к разным моментам времени.

Работа программы понятна из приведенных в ее тексте комментариев. Следует только сделать несколько замечаний. После запуска программа откроет графическое окно и нарисует в нем одно окно с системой координа, осями и координатной сеткой, после чего перейдет в режим ожидания (команда **pause**). Для продолжения расчета необходимо нажать любую клавишу, и программа начнет вывод фазовой траектории. Поскольку используется бесконечный цикл, выполнение программы можно прервать клавишей **Ctr+C**. Для повторного прогона с другими параметрами необходимо ввести соответствующие изменения в текст программы и перезапустить ее[11].

Задание 1. Уменьшая шаг **dt**, убедитесь, что фазовая траектория воспроизводится при этом без изменений в течение нескольких периодов (разумеется, с той точностью, какую допускает графическое изображение). Увеличивая шаг,

[11] **Можно, конечно, уже сейчас, используя опыт предыдущей задачи (Биения) и разработанные m-файлы, создать графический интерфейс и работать далее с ним. Но можно это сделать и позже или не делать вовсе. Это зависит от того, что проще — разрабатывать интерфейс или использовать примитивные приемы остановки и перезапуска программы.**

достигните такой его величины, чтобы появилось явно видимое искажение формы фазовой траектории.

Изобразите несколько разных фазовых траекторий.

Задавать начальные точки для вывода нескольких фазовых траекторий можно либо используя графический интерфейс (см. п. 2.3), либо с помощью повторного запуска программы, как это описано выше.

Постройте фазовые траектории разного типа, отвечающие колебаниям и вращению, а также сепаратриссу.

Включите в программу дополнительный цикл, который обеспечил бы построение целой серии фазовых траекторий с разными значениями начальной координаты (и/или импульса)[12].

Задание 2. Исследуйте, с какой точностью выполняется при расчете закон сохранения энергии. Удобно выводить зависимость энергии E от координаты x, чтобы точка (E, x) изображалась под точкой (p, x) (или над ней) в своем окне, а фазовая траектория - в своем. Для изображения двух разных графиков на одном листе можно открыть подрисунки на одном листе (функция **subplot**, см. п. 2.1). При этом для получения двух изменяющихся одновременно графиков можно использовать такой прием.

1. Вставить в начало программы определение двух подокон и определение двух разных дескрипторов для каждой из линий.

```
.....
subplot(2, 1, 1);          % определение 1-го подокна
axis([-pi pi -pi pi]);     % Задание диапазона осей
hl=line(x,p);              % Задание дескриптора 1-й линии
.....
subplot(2, 1, 2);          % определение 2-го подокна
axis([-pi pi 0.8 1.2]);    % Задание диапазона осей
he=line(x,E);              % Задание дескриптора 2-й линии
```

2. После соответствующих вычислений новых значений координат, импульсов и энергии внутри цикла нанести на каждый из подрисунков свою новую точку с помощью, например, такой последовательности операторов.

[12] **Интересно не писать внешний цикл, а сделать вектора x и p матрицами и написать аналог приведенной выше программы для матриц, выполняя тем самым в одном цикле расчет целого семейства траекторий.**

```
.....
set(hl,'XData',x,'YData',ρ);
set(he,'XData',x,'YData',E);
.....
```

Задание 3. Постройте изображение качающегося (в соответствии с расчетом) маятника, а также зависимость $x(t)$. При изображении качающегося маятника обратите внимание на различия в задании свойства 'EraseMode' - 'xor', 'background' или 'none'. Для построения движущегося отрезка, изображающего маятник, рекомендуется присвоить (с помощью оператора **set**) свойству 'EraseMode' значение 'background' для соответствующего объекта.

Изобразите также (в отдельном окне) зависимость x(t), чтобы можно было видеть движение маятника одновременно в трех разных формах.

Задание 4. Получите зависимость частоты колебаний маятника от амплитуды. Для этого удобно начальную точку фазовой траектории (x_0, p_0) задавать на оси $p = 0$ и определять полупериод колебаний $T/2$, подсчитывая число шагов, необходимых для смены знака у импульса p.

Для представления зависимости $\omega(x_0)$ удобно выделить отдельное графическое окно.

Задание 5. Включите в программу силу трения, пропорциональную скорости. Коэффициент пропорциональности следует включить в число параметров, доступных оперативному изменению (если вы работаете с графическим интерфейсом).

4.2. Вынужденные колебания

Будем рассматривать только колебания маятника под действием гармонической силы (и с учетом силы трения), а также будем иметь в виду ту модель, которая описана в (1). В этом случае безразмерное уравнение движения имеет вид

$$\ddot{x} = -\sin x - 2\lambda \cdot \dot{x} + f \cdot \cos x \cdot \sin \Omega t. \qquad (3)$$

4.2.1. Переходные колебания

Если маятник в начальный момент покоился, а в дальнейшем на него действует периодическая сила, то в течение некоторого времени он раскачивается, происходит нерегулярное движение. При этом возможен не только рост амплитуды, но

и ее периодическое нарастание и убывание. Спустя же какое-то время движение становится установившимся, периодическим, причем с частотой внешней силы Ω. Приближение к такому режиму асимптотическое, т.е. происходит, строго говоря, бесконечно долго, тем не менее вполне можно указать интервал времени, спустя который нерегулярные процессы оказываются незаметны (с обусловленной точностью). Обычно для этого нужно примерно такое же время, как для затухания свободных колебаний.

Задание 6. Получите различные режимы переходных колебаний — с монотонным ростом амплитуды и с ее осцилляциями. При этом предпочтительно выбрать небольшое значение амплитуды силы f, такое, чтобы даже попав в резонанс, эта сила не приводила маятник во вращение. Коэффициент в силе трения должен быть не слишком велик, чтобы свободные колебания затухали в течение многих периодов. Одна из возможностей наблюдения эффекта — выводить зависимость $x(t)$ в таком масштабе, чтобы график был сильно сжат в направлении оси t[13].

4.2.2. Резонанс

Если уравнение (1) для свободных колебаний математического маятника в принципе решается аналитически (решение выражается через специальные функции), то уравнение вынужденных колебаний с трением не решается и его можно исследовать качественно или численно. До начала численного решения полезно рассмотреть качественные особенности предполагаемого решения. Рассмотрим случай малых колебаний и установившийся (стационарный) режим. При малых углах отклонения маятника ($x \ll 1$) несложно проанализировать зависимость амплитуды установившихся колебаний маятника под действием гармонической силы $F(t) = f\,cos\,\omega t$ от частоты силы ω[14].

Если заменить $\sin\ x$ на x, то получим линейное относительно x уравнение (гармоническое приближение):

$$\ddot{x} + 2\lambda\dot{x} + x = f\cos\,\omega t.$$

Его решение удобно искать в виде

$$x = Ae^{i\omega t} + A^*e^{-i\omega t}, \tag{4}$$

[13] **В этом задании предполагается ограничиться исследованием, не затрагивающим всерьез особенности переходных процессов, связанные с нелинейными эффектами.**

[14] **Множитель** $\cos(x)$ **не играет роли в рассматриваемом нами процессе, поэтому мы этой зависимостью пренебрегаем, тем более, что при** $x \ll 1$ $\cos(x) \approx 1$**.**

представив при этом силу в форме $F = (f/2)(e^{i\omega t} + e^{-i\omega t})$. Приравняв слагаемые с одинаковой зависимостью от времени, получим

$$A = \frac{f}{2(1 - \omega^2 + 2i\lambda\omega)}\ , \tag{5}$$

откуда

$$x = a \cdot \cos(\omega t + \delta),$$

где

$$a = 2 \cdot |A| = f/\sqrt{(1 - \omega^2) + 4\lambda^2\omega^2}\,,$$

$$\delta = arcctg\frac{\omega^2 - 1}{2\lambda\omega}.$$

Учтем теперь следующий (ангармонический) член разложения $\sin x$, по-прежнему считая угол x малым:

$$\ddot{x} - 2\lambda\dot{x} + x - x^3/6 = f\cos\,\omega t.$$

Дополнительные слагаемые, получаемые за счет подстановки (4) в член $x^3/6$, будут иметь вид

$$x^3/6 = (A^3e^{3i\omega t} + 3A^2A^*e^{i\omega t} + 3AA^{*2}e^{-i\omega t} + A^{*3}e^{-3i\omega t})/6,$$

Слагаемые, пропорциональные $e^{\pm 3i\omega t}$, приведут к появлению в (4) малых добавок более высокого порядка. Это можно представить себе как появление в правой части уравнения силы, пропорциональной $\cos\ 3\omega t$, а поскольку их частота далека от резонанса, то и вклад их будет мал.

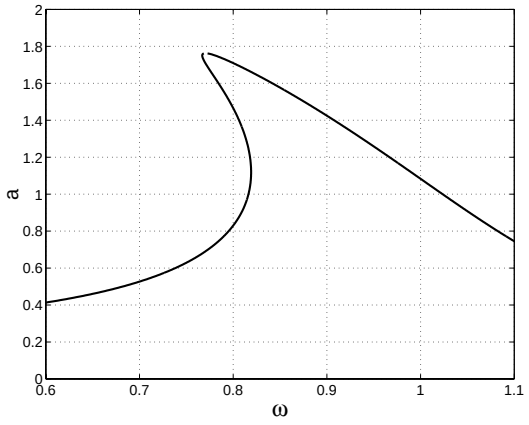


Рис. 6. Резонансная кривая при наличии гистерезиса

Теперь, как и раньше, собирая и приравнивая члены, имеющие одинаковую зависимость от времени, например при $e^{i\omega t}$, вместо (5) получаем $(1 - \omega^2 + 2i\lambda\omega + |A|^2/2)A = f/2$, откуда получаем уравнение

$$[(1 - \omega^2 + |A|^2/2)^2 + 4\lambda^2\omega^2]|A|^2 = f^2/4. \tag{6}$$

Это уравнение разрешаем относительно ω^2, после чего зависимость $a(\omega)$ легко представить с помощью **MATLAB** графически (рис 6). Аналитическое исследование этой зависимости приведено, например, в [7].

Зависимость $a(\omega)$ представляется существенно разной при различных значениях λ и f. Однако при относительно больших амплитудах сделанное нами при выводе (3) предположение о малости добавок к гармоническому колебанию нарушается, так что следует ожидать отклонения от этих кривых тех значений амплитуд, которые мы можем найти в компьютерном эксперименте (и которые являются, разумеется, истинными).

Появляются такие участки кривой, которые могут быть достигнуты при непрерывном изменении частоты либо только путем ее повышения, либо только путем понижения.

Для наблюдения такого явления нужно обеспечить, чтобы при изменении частоты Ω на малую величину $\Delta\Omega$ не происходило резких скачков силы $f \cdot \cos x \cdot \sin \Omega t$. Тогда переходной процесс будет завершаться быстро и точка $x_0(\omega)$ будет перемещаться вдоль одной ветви резонансной кривой (разумеется, исключая тот случай, когда скачок на другую ветвь становится неизбежен).

Чтобы задать такое плавное изменение силы, нужно избежать скачка фазы силы Ωt при изменении частоты $\Omega \to \Omega + \Delta\Omega$. Для этого в программе достаточно ввести переменную, равную этой фазе, скажем **wt**, и на каждом шаге по времени наращивать именно эту фазу: **wt=wt+dwt**, где **dwt=w*dt** — приращение фазы. При изменении же частоты **dw**$= \Delta\Omega$ следует просто изменять приращение фазы: **dwt=dwt+dw*dt**[15].

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Программа отрисовки зависимости стационарной амплитуды
% от частоты вынуждающей силы
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Обратите внимание на то, что lam -- это половина коэффициента
% при импульсе в выражении силы трения!

clear;
f=0.26;          % Амплитуда вынуждающей силы
lam=0.095;       % Половина коэффициента при импульсе
                 % в выражении силы трения!
a=0.05:0.001:1.0; aa=a.^2;
d=4*lam^4-4*lam^2*(1-0.5.*aa)+f^2./(4*aa);
k=find(d>=0);
aa=aa(k);
```

[15] Тем самым в качестве фазы будет использована величина $\int_0^t \omega(t)dt$.

```
a=a(k);
d=d(k);
w1=sqrt(1-2*lam^2-0.5.*aa-sqrt(d));
w2=sqrt(1-2*lam^2-0.5.*aa+sqrt(d));
h=plot(w1,2*a, w2, 2*a);
axis([0.6 1.0 0 2])
```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Задание 7. Получите зависимости амплитуды колебаний маятника от частоты внешней силы в случае малых колебаний и в случае, когда наблюдаются "скачки"амплитуды. Наложите эту зависимость на "теоретическую", полученную с помощью уравнения (6).

4.2.3. О случайном движении

Упомянем, наконец, об интересном явлении, имеющем место в исследуемой механической системе. При движении вблизи сепаратриссы (в отсутствие трения) маятник много времени проводит близко к вертикальному, неустойчивому положению грузом вверх. В этих условиях даже очень маленькое изменение начальных значений угла или скорости может привести к тому, что он пройдет через вертикальное положение и сделает оборот, вместо того чтобы качнуться обратно. Очень маленькое отклонение вырастает во много раз! Поскольку расчеты ведутся непременно с округлением, то спустя некоторое время начальные условия "забываются". Движение в ближайшей окрестности сепаратриссы оказывается в определенном смысле случайным.

Разумеется, если задать те же начальные значения x и p, при повторном счете зависимости $x(t)$ и $p(t)$ воспроизведутся полностью.

Включение же внешней периодической силы может заметно расширить эту область случайного движения на фазовой плоскости.

Однако наблюдать указанное явление непросто, так как непросто отличить истинно случайное движение от регулярного, но кажущегося случайным. Подробнее вопрос о возникновении хаотического движения в детерминированной системе будет затронут в работе “ШАРЫ”.

5. Движение частиц в центральном поле

Закон движения частицы $\mathbf{r}(t)$ в заданном поле $U(\mathbf{r})$ полностью определяется уравнением движения

$$m\ddot{\mathbf{r}} = -\frac{\partial U}{\partial \mathbf{r}} = \mathbf{F}(\mathbf{r}) \tag{1}$$

и начальными условиями $\mathbf{r}(0) = \mathbf{r}_0, \mathbf{v}(0) = \mathbf{v_0}$. В некоторых случаях дифференциальные уравнения (1) можно решить аналитически, в других возможно качественное исследование. Однако в сколько-нибудь сложных полях $U(\mathbf{r})$ приходится применять численные расчеты.

5.1. Траектория финитного движения

В данной работе изучается, в первую очередь, движение в центральных полях. Чтобы ориентироваться в ожидаемых результатах, необходимо вспомнить качественные способы исследования движения в центральном поле (см. [7]). В этом случае сохраняется момент импульса $\mathbf{M} = m[\mathbf{rv}]$, а траектория лежит в плоскости, перпендикулярной вектору $\mathbf{M}$. Мы ограничиваемся таким случаем, когда частица не удаляется от центра неограниченно (*финитное* движение) и не падает в центр поля. Тогда, вообще говоря, траектория оказывается расположенной между двумя окружностями $r_{min} \leq r \leq r_{max}$ и заполняет это кольцо, проходя как угодно близко к любой его точке.

Финитное движение в кулоновском поле $U_0 = -\alpha/r$, в отличие от этого общего случая, есть движение по замкнутой траектории – по эллипсу с полуосями: большой $a = \alpha/2 \mid E \mid$ и малой $b = M/\sqrt{2m \mid E \mid}$.

В других центральных полях траектории при выбранных наудачу значениях энергии и момента импульса окажутся почти замкнутыми лишь после очень большого числа оборотов [16].

Самый простой метод численного решения уравнения (1) (метод Эйлера) состоит в том, чтобы, выбрав малый "шаг"Δt, непосредственно определить приращения координат и скорости равенствами

$$\begin{aligned} \Delta\mathbf{r} &= \mathbf{v}\Delta t, \\ \Delta\mathbf{v} &= \frac{1}{m}\mathbf{F}(\mathbf{r}(t))\Delta t. \end{aligned} \tag{2}$$

[16] **Поле $U \propto r^2$, которое в данной работе не рассматривается, обладает такой же особенностью, как кулоновское.**

Многократно повторяя такое вычисление, получим ряд последовательных значений координат и скорости.

Гораздо большей точности можно добиться, если вместо (2) использовать разностную схему с перешагиванием

$$\Delta\mathbf{v} = \frac{1}{m}\mathbf{F}(\mathbf{r}(t+\Delta t)), \tag{3}$$

т.е. при вычислении силы использовать новые значения координат в момент времени $t + \Delta t$ (подробнее об этом см. в Приложении B).

Далее приводится отрывок текста программы, в которой реализован описанный выше алгоритм для вычисления и изображения на экране траектории частицы в кулоновском поле. В программе **alpha** обозначает α/m, а **dt**$= \Delta t$, **v=(vx,vy)**, $\mathbf{F}/m = \mathbf{a} =$**(ax,ay)**.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Учебная программа для решения задачи Планеты   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 clear;                          % Очистка рабочей области
% Задание начальных значений
 Alpha=1; dt=0.025;
x1=2.5; y1=0;
vx1=0; vy1=0.25;
Emax=-1e18;
Emin=1e18;
 r=sqrt(x1^2+y1^2);              % Расстояние до центра
 E1=(vx1^2+vy1^2)/4 ...
   - Alpha/r;                    % Полная энергия (почему так?)
 hl=line(x1,y1);                 % Задание дескриптора линии
 a=Alpha/(2*abs(E1));            % Большая полуось
 b=r*vy1/sqrt(2*abs(E1));        % Малая полуось
 axis([-2*a 2*a -1.2*b 1.2*b]);  % Задание масштаба
 set(hl,'EraseMode','none','Color','g');
ha=gca;
% Задание разметки осей
 set(ha,'XTick',[-a 0 a],'YTick',[-b 0 b]);
grid on;                         % Введение сетки
 pause                           % Пауза для обязательного вывода
 while 1                         % Бесконечный цикл
```

```
    x2=x1+vx1*dt;                  % Новые координаты
    y2=y1+vy1*dt;                  %
    r=sqrt(x2^2+y2^2);             % Расстояние до центра
    A = Alpha/r^2;                 % Модуль ускорения
    ax=-A*x2/r;                    % Компоненты вектора
    ay=-A*y2/r;                    % ускорения
    vx2=vx1+ax*dt;                 % Новые скорости
    vy2=vy1+ay*dt;                 %
    E2=((vx2+vx1)^2+(vy2+vy1)^2)/8 ...
      -Alpha/r;                    % Полная энергия
    if E2>  Emax                   % Границы изменения
      Emax=E2;                     % полной энергии
    end;
   if E2< Emin
     Emin=E2;
   end;
  % Обновление данных на графике
    set(hl,'XData',[x1 x2],'YData',[y1 y2]);
  % Переприсвоение
    x1=x2;
   y1=y2;
   vx1=vx2;
   vy1=vy2;
 end;                              % конец цикла 'while 1'
```

Работа программы понятна из приведенных в ее тексте комментариев. Следует только сделать несколько замечаний. Поскольку используется бесконечный цикл, выполнение программы можно прервать клавишей **Ctr+C**. Следует помнить, что в **MATLAB** все данные хранятся в рабочей области до тех пор, пока Вы не выйдете из командного окна. Поэтому после первого прогона программы можно остановить вывод командой **Ctr+C**, поменять требуемые параметры (например, шаг по времени), задать режим **hold on**(рисовать поверх), и снова запустить программу. График будет выведен на тот же рисунок[17].

Прежде чем выполнять задания, обсудим некоторые особенности начальной программы расчета траектории планеты. Прежде чем воспользоваться возможно-

[17] **Можно, конечно, уже сейчас, используя опыт предыдущей задачи (Биения, см. п. 2.3) и разработанные m-файлы, создать графический интерфейс и работать далее с ним. Но можно это сделать и позже или не делать вовсе. Это зависит от того, что проще – разрабатывать интерфейс или использовать примитивные приемы остановки и перезапуска программы.**

стями анимации (см. п. 2.4.2), необходимо приготовить дескриптор той линии или линий, на который потом можно будет ссылаться при смене координат. Кроме того, задается масштаб по осям и для проведения линий сетки только в определенных местах графика (а не по умолчанию) явно определяются места разметки осей (оператор **set(ha,'XTick',[-a 0 a],'YTick',[-b 0 b])** и следующий за ним оператор **grid on**.

Задание 1. Исследуйте влияние величины шага Δt на точность расчета траектории. При слишком большом значении Δt возникает прецессия орбиты, связанная с неточностью расчета.

Для оценки точности вычислений выведите на экран значение полной энергии в зависимости от координаты x. Это лучше сделать, разбив область графического вывода на две подобласти — для построения орбиты и для построения энергии. Это можно сделать с помощью функции **subplot** или явным заданием областей вывода графиков.

Пометьте на траектории другим цветом точки наибольшего и наименьшего удаления от центра.

Задание 2. Исследуйте влияние на траекторию частицы возмущения вида $\delta U = \beta/r^2$.

Предусмотрите вывод на экран «центра тяжести» витка траектории (виток — участок от одной точки траектории $r = r_{min}$ до следующей).

Если представить, что движущаяся частица — электрон в атоме, то интересующий нас «центр тяжести» определяет среднее по времени значение дипольного момента. Речь идет, таким образом, о центре тяжести отрезка кривой с учетом времени, проводимого частицей на каждом его участке.

Для исследования точности расчетов можно в некоторый момент изменить направление движения частицы на строго противоположное. В таком случае частица должна будет двигаться по уже «проложенной» траектории в обратном направлении. Проследить, насколько это выполняется, можно, изменив цвет следа.

5.2. Влияние малого возмущения

Интересно исследовать, как влияет на движение частицы малая добавка δU к потенциальной энергии U_0. Если добавка δU зависит только от $r = |\mathbf{r}|$, то поле $U = U_0 + \delta U$ остается центральным, а траектория перестает быть замкнутой и ее

можно представлять как *прецессирующий* эллипс, постепенно заполняющий кольцо $r_{min} \le r \le r_{max}$. Если же мы стартуем от поля, уже отличного от кулоновского, то та же добавка δU приведет к изменению скорости прецессии орбиты, но не вызовет качественного изменения траектории.

Таким образом, кулоновское поле в сравнении с другими центральными полями обладает той особенностью, что траектория финитного движения в нем оказывается гораздо более чувствительной к «возмущению» δU.

Подобная особенность кулоновского поля становится более яркой, если в качестве возмущения δU выбрано нецентральное поле. Например, добавление к кулоновскому очень слабого однородного поля, не параллельного плоскости орбиты, $\delta U = -\mathbf{fr}$, приводит не только к существенным деформациям орбиты, но и к значительным поворотам ее плоскости.

То же слабое однородное поле δU исказит траекторию в другом исходном центральном поле (скажем, $U_0 = -\alpha/r + \beta/r^2$) гораздо меньше[18]. В частности, не будет значительного поворота плоскости орбиты!

Такая повышенная чувствительность орбиты в кулоновском поле к возмущениям связана в конечном счете с тем, что орбита в кулоновском поле при любых значениях M и $E < 0$ замкнутая.

Чтобы понять причину таких различий, проследим за изменением момента импульса. Скорость его изменения равна моменту силы, действующей на частицу (причем силу притяжения к центру можно исключить): $\dot{\mathbf{M}} = [\mathbf{rf}]$. Приращение вектора $\mathbf{M}$ за один период движения в кулоновском поле мало, но за несколько периодов подобные приращения накапливаются и приводят к большому его изменению. В результате сильно изменяются положение плоскости орбиты (определяемое направлением $\mathbf{M}$) и малая полуось эллипса (определяемая его величиной). Если же исходное поле $U_0(r)$ заметно отличается от кулоновского, то траектория представляет собой "прецессирующий эллипс", и приращения $\mathbf{M}$ за несколько периодов взаимно компенсируются, так что плоскость орбиты лишь слегка покачивается.

Задание 3. Исследуйте влияние на орбиту частицы в кулоновском поле слабого однородного поля $\delta U = -\mathbf{fr}$. Если сила лежит в плоскости орбиты, то орбита искажается, постепенно превращаясь в вырожденный эллипс — отрезок. При этом происходит нарушение точности счета. Аккуратный расчет требовал бы в этом случае уменьшения шага и перехода к другой, более сложной вычислительной схеме.

[18] **Подразумевается, что возмущение δU намного меньше, чем отличие поля U_0 от кулоновского β/r^2.**

Лучше обойти эти трудности, исследуя движение в пространстве и под действием возмущающей силы $\mathbf{f}$, не параллельной плоскости траектории. В этом случае орбита искажается и поворачивается, но в отрезок не вырождается.

Для наблюдения орбиты, если она оказывается пространственной кривой, удобно изображать сразу три ее проекции на плоскости (X,Y), (Y,Z) и (X,Z), разместив их в трех расположенных рядом графических окнах. (Необходимо согласовать направление осей координат в разных окнах друг с другом.) Можно также с помощью свойства кривой 'ZData' строить пространственное изображение траектории (подробнее о построении трехмерной кривой см. Дополнение, п. 8.3).

В этом случае становится ясно, что анализ результатов — задача не менее сложная, чем построение модели. (Аналогично обстоит дело и в обычном, «железном» эксперименте.)

Задание 4. Повторите исследование влияния возмущения $\delta U = -\mathbf{fr}$ на финитное движение, если исходное поле сильно отличалось от кулоновского. Можно положить, например, $U_0 = -\alpha/r + \beta/r^2$.

5.3. Движение двух частиц

Движение нескольких взаимодействующих друг с другом тел лишь в исключительных случаях можно рассчитать аналитически. В то же время численный расчет оказывается совсем немногим более сложен, чем для одного тела, т.е. вполне доступен. При исследовании возможных движений в этом случае наиболее сложной частью задачи оказывается представление результатов — перебор и описание различных видов движения, понимание, какие из движений нужно считать качественно различными. Разумеется, такая постановка вопроса не предполагает однозначного ответа, да и исследование в рамках данного практикума не может быть полным. Это скорее иллюстрация наличия практически неограниченного числа возможностей.

Задание 5. Изобразите на экране движение одновременно двух точек в поле $U = -\alpha/r$.

Получите траекторию одной точки с точки зрения другой (т.е. траекторию Венеры, например, с точки зрения земного наблюдателя).

Введите взаимодействие частиц друг с другом: $U_{12} = \beta/|\mathbf{r_1} - \mathbf{r_2}|$.

"Выключив"поле $U(r)$ и сохраняя лишь взаимодействие двух тел, получите их траектории.

6. Случайные блуждания и диффузия

Хаотическое движение броуновских частиц в жидкости или газе представляет собой пример случайных блужданий. Теория броуновского движения была построена А.Эйнштейном и М.Смолуховским в 1905 - 1906 гг.

Задача о случайных блужданиях является одной из широко исследуемых задач теории вероятностей и находит множество других приложений.

6.1. Закономерности случайных блужданий

Закономерности случайных блужданий можно понять, используя простую модель, которая легко реализуется с помощью компьютера.

N частиц (которые в начальный момент для удобства наблюдений распределены на оси y) смещаются последовательными шагами Δx вдоль оси x. Каждый шаг каждой частицы выбирается случайным и независимым от других шагов. Однако распределение вероятностей при выборе любого шага одно и то же. Примем, что смещения в противоположные стороны равновероятны. Это значит, что среднее значение смещения

$$\langle \Delta x \rangle = 0. \tag{1}$$

Смысл этого равенства в том, что среднее арифметическое смещений Δx очень большого числа частиц приближается к нулю по мере роста этого числа. Так понимается усреднение и далее. Иногда такие средние величины называют *априорными* [19]. Кроме того, мы будем использовать «наблюдаемые средние» – средние арифметические для заданного числа частиц (как правило, очень большого). «Наблюдаемое среднее» смещения частицы $\langle \Delta x \rangle_{\text{н}}$ мало, но не равно нулю.

После каждого шага частицы будут «расползаться» от оси y. Обозначим $x(k)$ координату некоторой частицы через k шагов. Тогда

$$x(k+1) = x(k) + \Delta x. \tag{2}$$

Усреднив это равенство (вновь по множеству частиц), получаем

$$\langle x(k+1) \rangle = \langle x(k) \rangle,$$

т.е. среднее значение $\langle x(k) \rangle$ не изменяется от шага к шагу и, следовательно, равно $\langle x(0) \rangle = 0$. Наблюдаемое значение $\langle x \rangle_{\text{н}}$ для большого числа частиц

$$\langle x(k) \rangle_{\text{н}} = \frac{1}{N} \sum_{j=1}^{N} x_j(k) \tag{3}$$

[19] **Например, приступая к какой-то жеребьевке с помощью подбрасывания монеты, мы заранее предполагаем, что вероятность выпадения «орла» равна 1/2.**

окажется близким к нулю (здесь x_j - координата j-й частицы)[20].

Ширину полосы, по которой распределяются частицы после k-го шага, удобно характеризовать величиной $\langle x^2(k)\rangle$. Чтобы выяснить зависимость этой величины от числа шагов, возведем равенство (2) в квадрат и усредним:

$$\langle x^2(k+1)\rangle = \langle x^2(k)\rangle + 2\langle x(k)\Delta x\rangle + \langle(\Delta x)^2\rangle. \tag{4}$$

Ввиду независимости $x(k)$ и Δx имеем

$$\langle x(k)\Delta x\rangle = \langle x(k)\rangle\langle\Delta x\rangle = 0.$$

Обозначим $\langle(\Delta x)^2\rangle = a^2$. Из (4) следует

$$\langle x^2(k+1)\rangle = \langle x^2(k)\rangle + a^2,$$

т.е. средний квадрат координаты растет с каждым шагом на величину a^2. Значит,

$$\langle x^2(k)\rangle = ka^2. \tag{5}$$

Наблюдаемое значение

$$\langle x^2\rangle_{\text{н}} = \frac{1}{N}\sum_{j=1}^{N} x_j^2 \tag{6}$$

изменяется приблизительно пропорционально числу шагов.

Распределение частиц в занятой ими полосе более детально характеризуется функцией распределения $f(x)$, определяющей концентрацию частиц; $dW = f(x)dx$ — вероятность того, что координата j-й частицы после k-го шага окажется в интервале $x \le x_j \le x + dx$. Теория случайных блужданий дает для достаточно большого числа шагов k распределение Гаусса

$$f(x) = \frac{1}{\sqrt{2\pi ka^2}}\exp\left(-\frac{x^2}{2ka^2}\right). \tag{7}$$

Наблюдаемая функция распределения получается путем разбиения оси x на конечные интервалы и подсчета числа частиц в каждом из них. Результат подсчета представляется графически ступенчатой кривой — *гистограммой* (рис. 7).

Обратим внимание на одно свойство зависимости (5). Если укрупнить шаги по времени в l раз, то средний квадрат смещения за один шаг a^2 следует в соответствии с (5) заменить на $\tilde{a^2} = la^2$, а число шагов k — на $\tilde{k} = k/l$. В итоге $x^2(k) = la^2 \cdot k/l = \tilde{a^2}\tilde{k}$, т.е. вид зависимости (5) не изменяется при укрупнении шагов.

[20] **При заданном числе частиц N это справедливо для не слишком большого номера шага k.**

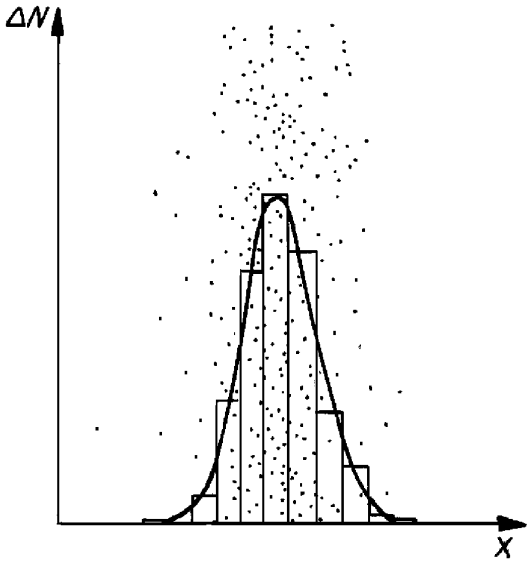


Рис. 7. Распределение частиц при диффузии (гистограмма и теоретическая кривая)

6.2. Оценка параметров движения броуновской частицы в жидкости

Приведем оценки для реального броуновского движения. Средняя скорость хаотического движения броуновской частицы v_T определяется так же, как средняя скорость молекулы, соотношением

$$mv_T^2 \sim k_Б T, \tag{8}$$

где m – масса частицы, $k_Б$ – постоянная Больцмана, T – абсолютная температура среды. Если скорость частицы $v \gg v_T$, то ее движение определяется уравнением

$$m\dot{\mathbf{v}} = \mathbf{F},$$

где $\mathbf{F} = -\alpha\mathbf{v}$ – сила трения[21]. Частица тормозится, и время τ, за которое ее скорость существенно уменьшится, можно оценить, подставив $\dot{v} \sim v/\tau$:

$$mv/\tau \sim \alpha v,$$

откуда

$$\tau \sim m/\alpha. \tag{9}$$

Если же скорость частицы близка к тепловой, $v \sim v_T$, то и сила, естественно, гораздо меньше, а отклонения ее от среднего значения $-\alpha v$ весьма существенны.

[21] **Для шарика радиуса R в жидкости с коэффициентом вязкости η согласно закону Стокса $\alpha = 6\pi\eta R$.**

Именно эти отклонения ответственны за безостановочное хаотическое движение частицы. Если речь идет о таком движении, то τ из (9) можно понимать как оценку времени, спустя которое частица «забывает» начальное направление движения. Но та же величина τ дает грубую оценку интервала времени, в течение которого частица «помнит» направление движения. (Может быть, для оценки времени «гарантированного забывания» стоило бы взять 2τ, а для оценки времени гарантированного сохранения направления $\tau/2$, но нас интересуют не «гарантированные», а средние времена, поэтому будем полагать, что коэффициенты 2, $1/2$ и т.п. лежат за пределами принятой точности оценок.)

За время τ частица проходит путь, равный по порядку величины

$$a \sim v_T \tau. \tag{10}$$

Смещения частицы за различные интервалы времени порядка τ мы можем рассматривать как случайные, подобные рассматривавшимся ранее Δx, только направленные не вдоль оси x, а в произвольном направлении (например, как три одновременных и независимых смещения вдоль трех осей координат). Движение частицы за время $t \gg \tau$ можно разбить на $k \sim t/\tau$ таких шагов. Смещение частицы за время t оценивается по аналогии с (5):

$$r^2(t) \sim k a^2 \sim (v_T \tau)^2 \frac{t}{\tau} \sim v_T^2 \tau\, t. \tag{11}$$

Этот результат обычно представляют в виде

$$\langle r^2(t) \rangle = 6 D\, t, \tag{12}$$

где D — коэффициент диффузии[22]. С учетом (8), (9), (11)

$$D \sim \frac{k_\text{Б} T}{\alpha}. \tag{13}$$

Если сначала частицы были сосредоточены в каком-то малом объеме, то со временем они распространяются все дальше, занимая область размера $\sim r(t)$.

Соотношения вида (12), (13), полученные Эйнштейном и Смолуховским, послужили основой экспериментов Перрена, в ходе которых была определена масса атомов и которые были приняты «научной общественностью» в качестве убедительного доказательства существования атомов.

Описанные выше закономерности следует понимать как предельный случай, отвечающий наблюдению бесконечного числа частиц. Реализация же случайных блужданий конечного числа частиц, совершающих броуновское движение (настоящих или «компьютерных»), демонстрирует лишь приближенное выполнение этих соотношений.

[22] **Для случайных блужданий в направлении оси x вместо (12) имеем $\langle x^2(t) \rangle = 2 D\, t$.**

6.3. Программа, изображающая случайные блуждания

В приводимой ниже программе смещения задаются датчиком случайных чисел. В качестве смещения задается величина $\Delta x = dh * (2.0 * rand - 1.0)$, где **rand** – квазислучайное число, вырабатываемое компьютером, **dh** – параметр случайного блуждания, масштаб «шага». При каждом обращении к процедуре **rand(m,n)** компьютер выдает матрицу **m x n** случайных чисел, лежащих в интервале от 0 до 1, причем для нашей задачи (и для множества других) выбор этих чисел неотличим от случайного, а распределение этих чисел в указанном интервале равномерное. Легко видеть, что распределение величин Δx окажется равномерным в интервале $-dh < \Delta x < dh$:

$$\frac{dw}{d\Delta x} = \begin{cases} 0 & \text{при } |\Delta x| > dh, \\ \frac{1}{2dh} & \text{при } |\Delta x| < dh. \end{cases} \tag{14}$$

Средний квадрат смещения при одном шаге

$$\langle(\Delta x)^2\rangle = \int\limits_{-\infty}^{\infty} (\Delta x)^2 \frac{dw}{d\Delta x} d\Delta x = \frac{dh^2}{3}. \tag{15}$$

Получаемые нами смещения на первых одном – двух шагах совсем не похожи на броуновское движение, так как функция распределения еще далека от гауссовой, однако спустя три – пять шагов функция распределения начинает отлично имитировать гауссову и смещения становятся практически такими же, как блуждания броуновских частиц. Так и должно получиться согласно теории вероятностей.

Предлагаемые ниже задания имеют главной целью иллюстрацию описанных закономерностей.

Задание 1. Получите на экране картину движения точек, моделирующих случайные блуждания в соответствие с описанной выше функцией распределения частиц по координате x. Для этого воспользуйтесь начальным вариантом программы **diffus.m**, имеющимся в пакете **MPP**, который приведен далее.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Стартовая программа для демонстрации случайных %
% блужданий                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

n =500;                      % Число частиц
dh =.02;                     % Параметр случайного распределения
% Задание вектор-столбцов координат точек
```

```
y =1:n;
y=y‘; x =zeros(size(y));
h=plot(x,y,’k.’);                  % Вывод начального положения точек
axis([-2 2 0 n+1 ]);               % Задание осей
% Определение режима перерисовки и размера точек
set(h,’EraseMode’,’background’,’MarkerSize’,3);
pause                              % Пауза для вывода графика на экран
i=0;                               % Начальное значение
while 1                            % Бесконечный цикл
i=i+1;
x=x+dh*(2*rand(n,1)-1);            % Случайные смещения x-координаты
                                   % каждой точки
% Смена координат точек на рисунке
set(h,’XData’,x,’YData’,y,’Color’,’k’)
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Задание 2. Выведите на экран гистограмму распределения частиц по координате x. Это можно сделать на том же рисунке, что и вывод самих частиц, или открыть для отрисовки гистограммы отдельное подокно (см. 2.1) или, подробно, в Дополнении (п. 8) описание функции **subplot**. При этом следует обратить внимание на то, что функция для отрисовки гистограммы (например **hist**) при использовании в стандартном виде отрисовывают гистограмму сразу после вызова, но как все функции верхнего уровня постоянно перерисовывают оси и, следовательно, вместо анимации получаются мигающая картина. Для построения нормально работающей динамической картины необходимо использовать функцию **hist** в виде **[n,x]=hist(y,m)**, что позволяет сначала насчитать параметры гистограммы, а потом, используя функции **stairs, line**, построить динамическую гистограмму с помощью оператора **set**, как это делалось ранее в п. 2.4.2 . Подробнее функции, используемые при отрисовке гистограмм, описаны в Дополнении, п. 8.2. Если при попытке нарисовать динамическую гистограмму возникнут проблемы, то можно воспользоваться помещенной в директорию **MPP** функцией **Hist_my** в качестве образца.

Выведите на экран кроме гистограммы «теоретическую» функцию вида (7). Для отрисовки функции можно либо насчитывать точки функции в виде вектора и использовать **line**, либо использовать функцию **fplot** (Дополнение,

п. 8). При этом следует выбирать масштабы так, чтобы на экране площадь под этой кривой была равна площади под гистограммой.

Площадь под гистограммой равна

$$S_{\text{гист}} = (xmax - xmin)N/L,$$

где xmin, xmax - область отрисовки гистограммы по оси x, N - число частиц, а L - число бинов, на которые разбивается ось x. Чтобы согласовать указанным образом масштаб функции (7), следует изображать функцию $w = S_{\text{гист}} f(x)$ в том же масштабе, что и гистограмму: $(xmin \le x \le xmax, 0 \le w \le wmax)$.

Сохранится ли описанное согласование масштабов, если значительная доля частиц «расползется» за пределы интервала [**xmin, xmax**]?

Задание 3. Получите на экране графики $\langle x \rangle_{\text{н}}$ и $\langle x^2 \rangle_{\text{н}}$ в зависимости от числа шагов.

Напишите функцию, реализующую метод наименьших квадратов (см. Приложение C), и с ее помощью рассчитайте коэффициенты соответствующих аппроксимирующих прямых. Постройте прямые, наилучшим образом аппроксимирующие зависимости $\langle x \rangle_{\text{н}}$ и $\langle x^2 \rangle_{\text{н}}$ от числа шагов.

Обратите внимание, что при небольшом числе частиц начинаются существенные отклонения от закона (5). Попытайтесь сделать соответствующие оценки и проверить их в ходе машинного эксперимента.

Задание 4. Получите на экране двумерную картину случайных блужданий частиц, вышедших из одной точки.

Если мы разобьем плоскость x, y на кольца одинаковой ширины и подсчитаем количества частиц в каждом из колец, то получим функцию распределения по расстоянию от начала координат $R = \sqrt{x^2 + y^2}$. Постройте функцию распределения по R.

Заметим, что предложенный закон блужданий обладает анизотропией. Например, после первого шага частицы заполнят квадрат (а не круг, как было при изотропных блужданиях). Однако спустя несколько шагов облако частиц становится изотропным. Это легко проверить, анализируя формулу (7).

Используя (7), получите и выведите на экран теоретическую функцию распределения по R.

Получите на экране зависимость концентрации частиц от R (в виде гистограммы). Для этого количество частиц в каждом из колец, найденное с помощью процедуры hist, следует разделить на площадь этого кольца и лишь затем воспользоваться процедурой для отрисовки полученной зависимости.

6.4. Броуновские частицы в поле тяжести

Если частицы находятся в поле тяжести, то кроме случайных блужданий они совершают еще и регулярное движение, направленное вниз. Если частицы к тому же отскакивают от дна сосуда, то в результате конкуренции случайных блужданий и регулярного смещения вниз устанавливается распределение концентрации частиц, быстро убывающее с высотой, называемое распределением Больцмана:

$$n(z) \propto \exp(-mgz/k_{\text{Б}}T). \tag{16}$$

Здесь $n(z)$ - концентрация частиц на высоте z над уровнем дна.

Подобное распределение концентрации с высотой можно получить и в нашей компьютерной модели.

Выберем теперь интервал времени, отвечающий одному «шагу», заметно бо̀льшим: $\Delta t \gg \tau$. Можно считать, что за такое время средняя скорость движения вниз $v_{\text{дрейф}}$ успеет установиться и будет определяться условием[23]

$$mg \sim \alpha v_{\text{дрейф}}, \tag{17}$$

означающим, что в среднем сила трения компенсирует силу тяжести. Регулярное смещение b частицы за время Δt оценивается как

$$b \sim v_{\text{дрейф}}\Delta t \sim g\tau\Delta t. \tag{18}$$

Однако величина Δt должна быть выбрана и не слишком большой: нужно, чтобы регулярное смещение было мало в сравнении с интервалом высоты, на котором заметно изменяется концентрация частиц. Для этого согласно (16) должно быть выполнено условие $mgb \ll k_{\text{Б}}T$, т.е. $gb \ll v_T^2$. Умножив обе стороны этого неравенства на $\tau\Delta t$ и учитывая (18), (11) (с заменой $t \to \Delta t$), получаем

$$b^2 \ll a^2,$$

т.е. регулярное смещение должно быть мало в сравнении со случайным.

[23] **Для упрощения формул мы не выписываем архимедову выталкивающую силу.**

Соответствующие численные оценки, например для условий работы, выполняемой в лабораторном практикуме, предлагаем проделать самостоятельно.

Для моделирования такого движения надо ввести на каждом шаге кроме случайного еще и постоянное смещение b, направленное вниз, а также обеспечить «упругое отражение» частиц от дна.

Что касается отражения от «дна», то закон такого отражения не предопределен моделью и его можно выбирать по-разному, нужно только, чтобы частицы не терялись. При этом в слое, прилегающем к поверхности, может оказаться «неестественно» много или мало частиц, однако отступая от «дна», экспоненциальное убывание концентрации с высотой воспроизводится хорошо. (Разумеется, такое распределение устанавливается не сразу.)

В итоге устанавливается распределение

$$n(x) = C \cdot \exp\left(-x/h\right). \tag{19}$$

Зависимость h от b и a легко установить, сопоставляя формулы (16) и (19) и пользуясь оценками для броуновского движения

$$h \sim \frac{k_{\text{Б}}T}{mg} \sim \frac{v_T^2 \tau \Delta t}{g\tau\Delta t} \sim \frac{a^2}{b}. \tag{20}$$

Задание 5. Получите на экране распределение частиц «в поле тяжести» и соответствующую наблюдаемую функцию распределения. Удобно направить «поле тяжести» вдоль оси x, чтобы затем изображения частиц согласовались с изображением гистограммы.

Найдите с помощью компьютерного эксперимента коэффициент пропорциональности в формуле (20). Для этого заметим, что величина h равна «средней высоте столбца частиц», которая определяется соотношением

$$\langle x \rangle = \frac{\int\limits_0^\infty xn(x)dx}{\int\limits_0^\infty n(x)dx} = \frac{\int\limits_0^\infty e^{-x/h}xdx}{\int\limits_0^\infty e^{-x/h}dx} = h.$$

Постройте теоретическую кривую вида (19) и гистограмму, согласовывая их масштабы и используя режим накопления данных в процедуре **hist**. (Удобно вывести на экран также точку, показывающую положение центра тяжести столба частиц, чтобы заметить момент, начиная с которого высота центра тяжести не будет изменяться регулярно, а будет лишь флуктуировать; тогда и можно будет начать накопление данных.) При этом в процедуре **hist** по мере

накопления данных удобно будет изменять масштаб, согласуя его с полным числом учтенных точек.

7. Броуновское движение

В гл. 6 дано качественное описание движения броуновской частицы. В этом описании существенную роль играет время τ, за которое частица «забывает» направление своего движения. Рассматривая положение частицы через интервалы времени $\Delta t \gg \tau$, получаем реализацию процесса случайных блужданий, т.е. случай, изученный в гл. 6.

7.1. Случайные силы

В данной работе предлагается моделировать движение броуновской частицы с масштабом времени $\Delta t \ll \tau$. В то же время величина Δt не слишком мала. В наиболее простом для понимания случае, движении броуновской частицы в разреженном газе, будем считать число ударов молекул о броуновскую частицу L за время Δt большим, $L \gg 1$. Изменение импульса частицы за время Δt за счет взаимодействия с молекулами среды $\Delta\mathbf{p} = m\Delta\mathbf{v}$ можно представить в виде $\langle\Delta\mathbf{p}\rangle + \Delta\mathbf{p}_{\text{случ}}$, где среднее значение определяет силу трения:

$$\langle\Delta\mathbf{p}\rangle = \mathbf{f}_{\text{тр}}\Delta t, \quad \mathbf{f}_{\text{тр}} = -\alpha\mathbf{v}, \tag{1}$$

а добавка $\Delta\mathbf{p}_{\text{случ}}$ ответственна за безостановочное броуновское движение. Такое безостановочное движение есть тепловое движение, и скорость его определяется температурой среды:

$$\frac{1}{2}m\langle v^2\rangle = \frac{3}{2}k_{\text{Б}}T. \tag{2}$$

Эта скорость устанавливается в результате «компромисса» между случайными толчками, в среднем разгоняющими частицу, и воздействием силы трения, тормозящим ее. Поэтому величина $\langle(\Delta\mathbf{p}_{\text{случ}})^2\rangle$ должна быть тем больше, чем выше температура и чем больше коэффициент α, определяющий силу трения.

Проведем соответствующую этим рассуждениям количественную оценку. Изменение скорости частицы за время Δt

$$\mathbf{v} \to \mathbf{v} - \frac{\alpha}{m}\mathbf{v}\Delta t + \frac{1}{m}\Delta\mathbf{p}_{\text{случ}} \tag{3}$$

не должно приводить к изменению $\langle v^2 \rangle$:

$$\langle v^2 \rangle \quad \to \quad \langle v^2 \rangle (1 - \frac{\Delta t}{\tau})^2 + \frac{1}{m} \langle (\Delta \mathbf{p}_{\text{случ}})^2 \rangle, \tag{4}$$

откуда

$$\langle (\Delta \mathbf{p}_{\text{случ}})^2 \rangle = 2m\alpha \langle v^2 \rangle \Delta t. \tag{5}$$

(Заведомо малое слагаемое с $(\frac{\Delta t}{\tau})^2 \ll 1$ отброшено.) Таким образом,

$$\langle (\Delta \mathbf{p}_{\text{случ}})^2 \rangle = 6\alpha k_{\text{Б}} T \Delta t. \tag{6}$$

Можно ввести «случайную силу» $\mathbf{f}_{\text{случ}} = \frac{\Delta \mathbf{p}_{\text{случ}}}{\Delta t}$, нужно только иметь в виду, что ее «амплитуда» зависит от Δt:

$$\sqrt{\langle f^2_{\text{случ}} \rangle} = \sqrt{\frac{6\alpha k_{\text{Б}} T}{\Delta t}}. \tag{7}$$

Характер зависимости (6) от Δt очевиден для случая, когда в качестве среды рассматривается разреженный газ. Тогда среднее значение числа ударов молекул за время Δt: $L \propto \Delta t$, а флуктуации этого числа, определяющие $\Delta \mathbf{p}_{\text{случ}}$, пропорциональны $\sqrt{L} \propto \sqrt{\Delta t}$.

Соотношение (3) можно интерпретировать следующим образом: точка, изображающая состояние броуновской частицы в пространстве скоростей, совершает случайные блуждания в «потенциальном поле» $\frac{1}{2}\alpha v^2$.

Описанный подход к изучению движения броуновской частицы, позволяющий продвинуться в области масштабов $\Delta t \ll \tau$, принадлежит П.Ланжевену.

Смещение частицы за время $\Delta t \ll \tau \sim \frac{m}{\alpha}$ находится, естественно, как

$$\Delta \mathbf{r} = \mathbf{v} \Delta t. \tag{8}$$

Подчеркнем различие в характере движения броуновской частицы, рассматриваемого в разных масштабах времени. При $\Delta t \ll \tau$ последовательные положения частицы образуют ясно выраженную траекторию, и при уменьшении Δt движение приближается к равномерному. Если же уменьшить Δt, не выходя из области $\Delta t \gg \tau$, то можно видеть, что каждый «шаг» является результатом нескольких более коротких, но столь же хаотических шагов.

Во избежание недоразумений отметим, что раздел теории вероятностей, называемый теорией броуновского движения, рассматривает случайные блуждания, воспроизводящиеся для сколь угодно малых Δt (что соответствует пределу $\tau \to 0$).

Таким же образом можно изучать тепловые флуктуации гармонического осциллятора. Вводя в выражение (3) вклад возвращающей силы $m\omega^2 x$, получим

$$v \to v - \frac{\alpha}{m} v\Delta t - \omega^2 x \Delta t + \frac{1}{m}\Delta p_{\text{случ}}. \tag{9}$$

Задание 1. Получите на экране траекторию броуновской частицы и «траекторию» в пространстве скоростей. Отметьте на траектории другим цветом положения частицы с интервалом времени τ. Как изменяется характер траекторий с изменением τ, L, Δt?

Задание 2. Получите зависимость $x(t), v(t)$ для гармонического осциллятора. Выведите для сравнения одновременно графики для двух одинаковых осцилляторов.

Задание 3. Рассматривая движение $N \sim 200$ броуновских частиц, получите зависимости $\langle v^2(t)\rangle$ и $\langle r^2(t)\rangle$. Определите коэффициент диффузии.

7.2. Корреляционные функции

Значения компоненты скорости частицы в моменты t_1 и t_2, разделенные интервалом $\xi = t_2 - t_1$, при $\mid \xi \mid \gg \tau$ статистически независимы:

$$\langle v_x(t_1) v_x(t_2)\rangle = \langle v_x(t_1)\rangle\langle v_x(t_2)\rangle = 0. \tag{10}$$

(Перемножаются компоненты скорости одной и той же частицы, усреднение же подразумевается по очень большому числу частиц.) При значениях $\mid \xi \mid \leq \tau$ компоненты скорости не успевают сильно измениться за время ξ; мерой их взаимной зависимости служит корреляционная функция[24]

$$\varphi(\xi) = \langle v_x(t) v_x(t+\xi)\rangle. \tag{11}$$

Поскольку мы рассматриваем движение броуновских частиц, статистические свойства которого не изменяются со временем (например, температура постоянна), функция φ фактически зависит лишь от ξ, а не от моментов t и $t+\xi$ по отдельности. Отсюда следует, в частности, что $\varphi(\xi)$ - четная функция; чтобы проверить это, достаточно заменить в (11) t на $t - \xi$.

[24] **Корреляционной функцией случайных величин $x(t), y(t)$ называют функцию $\langle (x(t) - \langle x(t)\rangle)(y(t) - \langle y(t)\rangle)\rangle$. Функцию вида (11) иногда называют автокорреляционной. В (11) мы учитываем, что $\langle x\rangle, \langle v_x\rangle = 0$.**

Функции $\langle x(t)x(t+\xi)\rangle, \langle x(t)v_x(t+\xi)\rangle$ и т.п. зависят не только от ξ, но и от t, и мы их не рассматриваем. Для гармонического осциллятора, в отличие от свободной частицы, эти функции не зависят от t и их интересно изучать наряду с корреляционной функцией скоростей.

Задание 4. Получите корреляционную функцию скоростей для броуновских частиц.

Для усреднения можно рассматривать одновременно движение многих частиц или суммировать вклады, получаемые за разные интервалы времени для одной частицы.

Задание 5. Получите для гармонического осциллятора корреляционные функции $\langle x(t)x(t+\xi)\rangle, \langle x(t)v(t+\xi)\rangle, \langle v(t)v(t+\xi)\rangle$.

При этом интересно рассмотреть случаи $\omega\tau \ll 1; \omega\tau \sim 1; \omega\tau \gg 1$.

8. Шары

В этой работе будем изучать «газ», образованный шарами, которые двигаются без трения по плоскому квадратному столу и абсолютно упруго сталкиваются друг с другом и со стенками («идеальный биллиард»). Едва ли можно наблюдать такой газ где-нибудь, кроме экрана дисплея. Тем не менее, это хорошая модель настоящего газа.

Пожалуй, самое интересное, что можно увидеть в этой работе, — это возникновение «молекулярного хаоса» (даже при движении всего лишь двух шаров).

Кроме того, можно получать функции распределения по скоростям, наблюдая, как с ростом числа шаров распределения приближаются к максвелловскому. Можно изучать смесь газов с частицами разной массы, получать распределение по высоте в поле тяжести, получать так называемые стохастический нагрев и стохастическое охлаждение и т.д.

8.1. Расчет движения шаров

8.1.1. Алгоритм расчета

Движение шаров за время t рассчитывается довольно просто. Определим моменты столкновения для каждой пары шаров без учета возможных помех со стороны других шаров и стенок, а также моменты соударения каждого из шаров с каждой из стенок, неограниченно продолженной, и снова без учета всех помех.

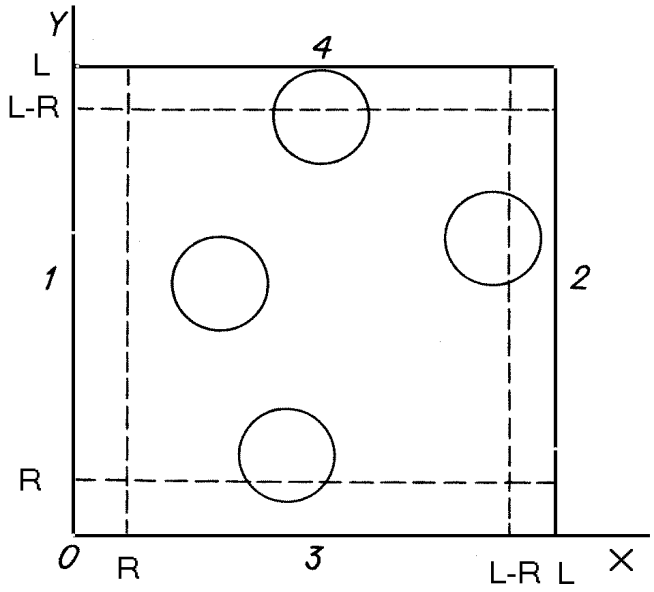


Рис. 8. Схема «стола» и системы координат

Далее выберем самое раннее из числа этих столкновений (отбросив предварительно те, которые происходили в прошлом). Ясно, что именно это столкновение и произойдет, так как никакое другое не успеет ему помешать. Если промежуток времени до этого столкновения t' больше, чем заданный интервал t, то в течение времени t не произойдет никаких столкновений, так что координаты в момент t определяются очевидным образом:

$$\mathbf{r}_i(t) = \mathbf{r}_i + \mathbf{v}_i\, t.$$

Если же $t' < t$, то нужно рассчитать смещение шаров за время t',— в результате этого смещения пара шаров (скажем, i-й и j-й) достигают соприкосновения. Затем рассчитываем изменение скоростей этой пары шаров, происходящее в результате столкновения. После этого задача сводится к предыдущей (только «контрольное время» стало меньше $t \to t - t'$): нужно вновь найти ближайшее столкновение, сравнить время до него с новым значением t и т.д. — пока заданный интервал времени не будет исчерпан.

Момент столкновения шара, например, с правой стенкой (см. рис. 8) определяется из уравнения $x + v_x t' = L - R$, где L— размер стола, R — радиус шара. Изменение скорости при таком столкновении $v_x \to -v_x$.

Момент столкновения пары шаров находится из квадратного уравнения $(\mathbf{r} + \mathbf{v}t')^2 = 4R^2$, где $\mathbf{r} = \mathbf{r}_i - \mathbf{r}_j, \quad \mathbf{v} = \mathbf{v}_i - \mathbf{v}_j$.

Изменение скоростей при столкновении

$$\mathbf{v}_i \to \mathbf{v}_i - \Delta\mathbf{v}, \quad \mathbf{v}_j \to \mathbf{v}_j + \Delta\mathbf{v},$$

где $\Delta\mathbf{v} = \mathbf{n}(\mathbf{nv}),\ \mathbf{n} = \mathbf{r}/(2R)$ [25]

Несложно «включить» также поле тяжести (иначе говоря, наклонить биллиардный стол). Скажем, если компонента ускорения, создаваемого силой тяжести в направлении против оси Y, равна g, то момент столкновения с «потолком» находится из уравнения $y + v_y t' - gt'^2/2 = L - R$.

8.1.2. Процедура Balls

Описанный алгоритм запрограммирован в виде довольно сложной функции Balls. Для желающих глубже разобраться в алгоритме далее приведен текст процедуры Balls2 на языке **MATLAB** для 2 шаров.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Функция расчета соударения двух шаров balls2       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function[r,v]=balls2(rin,vin,dt);
global rball lx ly ;             % Радиус шаров и размеры стола
a=rball; a2=4*a*a; eps=1E-10;
r=rin; v=vin;                    % Координаты и скорости шаров
ldl=[a a; a a];                  % Координаты левого нижнего края
                                 % области, доступной центрам шаров,
                                 % повторенные дважды - для каждого шара
lur=[lx-a ly-a ; lx-a ly-a];     % Правый верхний угол
t=dt;                            % Контрольное время - время до выхода из "balls"
while (t> 0)
% Учтем возможное наличие очень малых компонент скоростей
vm=(v.*v <  eps);
v1=v+vm;
tdl=(ldl-r)./v1;                 % Моменты столкновения со стенками l и d
                                 % учтем столкновения в прошлом и в очень нескорые)
```

[25] Если массы шаров различны, то

$$\mathbf{v}_i \to \mathbf{v}_i - \frac{m_j}{m_i + m_j}\Delta\mathbf{v}, \quad \mathbf{v}_j \to \mathbf{v}_j + \frac{m_i}{m_i + m_j}\Delta\mathbf{v}.$$

```
 tdl=tdl+1E10*((tdl< =0)+vm);
 tur=(lur-r)./v1;                % Столкновения со стенками r и u
 tur=tur+1E10*((tur< =0)+vm);
% Выбор самого раннего столкновения со стенкой
 [t1,j1]=min(tdl(:));
 [t2,j2]=min(tur(:));
 if(t1< t2)
  t0=t1; j0=j1;
 else
  t0=t2; j0=j2;
 end;
% Находим момент столкновения шаров друг с другом
 r0=r(:,1)-r(:,2); v0=v(:,1)-v(:,2);
 rr=r0'*r0; vv=v0'*v0;
 rv=r0'*v0; d=rv*rv-(rr-a2)*vv;
  if (d> 0) & (rv< 0) & (vv> 1E-10)
    tb=-(sqrt(d)+rv)/vv;
 else
    tb=inf;
  end;                          % if
% Выбор самого раннего соударения
    if(t< t0)&(t< tb)
    r=r+v.*t;                   % сдвиг шаров
    elseif(t0< tb)
      r=r+v.*t0;
% и изменение скорости при ударе о стенку
      v(j0)=-v(j0);
  else
      t0=tb;
      r=r+v.*t0;
      r0=r(:,1)-r(:,2);
      v0=v(:,1)-v(:,2);
      dv=r0'*v0/a2*r0;
      ddv=[-dv,dv];
% Изменение скоростей при столкновении шаров
      v=v+ddv;
    end;                        % if
```

```
% После столкновения сделаем маленький сдвиг
  r=r+v*eps;
  t=t-t0-eps;
  end;                          % while
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Подобная функция для произвольного числа шаров, написанная для ускорения счета на языке C, преобразована к виду, допускающему подключение ее к системе Matlab. Она вызывается из программы написанной на языке **MATLAB** как обычная функция. Ниже приведены необходимые для ее использования сведения. Функция

[xf,yf,vxf,vyf]= Balls(n,x,y,vx,vy,dt,R,Lx,Ly,m)

по известным значениям координат центров n шаров $\mathbf{r}_i = [x(i), y(i)])$ и компонент их скоростей $\mathbf{v}_i = [vx(i), vy(i)]), \ (i = 1, 2, ..., n)$ определяет значения этих же величин через заданный интервал времени dt и присваивает переменным $\mathbf{rf}_i = [xf(i), yf(i)] \ \ \mathbf{vf}_i = [vxf(i), vyf(i)])$. Шары движутся по прямоугольному столу размером $Lx \times Ly$, радиусы шаров одинаковы и равны R, значения масс заданы в массиве m и могут быть различными. [26]

Начало координат расположено в левом «нижнем» углу стола, оси координат параллельны его сторонам (см. рис. 8).

Предусмотрено число шаров $n \leq nmax = 25$. Переменные R, Lx, Ly, m можно не задавать. Тогда будут приняты значения $R = 20$, $Lx = Ly = 256$, а массы выбраны одинаковыми.

Начальные значения координат центров шаров должны быть выбраны в прямоугольнике $R \leq x(i) \leq Lx - R, \quad R \leq y(i) \leq Ly - R$ и притом так, чтобы расстояние между любыми центрами было не меньше, чем $2R$. [27]

Вычисления в процедуре **Balls** производятся по точным формулам.

Программа, которая демонстрирует движение дисков на экране, приведена в файле **GAS.m**.

---

[26] **Во избежание недоразумений отметим, что в процедуре Balls не предусмотрена возможность одновременного столкновения более чем двух шаров, поскольку такое столкновение представляется совершенно невероятным. Если даже будут заданы условия, заведомо приводящие к тройному соударению, то в процедуре Balls такое соударение будет трактоваться как последовательность парных.**

[27] **В противном случае, т.е. при выборе нефизических начальных условий, возможны неправильные результаты или сбой счета.**

8.2. Динамический хаос

Используя уравнения движения, удается рассчитать с высокой точностью движение планет на тысячи лет вперед (а также и в прошлом). Доказаны теоремы, утверждающие, что движение механической системы, описываемой уравнениями движения, полностью определяется значениями координат и скоростей всех ее точек в некоторый момент времени (теоремы единственности). Казалось бы движение системы 5-10 шаров тоже можно рассчитать хотя бы на сотни «периодов» (пробегов от стенки до стенки), тем более, что законы их движения и соударений очень просты. Однако оказывается, что рассчитать движение сталкивающихся шаров на сколько-нибудь длительное время практически невозможно.

8.2.1. Почему движение шаров становится непредсказуемым?

Чтобы понять это, представим себе, что направление движения одного из шаров (рис. 9) отклонилось от «правильного» на угол $\varphi_0 \sim 10^{-8}$, и исследуем, как будет изменяться угол отклонения при столкновениях.

При этом ограничимся самыми грубыми оценками. Будем иметь в виду, что средний путь шара между столкновениями — l — много больше радиуса шара, $l \gg a$. За время до очередного столкновения шар движется по прямой и центр его сместится от «правильного» положения на расстояние $OO' \sim l\varphi_0$. Тогда $AA' \sim$ $1/2 \quad ,OO' \sim l\varphi_0$ — смещение точки соприкосновения шаров при ударе. Участок поверхности второго шара в окрестности точки касания играет роль «зеркала». Это зеркало двигалось до удара, при ударе оно отскакивает, поэтому направление, в котором отскакивает шар, не определяется правилом «угол падения равен углу отражения». Тем не менее поворот «зеркала» на малый угол α ведет к изменению

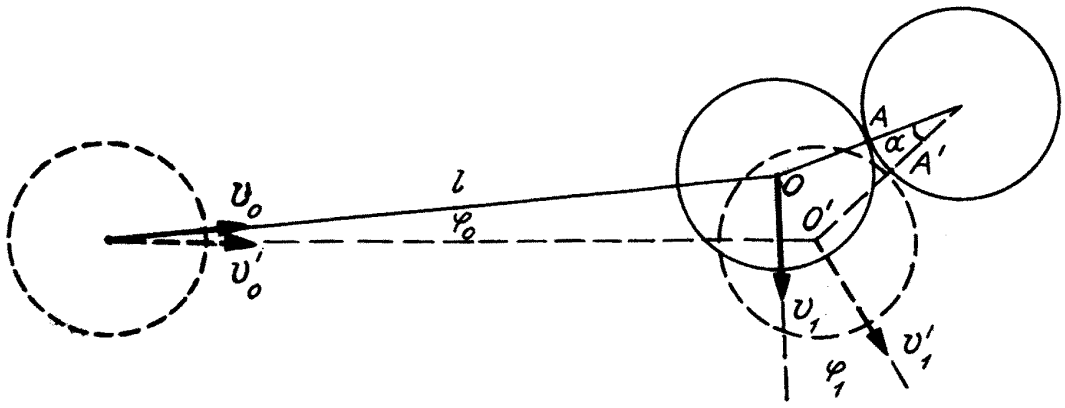


Рис. 9. Схема соударения шаров

направления движения отскочившего шара на угол $\varphi_1 \sim \alpha$ (может быть, в 1.5 - 2 раза больше или меньше — такой уровень точности в данном случае нас устраивает). Так как $\alpha \sim AA'/a$, получаем $\varphi_1 \sim (l/a)\varphi_0 \gg \varphi_0$ — при ударе угол отклонения скорости резко увеличивается. После k столкновений $\varphi_k \sim (l/a)^k\varphi_0$. Если $l/a \sim 10$, то достаточно 8-10 столкновений, чтобы стало $\varphi_k \sim 1$ и направление движения шара перестало иметь какое бы то ни было отношение к «правильному». Становится очевидным, что никакое разумное повышение точности расчетов не может значительно увеличить правильно рассчитываемый интервал движения шаров.

Чтобы предвидеть движение шаров после 100 соударений, согласно этой оценке нужно было бы задать их начальные скорости с фантастической точностью до 100 знаков.

Легко понять, что вывод о катастрофическом росте неопределенностей координат относится и к системе большого числа шаров. Относится он и к движению молекул настоящего газа. Только для молекул неопределенности возникают из-за всяческих возмущений, которыми во всех других отношениях можно пренебречь, а для нашего газа шаров — из-за ограниченной точности расчетов.

Таким образом, движение шаров (и молекул [28]) является вполне закономерным за относительно малый промежуток времени и случайным — за долгий промежуток. Отметим, что эта случайность реализуется в рамках закона сохранения энергии. [29]

8.2.2. Как убедиться в появлении хаоса?

Увидеть, что малое отклонение в направлении движения одного из шаров очень быстро вырастает, очень просто, проведя несколько «запусков» шаров с чуть различными начальными условиями.

Чтобы убедиться, что движение шаров спустя достаточный интервал времени становится непредсказуемым, нужно надежно предсказать «силой мысли» какое-то нетривиальное движение. Это совсем не трудно сделать. Представим себе, что в некоторый момент мы заменили направления скоростей всех шаров на противоположные: $\mathbf{v}_i \to -\mathbf{v}_i$. После этого шары должны двигаться строго по прежним траекториям и вернуться в начальное положение.

[28] **Для молекул есть еще одна причина хаотизации: их движение описывается не классической механикой, а квантовой, непременно включающей в описание понятие вероятности.**

[29] **Тот факт, что движение молекул газа является хаотическим, понимал и использовал в своих исследованиях еще Л. Больцман. Описанный выше механизм «потери памяти» понял и подробно изучил Н.С. Крылов, а исследовал это явление с математической строгостью Я.Г. Синай.**

Проделать именно такое наблюдение над нашим газом совсем просто. В результате мы обнаружим, что шары после нескольких столкновений друг с другом «сбиваются с пути».

Для качественного исследования достаточно зафиксировать пути частиц на экране (использовав вместо маркера 'o' или наряду с ним маркер '.' и выбрав для него вариант вывода без удаления предыдущего изображения: **set(hh,'EraseMode','none')**), а при повторном запуске шаров изменить цвет траекторий.

Задание 1. Предусмотрите в программе возможность в некоторый момент **tm** изменить скорости всех частиц на противоположные: $\mathbf{v}_i \longrightarrow -\mathbf{v}_i$ и проследите, будут ли шары возвращаться по «проложенным» ранее траекториям. Наблюдайте движение при различных значениях **tm**.

8.3. Функции распределения

Более пригодными, чем зависимости $\mathbf{r}_i(t)$ для описания движения частиц газа — как настоящих молекул, так и наших «шаров»,— являются средние величины и функции распределения. Будем отмечать на плоскости с координатами (v_x, v_y) точки, соответствующие скоростям шаров в разные моменты времени. Эти точки образуют «облако», концентрация точек в котором постепенно убывает к краям. Эта концентрация $f_0(\mathbf{v})$ становится величиной хорошо определенной, если число N_0 отмеченных точек очень велико. Обычно, функцией распределения по скорости $\mathbf{v}$ называют функцию $f(\mathbf{v}) = f_0(\mathbf{v})/N_0$, которая в нашем случае удовлетворяет условию

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\mathbf{v})dv_x dv_y = 1.$$

Вероятность того, что скорость шара окажется в элементе $dv_x dv_y$ пространства скоростей вблизи точки $\mathbf{v}$, равна

$$dw = f(\mathbf{v})dv_x dv_y. \tag{1}$$

Теоретический расчет, который приводить здесь было бы неуместно,[30] дает

$$f(\mathbf{v}) = \frac{m}{2\pi}\frac{N-1}{E}\left(1-\frac{\varepsilon}{E}\right)^{N-2}, \tag{2}$$

где $\varepsilon = mv^2/2$ — энергия шара; E — суммарная энергия всех N шаров.

[30] **Этот расчет можно найти в пособии: «Лекции по статистической физике» (Г.Л. Коткин,НГУ,1996), §5.1, предназначенном для студентов 3-го курса физфака НГУ.**

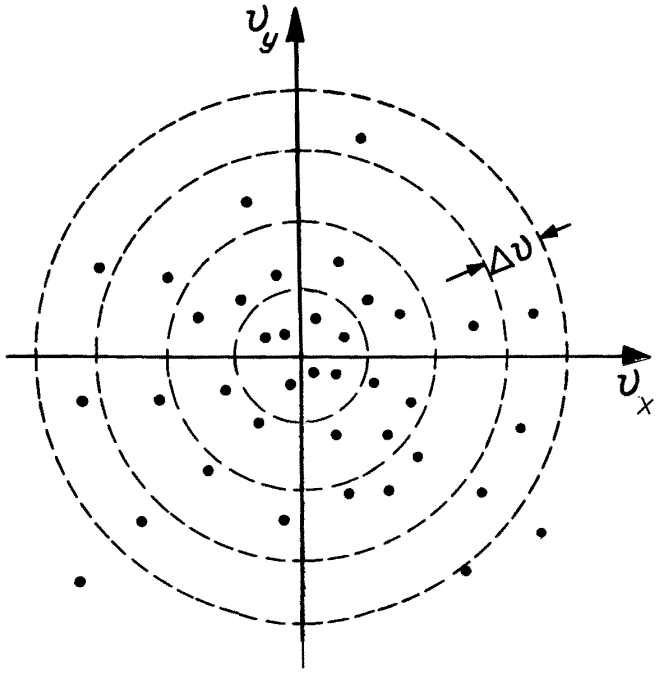


Рис. 10. Схема построения гистограммы модуля скорости

При $\varepsilon \ll E$ можно заменить $1 - \varepsilon/E$ на $\exp(-\varepsilon/E)$, так что

$$f = \frac{m}{2\pi}\frac{N-1}{E}\ \exp\left[-\frac{\varepsilon(N-2)}{E}\right].$$

Если к тому же $N \gg 1$, то можно ввести температуру газа: $k_Б T = E/N$, где $k_Б$ – постоянная Больцмана. (Для трехмерного газа было бы $(3/2)k_Б T = E/N$). Тогда (1) сводится к распределению Максвелла:

$$f(v_x, v_y) = \frac{m}{2\pi k_Б T}\exp\left[-\frac{m(v_x^2 + v_y^2)}{2k_Б T}\right].$$

Если разбить плоскость (v_x, v_y) на узкие кольца (рис. 10) одинаковой ширины Δv и подсчитать число точек в каждом из колец ΔN, то можно получить наблюдаемую функцию распределения $(1/N)(\Delta N/\Delta v)$. Соответствующая теоретическая зависимость получается из (1) заменой $dv_x dv_y$ на $2\pi v dv$.

Подобным же образом можно получить и распределение по энергии. Из равенства $\varepsilon = mv^2/2$ следует $d\varepsilon = mv\,dv$, поэтому теоретическая зависимость получается заменой $2\pi v\,dv$ на $(2\pi/m)d\varepsilon$. Для получения же наблюдаемой функции $(1/N)(\Delta N/\Delta\varepsilon)$ нужно разбить плоскость (v_x, v_y) на кольца равной площади.

Функция распределения по компоненте скорости v_x определяется с помощью разбиения плоскости(v_x, v_y) на полосы шириной Δv_x (рис. 11). Вид функции рас-

пределения :

$$\frac{dw}{dv_x} = \int\limits_{-\infty}^{\infty} f(v_x, v_y) dv_y.$$

Интеграл с помощью замены переменной

$$\sqrt{\frac{m}{2E}} v_y = x \sqrt{1 - \frac{m v_x^2}{2E}}$$

приводится к виду

$$\frac{dw}{dv_x} = \sqrt{\frac{2m}{E}} \frac{(N-1)}{\pi} \left(1 - \frac{m v_x^2}{2E}\right)^{N-\frac{3}{2}} I(N-2),$$

где [31]

$$I(N) = \int_0^1 (1-x^2)^N dx = \frac{(2N)!!}{(2N+1)!!},$$
$$N!! = N(N-2)(N-4)...$$

Задание 2. Получите на экране картину распределения частиц в плоскости (v_x, v_y). Для этого следует выводить и сохранять точки с координатами **(vx,vy)**. Интервал времени **dt** в функции **Balls** лучше выбрать большим, порядка среднего времени между столкновениями, чтобы каждый раз на экран выводились новые точки.

Задание 3. Получите «наблюдаемые функции распределения» (гистограммы) шаров по компоненте скорости v_x, по абсолютной величине скорости v, по энергии.

Выведите для сравнения также теоретические функции распределения (получаемые на основе формул (2)).

Интересно получить также распределение по скорости движения шаров друг относительно друга, по энергии относительного движения ($\frac{m}{4}(\mathbf{v}_1 - \mathbf{v}_2)^2$).

---

[31] **Для $I(N)$ с помощью интегрирования по частям легко получить рекуррентное соотношение**

$$I(N) = \frac{2N}{2N+1} I(N-1),$$

сразу же приводящее к (31)

При $N \gg 1$ основной вклад в $I(N)$ дает, очевидно, область $x \ll 1$. В этой области $1 - x^2$ можно заменить на $\exp(-x^2)$, после чего интервал интегрирования можно расширить до бесконечности. В итоге $I(N) \approx \int\limits_0^\infty \exp(-Nx^2) dx = \sqrt{\pi/N}/2$.

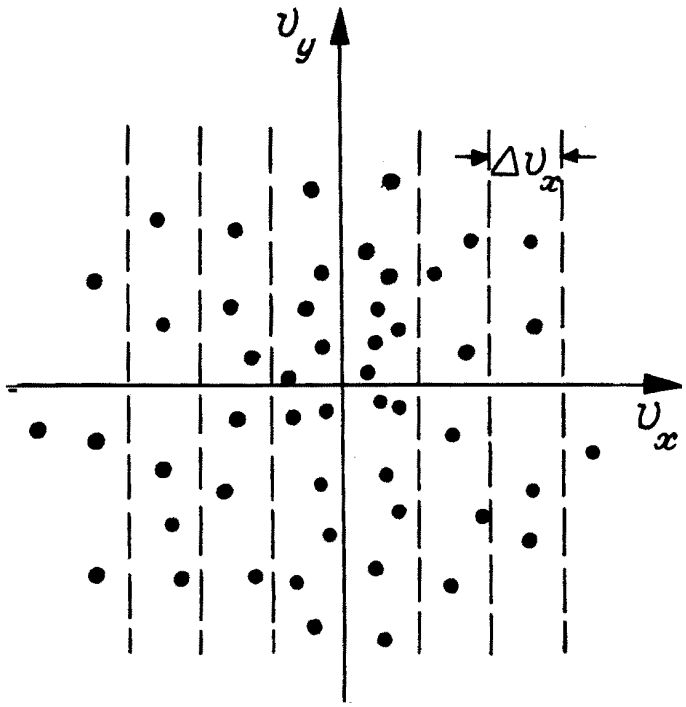


Рис. 11. Схема построения гистограммы по проекции скорости v_x

Задание 4. Получите функцию распределения по расстояниям между центрами шаров (усредненную за длительное время).

Укажите существенные отличия таких функций распределения при небольшой концентрации шаров и при условиях, когда среднее расстояние между шарами немногим больше их диаметра.

8.4. Стохастический нагрев и стохастическое охлаждение

Если биллиардный стол («ящик с газом») начинает двигаться, очень медленно наращивая скорость, а затем так же плавно останавливается, то вместе с ним, естественно, ускоряется и замедляется и газ, сохраняя в конечном счете свою энергию.

Если движение «ящика» не является медленным и плавным, то энергия газа в результате такого движения может как вырасти, так и уменьшиться. Какого изменения энергии газа можно ожидать при длительном периодическом движении «ящика»? В предельном случае, когда газа много, от движущихся стенок распространяются звуковые волны, затухающие в объеме газа, поэтому газ нагревается. А если частиц мало? Оказывается, и в этом случае энергия газа будет расти (хотя и не монотонно) — это так называемый стохастический нагрев газа.

Процедура **Balls** определяет движение шаров и в ящике, который движется с

постоянной скоростью, естественно, в системе отсчета, связанной с ним. Используя эту процедуру, можно найти движение шаров, если скорость «ящика» изменяется скачками.

Задание 5 Рассмотрите периодическое движение ящика вдоль одной из стенок, при котором ящик половину периода движется со скоорстью U, а второю — со скоростью $-U$. Постройте график зависимости энергии шаров от времени. Будет ли расти средняя энергия (температура) газа шаров?

Для исследования столкновений протонов с антипротонами на ускорителях со встречными пучками необходимо сделать летящие навстречу друг другу сгустки частиц очень концентрированными, а для этого нужно ослабить относительное движение частиц внутри каждого из сгустков.

Один из способов сделать это — это так называемое стохастическое охлаждение. Можно добиться замедления относительного движения частиц, если по-разному воздействовать на сгусток в зависимости от движения частиц в текущий момент. Такое воздействие достигается благодаря тому, что информация о текущем состоянии частиц, движущихся по дуге, успевает опередить сгусток, двигаясь по более короткому прямому пути. При этом можно фактически оперировать лишь информацией, относящейся только к сгустку в целом.

Управляя движением «ящика», можно смоделировать стохастическое охлаждение.

Задание 6 Получите стохастическое охлаждение газа шаров, запрограммировав выбор той или иной величины скорости «ящика» в зависимости от скорости центра масс шаров.

9. Потери пучка при прохождении через вещество

В этой работе можно познакомиться с основным методом моделирования, применяемым при исследовании прохождения пучков частиц через вещество — методом статистического моделирования (называемым методом Монте-Карло). При этом «судьба» каждой частицы «разыгрывается» с помощью случайного выбора, а полученные для множества частиц результаты подвергаются статистической обработке. Метод применяется, например, при проектировании ядерных реакторов, детекторов частиц на ускорителях и обработке получаемых результатов (а также во многих других случаях, скажем, при исследовании распространения мутаций в среде живых организмов).

Мы будем изучать, естественно, очень простой вариант задачи — прохождение пучка тяжелых частиц (A) через слой газа, состоящего из легких (O). К тому же будем полагать скорости частиц газа до столкновения с частицами пучка равными нулю. Не будем также учитывать изменений, происходящих в газе вследствие прохождения пучка. Это не ограничивает существенно общности задачи.

9.1. Эффективные сечения

Частицы можно представлять шариками радиусов R_A и R_O. Пусть частица A летит так, что центр ее должен пролететь на расстоянии ρ от центра частицы O. Столкновение произойдет, если $\rho < R$, где $R = R_A + R_O$. Фактически частица O образует для частиц A «преграду», площадь которой $\sigma = \pi R^2$. Эта величина называется эффективным сечением столкновения[32]. В зависимости от величины ρ (называемой прицельным параметром) определяются угол отклонения, передача энергии от частицы A частице O и т.п. Соответственно можно определить величины эффективных сечений, например, для потери частицей A определенной энергии от ε_1 до ε_2, отклонения на угол, больший данного и т.п. Определяются и дифференциальные эффективные сечения, например, дифференциальное эффективное сечение потери энергии частицей A

$$\frac{d\sigma}{d\varepsilon} = f(\varepsilon)$$

определяется так, что величина площадки $d\sigma = f(\varepsilon)d\varepsilon$ отвечает потере энергии в интервале от ε до $\varepsilon + d\varepsilon$ (при достаточно малой величине $d\varepsilon$). Подробнее об этом сказано в работе[7, §18], мы будем использовать взятые оттуда формулы.

Характерные размеры пучка частиц — сантиметры или даже микроны — во много раз превосходят характерные прицельные параметры. Именно поэтому здесь уместен статистический подход. Пусть концентрация газа-мишени составляет n_O частиц O на 1 см^3. Тогда в очень тонком слое (толщины dx) на площадь S приходится $n_O S\ dx$ частиц O, а перекрываемая ими площадь равна $dS = \sigma n_O S dx$. Можно сказать также, что $dW = dS/S = \sigma n_O\ dx$ представляет долю площади, перекрытую частицами O в слое dx. Иначе говоря, это вероятность столкновения частицы в данном слое. Имея в виду возможность выбрать достаточно тонкий слой, мы можем пренебречь возможностью, что какая-то из частиц O будет «затенена» другими. Фактически для этого достаточно условия $dW \ll 1$.

[32] **Для столкновений атомов характерные величины эффективных сечений имеют порядок 10^{-16} см^2, для нейтронов и атомных ядер — 10^{-26} см^2, на современных ускорителях изучаются сечения еще на 10 - 12 порядков меньшие.**

9.2. Потери частиц пучка при прохождении слоя

Начнем с простейшей задачи о выбывании частиц из пучка. Пусть любое столкновение ведет к выбыванию частицы – сечение выбывания равно σ. Задача об ослаблении пучка легко решается в таком случае аналитически. Пусть количество частиц A в исходном пучке равно N_{A0} . После прохождения слоя dx это число уменьшится на $dN_A = N_A dW$. Введя число частиц, оставшихся в пучке, $N_A(x)$, имеем

$$dN_A(x) = -N_A(x) n_O \sigma dx. \qquad (1)$$

Решение этого дифференциального уравнения, удовлетворяющее условию $N_A(0) = N_{A0}$,

$$N_A(x) = N_{A0} \exp(-bx), \qquad (2)$$

где $b = n_O\sigma$. Величина $1/b$ определяет толщину слоя, в котором пучок потеряет значительное число частиц (ослабнет в e =2.7 раза). Приведенное решение задачи относится к средним значениям числа частиц.

Теперь покажем, как решается эта задача методом Монте-Карло. Будем «пропускать» частицу через тонкие слои толщиной dx, пока она не пройдет слой x (или поглотится). «Розыгрыш» для слоя dx состоит в том, что мы выбираем случайное число с равномерным распределением от 0 до 1 (функция **rand** в **MATLAB**) и сравниваем с величиной dW; если **rand**$< dW$, то частица выбывает, в противном случае перемещается в следующий слой. Пропустив N_a частиц через слой толщины x, мы можем получить число оставшихся частиц $N_a(x)$.

Приведем отрезок программы, в котором через слой толщины Xmax «пропускается» N_a частиц. При реализации этой части программы предполагается, что для всех частиц заведены два вектора - вектор координат, достигнутых каждой из частицей к текущему моменту времени, и вектор состоящий из 0 и 1. Каждая 1 соответствует еще не выбывшей частице, а 0 - уже выбывшей из потока.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Программа расчета потерь пучка      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;
 Na=250; k=(1:Na)';
 no=100; Section=0.05;
 dx=0.01;
x=zeros(size(k));          % Начальный вектор координат
sc=ones(size(k));          % Начальный вектор-счетчик
```

```
dW=no*Section*dx;        % Вероятность поглощения на шаге dx
 hl=line(x,k);           % Подготовка Na точек для рисования
 set(hl,'Marker','o','MarkerSize',3);
 axis([0 1 0 Na+1]);     % Масштабирование осей
pause;                   % Пауза перед запуском основного цикла
% Далее выполняется цикл, на каждом шаге которого
% координата x возрастает на dx до тех пор, пока
% частица не пройдет путь Xmax или не поглотится
 while (any(sc)> 0)
  ra=rand(size(k));      % Расчет вероятностей с помощью
                         % датчика случайных чисел для
                         % всех частиц
  k1=find(ra-dW< 0);     % Определение номеров частиц
                         % подлежащих отбраковке
  sc(k1)=0;              % Зануление счетчика отбракованной частицы
  x=x+sc*dx;             % Продвижение на dx остальных частиц
  set(hl,'XData',x);     % Рисование
 end;                    % Конец цикла while
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Программа **Beam1** содержит моделирование выбывания частиц из пучка.

Заметим, что отклонения от (2) делают найденную зависимость более похожей на наблюдаемую экспериментально, поскольку здесь моделируется не только среднее, но и наличие флуктуаций относительно среднего. Величина же флуктуаций зависит от числа проходящих частиц и тоже может быть определена с помощью моделирования.

Число частиц N_f, прошедших через достаточно толстый слой, оказывается малым и сильно флуктуирует (изменяется от запуска к запуску). Распределение вероятностей для доли прошедших частиц $p = N_f/N_a$ — это распределение Бернулли.

Задание 1. Постройте согласно формуле (2) зависимость $N_a(x)$ в масштабе, удобном для сравнения с модельной зависимостью (гистограммой).

Постройте распределение для доли частиц, прошедших толстый слой. Наблюдайте изменение этого распределения при изменениях числа частиц в запуске.

Подобным же образом моделируются более сложные процессы. Пусть, например, при столкновении с частицей газа частица пучка A может не только поглотиться, но и превратиться в частицу B, летящую в том же направлении, причем

эффективное сечение такого превращения равно σ_{AB}, а эффективное сечение поглощения частиц A равно σ_A.

Приведем участок программы, моделирующей «судьбу» частиц в этом случае.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Моделирующая часть программы с поглощением А    %
% и превращением А -> В                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SecA - сечение поглощения А
% SecAB - сечение превращения А в В
% scA - счетчик числа частиц типа А
% scB - счетчик числа частиц типа В
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;                        % Очистка рабочей области
Na=25;                        % Начальное число частиц А
L=5; k=(1:Na)'; no=100;dx=0.01;
 SecA=0.1; SecAB=0.15;
x=zeros(size(k));             % Начальный вектор координат
scA=ones(size(k));            % Вектор-счетчик частиц А
scB=zeros(size(k));           % Вектор-счетчик частиц В
dWa=no*dx*SecA;               % Вероятность поглощения А на шаге dX
dWab=no*dx*SecAB;             % Вероятность А -> В на шаге dX
......................................
% Имитация поглощения и превращения частиц
% Пока есть частицы А и не пройден путь L
while (any(scA)> 0 & all(x)< L)
  ra=rand(size(k));
  ka=find(ra-dWa< 0);         % Номера поглощенных частиц А
  scA(ka)=0;                  % Выбывание частиц типа А
  kb=find(dWa< ra & ra< dWa+dWab)
  scB(kb)=scA(kb);            % Превращение А в В
  scA(kb)=0;                  % Исчезновение этих А
  x=x+scA*dx+scB*dx;
  pause(1);
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Для подобной модели несложно также составить и решить уравнения, описывающие изменения среднего числа частиц A и B, подобно уравнениям (1), (2).

Стоит также отметить близость рассматривавшейся задачи к задаче о радиоактивном распаде атомных ядер (для перехода к ней нужно понимать под x время, прошедшее с начала наблюдения).

Задание 2. Добавьте в модель поглощение частиц B и обратные переходы B в A.

9.3. Потери энергии

Теперь перейдем к моделированию потерь энергии тяжелыми частицами A при прохождении слоя газа, состоящего из легких.

Обозначим массу легкой частицы (O) m, массу тяжелой (A) M, ее скорость V, а энергию E ($m \ll M$).

Оценка, приведенная ниже, показывает, что отклонение траектории частицы A от прямой будет невелико даже при значительной потере энергии. Поэтому будем контролировать прохождения частицей слоя газа тем же способом, что и раньше.

Покажем, как можно моделировать потери энергии. Обозначим полное сечение соударений σ, а дифференциальное сечение потери энергии $d\sigma/d\varepsilon = F(E,\varepsilon)$, где E – энергия частицы перед столкновением, а ε – потеря энергии при столкновении.

Сначала следует определить, как и в предыдущих случаях, произошло ли на данном участке dX столкновение. Если же оно произошло, то следует разыграть, какова именно величина потери энергии. Эта операция производится методом браковки, описанным в Приложении D.

```
.....................................
ra=rand(size(k));
ka=find(ra-Na*Sect*dX< 0);% Номера рассеявшихся частиц
for ii=ka                 % Цикл по рассеявшимся на этом шаге частицам
 % Генерация случайной величины потерянной энергии,
 % распределенной с плотностью вероятности F(E,eps)
  while 1                 % Бесконечный цикл, выход с помощью break
    eps(ii)=epsmax*rand;  % Выбор величины потери энергии
    if Fmax*rand <  F(E(ii),eps(ii))
      break;
    end;                  % Конец if
```

```
  end;                          % Конец while
    E(ii)=E(ii)-eps(ii);        % Вычисление потери энергии
end;                            % Конец for
.....
```

Здесь **epsmax**=ε_{max} – максимально возможное значение потери энергии, а **Fmax** =F_{max} – число, не меньшее, чем максимум функции $F(E,\varepsilon)$.

При столкновениях упругих шариков реализуется особый случай: функция $F(E,\varepsilon)$ во всем интервале $0 < \varepsilon < \varepsilon_{max}$ фактически от ε не зависит ($\varepsilon_{max} = 4mME/(m+M)^2 \approx 4(m/M)E$) (см. [7, §18, задача 2]). В этом случае вместо приведенного выше цикла while - end следует сохранить только отмеченную комментарием строчку вычисления случайной потери энергии.

При столкновениях заряженных частиц $F(E,\varepsilon)$ неограниченно возрастает при $\varepsilon \to 0$. Это связано с очень слабым рассеянием при больших прицельных параметрах. В таком случае можно ввести добавочное ограничение, выбрав какое-то значение ε_{min} и отказавшись от учета меньших ε. Тогда $F_{max} = F(E,\varepsilon_{min})$.

С точки зрения физического смысла задачи величины ρ и ε должны быть ограничены по ряду причин: при больших прицельных параметрах во взаимодействиях заряженных частиц существенно влияние других частиц, небольшие потери энергии не заметны на фоне неизбежного разброса энергий начального пучка и т.п. С точки зрения построения моделирующей программы в предлагаемом способе моделирования также не обойтись без такого ограничения. Но если окажется, что полная величина потери, которая при $\varepsilon < \varepsilon_{min}$ могла бы быть мала в сравнении с характерной точностью, принятой при анализе распределения по энергиям $\varepsilon_{min}X_{max}/dX \ll E/l$, то малые значения ε заведомо можно не учитывать. В то же время ставить границу ε_{min} слишком низко невыгодно, так как это приведет к замедлению выбора ε: слишком малую долю будет составлять площадь под кривой $F(E,\varepsilon)$ от площади прямоугольника $\varepsilon_{min} < \varepsilon < \varepsilon_{max},\ 0 < F < F_{max}$.

Задание 3. Определите распределение по энергиям частиц, прошедших слой **Xmax**.

Предлагается два варианта:

- частицы - абсолютно упругие шарики;
- частицы - заряженные точки, взаимодействующие по закону Кулона $U = \alpha/r$. Для этого понадобится выражение для дифференциального эффективного сечения потери энергии при столкновении частиц, которое можно взять в [7, §19]:

$$\frac{d\sigma}{d\varepsilon} = \frac{2\pi\alpha^2 M}{mE\varepsilon^2} \quad \text{при} \quad \varepsilon < \varepsilon_{max} = 4\frac{m}{M}E.$$

Заметим, что в этой задаче, в отличие от предыдущих, моделирование заведомо является самым простым способом исследования.

9.4. Распределение по углам и энергиям

Сначала сделаем грубую оценку, показывающую, что даже при значительной потере энергии можно пренебречь отклонением направления движения частиц A от первоначального. Оценка основана на том, что при каждом соударении угол отклонения изменяется мало, причем направление движения может как удаляться от первоначального, так и приближаться к нему.

При столкновении легкая частица получает скорость порядка V, т.е. импульс $p \approx mV$ и энергию порядка $\varepsilon \approx \frac{mV^2}{2}$. Частица потеряет энергию порядка первоначальной за $N \approx \frac{MV^2}{2}/\varepsilon \approx M/m$ соударений. Угол отклонения тяжелой частицы при одном столкновении $\theta_1 \approx p/MV \approx m/M$. Отклонения при разных столкновениях происходят в разные стороны по случайным направлениям, поэтому складываются не углы, а их квадраты — в плоскости (V_y, V_z) происходит диффузия. Тогда угол отклонения за N соударений $\theta \approx \theta_1\sqrt{N} \approx \sqrt{m/M} \ll 1$.

Задавать направление движения частицы можно полярными углами вектора ее скорости θ и φ. Однако удобнее будет ввести углы $\theta_y = \theta\,\cos\varphi \approx V_y/V$ и $\theta_z = \theta\,\sin\varphi \approx V_z/V$.

Чтобы моделировать отклонение направления скорости частицы при многократных столкновениях, следует воспользоваться дифференциальным эффективным сечением рассеяния на данный угол $\frac{d\sigma}{d\theta} = f(\theta)$[33].

Поскольку мы принимаем углы θ небольшими, каждый добавочный угол отклонения θ_1 можно разыгрывать так же, как первое отклонение от первоначального направления вдоль оси X.

Кроме того, следует произвести также выбор азимутального угла φ нового отклонения[34]:

```
.................
 phi = 2*pi*rand;
.................
```

[33] **Другой вариант расчета основан на том, что угол отклонения частицы в лабораторной системе отсчета θ, угол в системе центра масс χ и потеря энергии ε связаны друг с другом простыми соотношениями (см. [7, §17]). Поэтому можно сначала «разыграть» χ , а затем вычислить ε и θ (либо наоборот, разыграть ε, а затем вычислить χ и θ).**

[34] **Если частицы A или B определенным образом ориентированы и взаимодействие их не сводится к центральному полю, то возможна зависимость дифференциального эффективного сечения от угла φ. Тогда этот угол также следует разыгрывать методом браковки.**

Теперь можно найти значения добавок $\theta_{1y} = \theta_1 \cos\varphi$, $\theta_{1z} = \theta_1 \sin\varphi$ и получить новые значения θ_y, θ_z:

$$\theta_y \rightarrow \theta_y + \theta_{1y} \ , \theta_z \rightarrow \theta_z + \theta_{1z}.$$

В результате получится уже упомянутая диффузия в плоскости (V_y, V_z).

После первого столкновения угол отклонения и потеря энергии жестко связаны друг с другом, но при повторных столкновениях эта связь разрушается.

Задание 4. Определите распределение по углам частиц, прошедших слой X_{max}, и среднее значение квадрата угла отклонения (выбрав один из вариантов предыдущего задания).

Выведите в плоскости (E, θ) точки, демонстрирующие энергетически-угловое распределение частиц, прошедших сквозь слой.

Задание 5. Считая начальный пучок тонким, получите картину распределения частиц пучка в поперечной плоскости после прохождения слоя x. Для этого понадобится следить за поперечным смещением каждой частицы на каждом шагу dx.

10. Работа с сигналами и модель диодного выпрямителя

В инженерных и научных приложениях часто встречаются различные электрические сигналы. В системе **MATLAB** имеется специальный пакет (*Signal Processing Toolbox*), предназначенный для работы с сигналами. При работе с задачами в настоящем параграфе[35] не предполагается использование специальных средств. Все задания, предложенные в настоящем параграфе, будут решаться стандартными методами основного пакета **MATLAB**.

10.1. Работа с сигналами

Хотя большинство сигналов в цепях, которые мы будем изучать, являются непрерывными или *аналоговыми* сигналами, при компьютерном моделировании и/или обработке таких сигналов используется их дискретная выборка, представляющая непрерывный сигнал как вектор, каждый элемент которого представляет собой значение сигнала в некоторый момент времени. При этом, как правило, используются равноотстоящие моменты времени, поэтому вектор времени характеризуется

[35] **Данная задача была разработана совместно с Ю. М.Прокопьевым. При этом частично использовались материалы из [11]** .

всего тремя параметрами — временем начала, временем конца и шагом по времени. Если же сигнал периодический, а как правило изучаются именно такие сигналы, то сигнал представляется в виде вектора значений в моменты времени, распределенные в течение одного-двух периодов.

Задание 1. Создать вектор-столбец со значениями от 0 до 50 с шагом 0,2. Используя его в качестве аргумента записать выражения для вычисления значений следующих сигналов[36]:

1) $f(t) = A$ при $0 \le t \le T$ — постоянная функция;
2) $f(t) = A/\tau \cdot t$ при $0 \le t \le T$ — линейная функция;
3) $f(t) = A \cdot Sin(2\pi t/\tau)$ при $0 \le t \le T$ — синусоидальный сигнал;
4) $f(t) = \begin{cases} 0 \text{ если } t \le \tau \\ A \text{ если } t \succ \tau \end{cases}$ при $0 \le t \le T$ — ступенька;
5) пилообразный сигнал;
6) прямоугольные импульсы.

Изобразить перечисленные выше сигналы на одном или двух рисунках, отрисовывая каждый из сигналов другим цветом или в отдельном окне.

Далее приведен текст примерной программы, которая строит два из перечисленных выше сигналов (синусоидальный и линейный) на одном рисунке.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Примерная программа вывода двух сигналов %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;                      % Очистка рабочей области
% Задание начальных значений
a1=1.5;                     % Амплитуда для синуса
a2=0.15;                    % Амплитуда для прямой
tau=10.0;                   % Период
t=0:0.2:50;                 % Вектор времени
y1=a1*sin(2*pi*t/tau);      % Первый сигнал
y2=a2/tau*t;                % Второй сигнал
plot(t,y1,t,y2);            % Отрисовка двух сигналов
axis([0 50 -2 2])           % Задание масштабов по осям
```

[36] **Для записи сигналов без явного использования циклов рекомендуется познакомиться и использовать такие *поэлементные* функции как sin, mod и sign**

```
grid on                                  % Нанесение сетки
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Для дальнейшей работы необходимо научиться работе с сигналами в том виде, как они представлены в **MATLAB**, т.е. в виде векторов. Необходимо различать поэлементные и матричные операторы. Подробнее об операциях с матрицами в **MATLAB** можно познакомиться в п. 2 и в Дополнении, п. 2.

Задание 2. Используя ранее написанную функцию генерации сигналов, сформировать, используя (:), матрицу, в первом столбце которой помещено время, во второй столбец — сигнал 1, в третий столбец — второй сигнал и т.д.

Задание 3. Сформировать массив, десять элементов которого образуют прямоугольный импульс, и создать вектор с периодическим прямоугольным импульсом (10 импульсов).

Задание 4. Создать высокочастотный сигнал, модулированный Гауссом, прямоугольным импульсом, треугольным сигналом (поэлементное умножение двух векторов).

Задание 5. Используя функцию **diff(x)**, вычисляющую разность между соседними элементами вектора-столбца, рассчитать производную от каждого из сигналов в матрице сигналов, вывести графики производной. Используя функцию **sum(x)**, вычислить интеграл (по формуле прямоугольников или лучше трапеций) для всех сигналов в матрице и нарисовать их. На каждом рисунке (**figure**) должно быть три окна, в которых соответственно представлены сигнал, производная и интеграл от него. Сверху над рисунком должна быть надпись (используйте функцию **title**).

10.2. Расчет простейших цепей

В качестве простейшей цепи для начала будем рассматривать последовательно соединенные источник э.д.с. переменного напряжения $E(t)$ (сигнала), постоянного активного сопротивления R и конденсатора емкостью C. Эту схему описывает следующая система уравнений:

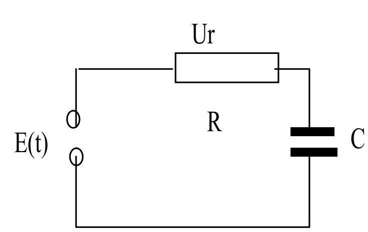


Рис. 12. Простейшая схема RC-цепочки

$$\begin{cases} \frac{dQ}{dt} = i \\ E(t) = U_r + U_c \\ U_r = I \cdot R \\ U_c = \frac{Q}{C} \end{cases}$$

Для расчета временной зависимости падения напряжения U_r и U_c можно воспользоваться простейшим алгоритмом, реализованным в следующем примере.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Примерная программа расчета RC-цепочки   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 clear;                      % Очистка памяти
% Задание параметров цепи
 C=0.1;                      % Это емкость конденсатора в фарадах
 R=1;                        % Это сопротивление в омах
 tau=R*C;
% Задание начальных значений
 t0=0; Q0=0;


% Задаем начальные значения времени t, тока I и заряда Q

 t(1)=t0; I(1)=0; Q(1)=Q0;
 dt=0.02; t=0:dt:4;
 E=signale(t,3);
 ss(1)=0.0
 for k=2:length(t)
    Q(k)=Q(k-1)+I(k-1)*dt;
    Uc(k)=Q(k)/C;
    Ur(k)=E(k)-Uc(k);
    I(k)=Ur(k)/R;
    ss(k)=ss(k-1)+E(k)*dt;
 end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Задание 6. Написать программу, реализующую расчет RC цепи по описанному выше алгоритму. Вывести на экран сигнал E, производную, интеграл, Uc, Ur. Проанализировать, когда Uc пропорционально производной, а когда Ur пропорционально интегралу от сигнала. Удобнее всего это задание выполнить с

помощью описанного в п. 2.3 графического интерфейса, хотя это и не обязательно.

Задание 7. Рассмотреть самостоятельно схему, аналогичную RC-цепочке, у которой вместо конденсатора C стоит индуктивность L (RL-цепочка), и выполнить все упражнения пункта 5. Система уравнений, описывающая эту цепочку, имеет вид

$$\begin{cases} U_r = i \cdot R \\ U_l = E - U_r \\ \frac{di}{dt} = \frac{U_l}{L} \end{cases}$$

10.3. Статическая модель диода. Решение нелинейных уравнений

Если напряжение на диоде равно u, то ток через диод определяется известным соотношением:

$$I = I_0 \cdot \left(e^{\frac{u}{\varphi_T}} - 1 \right) , \tag{3}$$

где I_0 - тепловой ток,

φ_T - температурный потенциал.

$$\varphi_T = \frac{kT}{q} = \frac{T}{11600} .$$

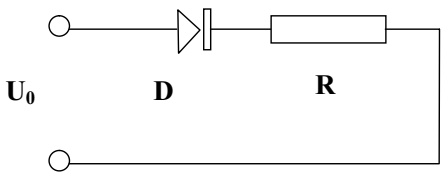


Рис. 13. Простейшая схема включения диода с нагрузкой

Название «температурный потенциал» для величины φ_T вполне оправдано, поскольку она имеет размерность напряжения и пропорциональна температуре. С физической точки зрения температурный потенциал есть выраженная в электрических единицах статистическая температура или близкая к ней средняя кинетическая энергия «свободного» электрона в электронном газе. При T=300 K, $\varphi_T(300K) = 0,025$ В. Ток I_0, определяющий «масштаб» характеристики, называется тепловым током. Термин «тепловой» отражает сильную температурную зависимость тока I_0, а также тот факт, что он равен нулю при абсолютном нуле температуры.

Другим распространенным термином является «обратный ток насыщения», происхождение которого связано с тем, что при отрицательном напряжении $|u| \gg \varphi_T$ обратный ток идеализированного диода равен I_0 и не зависит от напряжения, по

крайней мере, до тех пор, пока не происходит обратный пробой. Мы примем величину $I_0 = 1$мкА, что является характерным значением для большинства кремниевых диодов.

Для цепи, изображенной на рис. 13, нетрудно написать систему уравнений для нахождения тока через диод.

$$\begin{aligned} I &= \tfrac{1}{R}(U_0 - u) \\ I &= I_0 \cdot (e^{u/\varphi_T} - 1)\,. \end{aligned}$$

Задание 8. Выведите график зависимости $I(u)$, полученной из уравнения (3). Определите оптимальные масштабы отображения, при которых можно различать прямую и обратную ветви. $I_0 = 1$мкА, резистор $R = 1$кОм.

Задание 9. Решите уравнение графически, определив рабочую точку, которая является точкой пересечения двух графиков — характеристики диода и нагрузочной прямой $I = \frac{1}{R}(U_0 - u)$.

Для расчета режима работы цепи, включающей диод и другие элементы, необходимо на каждом шаге решать нелинейное уравнение вида $x = f(x)$. Это можно сделать с помощью метода *простой итерации*, суть которого очень проста. Начальное приближение x_0 подставляется в правую часть, находится x_1, которое затем используется в следующей итерации как начальное приближение и т. д. Критерием завершения итераций является малое отличие x_i и x_{i+1}. Конечно, процесс может и не сойтись. Это зависит от вида уравнения и от начального приближения. Но даже если процесс сходится, требуется большое количестве итераций и соответственно времени для их выполнения. Можно, конечно, использовать какую-либо стандартную процедуру **MATLAB**, например **fzero**, но в данном случае мы используем простейший вариант метода Ньютона.

Для уравнения $f(x) = 0$, а наше уравнение легко привести к такому виду, можно записать разложение в ряд Тейлора вблизи точки x_0, которую мы считаем начальным приближением:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \ldots$$

Пренебрегая членами разложения выше первого порядка, можно написать:

$$x = x_0 - f(x_0)/f'(x_0).$$

Соответственно итерационная формула будет выглядеть следующим образом:

$$x_{i+1} = x_i - f(x_i)/f'(x_i).$$

Критерий завершения итераций такой же, как и в предыдущем примере. Как правило, этот метод обеспечивает существенно более быструю сходимость, правда, требует знания производной исходной функции. Иногда используют методы Ньютона более высоких порядков, которые основаны на учете следующих членов разложения.

Задание 10. По аналогии с RC и RL цепями запишите уравнение для переменного тока в цепи, состоящей из последовательно соединенного диода ($I_0 = 1$мкА) и резистора ($R = 1$кОм) с приложенным (входным) переменным напряжением. Пусть входное напряжение имеет амплитуду 10 В и является синусоидальным с частотой 50 Гц. Найдите ток через резистор. Для этого необходимо решать нелинейное уравнение, определяющее рабочую точку диода, на каждом шаге по времени. Постройте графики входного напряжения и тока через резистор.

Рассмотрим наконец модель выпрямителя с фильтром. Теперь слегка усложним нашу схему с диодом и резистором, добавив нагрузочный резистор R_2 и емкость C, которая служит для сглаживания пульсаций выходного напряжения (рис. 14).

В таком варианте можно рассматривать влияние гораздо большего количества параметров на работу выпрямителя, в частности влияние сопротивления нагрузки или емкости конденсатора на качество выпрямления. Качества выпрямления можно характеризовать амплитудой пульсаций напряжения на нагрузке R_2. Система уравнений, описывающая нестационарную работу такой схемы, выглядит следующим образом:

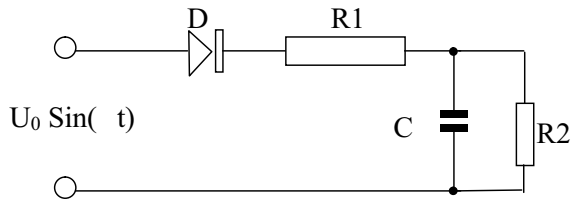


Рис. 14. Простейшая схема включения диода с нагрузкой и фильтром

$$\begin{cases} u + iR_1 + U_c = U_0 Sin(\omega t) \\ i_r R_2 = U_c \\ i = i_r + i_c \\ i_c = C\frac{dU_c}{dt} \\ i = F(u) \end{cases},$$

где $F(u)$ - правая часть уравнения (3).

Алгоритм решения этой системы уравнений может выглядеть, например, так:

1. $U_c = 0$
2. Решить нелинейное уравнение $u + F(u)R_1 = U_0 \sin(\omega t) - U_c$

3. $i = F(u)$
4. $i_r = \frac{U_c}{R_2}$
5. $i_c = i - i_r$
6. $\tilde{U_c} = U_c + \frac{i_c}{C} dt$

В результате такой последовательности вычислений мы получаем значение напряжения на конденсаторе U_c в момент времени $t + dt$ и снова переходим к пункту 2. Фактически это означает, что мы ищем мгновенные значения параметров как для статической схемы в предположении, что напряжение на конденсаторе берется в предыдущий момент времени. Для такого рассмотрения необходимо, чтобы шаг по времени был много меньше, чем постоянные времени зарядки и разрядки конденсатора (R_1C, R_2C).

Задание 11. Задайте приемлемые, на ваш взгляд, параметры элементов схемы. Входное напряжение синусоидальное, частота 50 Гц, амплитуда 10 В. Нарисуйте графики зависимости от времени входного напряжения, тока через диод и напряжения на емкости. Рассмотрите две временные области. Первая должна демонстрировать процесс установления напряжения на емкости в начальный период времени. Во второй постарайтесь показать пульсации напряжения на нагрузке около установившегося выходного напряжения.

ПРИЛОЖЕНИЕ

А. Дополнительные задачи

Здесь приведены примеры задач, предлагаемых студентам для самостоятельной работы.

Свободные колебания

1. Изобразите на экране монитора нормальные колебания двойного маятника (см. [9, задача 6.3]).

2. Колебания молекулы CO_2 можно представлять как наложение гармонических колебаний. Изобразите каждое из этих колебаний и их суперпозицию (см. [7, §24, задача 1]).

Электрические и магнитные поля

3. Изобразите силовые линии магнитного поля, создаваемого парой прямых токов и однородным полем, перпендикулярным этим токам (магнитная ловушка), для следующих случаев:

   а) токи текут в направлении, перпендикулярном плоскости XУ;

   б) токи текут по прямым, скрещивающимся в пространстве.

Частица в магнитной ловушке

4. Исследуйте движение заряженной частицы в магнитных полях, заданных в задачах 3а, 3б.

   Сила, действующая на частицу в магнитном поле, - сила Лоренца $\vec{F} = \frac{e}{c}[\vec{v}\vec{B}]$. Так как она зависит от скорости, схема расчета, используемая в работе "МАЯТНИК" и "ПЛАНЕТА", оказывается здесь недостаточно точной. Можно рекомендовать схему расчета, основанную на равенстве вида

$$x(t+h) = x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \frac{h^3}{6}\dddot{x}(t) + O(h^4),$$

   причем $\dot{x}(t) = v(t), \ddot{x}(t) = f(t) = F(t)/m$, а $\dddot{x}(t)$ с той же точностью выражается как $\frac{f(t)-f(t-h)}{h}$.

5. Исследуйте движение заряженной частицы в поле магнитного диполя (моделируя тем самым движение частиц в радиационных поясах).

Фокусировка пучков частиц

6. Пучок частиц, летевших параллельно оси X, фокусируется в поле $U(x, r)$. Изобразите движение группы частиц, стартовавших с большого расстояния с разными прицельными параметрами одновременно:

   а) $U = \frac{Ay^2}{r^2+B^2}$, б) $U = \frac{Ay^2}{(r^2+B^2)^2}$.

7. Пучку частиц, фокусируемых в поле, можно сопоставить картину фокусировки волн в неоднородной среде. Можно показать (но мы не будем этого делать), что длина волны должна быть обратно пропорциональна скорости частицы. Иначе говоря, фронт волны должен смещаться в направлении $\vec{v}$, но на расстояние

   $$\Delta\vec{r} \sim \vec{v}\Delta t/v^2.$$

   Получите картину волновых поверхностей, соответствующих фокусировке пучка в полях, заданных в задаче 9.

8. Пучок частиц летит параллельно оси X и рассеивается в поле

   $$U(r) = -\alpha/\sqrt{r^2 + a^2}.$$

   Определить распределение рассеянных частиц по углам. Считать, что падающий поток частиц равномерно распределен по площади своего поперечного сечения.

9. Движение частиц в стоячей волне на поверхности жидкости

   $$x = x_0 + ae^{kz_0}\cos kx_0\cos\omega t, z = z_0 + ae^{kz_0}\sin kx_0\cos\omega t.$$

   Получите картину движения частиц. Дуги траекторий, прорисованные частицами за время $\tau \ll 1/\omega$, дадут представление о линиях тока жидкости.

Концентрация частиц

10. Рассматривается диффузия (случайные блуждания) частиц в поле $U = \frac{k}{2}(x^2 + y^2 + z^2)$. Определите наблюдаемую концентрацию частиц в зависимости от $r = \sqrt{x^2 + y^2 + z^2}$.

В. Приближенные методы решения систем дифференциальных уравнений

В.1. Примеры численного решения дифференциальных уравнений

Во многих случаях бывает нужно исследовать так называемые *динамические системы* — системы, движение которых определяется дифференциальными уравнениями. Рассмотрим простой пример — движение частицы в поле сил, причем ограничимся движением вдоль одной прямой. Координата частицы x и ее скорость v определяются уравнениями

$$\frac{dx}{dt} = v, \quad \frac{dv}{dt} = a(x),$$

где $a(x) = F(x)/m$, $F(x)$ — действующая на частицу сила, m — масса частицы. Задача состоит в вычислении зависимостей $x(t)$, $v(t)$ при условии, что заданы начальные значения координаты и скорости значения $x(0)$ и $v(0)$.

Вид этих уравнений представляет собой очевидный намек на возможный способ вычислений: выбрать достаточно малое конечное значение величины dt, а затем воспользоваться соотношениями

$$x(t+dt) \approx x(t) + \frac{dx}{dt}dt = x(t) + v(t)dt,$$

$$v(t+dt) \approx v(t) + \frac{dv}{dt}dt = v(t) + a(t)dt.$$

Многократно применяя эти соотношения, можно рассчитать значения x и v в ряде дискретных, но достаточно близких друг к другу точек. Чтобы оценить величину отклонения от истинного закона движения, запишем, например для x, более точное равенство

$$x(t+dt) \approx x(t) + \frac{dx}{dt}dt + \frac{1}{2}\frac{d^2x}{dt^2}(dt)^2 = x(t) + v(t)dt + \frac{1}{2}a(t)(dt)^2. \quad (*)$$

Из него видно, что неточность на каждом шаге пропорциональна $(dt)^2$; для продвижения на время t необходимо число шагов, равное t/dt, так что неточность окажется пропорциональна $t \cdot dt$. Если необходимо (при заданном интервале t) улучшить точность в 10 раз, то нужно в 10 раз уменьшить шаг dt. Во столько же раз вырастет число шагов и время, необходимое для расчета.

Смысл равенства $(*)$ состоит в учете ускорения на интервале dt. Можно добиться примерно той же точности, что и в $(*)$, если взять среднее значение скорости на том же интервале или, что удобнее всего, значение скорости в середине интервала, $v(t+dt/2)$. Ускорение же, необходимое для выполнения сдвига на шаг по

времени для скорости, нужно будет вычислять в середине интервала $(t + dt/2, t + dt/2 + dt)$, т.е. в момент $t + dt$, что нас тоже устраивает, так как это позволит найти $x(t + dt)$.

Таким образом, для увеличения точности достаточно вычислять значения координат в моменты времени

$$t, \qquad t + dt, \qquad t + 2dt, \qquad t + 3dt, \ldots ,$$

а значения скорости — в моменты

$$t + dt/2, \quad t + 3dt/2, \quad t + 5dt/2, \quad t + 7dt/2, \ldots .$$

Для оценки неточности расчета запишем (вновь только для x)

$$x(t + dt) \approx x(t) + v(t + dt/2)dt \approx x(t) + \left(v(t) + \frac{dv}{dt}dt/2\right) dt,$$

что совпадает с $(*)$. Неточность имеет порядок $(dt)^3$.

Очевидно, что такой способ расчетов применим и для векторных величин.

Однако этот прием не срабатывает, если сила, действующая на частицу, зависит не только от координат, но и от скорости.

B.2. Численное решение обыкновенных дифференциальных уравнений в системе MATLAB

Выработано много удобных и относительно экономных способов проводить подобные вычисления. С ними вам предстоит познакомиться в курсе "Вычислительные методы". В системе **MATLAB** существует целый пакет процедур, предназначенных для решения систем обыкновенных дифференциальных уравнений, а если формулировать еще более точно, то для решения задачи Коши для системы обыкновенных дифференциальных уравнений (ОДУ). Причем этот пакет обеспечивает по вашему выбору разные алгоритмы решения такой задачи, но для экономии усилий и облегчения работы обращение и написание требуемого дополнительного кода одинаково и не зависит от алгоритма. Поэтому рассмотрим здесь метод подготовки M-файлов для решения таких систем независимо от используемого алгоритма решения.

Из приведеных выше примеров понятно, что большой класс ОДУ, а именно уравнения с одной независимой переменной, которую чаще всего именуют временем t, разрешенные относительно старшей производной могут быть сведены к системе дифференциальных уравнений первого порядка вида

$$\mathbf{y}'(t) = \mathbf{F}(t, \mathbf{y}(t))$$

с начальными условиями

$$\mathbf{y}(t_0) = \mathbf{y}_0.$$

Если правые части соответстующей системы достаточно гладки, то описанная система имеет единственное решение, которое может быть получено численно с помощью одного из алгоритмов, используемых в системе **MATLAB**.

B.3. Подготовка M-файла для решения ОДУ(пример)

Рассмотрим пример подготовки M-файла для решения уравнения Ван-дер-Поля [37].

$$y'' - \mu(1 - y^2)y' + y = 0, \quad \mu > 0\,.$$

Сделав очевидные замены $y_1 = y, y_2 = y'$, можно переписать это уравнение в виде системы двух уравнений:

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= \mu(1 - (y_1)^2)y_2 - y_1\,. \end{aligned}$$

Для решения этой системы уравнений необходимо написать M-файл, который описывал бы правую часть системы уравнений. Для нашего случая эта функция ($\mu = 1$, а y_1 и y_2 становятся элементами двумерного вектора `y(1)` и `y(2)`) должна выглядеть следующим образом:

```
function dy = vdp1(t,y)
dy = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

Хотя в рассматриваемом случае правая часть системы не зависит явно от t, а для некоторых систем уравнений может не зависеть от `y`, функция должна иметь не менее двух формальных параметров **t** и **y**.

Для решения системы уравнений на временном интервале $[0\ 20]$ и с начальными значениями $y_1(0) = 2, y_2(0) = 0$ необходимо написать

```
[T, Y]= ode45('vdp1',[0 20],[2;0]);
```

В результате решения получим вектор-столбец **T**, содержащий точки на заданном временном интервале, и матрицу **Y**, каждая строка которой соответствует времени из **T**. Для того чтобы увидеть результат решения, можно написать такую последовательность команд

[37] **Известное в литературе уравнение Ван-дер-Поля характеризуется тем, что в зависимости от значения параметра μ оно имеет разную степень жесткости и в связи с этим часто используется для тестирования численных решений.**

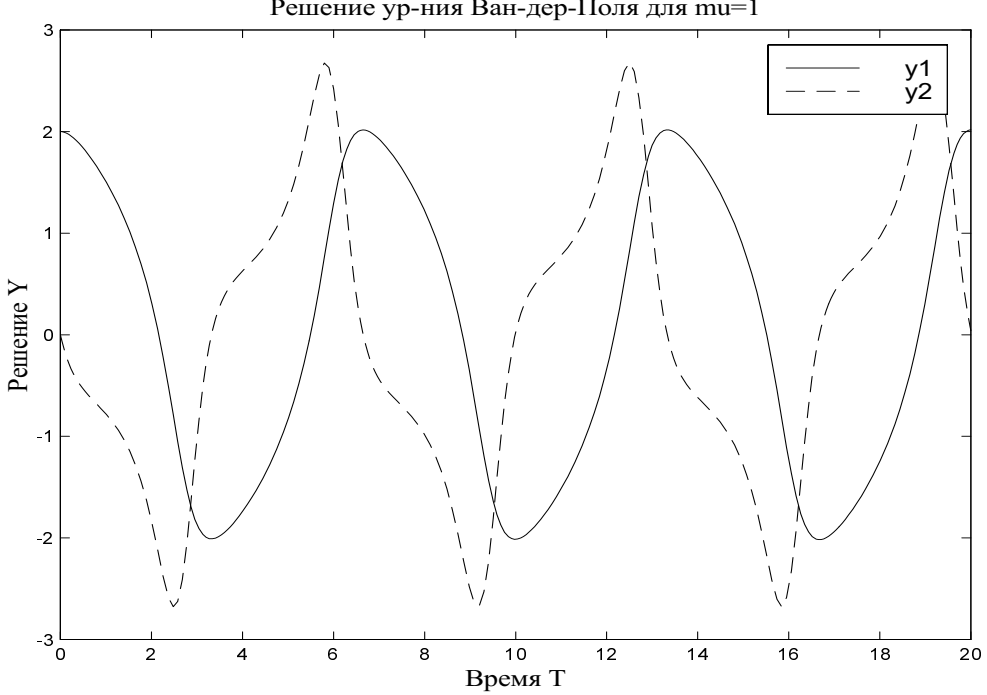


Рис. 15. Представление результатов решения уравнения Ван-дер-Поля для mu=1 на экране

```
.....................................
% Вывод результатов решения
 plot(T,Y(:,1),'-',T,Y(:,2),'--');
  title('Решение ур-ния Ван-дер-Поля для mu=1');
  xlabel('Время T');
  ylabel('Решение Y');
  legend('y1','y2');
.....................................
```

В результате получим картинку, подобную рисунку 15.

B.4. Методы решения ОДУ

При решении тех или иных задач можно выбирать разные процедуры численного решения ОДУ. Так, в приведенном выше примере использовалась функция **ode45**. Для решения нежестких систем уравнений в **MATLAB** имеются следующие функции

ode45 базируется на явном методе Рунге-Кутта. Это одношаговый алгоритм - для вычисления $y(t_n)$ необходимо знание решения в одной предыдущей точке $y(t_{n-1})$. Эта функция наиболее удобна для первого, 'пристрелочного' решения большинства задач.

ode23 тоже базируется на явном методе Рунге-Кутта, но меньшего порядка, поэтому бывает более подходящим для получения более грубого решения (с меньшей точностью) и при наличии небольшой жесткости. Также является одношаговым методом.

ode113 использует метод переменного порядка Адамса-Бэшфорта-Милтона. Он может оказаться более эффективным, чем метод **ode45**, особенно при высоких точностях и при сложности вычисления правых частей уравнений. Метод многошаговый, поэтому для начала решения необходимо знание решения в нескольких начальных точках.

Для решения жестких систем уравнений в системе **MATLAB** предусмотрены 4 функции.

ode15s базируется на методе численного дифференцирования назад, известного как метод Гира. Также, как и метод **ode113**, этот метод является многошаговым. Если вы считаете, что у вас жесткая задача или не смогли ее решить с помощью **ode45**, попробуйте **ode15s**.

ode23s использует метод Розенброка второго порядка. Поскольку это одношаговый метод, он может быть более эффективен, чем метод **ode15s**, для случаев невысокой точности.

ode23t является реализацией правила трапеций со свободным множителем. Этот метод имеет смысл использовать только если задача умеренно жесткая и нет нужды в численном демпфировании решения.

ode23tb реализует двухстадийное решение по неявной формуле Рунге-Кутта. Как и метод **ode23s** этот метод эффективен при невысокой требуемой точности решения.

B.5. Общие правила вызова решателей ОДУ

Все перечисленные выше функции вызываются одинаковым образом. В простейшем виде это выглядит следующим образом:

[T,Y] = odeXX('F',tspan,y0),

где

odeXX - любая из функций, перечисленных выше;

'F' - строка, содержащая имя файла с описанием правых частей системы;

tspan - вектор, определяющий интервал интегрирования. Если вектор **tspan = [t0 tfinal]** имеет всего два элемента, то интегрирование идет от **t0** до **tfinal**. Если вектор **tspan** имеет более двух элементов, то функция **odeXX** выдает решение во всех точках, которые перечислены в векторе **tspan**, отметим, что **t0 > tfinal** допустимо;

y0 - вектор начальных условий задачи;

T - вектор-столбец моментов времени;

Y - матрица решений. Каждая строка матрицы содержит вектор решений (все **y(i)**) для соответствующего момента времени.

В дополнение к простейшему варианту вызова решателя может использоваться более сложный синтаксис, который расширяет наши возможности по управлению решением. Любая из функций **odeXX** может содержать четвертый и последующие аргументы, т.е. обращение может выглядеть следующим образом:

[T,Y] = odeXX('F',tspan,y0,options,p1,p2,...).

Аргумент **options** задается специальным образом с помощью функции задания опций (см. в помощи **odeset Function**). Больший интерес представляют дополнительные параметры **p1,p2,...**. Дело в том, что эти параметры передаются в функцию определения правых частей в виде

F(t,y,flag,p1,p2,...).

Подробнее способы описания правых частей системы и ряда дополнительных параметров приведены в следующем параграфе.

B.6. Общие правила определения функции правых частей

При написании функции правых частей для каждой задачи необходимо руководствоваться следующими правилами.

- Функция описания правых частей должна содержать не менее двух входных аргументов **t** и **y**, даже если какой-то из них не используется явно при вычислении правых частей.
- Правые части системы, которые вычисляются этой функцией $F(t, y)$, должны образовывать вектор-столбец.

- Любые дополнительные параметры, которые необходимо передавать функции $F(t, y)$, должны быть в конце списка параметров самой функции (после специального параметра **flag**) и в списке аргументов вызова решателя там же.

Рассмотрим ранее описанный пример функции для системы Ван-дер-Поля. В приведенном ранее примере предполагалось, что $\mu = 1$. Рассмотрим вариант с передачей этого параметра явно.

```
function [out1,out2,out3] = vdpode(t,y,flag,mu)
if nargin < 4 | isempty(mu) % Если нет 4-го параметра,
 mu = 1;                    % то mu=1
end
% Если нет 3-го параметра
if nargin < 3 | isempty(flag)
% вычисляем правые части dy/dt = F(t,y)
 out1 = [y(2); mu*(1_y(1)^2)*y(2)_y(1)];
% Если аргумента 3 и flag='init'
elseif strcmp(flag,'init')
% Вывод [tspan,y0,options].
% Начальные условия включены в функцию vdpode
out1 = [0; 20]; % tspan
out2 = [2; 0];  % начальные условия
% дополнительные опции (точность)
out3 = odeset('RelTol',1e_4);
end
```

C. Метод наименьших квадратов

С помощью метода наименьших квадратов находится прямая на плоскости XY(рис. 16), проходящая в определенном смысле наиболее близко к заданным N точкам с координатами (x_i, y_i). Речь идет о такой прямой, чтобы сумма квадратов отклонений от нее по вертикали Δy_i принимала наименьшее значение.

Такая задача возникает, если мы хотим аппроксимировать линейной функцией $y = a_0 + a_1 x$ экспериментально найденную зависимость $y(x)$, причем точность определения величин x была высокой, а величины y находились приближенно. Минимум функции $F = \sum_i (a_0 + a_1 x_i - y_i)^2$ определяется условиями

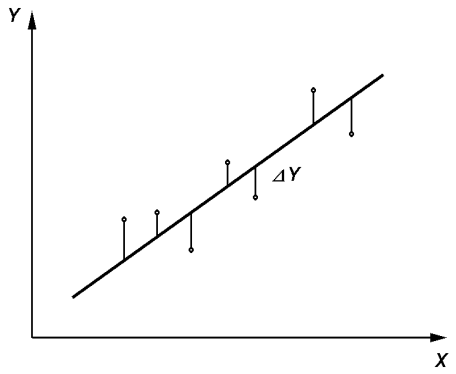


Рис. 16. Аппроксимация данных методом наименьших квадратов

$$\partial F/\partial a_0 = \partial F/\partial a_1 = 0,$$

которые приводят к системе линейных уравнений

$$N a_0 + a_1 \sum x_i = \sum y_i,$$
$$a_0 \sum x_i + a_1 \sum x_i^2 = \sum x_i y_i.$$

Из этой системы уравнений находятся коэффициенты a_0 и a_1, которые определяют искомую прямую.

Пусть данные (вектора x_i и y_i) представляют собой вектор-столбцы. Введя еще один вектор-столбец **od**, совпадающий по длине с длиной **x**, состоящий из единиц, можно записать вышеприведенную систему уравнений (точнее, вычисление ее коэффициентов) в виде

```
od = ones(size(x));
s(1,1) = od*(od');
s(2,1) = od*(x');
s(1,2) = s(2,1);
s(2,2) = x*(x');
ρ = [od*(y'); x*(y')];
```

Тогда саму систему уравнений можно записать в матричном виде

$$\mathbf{s}a = p,$$

а решение ее в виде

$$a = \mathbf{s}^{-1} p\,.$$

Используя правила обращения матриц в **MATLAB**, это решение можно записать в виде **a = s \ p;**. Полученный вектор-столбец **a** и будет содержать требуемые коэффициенты.

Можно вычислить соответствующие коэффициенты и насчитать полученную «теоретическую» прямую (в общем случае - кривую) с помощью функций **polyfit** и **polyval**, например, следующим образом:

```
a = polyfit(x,y,1);    % Вычисление коэффициентов
y1 = polyval(a,x);     % Расчет аппроксимирующих значений
```

Подробнее об этих функциях и их возможностях вы можете узнать во встроенной помощи (**help polyfit**) или в [2, 3, 6].

D. Моделирование распределения случайных величин

Датчик случайных чисел **rand** является встроенной функцией системы **MATLAB**. При каждом обращении возвращает случайное число из равномерного распределения на интервале (0,1).

Легко построить функцию, генерирующую равномерное распределение на произвольном интервале [A,B]:

X=A+(B-A)*rand.

Пусть нужно получить значения случайной величины X, распределенной на интервале [A,B] с плотностью $f(X)$ (рис. 17). Выберем прямоугольник ABCD, высота которого H не меньше максимального значения функции $f(X)$ на [A,B].

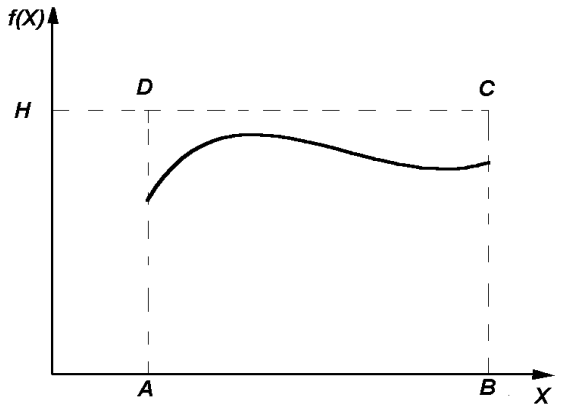


Рис. 17. Функция отбраковки

Будем набрасывать на прямоугольник точки равномерно по его площади и браковать те из них, которые попадают выше кривой $f(X)$. Значения X для *принятых* точек и будут реализовать искомое распределение, т.е получаемые значения X будут распределены на интервале [A,B] с плотностью $f(X)$.

Участок программы, реализующий этот алгоритм, может выглядеть, например, следующим образом:

```
.....................
  x=A+(B-A)*rand;
  while (f(x) < H*rand)
    x=A+(B-A)*rand;
  end;
.....................
```

При этом функция $f(X)$ должна быть описана в смысле функции **MATLAB**. При каждом обращении он будет давать значение X из требуемого распределения. Ясно, что чем больше площадь над кривой $f(X)$, тем реже мы будем браковать точки. Поэтому лучше всего выбрать H равным максимальному значению $f(X)$. Указанный способ позволяет генерировать и многомерные распределения случайных величин.

Еще одна важная задача, возникающая при моделировании угловых распределений, — это реализация равномерного (изотропного) распределения в пространстве. Если задавать направление единичным вектором, выходящим из начала координат, то концы таких векторов расположены на поверхности единичной сферы. Равная вероятность для любого направления означает, что конец вектора — случайная точка Ω, равномерно распределенная на поверхности этой сферы. Вероятность того, что Ω окажется в любом элементе поверхности dS , равна $dS/4\pi$. Выбирая на поверхности сферы сферические координаты (θ, φ) с полярной осью OZ, имеем $dS = \sin\theta d\theta d\varphi$, где $0 \leq \theta \leq \pi, 0 \leq \varphi \leq 2\pi$. Для получения (розыгрыша) случайного направления нам понадобятся два случайных числа. Соответствующий участок программы имеет вид

```
.................
  C=2.0*rand-1;
  Fi=2.0*pi*rand;
.................
```

Здесь **Fi** — угол φ , а **C** — косинус угла θ.

В системе **MATLAB** есть также встроенная функция, задающая нормальное распределение — **randn(m,n)**. Эта функция генерирует матрицу **m x n** случайных чисел с нормальным распределением. Это распределение имеет нулевое среднее и единичную дисперсию.

E. Краткий справочник по функциям MATLAB

Краткий справочник по функциям MATLAB содержит 20 основных групп функций. По команде **help** осуществляется вывод списка этих групп. По команде **help <имя группы>** выводится список функций, содержащихся в этой группе. Порядок, в котором приведены далее группы функций, может служить указателем для всего краткого справочника.

*Ф*ункция	*Д*ействие
Группы функций	
general	команды общего назначения
ops	операторы и специальные символы
lang	конструкции языка
elmat	элементарные матрицы и действия с матрицами
specmat	специальные матрицы
elfun	элементарные математические функции
specfun	специальные математические функции
matfun	матричные функции - численные методы линейной алгебры
datafun	обработка данных и преобразование Фурье
polyfun	полиномиальные функции и интерполяция
sparfun	работа с разреженными матрицами
plotxy	двумерная графика
plotxyz	трехмерная графика
graphics	графические функции общего назначения
color	управление цветом и освещением
sounds	функции работы со звуком
strfun	функции работы со строковыми переменными
iofun	функции ввода/вывода низкого уровня
demos	демонстрации и примеры
Клавиши редактирования	
↑ (Ctrl+P)	Вызов предыдущей строки
↓ (Ctrl+N)	Вызов следующей строки
Команды общего назначения	
Управление командами и функциями	
demo	запуск демонстрационных примеров
help	онлайновая помощь
lookfor	поиск ключевых слов в помощи

Функция	*Действие*
path	исполняемая команда операционной системы
what	список M-, MAT-, и MEX-файлов в директории
Управление переменными и рабочим пространством	
clear	очистка переменных и функций в памяти
length	длина вектора
load	восстановить переменные с диска
save	сохранить переменные рабочей области на диске
size	размерность матрицы
who	список текущих переменных
whos	список текущих переменных, развернутая форма.
Работа с файлами и операционной системой	
cd	сменить текущую рабочую директорию
delete	удалить файл
diary	сохранить в файле протокол текущей сессии
dir	листинг текущей директории
!	Выполнять команду DOS
Управление командным окном	
clc	Очистить командное окно
echo	Печатать выполняемые команды программы
Запуск и выход из MATLAB'а	
quit	выйти из MATLAB'а
startup	файл, выполняемый при запуске MATLAB'а
Операторы и специальные символы	
Арифметические и матричные операторы	
*	умножение матриц
.*	поэлементное умножение матриц
^	возведение матрицы в степень
.^	возведение элементов матрицы в степень
kron	тензорное произведение матриц
\	левое деление матриц
/	правое деление матриц
./	поэлементное деление матриц
'	эрмитовское сопряжение
.'	транспонирование
Операторы отношения	

Функция	*Действие*
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно
==	равно
~=	не равно
	Логические операторы
&	и (AND)
|	или (OR)
~	не (NOT)
xor	исключающее или (XOR)
	Специальные символы
.	десятичная точка
..	родительская директория
...	продолжение (перенос на следующую строку)
%	комментарий
	Логические функции
all	истина, если верно для всех элементов матрицы
any	истина, если верно хотя бы для одного элемента матрицы
exist	проверяет, существует ли переменная или функция
find	находит индексы ненулевых элементов
finite	истина для конечных элементов
	Языковые конструкции и отладка
	MATLAB как язык программирования
eval	выполнить строку
feval	выполнить функцию
function	определить функцию
global	сделать переменную глобальной
nargchk	установить кол-во входных аргументов
	Операторы управления
break	выйти из цикла
end	конец цикла или условного оператора
error	выдать сообщение и прекратить выполнение функции
for	цикл с заданным числом повторений
if,else, elseif	условные операторы

Функция	*Действие*
return	возврат в вызвавшую функцию
while	цикл с неопределенным числом повторений
Интерактивный ввод	
input	ожидает ввод пользователя
pause	ожидает нажатия клавиши (или заданное время)
Матрицы и операции с ними	
Элементарные матрицы	
eye	единичная матрица
linspace	вектор с равноотстоящими элементами
logspace	вектор с равноотстоящими в логарифмическом масштабе элементами
meshgrid	задает двумерную сетку для 3d-графика
ones	матрица единиц
rand	равномерно распределенные случайные числа
randn	нормально распределенные случайные числа
zeros	нулевая матрица
:	вектор с равноотстоящими элементами
Специальные переменные и константы	
ans	результат последней операции
eps	относительная точность вычислений с плавающей запятой
flops	счетчик числа операций с плавающей запятой
i,j	мнимая единица
inf	бесконечность
nargin	кол-во входных параметров функции
nargout	кол-во выходных параметров функции
pi	число π
realmax	максимальное действительное число
realmin	минимальное действительное число
Дата и время	
clock	время
cputime	затраченное время процессора
date	дата
etime	затраченное время
tic,toc	запуск-останов времени
Операции с матрицами	

Функция	*Действие*
diag	выделить диагональ
fliplr	отразить матрицу по строкам
flipud	отразить матрицу по столбцам
reshape	изменить размер
rot90	повернуть матрицу на 90 градусов
tril	выделить нижнюю треугольную часть матрицы
triu	выделить верхнюю треугольную часть матрицы
Математические функции	
Элементарные математические функции	
abs	модуль
acos	арккосинус
acosh	гиперболический арккосинус
angle	угол
asin	арксинус
asinh	гиперболический арксинус
atan	арктангенс
atanh	гиперболический арктангенс
conj	сопряжение
cos	косинус
cosh	гиперболический косинус
exp	экспонента
imag	мнимая часть
log	натуральный логарифм
log10	десятичный логарифм
real	действительная часть
rem	остаток от деления
round	округление до ближайшего целого
sign	сигнум-функция (знак)
sin	синус
sinh	гиперболический синус
sqrt	квадратный корень
tan	тангенс
tanh	гиперболический тангенс
Спецфункции	
bessel	функция Бесселя
besselh	функция Ганкеля

Функция	*Действие*
betainc	неполная бета-функция
betaln	логарифм бета-функции
ellipj	эллиптическая функция Якоби
ellipke	полный эллиптический интеграл
erf	функция ошибок
erfinv	обратная функция ошибок
gamma	гамма-функция
gammaln	логарифм гамма-функции
gammainc	неполная гамма-функция
Функции от матриц - линейная алгебра	
Матричный анализ	
cond	число обусловленности
det	определитель
norm	норма
orth	ортогонализация
rank	ранг
trace	сумма диагональных элементов
Системы линейных уравнений	
inv	обратная матрица
\ or /	решение линейной системы
Собственные значения	
eig	собственные значения и собственные вектора
poly	характеристический полином
Функции от матриц	
expm	матричная экспонента, 4 варианта
funm	любая функция от матриц
logm	логарифм матрицы
sqrtm	корень из матрицы
Обработка данных и преобразование Фурье	
Основные операции	
cumprod	произведение элементов с накоплением
cumsum	сумма элементов с накоплением
max	максимальный элемент
mean	среднее значение
min	минимальный элемент

Функция	*Действие*
prod	произведение всех элементов
sort	сортировка по возрастанию
std	дисперсия
sum	сумма всех элементов
trapz	интегрирование методом трапеций
Конечные разности	
diff	численное дифференцирование
gradient	градиент
Корреляция	
corrcoef	коэффициенты корреляции
cov	матрица ковариации
Фильтрация и конволюция	
conv	конволюция и умножение полиномов
deconv	деконволюция и деление полиномов
filter	цифровой фильтр
Преобразование Фурье	
abs	модуль
angle	фаза
fft	дискретное преобразование Фурье
fftshift	сдвигает 0 в середину спектра
ifft	обратное дискретное преобразование Фурье
nextpow2	ближайшая большая степень двойки
Полиномиальные и интерполяционные функции	
Полиномиальные функции	
conv	умножение полиномов
deconv	деление полиномов
poly	задать полином с известными корнями
polyder	производная полинома
polyfit	аппроксимация полиномом данных
polyval	вычислить полином
polyvalm	вычислить полином от матрицы
roots	корни полинома
Интерполяция данных	
interp1	интерполяция
Функции от функций	

Функция	*Действие*
Нелинейные численные методы	
fmin	минимизирует функцию одной переменной
fmins	минимизирует функцию нескольких переменных
fzero	находит 0 функции
ode23	решает систему дифференциальных уравнений (несколько вариантов)
quad	численно вычисляет интеграл
Двумерная графика	
Элементарная x-y графика	
comet	отрисовка плоской траектории
fill	рисует закрашенный многоугольник
line	рисует график
loglog	график с логарифмическими осями
plot	линейный график
semilogx	график с логарифмической осью x
semilogy	график с логарифмической осью y
Специализированные графики	
bar	рисует диаграмму
errorbar	график экспериментальных данных с ошибкой измерения
fplot	график функции
hist	насчитывает и рисует гистограмму
polar	график в полярных координатах
Разметка графика	
grid	показать сетку
gtext	выводит текст, перемещаемый с помощью мыши
text	выводит текст в заданные координаты
title	выводит название графика
xlabel	подпись оси x
ylabel	подпись оси y
Трехмерная графика	
comet3	отрисовка 3-мерной траектории
plot3	рисует линии и точки в 3d
contour	рисует изолинии
image	выводит рисунок
mesh	3d поверхность

Функция	*Действие*
surf	3d поверхность с тенями
colormap	задает цветовую палитру
Графические функции общего назначения	
Создание и контроль окна графика	
clf	очистить окно
close	закрыть окно
figure	создать окно
gcf	получить дескриптор активного окна
Создание и контроль осей	
axes	создать оси
axis	задать масштаб и внешний вид осей
cla	очистить активные оси
gca	получить дескриптор активных осей
hold on/off	рисовать поверх или стирая
subplot	разбить окно на несколько графиков
Операции управления графикой	
delete	удалить объект
drawnow	нарисовать всю накопленную графику
get	получить свойства объекта
reset	восстановить свойства объекта (по умолчанию)
set	установить свойства объекта
print	печатать график
Строковые функции	
setstr	преобразует число в строку
strcmp	сравнить строки
int2str	преобразует целое число в строку
num2str	преобразует число в строку
str2num	преобразует строку в число
Функции ввода/вывода	
fclose	закрыть файл
fopen	открыть файл
fread	читать бинарный файл
fwrite	писать в бинарный файл
fgetl	прочитать строку из файла
fprintf	писать данные в файл по формату

Функция	*Действие*
fscanf	читать данные из файла по формату

F. Компиляция файлов MATLAB

MATLAB - язык, ориентированный на работу с матрицами. Вычисления, которые удается записать в матричном виде, выполняются достаточно быстро. Однако векторизовать программу удается не всегда, а в этом случае **MATLAB** будет проигрывать в скорости языкам более низкого уровня, например C.

Существует несколько способов ускорить вычисления:

- перевести текстовые m-файлы в p-файлы, содержащие внутренний код **MATLAB** (*pcode*)
- откомпилировать m-файлы в dll-библиотеки (*mcc*)
- написать наиболее медленные части программы в виде функций на C или Фортране, откомпилировать их в dll-библиотеки и вызывать из **MATLAB**.

F.1. Использование команды pcode

Этот способ дает выигрыш в скорости, если программа очень длинная или имеется много вызовов различных функций в m-файлах. Если в программе многократно вызывается одна и та же функция, то она переводится в p-файл при первом вызове и хранится в этом виде для будущих вызовов. В этом случае выигрыш в скорости будет мал. Возможные форматы вызова:

- **pcode file1 (file2 ...)** переводит file1.m в file1.p и т.д., file1.m может находиться в любой директории, путь на которую указан в pathdef.m, file1.p создается в текущей директории
- **pcode *.m** переводит все m-файлы в текущей директории в p-файлы
- **pcode file1 (file2 ...) -inplace** создает p-файл в той же директории, где был m-файл.

F.2. Использование команды mcc

Второй способ позволяет перевести функции MATLAB в исполняемый код, что позволяет ускорить программу на 20-40%. Кроме того, полученную dll-библиотеку можно использовать в любом языке: например, вызывать графические функции

MATLAB из программы на C или Фортране. Для компиляции m-файлов необходимы:

- компилятор C, поддерживающий создание dll-библиотек (Borland C/C++ 5.x, Microsoft Visual C++ 4.2 или 5.0, Watcom C/C++ 10.6 или 11)
- MATLAB Compiler (необходимо указать при установке MATLAB)

Команда mcc с различными ключами также позволяет перевести m-файл в C-файл, откомпилировать программу MATLAB в исполняемый файл (который можно будет выполнять на компьютерах, где MATLAB не установлен), и др.

F.3. Использование команды mex

Третий способ не имеет прямого отношения к MATLAB. Можно написать любую функцию на C (или Фортране), включить ее в dll-библиотеку, а затем вызывать из MATLAB. Для создания dll-библиотеки необходимо кроме самой функции на C написать еще одну функцию, передающую входные аргументы вызывающей программы в функцию и возвращающую выходные аргументы вызывающей программе. При этом файл с программой на C (timestwo.c) имеет следующий формат:

%%

```
#include "mex.h"
/*
* timestwo.c - пример взят из API-guide
*
* Функция умножает скаляр на 2.
*
* Это MEX-file для MATLAB.
* Copyright (c) 1984-1998 The MathWorks, Inc.
*/
/* $Revision: 1.5 $ */
void timestwo(double y[], double x[])
{
y[0] = 2.0*x[0];
}
void mexFunction( int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[] )
{
```

```
double *x,*y;
int mrows,ncols;
/* Check for proper number of arguments. */
 if(nrhs!=1)
{
mexErrMsgTxt("One input required.");
}
else
  if(nlhs>1)
  {
  mexErrMsgTxt("Too many output arguments");
  }
/* The input must be a noncomplex scalar double.*/
 mrows = mxGetM(prhs[0]);
ncols = mxGetN(prhs[0]);
if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
!(mrows==1 && ncols==1) ) {
mexErrMsgTxt("Input must be a noncomplex scalar double.");
}
/* Create matrix for the return argument. */
 plhs[0] = mxCreateDoubleMatrix(mrows,ncols, mxREAL);
/* Assign pointers to each input and output. */
 x = mxGetPr(prhs[0]);
y = mxGetPr(plhs[0]);
/* Call the timestwo subroutine. */
 timestwo(y,x);
}
```

Здесь описана функция **timestwo**, умножающая число на два, и функция **mexFunction**, которая принимает входной аргумент **x** и возвращает выходной **y**. Аргументами последней являются:

nlhs - число выходных аргументов dll-функции

nrhs - число входных аргументов

***plhs[ ] -** массив указателей на первые элементы выходных аргументов

***prhs[ ] -** массив указателей на первые элементы входных аргументов.

Имеется множество функций для передачи переменных разнообразных типов и вывода сообщений (например **mxGetPr** и **mxGetPi** - для получения указателей на реальную и мнимую часть входного массива, **mxGetString** - для получения указателя на строку, **mxGetN** и **mxGetM** для получения размеров массива, **mxCreateNumericArray**,
mxCreateDoubleMatrix, **mxCreateString** - для задания указателей на массивы, матрицы и строки, и т.д. Полный список можно посмотреть в файле

$$\backslash matlab5.2\backslash help\backslash techdoc\backslash APIREF\backslash APIREFTOC.HTML$$

Если откомпилировать этот файл, то функцию **timestwo** можно будет вызывать из **MATLAB**. Формат вызова mex:

mex -setup необходимо запускать после установки нового компилятора C (или фортрана), а также один раз после установки **MATLAB**.

mex file компилирует **file.c** и создает **file.dll**.

Различные дополнительные ключи можно узнать, набрав **help mex** в окне **MATLAB**.

Список литературы

[1] Коткин Г.Л., Черкасский В.С. Численное моделирование физических процессов. Новосибирск: НГУ, 1998. 123 с.

[2] Потемкин Г.В. Matlab 5 для студентов. М.: Диалог-МИФИ, 1998. 314 с.

[3] Потемкин Г.В. Система MATLAB. Справочное пособие. М.: Диалог-МИФИ, 1998. 314 с.

[4] Мартынов Н.Н., Иванов А.П. MATLAB 5.X. Вычисления, визуализация, программирование. М.: КУДИЦ-ОБРАЗ, 2000. 332 с.

[5] Marchand Patrick. Graphics and GUI with MATLAB. Second edition. CRC Press LLC, New York; Washington, 1999. 445 p.

[6] Using MATLAB. Version 5.2. The MathWorks, Inc. Natick, MA, 1998.

[7] Ландау Л.Д., Лифшиц Е.М. Механика. М.: Наука, 1976.

[8] Коткин Г.Л. Лекции по статистической физике. Новосибирск: НГУ, 1996.

[9] Коткин Г.Л., Сербо В.Г. Сборник задач по классической механике. М.:Наука, 1977.

[10] Котельников И.А., Чеботаев П.З. Введение в систему LaTeX. Новосибирск: Сиб. хронограф, 1999.

[11] Вельтмандер П.В., Фруми Л.Л. Моделировыание в курсе электротехники и электроники. Новосибирск: НГУ, 1995. 65 с.

ДОПОЛНЕНИЕ
ВВЕДЕНИЕ В СИСТЕМУ MATLAB

MATLAB является интерактивной, матрично-ориентированной системой для научных и инженерных расчетов. Система позволяет решать сложные численные проблемы без написания каких-либо программ. Имя **MATLAB** является аббревиатурой двух слов **MATrix LABoratory** (МАТричная ЛАБоратория). Цель этой главы - помочь начать работать с системой **MATLAB**. Лучше всего это делать прямо на компьютере. Желательно изучать это руководство и сразу выполнять все примеры, экспериментируя с ними. Во время работы с системой можно пользоваться встроенной оперативной помощью, которая содержит много подробностей. После входа в систему, как это описано в п. 1.4, команда **help** выведет список групп, в которые объеденены функции. Команда **help <имя_группы>** выведет на экран список функций, размещенных в этой группе с краткими пояснениями. Команда **help <имя_функции>** выдаст подробную информацию о функции. Например, команда **help eig** выдаст информацию о функции вычисления собственных значений матрицы **eig**. Вы можете познакомиться с некоторыми возможностями системы **MATLAB** с помощью команд **intro** и **demo**. Для более глубокого знакомства с системой желательно ознакомиться с книгой «MATLAB User's Guide» (Руководство пользователя). Система **MATLAB** может работать на таких платформах как Sun/Apollo/VAXstation/HP workstations, VAX, MicroVAX, Gould, PC и AT совместимые, 80386 и 80486 компьютеры, Apple Macintosh, и на ряде параллельных машин. В настоящем руководстве будут описаны основные свойства системы, которые одинаково применимы при использовании версий 5.0., 5.1 и 5.2. **MATLAB** является собственным знаком фирмы MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, MA 01760, (508)653-1415, Fax: (508)653-2997,
Email: info@mathworks.com.

1. Работа в командном окне

1.1. Вход в систему MATLAB

В большинстве систем после входа в саму операционную среду войти в **MATLAB** можно, набрав в ответ на системный запрос команду **matlab**. При работе в Windows 3.1 или в Windows-95 необходимо найти соответствующую иконку и кликнуть на ней. Выход осуществляется с помощью команды **quit**. При работе в Windows 3.1 (версии 4.0 и 4.2) в качестве редактора m-файлов используется Notepade, а

в Windows-95 (версии 5.0 и выше) используется собственный встроенный редактор[1]. В обоих случаях командное окно системы **MATLAB** находится в одном окне, а редактор - в другом. В окне **MATLAB** помимо собственно команд **MATLAB** можно использовать системные команды **DOS**. Например, команда **dir** выводит на экран содержимое текущей директории, команда **what** выводит только список m-файлов[2] текущей директории. Команда **cd** позволяет сменить текущую директорию, а команды **delete** и **type** стирают и печатают на экране содержимое файла соответственно.

1.2. Интерактивный доступ к справочной информации и документации

Существуют следующие способы получить информацию о функциях системы **MATLAB** в процессе работы:

- команда **help**;
- команда **lookfor**;
- меню **Help**;
- просмотр и вывод на печать страниц документации;
- обращение к WEB-серверу фирмы The MathWorks.

1.2.1. Команда help

Основной и наиболее быстрый способ выяснить синтаксис и особенности применения m-функции - это использовать команду **help** <**имя m-функции**>. Соответствующая информация появляется непосредственно в командном окне.

Например, команда **help magic** выведет в командное окно следующую информацию на английском языке:

```
MAGIC  Magic square.
  MAGIC(N) is an N-by-N matrix constructed from
  the integers 1 through N^2 with equal row, column,
  and diagonal sums.
  Produces valid magic squares for N = 1,3,4,5,...
```

[1] **Далее речь будет идти только о версии 5.x и работе в системе Windows-95.**

[2] **Здесь и далее m-файлом мы будем называть любой текстовой файл, содержащий набор команд MATLAB и имеющий расширение .m. Подробнее об этих файлах и их роли в системе MATLAB см. п. 6 этого Дополнения.**

Следует обратить внимание, что текст интерактивной справки использует верхний регистр для написания имен функций и переменных, чтобы выделить их из основной части текста. Однако при использовании функций их имена необходимо вводить с помощью символов нижнего регистра.

Команда **help** без аргументов выводит на экран список каталогов, которые имеются в системе с кратким описанием их содержимого. Повторный набор этой команды с именем каталога, например **help elmat**, выведет список функций, предназначенных для создания и работы с матрицами специального вида. Ввод команды с именем определенной функции выдаст на экран описание этой функции. Следует особо обратить внимание, что в качестве ответа на запрос о помощи выводятся все строки комментариев, которые написаны в начале каждой функции - как созданной разработчиками системы, так и собственными функциями пользователя.

1.2.2. Команда lookfor

Эта команда позволяет выполнить поиск m-функции по ключевому слову; при этом анализируется первая строка комментария, и она же выводится на экран, если в ней встретилось ключевое слово. Например, в системе **MATLAB** нет m-функции с именем **inverse**, и поэтому на команду **help inverse** ответом будет - **inverse.m not found** (inverse.m не найден).

Однако команда **lookfor inverse** найдет не менее дюжины совпадений, и это будет зависеть от того, какие ППП (пакеты прикладных программ) подключены к системе **MATLAB**. Добавление к команде **lookfor** опции **-all** в виде

lookfor <слово> -all

расширяет область поиска - <слово> ищется в первом блоке комментариев, т.е. в блоке комментариев между заголовком функции и первым оператором.

1.2.3. Меню Help

Это меню командного окна системы **MATLAB** позволяет активизировать следующие окна:

- **Help Window**
- **Help Tips**
- **Help Desk(HTML)**
- **Examples and Demos**

- **About MATLAB**
- **Subscribe (HTML)**

Окно справки **Help Window** позволяет получить в отдельном окне то же самое, что и команда **help**. Отличие состоит в том, что в этом окне можно погружаться внутрь раздела с помощью двойного щелчка мыши, а не повторно набирая команду **help** с новыми аргументами.

В пункте меню **Help Tips** приведена краткая справка по использованию помощи, т.е. описаны все пункты этого меню.

Вызов пункта меню **Help Desk** позволяет получить доступ к большому объему справочной информации и к документации по системе, размещаемой на жестких дисках, либо на диске CD-ROM в формате HTML. При использовании этого пункта меню желательно, чтобы в системе было установлено какое-либо средство работы в Internet (например, Internet Explorer или Netscape Navigator). Все операторы и функции системы **MATLAB** описаны подробно и с большим числом примеров. Реализована поисковая система, позволяющая выполнить необходимые запросы.

В пункте меню **Examples and Demos** приведено большое число примеров использования системы **Matlab** для решения задач из разных областей.

Пункт меню **About Matlab** выводит на экран стандартную заставку **Matlab**, а в пункте меню **Subscribe HTML** предлагается подписаться (при наличии лицензии на продукт) на доступ через интернет к услугам фирмы **MathWorks**.

Просмотр и распечатка документации. Версии справочной документации доступны для просмотра и распечатки в формате **PDF** с помощью средства **Adobe Acrobat**. Оно позволяет просматривать текст в формате печатной страницы, с полным набором шрифтов, графики и изображений, с полным ощущением чтения книги. Одновременно это и наилучший способ получения копий нужных страниц.

1.3. Редактирование и перевызов командной строки

Командная строка **MATLAB** легко редактируется. Курсор можно перемещать с помощью стрелок $\leftarrow\rightarrow$ и удалять неправильно набранные символы с помощью клавиш **Backspace** или **Delete**. Другие редактирующие возможности тоже доступны. Если вы работаете на PC, то доступны клавиши **Home, End** и **Delete**. При работе на других системах необходимо использовать команду **edit** для ознакомления с доступными командами. Удобным свойством системы является возможность использовать клавиши-стрелки $\uparrow\downarrow$ для доступа к стеку с ранее введенными командами.

Таким образом, имеется возможность перевызвать ранее вызванную команду, отредактировать ее и снова выполнить. Для небольших процедур это гораздо удобнее, чем писать и отлаживать m-файлы, что требует постоянного перехода из окна **MATLAB** в окно текстового редактора (см. п. 7 и 7.2). Например, при необходимости сосчитать сколько флопсов (см. п. 7.4) требуется для обращения матриц разного размера, можно использовать приведенную далее командную строку, и перевызывать ее с помощью стрелки вверх с последующим редактированием и повторным запуском.

>> a = rand(8); flops(0), inv(a); flops

Если вы хотите сравнить графики функций $\sin(mx)$ и $\sin(nx)$ на интервале $[0, 2\pi]$ для различных значений n и m, то можете воспользоваться командной строкой

m=2;n=3;x=0:.01:2*pi;y=sin(m*x);z=sin(n*x); plot(x,y,x,z)

с последующим редактированием и перевызовом этой же самой строки. Проделайте все описанное выше самостоятельно.

1.4. Формат вывода

Поскольку все вычисления в **MATLAB** выполняются с двойной точностью, формат вывода может управляться с помощью следующих команд.

format short	с фиксированной точкой и 4 знаками после точки (по умолчанию)
format long	с фиксированной точкой и 14 знаками после точки
format short e	научная нотация с 4 десятичными знаками
format long e	научная нотация с 15 десятичными знаками

После вызова одного из приведенных выше форматов он сохраняется до вызова другого. Команда **format compact** подавляет большинство пустых строк, позволяя большее количество информации вывести на экран или страницу. Она не зависит от других команд формата.

1.5. Копия протокола сессии

Легче всего протокол сессии получить с помощью команды **diary**. Вызов команды **diary <имя_файла>** приведет к тому, что все появившееся далее на экране (кроме графики) будет записано в файл **<имя_файла>**. Если имя файла в команде будет опущено, то протокол сессии будет записан в файл с именем **diary**. Команда

diary off приведет к выключению записи, а команда **diary on** к восстановлению ее функции и т.д. После завершения вы можете отредактировать этот файл как обычный текстовой файл и распечатать, если необходимо, или использовать для последующего написания m-файла.

2. Введение матриц

MATLAB работает практически с одним видом объектов - с числовыми прямоугольными матрицами, элементами которых могут быть в общем случае комплексные числа. Все переменные представляют собой матрицы. В некоторых случаях матрицы 1x1 интерпретируются как скаляры, а матрицы с одной строкой или одним столбцом интерпретируются как вектора. В системе **MATLAB** матрицы могут быть введены разными способами.

- Введены явно с помощью списка элементов;
- Сгенерированы встроенными операторами или функциями;
- Созданы в m-файлах (см. п. 6 и 7 далее);
- Загружены из внешнего файла данных.

2.1. Явное определение матриц

Например, любой из приведенных далее операторов

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

или

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

создает матрицу 3x3 и присваивает ее значение переменной. Попробуйте сами поэкспериментировать с подобными определениями.

Элементы внутри строки матрицы могут отделяться друг от друга не только пробелами, но и запятыми. При вводе чисел в экспоненциальной форме (например, 2.34e-9) не следует использовать пробелы. Ввод больших матриц лучше выполнять с помощью m-файлов, в которых легко находить и исправлять ошибки (см.

п. 7). Такие встроенные функции как **rand, magic** и **hilb** позволяют легко сгенерировать матрицы, с которыми удобно поэкспериментировать. Команда **rand(n)** создает матрицу **nxn**, каждый элемент которой - случайное число с равномерным распределением в диапазоне [0 1], в то время как команда **rand(m,n)** создает такую же матрицу размера **mxn**. Команда **magic(n)** создает матрицу **nxn**, которая является магическим квадратом (суммы элементов по строкам и столбцам равны). Команда **hilb(n)** создает матрицу Гильберта размером **nxn**, которая является королевой плохо определенных матриц (**m** и **n** являются, конечно, целыми положительными числами). Матрицы могут быть сгенерированы также с помощью цикла **for**(см. п. 4).

Ссылки на отдельные элементы матриц и векторов осуществляются с помощью индексов в круглых скобках обычным образом. Например, **A(2,3)** означает элемент матрицы, стоящий на 2-й строке и 3-м столбце матрицы **A**, а **x(3)** означает 3-й элемент вектора **x**. Попробуйте сами поэкспериментировать с элементами матриц. В качестве индексов векторов и матриц могут использоваться только положительные числа. На элементы матрицы **A** можно ссылаться, используя единственный индекс, **A(k)**. Это обычный способ, когда речь идет о ссылке на элементы вектора. Но также можно ссылаться и на элементы двумерной матрицы, и в этом случае эта матрица рассматривается как один длинный вектор-столбец, сформированный из столбцов исходной матрицы. В приведенной матрице на элемент 5 можно сослаться либо **A(2,2)**, либо **A(5)**. И то и другое правильно.

2.2. Подматрицы и использование двоеточия (:)

Вектора и подматрицы часто используются в системе **MATLAB**, чтобы получить компактную запись алгоритмов сложной обработки данных. Использование нотации с двоеточием (которая используется и для генерации векторов и подматриц) и векторов вместо индексов является ключом к эффективной манипуляции этими объектами. Творческое использование этих возможностей позволяет минимизировать число явных циклов (использование явных циклов замедляет работу **MATLAB**) и делает написанную программу простой и легко читаемой. Правда, необходимы специальные усилия для овладения этими возможностями. Выражение **1:5** фактически является вектор-строкой **[1 2 3 4 5]**. Числа в этом векторе не обязательно целые и инкремент не обязательно равен единице. Например, оператор **x=0.2:0.2:1.2** дает вектор **x** равный **[0.2, 0.4, 0.6, 0.8, 1.0, 1.2]**, а оператор **5:-1:1** дает в результате вектор **[5 4 3 2 1]**. Приведенный далее оператор, например, дает таблицу синусов. Попробуйте выполнить все приведенные выше примеры.

```
x = [0.0:0.1:2.0]';
y = sin(x);
[x y]
```

Отметим, что поскольку **sin** является скалярной функцией (т.е. действующей поэлементно), если аргументом ее является вектор **x**, то результатом будет вектор **y**. Двоеточие может быть использовано для доступа к подматрицам. Например, **A(1:4,3)** является вектор-столбцом, состоящим из четырех первых элементов третьего столбца матрицы **A**. Двоеточие само по себе означает всю строку или весь столбец. Например, **A(:,3)** является третьим столбцом **A**, а **A(1:4,:)** представляет собой первые четыре строки матрицы. Произвольный целый вектор может использоваться в качестве индекса подматрицы. Например, **A(:,[2 4])** является матрицей из двух столбцов, 2-го и 4-го столбцов матрицы **A**. Такое индексирование, как и нотация с двоеточием, может использоваться с обеих сторон знака присваивания. Так, оператор **A(:,[2 4 5]) = B(:,1:3)** заменяет 2,4 и 5-й столбцы матрицы **A** на первые три столбца матрицы **B**. Заметим, что при выполнении такого оператора вся измененная матрица **A** будет выведена на экран. Попробуйте выполнить сами эту и подобные команды. Столбцы 2-й и 4-й матрицы **A** могут быть умножены справа на матрицу **[1 2; 3 4]** размером **2x2**- такой оператор имеет вид **A(:,[2,4]) = A(:,[2,4])*[1 2; 3 4]**. В этом случае, как и в предыдущем, вся измененная матрица будет выведена на экран. Попробуйте понять, а потом проверьте, что будет с вектором длины **n**, если будет выполнен оператор **x = x(n:-1:1)**. Для того чтобы лучше оценить описанные возможности **MATLAB**, попробуйте выполнить те же самые действия с помощью операторов Паскаля, Фортрана или С.

2.3. Функции построения матриц

Имеются следующие стандартные функции для построения матриц

eye	единичная матрица
zeros	нулевая матрица
ones	матрица единиц
diag	см. ниже
triu	верхняя треугольная часть матрицы
tril	нижняя треугольная часть матрицы
rand	матрица со случайными элементами

Например, команда **(m,n)** создает **mxn** матрицу нулей, а команда **zeros(n)** генерирует матрицу **nxn**; если **A** — матрица, то команда **zeros(size(A))** создаст

матрицу нулей такой же размерности, как и **A**. Если **x** – вектор, то **diag(x)** – это диагональная матрица, у которой на главной диагонали стоит вектор **x**; если **A** – квадратная матрица, то **diag(A)** – это вектор, элементы которого состоят из диагональных элементов **A**. Проверьте, чему равна **diag(diag(A))**? Матрицы можно строить из блоков. Например, если **A** – матрица **3x3**, тогда команда

```
B=[A, zeros(3,2); zeros(2,3), eye(2)]
```

создает новую матрицу **5x5**. Попробуйте выполнить эту команду сами.

3. Операции, выражения и переменные

3.1. Правила записи операторов

MATLAB является интерпретирующим языком непосредственных вычислений, т.е. выражения, которые вы вводите, интерпретируются и вычисляются. Операторы **MATLAB** обычно имеют форму

переменная = выражение или просто

выражение

Выражение, как правило, формируется из операторов, функций и имен переменных. После выполнения выражения генерируется матрица, которая выводится на экран и присваивается соответствующей переменной для последующего использования. Если имя переменной в левой части и знак = отсутствуют, автоматически генерируется переменная **ans** (**answer** - ответ), которой присваивается результат вычислений.

Обычно оператор завершается клавишей **Enter**. Однако в случае необходимости оператор может быть продолжен на следующей строке, для чего его необходимо завершить тремя или более точками, после которых следует **Enter**. С другой стороны, в одной строке может быть несколько операторов, разделенных запятой или точкой с запятой.

Если последним символом в строке является точка с запятой, то вывод значений результата не производится, но присвоение выполняется. Это помогает подавить вывод ненужных промежуточных результатов. **MATLAB** различает большие и маленькие буквы в именах команд, функций и переменных. Например, **solveUT не** то же самое, что **solveut**. Команда **who** выводит список всех переменных в рабочем пространстве. Переменная может быть удалена из рабочего пространства командой **clear <имя_переменной>**. Команда **clear** без аргументов очищает все непостоянные переменные в рабочем пространстве. Постоянная переменная **eps**

(epsilon) представляет машинную точность - порядка 10^{-6} на большинстве машин. Эта переменная является полезной при определении сходимости итеративных процессов. Вывод на дисплей или вычисления могут быть прерваны на большинстве компьютеров, не покидая **MATLAB**, с помощью комбинации клавиш **Ctrl+C** (**Ctrl+Break** на PC).

3.2. Матричные операции

Следующие матричные операции доступны в **MATLAB**:

+	сложение
-	вычитание
*	умножение
^	степень
'	транспонирование
\	левое деление
/	правое деление

Эти матричные операции применимы, конечно, и к скалярам (матрицам 1x1). Если размерность матриц не соответствует используемой операции, то система генерирует сообщение об ошибке за исключением случаев, когда одним из операндов является скаляр, потому что в этом случае операция выполняется между скаляром и каждым элементом матрицы второго операнда.

Операция «матричное деление» требует специальных комментариев. Если A является обращаемой квадратной матрицей, а b - вектор-столбец или вектор-строка соответственно, тогда $x = A\backslash b$ является решением уравнения $A * x = b$, а $x = b/A$ является решением уравнения $x * A = b$. Если A - квадратная матрица, то при левом делении для факторизации используется метод исключения Гаусса и эта факторизация позволяет решить уравнение $A * x = b$. Если матрица не квадратная, то для ее факторизации используется метод ортогонализации Хаусхольдера с ведущим столбцом, а приведенная матрица используется для решения недо- или переопределенной системы уравнений в смысле наименьших квадратов. Правое деление определяется в терминах левого деления по формуле $b/A = (A'\backslash b')'$.

3.3. Операции с массивами

Матричные операции сложения и вычитания действуют поэлементно, а остальные приведенные выше операции - нет, они являются матричными операциями. Следует отметить, что приведенные выше операции *, ^, \, / могут стать поэлементными,

если перед ними поставить точку. Например, операция **[1,2,3,4].*[1,2,3,4]** или **[1,2,3,4].^2** дадут один и тот же результат **[1,4,9,16]**. Попробуйте эти операции или им подобные выполнить самостоятельно. Эти действия в особенности полезны при использовании графики.

3.4. Сохранение данных из рабочей области

При выходе из системы **MATLAB** все переменные рабочей области теряются. Однако при вызове команды **save** перед выходом все переменные рабочей области записываются на диск в нетекстовом формате в файл с именем **matlab.mat**. Если впоследствии загрузить **MATLAB**, то команда **load** восстановит все переменные рабочего пространства.

4. Операторы for, while, if, case и операторы отношения

При использовании в своей основной форме перечисленные выше операторы управления **MATLAB** работают так же, как и в большинстве языков программирования.

4.1. Цикл for

Например, для данного **n**, оператор

```
x = [ ]; for i = 1:n,x=[x,i^2], end
```

или

```
x = [ ]; for i = 1:n x = [x,i^2] end
```

создает определенный вектор размерности **n**, а оператор

```
x = []; for i = n:-1:1, x=[x,i^2], end
```

создает вектор с теми же элементами, но размещенными в обратном порядке. Попробуйте выполнить это сами. Заметим, что матрица может быть пустой (например, в случае оператора **x = [ ]**.) Последовательность опреаторов

```
 for i = 1:m
   for j = 1:n
     H(i, j) = 1/(i+j-1);
   end
end
 H
```

создаст и напечатает на экране матрицу Гильберта размерности **mxn**. Точка с запятой, которая завершает внутренний оператор, предотвращает вывод на экран

ненужных промежуточных результатов, в то время как последний оператор **H** выводит на экран окончательный результат.

4.2. Цикл while

В общем виде цикл **while** записывается в виде

```
while <условие>
<операторы>
end
```

<Операторы> будут повторяться до тех пор, пока **<условие>** будет оставаться истинным. Например, для заданного числа **a** приведенная далее последовательность операторов вычислит и выведет на дисплей наименьшее неотрицательное число **n**, такое что $2^n < a$:

```
n = 0;
while  2^n < a
  n = n + 1;
end
n
```

4.3. Условный оператор if

В общем виде простой оператор **if** используется следующим образом:

```
if <условие>
 <операторы>
end
```

<Операторы> будут выполняться только если **<условие>** истинно. Возможно также множественное ветвление, что демонстрируется приведенным далее примером.

```
if n < 0
 parity = 0;
elseif rem(n,2) == 0
 parity = 2;
else
 parity = 1;
end
```

При использовании двухвариантного условного оператора часть, связанная с **elseif**, конечно, не используется.

4.4. Оператор переключения case

При необходимости построить конструкцию ветвления с более чем двумя логическими условиями удобнее использовать не вложенные операторы **if**, а оператор переключения **switch ... case**. Этот оператор имеет следующую структуру:

```
switch <выражение>
   % <выражение> - это обязательно скаляр или строка
case <значение1>
 операторы
   % выполняется, если <выражение>=<значение1>
case <значение2>
 операторы
   % выполняется, если <выражение>=<значение2>
 ...
otherwise
 операторы
   % выполняется, если <выражение> не совпало
   % ни с одним значением
end
```

4.5. Условия (операторы отношения)

В **MATLAB** используются следующие *операторы отношения*:

<	меньше чем
>	больше чем
<=	меньше или равно
>=	больше или равно
==	равно
~=	не равно

Отметим, что знак **=** используется в операторах присваивания, в то время как знак **==** используется в операторах отношения. Операторы отношения (или, другими словами, логические переменные, которые они создают) могут объединяться с помощью следующих логических операторов:

&	И
|	ИЛИ
~	НЕ

Когда эти операторы применяются к скалярам, то результатом является тоже скаляр **1** или **0** в зависимости от того, является ли результат **истиной** или **ложью**. Попробуйте вычислить **3 < 5, 3 > 5, 3 == 5**, и **3 == 3**. Когда операторы отношения применяются к матрицам одного размера, результатом является матрица того же размера, у которой в качестве элементов стоят **0** или **1**, в зависимости от соотношения между соответствующими элементами исходных матриц. Попробуйте вычислить **a=rand(5), b=triu(a), a == b**. Операторы **while** и **if** интерпретируют отношение между матрицами как истинное в том случае, если результирующая матрица не имеет нулевых элементов.

Так, если вы хотите выполнить оператор в том случае, когда матрицы **A** и **B** полностью совпадают, вы можете написать

if A == B <операторы> end

но если вы хотите выполнить оператор в том случае, когда матрицы **A** и **B** не равны, вы должны ввести **if any(any(A ~= B)) <оператор> end** или, что проще, **if A == B else <оператор> end**. Заметим, что конструкция **if A ~= B, <оператор>, end** почти наверняка не даст того, что нужно, поскольку оператор будет выполняться только если каждый элемент матрицы **A** будет отличаться от соответствующего элемента матрицы **B**. Для сведения матричных отношений к вектору или скаляру можно воспользоваться функциями **any** и **all**. В предыдущем примере необходимо использование функции **any** два раза, поскольку эта функция - векторная (см. 5.2). Оператор **for** допускает использовать любую матрицу вместо **1:n**. Для более полного знакомства с возможностями, расширяющими мощь оператора **for**, см.[6].

4.6. Функция find

Хотя функция **find** не относится формально к условным операторам или операторам отношения, тем не менее мы решили поместить ее в этот раздел, поскольку ее использование весьма полезно для работы с циклами и условными операторами.

Оператор **k=find(x)** возвращает вектор **k** номеров ненулевых элементов вектора/матрицы **x**. Если **x**- матрица, то при определении индексов она рассматривается как вектор, образованный последовательно соединенными столбцами матрицы. Вектор **find(x)** можно использовать совместно с операторами отношения, поскольку результатом применения оператора отношения к матрицам является мат-

рица из **0** и **1** (**ложь** или **истина**). Таким образом можно с помощью одного оператора **find** определить и записать сразу все индексы матрицы, удовлетворяющие некоторому условию. Если при этом вспомнить, что оператор цикла **for** допускает форму **for k=KK**, где **KK** - целый вектор, то удобно использовать их вместе. Например если вам необходимо выполнить **<оператор>** только для тех элементов матрицы, которые больше **3**, то удобно это сделать следующим образом:

```
for i=find(A>3)
    <оператор>
end;
```

5. Функции MATLAB

В системе **MATLAB** существует большое количество функций, подготовленных разработчиками системы (см. п. E). Большинство из них предоставлено в виде исходных текстов. Можно эти функции классифицировать по областям их использования (тригонометрические, спецфункции, функции линейной алгебры и т.д.), но здесь мы вкратце опишем действия функций по отношению к матричному характеру переменных **MATLAB**.

5.1. Скалярные функции

Определенные функции **MATLAB** действуют только на скаляры, но когда аргументом их является матрица, то они действуют поэлементно. К таким функциям относятся

sin	asin	exp	abs	round
cos	acos	log (натуральный)	sqrt	floor
tan	atan	rem (остаток)	sign	ceil

5.2. Векторные функции

Существуют другие функции, аргументами которых являются вектора (строки или столбцы), но если эти функции действуют на матрицу размера **mxn (m >= 2)**, то они действуют постолбцово, т.е. результатом действия является вектор-строка, каждый элемент которой является результатом действия этой функции на соответствующий столбец. Построчное действие такой функции (если необходимо) может быть достигнуто использованием операции транспонирования. Например, **mean(A')**. Некоторые из этих функций приведены далее:

max	sum	median	any
min	prod	mean	all
sort	std		

Например, максимальный элемент прямоугольной матрицы находится с помощью команды **max(max(A))**, а не с помощью **max(A)**. Попробуйте сами выполнить эти операции.

5.3. Матричные функции

Наибольшую мощь системе **MATLAB** дают матричные функции. Наиболее употребительные приведены в следующей таблице:

eig	собственные значения и собственные вектора
chol	факторизация Холецкого
svd	сингулярная декомпозиция
inv	обратная матрица
lu	LU-факторизация
qr	QR-факторизация
hess	форма Хессенберга
schur	декомпозиция Шура
rref	приведение к треугольной форме методом Гаусса
expm	матричная экспонента
sqrtm	матричный корень квадратный
poly	характеристический полином
det	определитель
size	размерность
norm	норма вектора или матрицы
cond	число обусловленности
rank	ранг матрицы

Функции **MATLAB** могут иметь один или несколько результатов. Например, функция **y = eig(A)**, или просто **eig(A)** генерирует вектор-столбец, содержащий собственные значения матрицы **A**, в то время как оператор **[U,D] = eig(A)** генерирует матрицу **U**, чьи столбцы являются собственными векторами **A**, а диагональная матрица **D** содержит на главной диагонали собственные значения этой матрицы. Попробуйте вычислить это сами.

6. M-файлы

MATLAB может выполнять последовательность операторов, записанных в файл на диске. Такие файлы называются *m-файлами*, потому что имена этих файлов имеют вид **<имя>.m**. Большая часть вашей работы в **MATLAB** будет состоять в создании, редактировании и выполнении таких m-файлов. Имеется два типа m-файлов: *файлы-программы*, или сценарии, и *файлы-функции*.

6.1. Файлы-программы, или сценарии

Файлы-программы состоят из последовательности обычных операторов **MATLAB**. Если файл с таким сценарием имеет имя, например, **rotate.m**, то команда **rotate**, введенная в командной строке, вызовет выполнение соответствующей последовательности операторов. Переменные в программе являются глобальными и изменят значения таких же переменных (если таковые есть) в рабочей области текущей сессии. Программы или сценарии часто используются для ввода данных в большие матрицы; в таких файлах легко исправить ошибки ввода. Если, например, файл на диске с именем **data.m** содержит строки

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

тогда команда **data** приведет к тому, что написанное выше присвоение будет выполнено. Внутри m-файлов можно ссылаться на другие m-файлы, в том числе и на самого себя рекурсивно.

6.2. Файлы-функции

Файлы-функции фактически дают возможность расширять **MATLAB**, поскольку определенные вами новые функции, специфические для решения ваших задач, имеют тот же статус, что и другие функции **MATLAB**. Переменные в функциях являются по умолчанию локальными, но в версиях 4.0 и выше разрешено объявлять требуемые переменные глобальными (**global**). Для начала рассмотрим простой пример файла-функции.

```
function  r = randint(m,n)
% RANDINT случайная матрица с целыми элементами.
% randint(m,n) возвращает матрицу mxn с целыми
```

```
% элементами между 0 и 9.
a = floor(10*rand(m,n));
```

Более общая версия такой функции может иметь такой вид:

```
function  r = randint(m,n,a,b)
% RANDINT случайная матрица с целыми элементами.
% randint(m,n) возвращает матрицу mxn с целыми
% элементами между  0 и 9.
% rand(m,n,a,b) возвращает матрицу с целыми
% элементами в диапазоне между целыми a и b.
if nargin < 3, a = 0; b = 9; end
r = floor((b-a+1)*rand(m,n)) + a;
```

Этот текст должен быть записан на диск в виде файла с именем **randint.m** (в соответствие с именем функции - это обязательное условие для функции).

Первая строка функции - объявление имени функции, входные аргументы, выходные аргументы. Без такой строки весь следующий файл является программой, или сценарием, а не функцией. Так, например, оператор **z = randint(4,5)** приведет к передаче чисел 4 и 5 переменным **m** и **n**, а выходной результат будет передан переменной **z**. Поскольку переменные в файле-функции локальные, их имена никак не влияют на имена и значения переменных в текущей рабочей области **MATLAB**.

Отметим роль функции **nargin** (число входных аргументов). Использование этой функции в данном примере позволяет установить значение отсутствующих входных аргументов по умолчанию — таких переменных, как **a** и **b** в примере. В общем случае наличие такой функции позволяет использовать функции с переменным числом входных аргументов и в зависимости от их числа направлять вычисления по разным логическим веткам функции.

Функция может иметь множественные выходные аргументы. Например

```
function  [mean, stdev] = stat(x)
%  STAT  Среднее и стандартное отклонение
%  Для вектора x, stat(x) возвращает среднее
%  значение и стандартное отклонение  x.
%  Для матрицы x, stat(x) возвращает два
%  вектора-строки, содержащие, соответственно,
%  среднее и стандартное отклонение каждого
%  столбца матрицы x.
[m  n] = size(x);
```

```
if m == 1
  m = n; % обработка в случае ввода вектор-строки
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m - mean.^2);
```

Если этот файл записать на диск под именем **stat.m**, то команда **[xm,xd]=stat(x)** присвоит среднее значение элементов **x** переменной **xm**, а стандартное отклонение переменной **xd**. Несмотря на наличие нескольких выходных аргументов, можно присвоить значение функции одной переменной. Например, **xm = stat(x)** (никаких скобок вокруг **xm** не требуется) присвоит **xm** среднее значение **x**. Символ **%** указывает на то, что вся строка символов после него является комментарием и игнорируется при исполнении. Тем не менее, несколько первых строк-комментариев, которые являются кратким описанием данной функции, доступны при вводе оператора, т.е. являются той помощью, которая вызывается с помощью команды **help stat**. Такую документацию всегда следует включать в файл-функцию. Приведенная выше функция демонстрирует некоторые возможности **MATLAB**, которые позволяют написать эффективный код. Отметим, например, оператор **x.^2**, который является возведением в степень каждого элемента вектора **x**, оператор с функцией **sum**, которая является векторной функцией (п. 5.2), функцию **sqrt**, которая является скалярной функцией (см. 5.1), и деление в выражении **sum(x)/m**, которое является матрично-скалярной операцией.

Приведенная далее функция, которая вычисляет наибольший общий делитель двух целых чисел с помощью алгоритма Евклида, иллюстрирует использование функции вывода сообщений об ошибках (см. п. 6.3).

```
function  a = gcd(a,b)
% GCD  Наибольший общий делитель.
%   gcd(a,b) является наибольшим общим делителем
%   целых чисел  a и b, оба не равны нулю.
 a = round(abs(a));  b = round(abs(b));
if  a == 0 & b == 0
  error('gcd не определено, если оба числа равны нулю')
else
  while b ~= 0
    r = rem(a,b);
    a = b;  b = r;
  end
end
```

Некоторые дополнительные продвинутые возможности иллюстрируются в следующей функции. Как было отмечено ранее, некоторые входные аргументы функции, например такие, как **tol**, могут быть сделаны необязательными с помощью функции **nargin** (число входных аргументов). Аналогично может быть использована функция **nargout** (число выходных аргументов). Отметим, что используется тот факт, что результатом оператора отношения является число (1 - если истинно и 0 - если ложь). Так, если в операторах **while** или **if** используются отношения, то **nonzero** означает "истина", а 0 означает "ложь". Использованная в конце функция **feval** позволяет в качестве входной переменной использовать строку, содержащую имя другой функции.

```
function [b, steps] = bisect(fun, x, tol)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BISECT Нуль функции одной переменной методом
% деления пополам.
%  bisect(fun,x) возвращает нуль функции.
%  fun - строка, содержащая имя вещественной
%  функции одной вещественной переменной;
%  обычно эта функция определяется в m-файле.
%  x  начальное приближение. Возвращается значение
%  вблизи точки, в которой fun меняет знак.
%  Например, bisect('sin',3) равна pi.
%  Обратите внимание на кавычки вокруг sin.
%  Необязательный третий аргумент определяет
%  относительную точность результата. По умолчанию
%  эта точность eps.
%  Второй выходной аргумент (необязательный)
%  содержит протокол процесса;
%  строки матрицы имеют вид  [c f(c)].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Инициализация
 if nargin < 3, tol = eps; end
trace = (nargout == 2);
if x ~= 0, dx = x/20; else, dx = 1/20; end
a = x - dx;  fa = feval(fun,a);
b = x + dx;  fb = feval(fun,b);
% Поиск изменения знака.
 while (fa > 0) == (fb > 0)
```

```
dx = 2.0*dx;
a = x - dx;  fa = feval(fun,a);
if (fa > 0) ~= (fb > 0), break, end
b = x + dx;  fb = feval(fun,b);
end
if trace, steps = [a fa; b fb]; end
%
% Основной цикл
 while  abs(b - a) > 2.0*tol*max(abs(b),1.0)
c = a + 0.5*(b - a);  fc = feval(fun,c);
if trace, steps = [steps; [c fc]]; end
if (fb > 0) == (fc > 0)
  b = c;  fb = fc;
else
  a = c;  fa = fc;
end end
```

Некоторые функции **MATLAB** являются встроенными, в то время как другие поставляются в виде m-файлов. Текст реально имеющихся m-файлов (системы **MATLAB** или ваших собственных) можно просмотреть с помощью команды **type <имя_функции>**. Попробуйте команды **type eig, type vander**, и **type rank**.

6.3. Текстовые строки, сообщения об ошибках, ввод

Текстовые строки вводятся в **MATLAB** в виде текста в одинарных кавычках. Например, оператор **s = 'This is a test'** присваивает данный текст переменной **s**. Вывод текстовой строки осуществляется с помощью оператора **disp**. Например, оператор **disp('this message is hereby displayed')** выводит соответствующее сообщение на экран. Сообщения об ошибках лучше выводить с помощью функции **error**. Например, если в процессе выполнения m-файла будет выполнен оператор **error('Sorry, the matrix must be symmetric')**, то после вывода сообщения на экран выполнение m-файла будет прекращено. В m-файле может быть запрос на интерактивный ввод данных, организованный с помощью оператора **input**. Когда вводится оператор **iter = input('Введите число иттераций: ')**, на экран выводится запрос на ввод, и выполнение программы приостанавливается до того момента, пока пользователь не введет с клавиатуры требуемые входные данные. После нажатия клавиши **Enter** данные присваиваются переменной **iter**, и выполнение программы продолжается.

7. Работа с m-файлами

Во время работы в **MATLAB** часто необходимо создавать или редактировать m-файлы, а после этого возвращаться в командное окно **MATLAB** для отладки или вычислений. В версии 5.0 или старше (в системе Windows-95) имеется специальный редактор/отладчик (см. п. 7.2), в котором можно исправлять текст и выполнять пошаговую отладку программы. После исправления необходимо сохранить сделанные изменения. Как использовать средства отладки в версии 5.0 и старше будет рассказано далее (п. 7.3).

M-файлы, с которыми вы работаете, должны быть доступны. Для этого либо текущая директория должна быть директорией с вашими файлами, либо необходимо проложить туда путь (в смысле **DOS**). Это можно сделать либо с помощью команд **DOS** непосредственно из командного окна (команды **cd**), либо с помощью пункта меню **File/Set Path**, который позволяет сделать необходимые директории доступными.

7.1. Список путей доступа

Для поиска m-файлов система **MATLAB** использует механизм путей доступа поскольку m-файлы записываются в каталоги или папки файловой системы. Например, при поиске файла с именем **foo MATLAB** выполняет следующие действия:

1) просматривает, не является ли **foo** именем переменной;

2) просматривает, не является ли **foo** встроенной функцией;

3) ищет в текущем каталоге m-файл с именем **foo.m**;

4) ищет m-файл с именем **foo.m** во всех каталогах *списка путей доступа*.

Реально применяемые правила поиска являются более сложными из-за ограничений, которые связаны с использованием *подфункций*[3], личных (private) функций и объектно-ориентированных механизмов. Однако приведенный выше упрощенный порядок поиска точно отражает механизм поиска m-файлов, с которыми обычно работает пользователь.

7.1.1. Работа со списком путей доступа

В процессе сеанса работы можно вывести на терминал или внести изменения в список путей доступа, используя следующие функции:

[3] **Внутри функций допускается определение других функций, только следует иметь в виду, что доступ к таким функциям возможен только из функций, внутри которых они определены.**

- **path** - выводит на экран список путей доступа;
- **path (s)** - заменяет существующий список списком **s**;
- **addpath /home/lib** и **path(path, '/home/lib')** - добавляют новый каталог текущего подкаталога в список путей доступа;
- **rmpath /home/lib** - удаляет путь **/home/lib** из списка.

Список путей доступа, используемый по умолчанию, определен в файле **pathdef.m**, который размещен в каталоге **local**; этот файл выполняется при каждом запуске системы **MATLAB**.

Кроме работы из командной строки существует средство просмотра путей доступа **Path Browser** (см. далее), которое поддерживает удобный графический интерфейс для просмотра и изменения списка путей.

7.1.2. Текущий каталог

Система **MATLAB** использует понятие *текущего каталога* при работе с m- и mat-файлами во время сеанса работы.

Начальный текущий каталог определен в файле запуска, который ассоциирован с ярлыком запуска системы **MATLAB**, расположенном на рабочем столе. Для вывода текущего каталога на экран терминала предназначена команда **cd**. Для изменения текущего каталога следует использовать команду **cd<новый путь доступа>**.

7.1.3. Средство просмотра и редактирования путей доступа Path Browser

Как было указано выше, при работе в системе Windows-95 имеется специальное средство для просмотра и изменения путей доступа **Path Browser** (рис. 1). Показанное далее окно открывается либо из меню **File/Set Path** командного окна, либо с помощью кнопки на инструментальной панели.

После дополнения списка путей доступа необходимо сохранить новый путь с помощью пункта меню **File/Save Path**, в противном случае установленный путь будет известен системе только на время одного сеанса работы[4].

[4] **Работающие в терминальном классе должны иметь в виду, что сохранение пути доступа в настоящей конфигурации системы невозможно, поскольку запись в сетевую директорию local запрещена.**

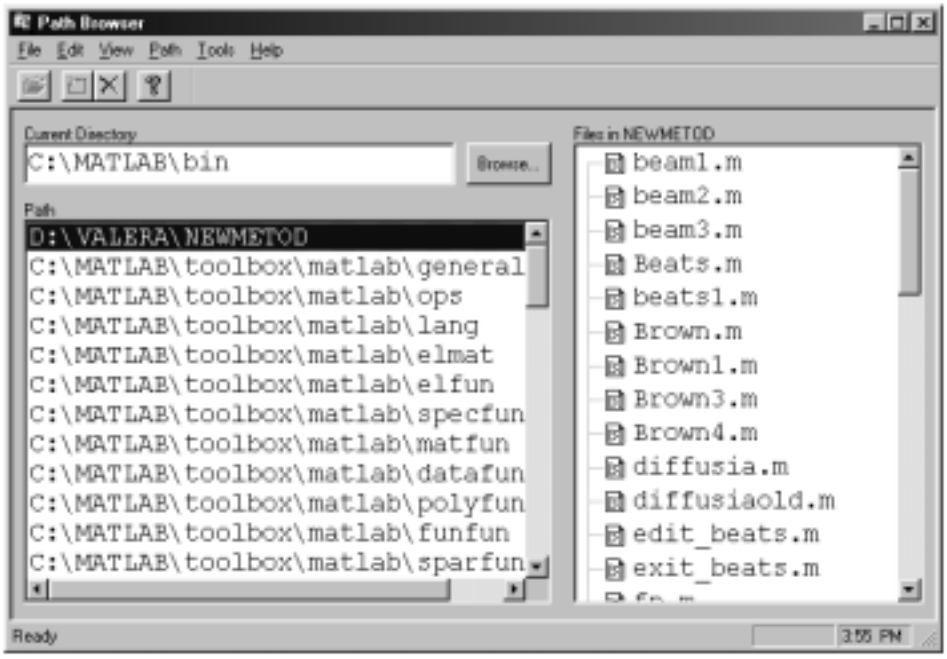


Рис. 1. Окно **Path Browser**

7.2. Использование редактора/отладчика

Редактор/отладчик предоставляет как средства редактирования текста m-файла, так и средства пошаговой его отладки. Один из способов вызова редактора - вызов из командной строки **MATLAB** с помощью команды **edit**. Например, команда **edit poof** откроет встроенный редактор для редактирования файла **poof.m**, если в меню **File** в диалоговом окне **Preferences** не установлен вами другой редактор. Можно открыть редактор и другим способом — с помощью меню **File/New** или кнопки **New File** на панели инструментов (см. далее). Для открытия существующего m-файла выберите пункт **File/Open** или щелкните на кнопке **Open File**. После вызова редактор/отладчик будет иметь вид, показанный на рис. 2

Редактор, используемый в системе, имеет синтаксическую раскраску, т.е. слово или символ по мере ввода приобретают тот цвет, который соответствует их типу. Редактор различает такие типы вводимых слов:

- Комментарии
- Ключевые слова
- Незаконченные строки

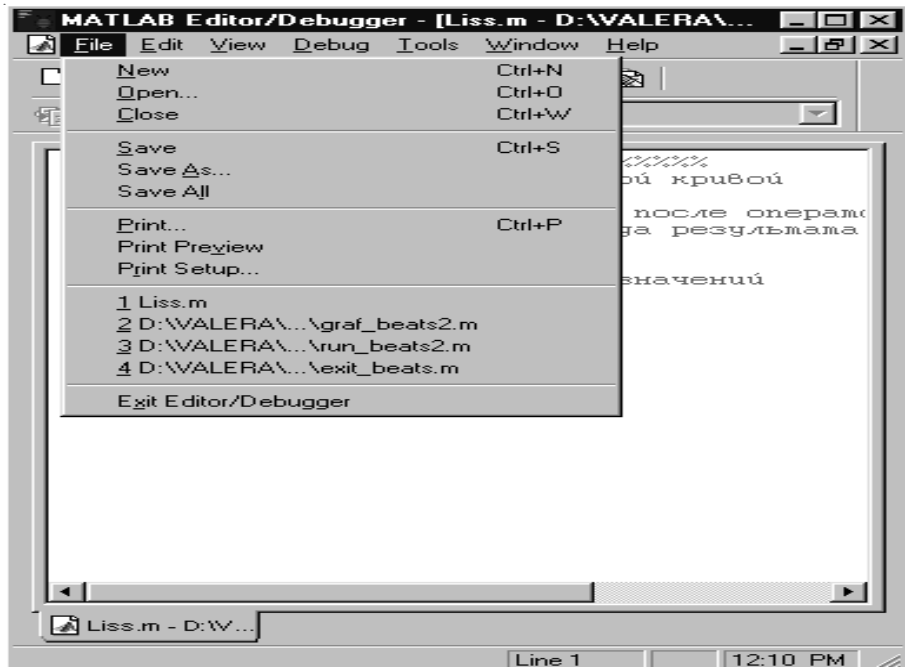


Рис. 2. Общий вид редактора/отладчика

- Законченные строки
- Другой текст

С помощью пункта меню **Tools/Fonts** можно настроить такие важные параметры, как используемый шрифт. Это особенно важно для работы с русским текстом, поскольку не все шрифты правильно воспроизводят русский текст. В остальном данный редактор не отличается от обычного многооконного текстового редактора - в нем работают все редактирующие клавиши (**Del, Bspace, Home** и т.д.). При редактировании файлов вы можете непосредственно перейти к требуемой строке при помощи пункта меню **Edit/GoTo Line** и указать номер требуемой строки в появившемся окне. После редактирования файла и повторного его запуска из командного окна желательно *предварительно* сохранить новый вариант файла. Но можно запускать редактируемый файл на счет не выходя из редактора (т.е. не переходя в командное окно) с помощью пункта меню **Tools/Run**. При этом предварительное сохранение текста исправлений не требуется.

Одной из важных особенностей данного редактора является то, что после проведения вычислений можно в редакторе просмотреть значения переменных, которые они имеют в текущий момент в рабочей области. Для этого достаточно установить курсор мыши на этой переменной, и появится прямоугольник с желтым фоном, на котором выводится текущее значение переменной. Если переменная представляет из себя большую матрицу, то таким образом увидеть целиком ее не удастся. Для

просмотра (и возможного исправления при отладке) всех значений матрицы есть специальный пункт меню **View/Workspace Browser**. Имеется еще два интересных пункта в меню **View**. Это пункт **Evaluate Selection**, который позволяет вычислять значение выделенного выражения и помещать результат в командное окно, и пункт **Auto Indent Selection**, который выполняет автоматическое форматирование текста программы с отступами в соответствии с правилами **MATLAB**. Выбор пункта меню **View/Options** позволяет получить доступ к диалоговому окну, в котором можно настроить параметры редактора.

7.3. Отладка m-файлов

Отладка программного кода - это процесс, в ходе которого могут быть выявлены ошибки двух видов:

- *синтаксические*, которые связаны с неточностью записи имен m-функций или арифметических выражений. **MATLAB** обнаруживает большинство синтаксических ошибок, сопровождая их сообщением об ошибке с указанием номера строки соответствующего m-файла;

- *ошибки времени выполнения*, которые, как правило, имеют алгоритмическую природу и проявляются в том, что приводят к непредвиденным результатам.

Достаточно легко можно исправить синтаксические ошибки, которые сопровождаются сообщениями о причинах их возникновения. Ошибки времени выполнения выявить более сложно, потому что локальная рабочая область m-функции оказывается потерянной, если ошибка приводит к возврату в рабочую область системы **MATLAB**. Чтобы определить причину такой ошибки, можно использовать один из следующих приемов:

- реализовать вывод результатов промежуточных вычислений на дисплей, удалив в соответствующих операторах точки с запятой, которые подавляют вывод на экран промежуточных результатов;

- добавить в m-файл команды **keyboard**, которые останавливают выполнение m-файла и разрешают проверить и изменить переменные рабочей области вызываемой m-функции. В этом режиме появляется специальное приглашение K». Возврат к выполнению функции реализуется командой **return**;

- закомментировать заголовок функции и выполнить m-файл как сценарий. Это позволяет проследить результаты промежуточных вычислений в рабочей области системы;

- использовать отладчик системы **MATLAB**.

Отладчик полезен для исправления ошибок во время выполнения пограммы именно потому, что он дает возможность отслеживать рабочие области функции и проверять или изменять значения соответствующих переменных. Отладчик позволяет устанавливать и удалять контрольные точки, то есть специальным образом помеченные строки m-файла, в которых выполнение останавливается. Это дает возможность изменять содержимое рабочей области, просматривать стек вызова m-функций и выполнять m-файл построчно. Отладчик может функционировать как в режиме командной строки, так и в режиме графического интерфейса пользователя. Далее мы рассмотрим отладку только в режиме графического интерфейса пользователя, поскольку он наиболее прост и нагляден.

Рассмотрим возможности отладки, котрые нам предоставляет **Editor/Debuger**. Для его запуска используется команда **edit <имя_файла>** или пункт меню **File/Open.** Можно открыть окно редактора/отладчика и с помощью пункта меню **File/New/M-file.** При таком варианте имя отлаживаемого файла открывается уже из меню самого редактора/отладчика. Общий вид окна редактора показан на рис. 2, а кнопки панели инструментов, которые удобнее всего использовать при отладке, показаны на рис. 3.

Панель инструментов редактора/отладчика выглядит следующим образом:

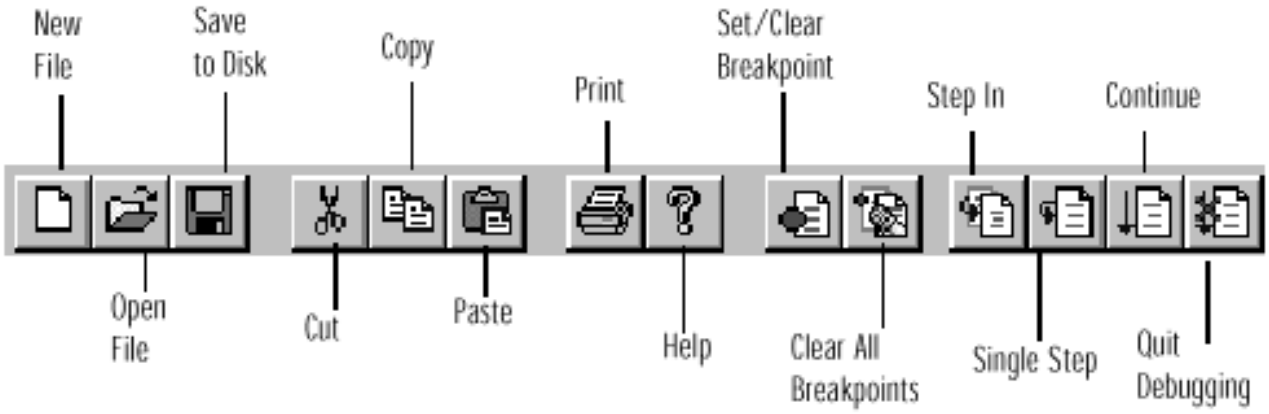


Рис. 3. Кнопки основной панели редактора/отладчика

Способ использования этих кнопок понятен из их названия. Часть кнопок представляют собой обычные редактирующие кнопки, используемые при сохранении, копировании, печати и поиске файлов. Другая часть связана непосредственно с отладкой. Это кнопки установки и очистки точек остановки, кнопки пошагового перемещения по программе с заходом в подпрограммы (**Step in**) и без захода в подпрограммы (**Single Step**), кнопка продолжения вычислений (**Continue**) и кнопка

остановки отладки (**Quit Debugging**). Как было отмечено выше, с помощью пункта меню **View/Workspace Browser** можно не только просмотреть, но и изменить значение любой переменной. При этом в одном из окон редактора открывается таблица, подобная электронной таблице, и в ней можно не только просматривать, но и исправлять значения перемнных.

7.4. Сравнение алгоритмов: flops и etime

Для определения эффективности алгоритмов принято использовать два параметра - число операций с плавающей запятой (**flops**) и затраченное время. Функция **MATLAB flops** сохраняет полное число выполненных операций с плавающей запятой. Команда **flops(0)** (не **flops = 0**!) обнуляет счетчик выполненных операций. Таким образом, если вы в начале исполнения вашего алгоритма введете команду **flops(0)**, то команда **flops** сразу после его завершения даст число операций. Функция **clock** возвращает текущее время с точностью до сотых долей секунды (см. **help clock**). При наличии двух таких времен **t1** и **t2** функция **etime(t2,t1)** выдает время, прошедшее от **t1** до **t2**. Можно, например, измерить время, требуемое для решения данной линейной системы $Ax = b$ методом исключения Гаусса следующим образом: **t = clock; x = A\ b; time = etime(clock,t)**. Вы можете сравнить это время и число **flops** с временем решения этой же системы с помощью оператора **x = inv(A)*b;**. Попробуйте сами сделать это. В версии 4.0 и выше имеются более удобные функции **tic** и **toc**.

8. Графика

MATLAB может создавать как плоские графики, так и трехмерные сетчатые поверхности, а также движущиеся графики, или анимацию. Для знакомства с некоторыми графическими возможностями имеющейся у вас версии запустите программу **demo**.

Основными и наиболее часто употребительными являются графики, которые представляют собой линии, описывающие те или иные численные данные. **MATLAB** предоставляет набор команд высокого уровня, которые используются для построения таких линий. Это такие команды, как **plot, title, axis, text, hist, contour** и ряд других, описанных в следующих двух разделах.

8.1. Плоские графики

8.1.1. Команда plot

Команда **plot** создает **x-y** график в линейных осях; если **x** и **y** являются векторами одинаковой длины, то команда **plot(x,y)** открывает графическое окно и рисует зависимость **y(x)**. Вы можете нарисовать график синуса на интервале от -4 до 4 с помощью команды

```
x = -4:.01:4;  y = sin(x); plot(x,y)
```

Попробуйте это сделать сами. Вектор **x** является набором равноотстоящих точек с шагом 0.01, а **y** - вектор со значениями функции синуса в этих точках (вспомните, что синус - поэлементная функция). Рисунок при создании открывает отдельное окно. Переход между окнами (возврат в окно **MATLAB** или переход от рисунка к рисунку осуществляется в соответствии с правилами среды, например, с помощью комбинации **Alt+Tab** или с помощью мыши). В качестве второго примера вы можете нарисовать график e^{-x^2} на интервале от -1.5 до 1.5 следующим образом: **x = - 1.5:.01:1.5; y = exp(-x.^2); plot(x,y)**. Обратите внимание на то, что точка перед ^ обязательна, поскольку мы хотим, чтобы возведение в степень выполнялось поэлементно (см. п. *3.3*). Можно, например, рисовать кривые, заданные параметрически. Попробуйте, например, выполнить такой оператор **t=0:.001:2*pi; x=cos(3*t); y=sin(2*t); plot(x,y)**.

Аргументами функции **plot** могут быть различные комбинации векторов и матриц. Возможны следующие варианты:

- **plot(y)**
  - Если **y** - вектор, то будет нарисована кривая **y** как функция номера элемента в **y**.
  - Если **y** - матрица, то будет сгенерирован набор кривых, каждая из которых представляет собой зависимость столбца матрицы от номера строки.
- **plot(x,y)**
  - Если **x** и **y** - вектора одинаковой длины и размерности (оба строки или оба столбцы), то будет нарисована кривая **y** от **x**.
  - Если **x** - вектор, а **y** - матрица, строки или столбцы **y** будут нарисованы в зависимости от **x**. Если столбец матрицы **y** имеет ту же длину, что и вектор **x**, то будет построен набор кривых, представляющий зависимость каждого из столбцов от **x**. Если строка матрицы **y** имеет ту же длину,

что и вектор **x**, то будет построен набор кривых, представляющий зависимость каждой из строк от **x**. Если число строк и столбцов **y** одинаково, то строятся столбцы от **x**.

- Если **x** - матрица, а **y** - вектор, то будет построено несколько кривых представляющих зависимость **y** от строк или столбцов матрицы **x** по правилу, описанному в предыдущем пункте.
- Если **x** и **y** - матрицы одинаковой размерности, то будет построен набор кривых, представляющих столбцы **y** от столбцов **x**.

Если у вас цветной монитор, то вы уже обратили внимание, что при описанном способе вызова функции **plot** различные кривые на одном и том же графике отрисовываются разным цветом. Перебор цветов выполняется автоматически, а при использовании соответствующих аргументов у команды **plot** эти цвета можно выбирать (см. далее).

Число аргументов у команды **plot** не ограничивается двумя. Можно использовать эту команду в формате **plot(x1,y1,x2,y2,...)**, причем правила, описанные выше, относятся к каждой паре аргументов.

Иногда для отрисовки кривой используется функция **line** с теми же аргументами, что и функция **plot**. Различие станет ясно после знакомства с дескрипторной графикой низкого уровня (см. п. 8.4).

Представляет интерес познакомиться с функцией **comet(x,y)**, которая строит движущуюся двумерную кривую с головой другого цвета. И хотя такое отображение не является анимацией в полном смысле слова, поскольку выполняется отрисовка уже насчитанных точек, тем не менее при построении сложных параметрических кривых полезно использовать эту функцию (например, в задаче исследования кривых Лиссажу).

8.1.2. Разметка графика и надписи

Команда **grid** поместит сетку на график. График может быть снабжен заголовком, именами осей, и на сам график может быть помещен дополнительный текст с помощью команд вывода текста. Аргументами всех этих команд является текстовая строка. Например, команда **title('График наилучшего приближения')** добавит к вашему графику заголовок. Команда **gtext('Пятно')** позволяет с помощью мыши или клавишного курсора разместить на рисунке индикаторный крест, в месте размещения которого и будет помещен текст после нажатия произвольной клавиши. При необходимости сделать подписи осей используются команды **Xlabel('ПодписьX')**, **Ylabel('ПодписьY')**. При необходимости вывести в каче-

стве надписи какую-либо комбинацию символов, которые отсутствуют в стандартном текстовом процессоре (например, верхние и нижние индексы, формулы и т.д.), вы можете использовать стандартную нотацию **LaTeX**а (подробнее о **LaTeX** вы можете узнать, например в [10]). Если выведена неправильная надпись, то можно ее стереть или заменить на другую, повторив такую же команду вывода текста, но с пустой или новой строкой в качестве аргумента.

Для изображения нескольких кривых на одном рисунке существуют два способа, которые иллюстрируются следующими примерами.

1. ```
   x=0:.01:2*pi;
   y1=sin(x);
   y2=sin(2*x);
   y3=sin(4*x);
   plot(x,y1,x,y2,x,y3)
   ```

   или, что полностью эквивалентно, формируется матрица **Y**, содержащая значения изображаемых функций в виде столбцов, и после этого рисунок создается последовательностью команд

   ```
   x=0:.01:2*pi;
   Y=[sin(x)', sin(2*x)', sin(4*x)'];
   plot(x,Y)
   ```

2. Другим способом такого рисования является использование команды **hold on**, которая замораживает текущий график, так что последующие кривые помещаются на этот же график. При этом масштаб и разметка осей изменяются, если новая кривая не вписывается в нарисованные оси. Команда **hold off** приводит к тому, что любой последующий вызов команды **plot** создает новый рисунок на этом же листе, т.е. предыдущий график стирается.

При выводе графика можно сменить принятый по умолчанию тип точек, с помощью которых рисуется данный график. Например, последовательность команд

```
x=0:.01:2*pi;
y1=sin(x);
y2=sin(2*x);
y3=sin(4*x);
plot(x,y1,'--',x,y2,':',x,y3,'+')
```

приведет к тому, что первый график будет нарисован пунктиром, второй — точками, а третий — символами + (рис. 4). В общем случае каждая линия на графике определяется триплетом **x,y,s**, где **x** и **y** - это вектора с координатами, а **s** - строковая
```

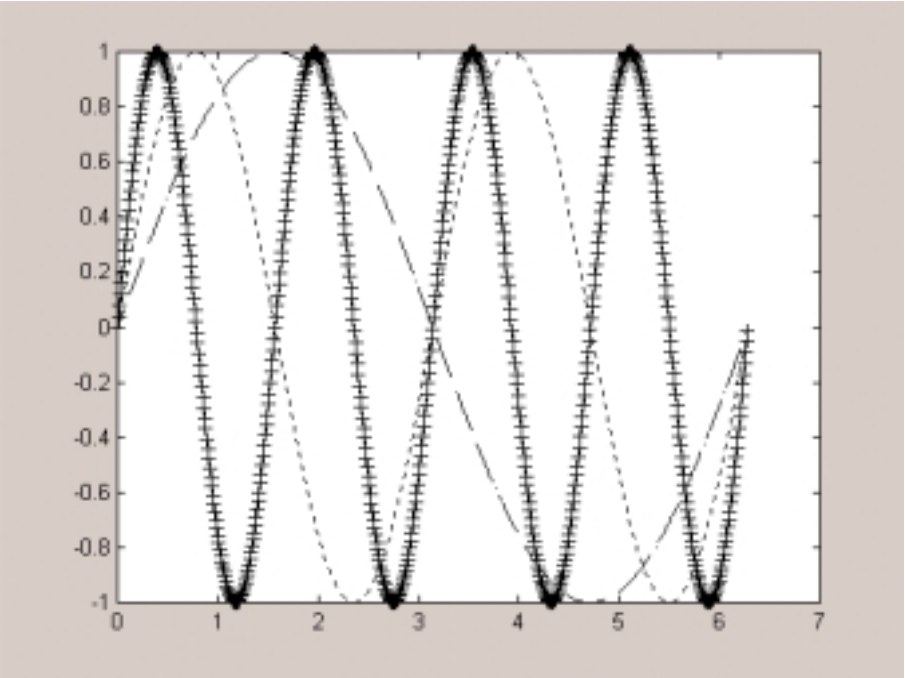


Рис. 4. Пример графика Sin(x), Sin(2x) и Sin(3x)

переменная, образованная любыми комбинациями из каких-нибудь приведенных ниже столбцов.

S	Цвет	S	Маркер	S	Тип линии
y	желтый	.	точка	-	сплошная
m	фиолетовый	o	кружок	:	точечная
c	голубой	x	x-метка	-.	штрих-пунктирная
r	красный	+	плюс	- -	пунктирная
g	зеленый	*	звездочка		
b	синий	s	квадрат		
w	белый	d	алмаз		
k	черный	v	треугольник (вниз)		
		^	треугольник (вверх)		
		<	треугольник (влево)		
		>	треугольник (вправо)		
		p	шестиугольник		
		h	восьмиугольник		

8.1.3. Управление осями при выводе графиков

По умолчанию масштаб осей выбирается автоматически так чтобы график целиком поместился в окне, причем с разумным запасом. Кроме того, оси автоматически

размечаются и по умолчанию выбирается декартова система координат с началом координат [5], в левом нижнем углу. Эта автоматическая установка может быть изменена с помощью команды **axis**. Использование этой функции применительно к плоским графикам описано в следующей далее таблице.

Обращение	Результат
axis([xmin xmax ymin ymax])	Устанавливает пределы изменения соответственно по **x** и **y** координатам соответственно. Если максимальный предел по любой координате установить равным **Inf**, то соответствующий предел будет определяться автоматически. Аналогичное справедливо для нижнего предела, если его установить равным **-Inf**.
axis('square')	Делает область вывода квадратной.
axis('equal')	Делает единицы измерения по оси **x** и **y** одинаковыми.
axis('normal')	Восстанавливает режим по умолчанию.
axis('ij')	Помещает начало координат в левый верхний угол. Направление оси **x** - слева направо, направление оси **y** - сверху вниз.
axis('xy')	Восстанавливает стандартную декартову систему координат с началом в левом нижнем углу и направлением оси **y** снизу вверх.
axis('tight')	Устанавливает пределы по осям точно равными максимальным и минимальным значениям соответствующих переменных.

[5] **Началом координат считается точка с координатами** x_{min}, y_{min}.

Обращение	Результат
axis('off')	Делает невидимыми оси, метки осей и надписи на осях.
axis('on')	Включает оси и их разметку.

8.1.4. Несколько графиков на листе

При необходимости вывести на один экран несколько графиков, т.е. разбить графическое окно на несколько отдельных частей, каждая из которых со своими осями, используется команда **subplot(m n p)** или, что тоже самое, **subplot(m,n,p)**. Значение **m** указывает, на сколько частей окно разбивается по вертикали, **n** - по горизонтали, а **p** - порядковый номер подокна при счете слева направо и сверху вниз. Команда **subplot** используется как для создания нового подокна, так и для перехода от одного подокна к другому. После вызова этой команды **plot** нарисует график или графики в соответствующем подокне. Например, последовательность команд

```
x= -1:.1:1;
y1=sin(x);
subplot(2, 1, 1), plot(x, y1);
y2=log(abs(y1));
subplot(2, 1, 2), plot(x,y2);
```

строит два рисунка в верхней и нижней части экрана. В верхней части строится график **sin(x)**, а в нижней части экрана - зависимость **log(abs(sin(x)))**. Как видно из приведенной выше последовательности команд, при **x=0** вычисляется **log(0)**. При этом в командном окне появляется **предупреждение**, а график строится в точках $-1 \leq x \leq 0.1$ и $0.1 \leq x \leq 1$.

8.2. Специальные виды графиков

Существует целый ряд функций высокого уровня, которые позволяют построить такие виды зависимости, как столбиковые диаграммы, гистограммы, ступенчатые зависимости и статистические кривые с указанием погрешностей.

8.2.1. Столбиковые диаграммы

Как правило, построение таких диаграмм у большинства ассоциируется с электронными таблицами или другими подобными средствами. Но поскольку **MATLAB**

используется и для финансового анализа (а также эта функция удобна для построения *гистограмм*), мы здесь с ней познакомимся.

Если необходимо построить столбиковую диаграмму, показывающую зависимость какой-либо величины, хранящейся в векторе **bar_h**, от номера каждого элемента, то это выполняется командой **bar(bar_h)**. Если необходимо построить эту же зависимость, но от другой величины, то используется функция **bar(bar_h,x)**. Если вам необходимо рисовать столбики определенного цвета или с помощью определенной линии, то можно использовать еще один аргумент у функции **bar(x, y, line_style_string)**, который полностью эквивалентен такому же аргументу у команды **plot**(см. п. 8.1.2). Существуют еще разные возможности построения столбиковых диаграмм, когда аргументом является не вектор, а матрица. При этом возникают различные возможности группировки этих столбиков. Поскольку это выходит за рамки наших интересов, рекомендуем при необходимости ознакомиться с этими возможностями с помощью встроенной помощи(**help bar**).

8.2.2. Ступенчатые кривые

В системе **MATLAB** существует функция, которая создает ступенчатое изображение ваших данных. Например, вместо соединения каждой пары точек из векторов-аргументов прямыми линиями (как это делает функция **plot** или **line**) функция **stairs** изображает ваши данные в виде горизонтальных отрезков на уровне y_i, причем каждый отрезок горизонтальной линии длится от i до $i+1$, если обращение имеет вид **stairs(y)**, и от x_i до x_{i+1}, если обращение имеет вид **stairs(x,y)**. Значения x_i не должны быть равноотстоящими и не должны быть упорядочены по возрастанию. Вывод графика на экран можно предотвратить, используя обращение вида **[xs,ys]=stairs(x,y)**. Потом этот график может быть выведен на экран с помощью команды **plot**, **line** или другим каким-нибудь способом (см., например, п. 8.6)

8.2.3. Гистограммы

Гистограмма является специальным видом столбиковой диаграммы. При построении гистограмы (т.е. графического изображения распределения некоторой величины) необходимо задавать определенное число бинов (иногда говорят: каналов гистограммы)[6], определяя тем самым, сколько данных попадет в каждый бин, и

[6] **Т.е. число разбиений переменной на интервалы, относительно которой и будет вычисляться и строиться распределение. Например, если студенты имеют рост от 150 см до 200 см, то можно разбить этот интервал ростов на 10 бинов, по 5 см в каждом, т.е это интервалы от 150 см до 155 см и т.д. Так вот, гистограмма показывает, сколько студентов попадает в каждый**

графически изображать это в виде столбиковой или ступенчатой диаграммы. В **MATLAB** существует функция **hist**, которая при обращении к ней в виде **hist(y)** вычисляет и рисует гистограмму с 10 бинами, равномерно распределенными между y_{max} и y_{min}. Кроме того, функция **hist(y)** может иметь второй аргумент. Если этот аргумент - целое число, то это число определяет число бинов. Если второй аргумент - вектор, то этот вектор определяет центры используемых бинов. В этом случае центры бинов должны быть равноотстоящими, а координаты этих центров должны быть расположены в возрастающем порядке. При нарушении любого из этих условий результат становится непредсказуемым.

Как и при использовании команды **bar**, можно блокировать вывод на экран гистограммы, присвоив результат выполнения функции **hist(y)** двум выходным аргументам. При обращении вида **[n,x]=hist(y)** или **[n,x]=hist(y,num_of_bins)** или **[n,x] = hist(y,bin_centers)** вычисляются два вектора. Вектор **n** содержит число попаданий величины **y** в каждый из бинов, центры которых находятся в векторе **x**. Такое использование функции **hist** позволяет использовать для отрисовки гистограммы не столбиковую диаграмму (функцию **bar**), а, например, ступенчатую кривую (функцию **stairs**).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Пример постороения гистограммы с помощью STAIRS    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y=randn(100,1);          % Генерация случайных чисел
dx=0.5;                  % Ширина бинов
x=-2.5:dx:2.5;           % Центры бинов
[ny,xh]=hist(y,x);       % Вычисление параметров гистограммы
xh=xh-dx/2;              % Сдвиг координа по оси x на половину ширины бина
[xl,yl]=stairs(xh,ny);   % Вычисление координат ломаной
hl=line(xl,yl);          % Создание дескриптора ломаной линии и отрисовка ее
```

При использовании подобной конструкции будет создан дескриптор **hl**, который может быть далее использован для построения динамических гистограмм с помощью оператора **set**.

8.2.4. Изображение кривых с погрешностями («усами»)

При необходимости изобразить возможный диапазон погрешности кривой, полученной статистическими методами, используется функция **errorbar(x,y,e)**. При

интервал ростов, или бинов.

вызове этой функции будет нарисована кривая, представляющая собой зависимость $y(x)$, а в каждой точке **x,y** изображается вертикальная прямая, имеющая длину, равную удвоенному значению соответствующего элемента вектора **e**. Если **x,y,e** представляют собой матрицы, тогда строится несколько кривых с погрешностями, причем каждая соответствует столбцам матриц.

8.2.5. Изображение функций

Функция **fplot** предоставляет альтернативную возможность изображения функций по сравнению с вычислением вектора **y** по **x** и последующим изображением этой кривой с помощью функции **plot**. Эта функция бывает особенно полезной, когда кривая имеет несколько разных скоростей изменения и заранее не ясно, в скольких и каких точках необходимо вычислять и выводить кривую. Этой функции необходимо передавать строку, описывающую требуемую функцию в виде $f(x)$. Строка, описывающая $f(x)$, может содержать любые допустимые в **MATLAB** операции и/или функции. Функция $f(x)$ должна возвращать вектор той же размерности, что и x, или матрицу, каждый столбец которой имеет столько же элементов, сколько и x.

Например, для того чтобы нарисовать кривую $y = sin(x)cos(2x)$ в диапазоне x от 0 до 5π, необходимо вызвать функцию **fplot ('sin(x) .*cos(2x)', [0 5*pi])**. Функция **fplot** имеет еще два дополнительных (необязательных) аргумента. Один из них - это строка, описывающая тип и цвет линии (аналогично функции **plot**), а вторая - точность. По умолчанию точность равна $2 \cdot 10^{-3}$, и она определяет, на сколько точек делить интервал, чтобы погрешность от линейной интерполяции не превосходила этой заданной точности. Чем меньше точность, тем дольше вам придется ждать вывода графика на экран. При необходимости использовать эту функцию только для вычисления координат рисуемой линии (например, при использовании ее в анимации) используется обращение в виде **[Xp,Yp] = fplot(...)**. В этом случае кривая не рисуется, а соответствующие координаты заносятся в массивы **Xp** и **Yp** соответственно.

8.3. Трехмерные изображения

Не всегда весь объем визуализируемой информации удобно представить в виде набора двумерных кривых. Иногда, по смыслу задачи, данные должны быть представлены как объекты трехмерного пространства. Для изображения таких объектов мы рассмотрим несколько функций.

8.3.1. Одномерная кривая

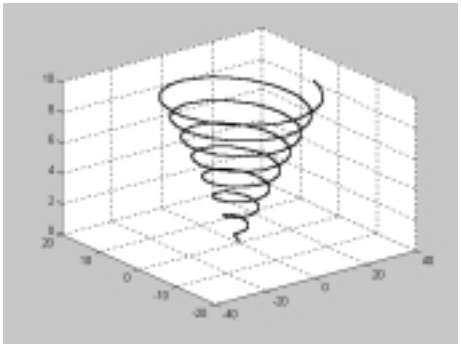


Рис. 5. Пример вывода трехмерной кривой

Для изображения одномерной кривой в трехмерном пространстве используется естественное обобщение функции **plot(x,y)**, которое называется **plot3(x,y,z)**. Для получения рис. 5 использовалась последовательность команд **t=0:0.1:50; x=0.5*t.*cos(t); y=0.6*t.*sin(t); z=0.2*t; plot3(x,y,z);** Способ обращения к команде **plot3**, а также список дополнительных параметров полностью совпадает с описанным в п. 8.1.1 и далее. Аналогично функциям **plot - plot3** существует пара функция **comet - comet3**. Функция **comet3** используется так же, как и **comet** (см. Дополнение, п. 8.1). При использовании для отрисовки трехмерной кривой функции (объекта) **line** можно обращаться к этой функции как к функции **plot**, т.е. **line(x,y,z)**. Возможно также при создании анимации трехмерной кривой использовать изменение свойств объекта с помощью оператора **set**, как это показано в примере к п. 5.1. В этом случае для отрисовки трехмерной кривой необходимо использовать свойство **'ZData'** (по аналогии с **'XData'** и **'YData'**), значением которого должны быть z-координаты соответствующей кривой. Следует отметить, что даже нарисовав такую картинку на плоском экране зачастую трудно представить себе ее истинный вид.

8.3.2. Сеточные поверхности

Трехмерные сеточные поверхности изображаются с помощью функции **mesh**. Команда **mesh(z)** изображает в трехмерной перспективе поверхность, описываемую элементами матрицы **z**. Эта поверхность определяется Z-координатами точек над прямоугольной сеткой в X-Y плоскости. Попробуйте выполнить самостоятельно команду **mesh(eye(10))**. Для того чтобы нарисовать функцию **z=f(x,y)** над прямоугольником, необходимо определить вектора **xx** и **yy** , которые определят разбиение сторон прямоугольника. С помощью команды **meshgrid** можно создать матрицу **x**, каждая строка которой будет совпадать с **xx**, а размер столбцов которой будет совпадать с длиной вектора **yy**, и, аналогично, матрицу **y**, каждый столбец которой совпадает с **yy**, следующим образом **[x,y] = meshgrid(xx,yy);**. После этого необходимо определить матрицу **z**, вычисляя каждый ее элемент как функцию **f** в соответствующих точках, определяемых матрицами **x** и **y**, после чего использовать команду **mesh**. Вы можете, например, нарисовать поверхность на квадрате **[-2,2] x [-2,2]** с помощью следующего набора команд (попробуйте это

выполнить)

```
xx = -2:.1:2;
yy = xx;
[x,y] = meshgrid(xx,yy);
z = exp(-x.^2 - y.^2);
mesh(z)
```

Конечно, первые три строки можно заменить на **[x,y] = meshgrid(-2:.1:2, -2:.1:2);** Более полно с возможностями изображения трехмерных поверхностей можно познакомиться с помощью оперативной помощи (**help plot3/mesh/surf**), по руководству пользователя [6] или с помощью книги [5].

8.3.3. Изолинии

Одним из популярных способов визуализации поверхностей является изображение изолиний. **MATLAB** предоставляет возможность построения изолиний двух типов - двумерных или плоских, фактически являющихся проекциями соответствующих линий постоянного значения на плоскость X-Y, и трехмерных изолиний, нарисованных в какой-либо перспективе.

Простейшим способом изобразить изолинии на плоскости - это обратиться к функции **contour(Z)**, где **Z** - это матрица, содержащая значения исследуемой поверхности на равномерной сетке, т.е. $Z_{ij} = f(i,j)$. При таком обращении система **MATLAB** сама выберет число изолиний и значения функции, при которой они будут построены. Если вы хотите сами задать число выводимых изолиний, то необходимо обращение вида **contour(Z,n_of_lines)**, если же необходимо задать сами эти уровни, то это можно сделать с помощью обращения **contour(Z,values _of _levels)**. Если вам необходимо нарисовать одну изолинию с определенным значением функции, то вектор, содержащий уровни, должен иметь два одинаковых элемента, равных этому уровню.

Отмеченные выше три способа вызова функции построения изолиний стороят их по отношению к номерам строк и столбцов матрицы **Z** так, что элемент **Z(1,1)** будет помещаться в левом нижнем углу рисунка. Можно построить изолинии относительно выбранных вами масштабов по оси X и Y. Для этого необходимо передать функции вектора или матрицу координат, определяющие координаты каждого из элементов матрицы **Z**. Эта передача осуществляется путем одного из следующих обращений:

```
contour(x,y,Z); contour(x,y,Z,n_of_lines);
contour(x,y,Z,values_of_levels);
```

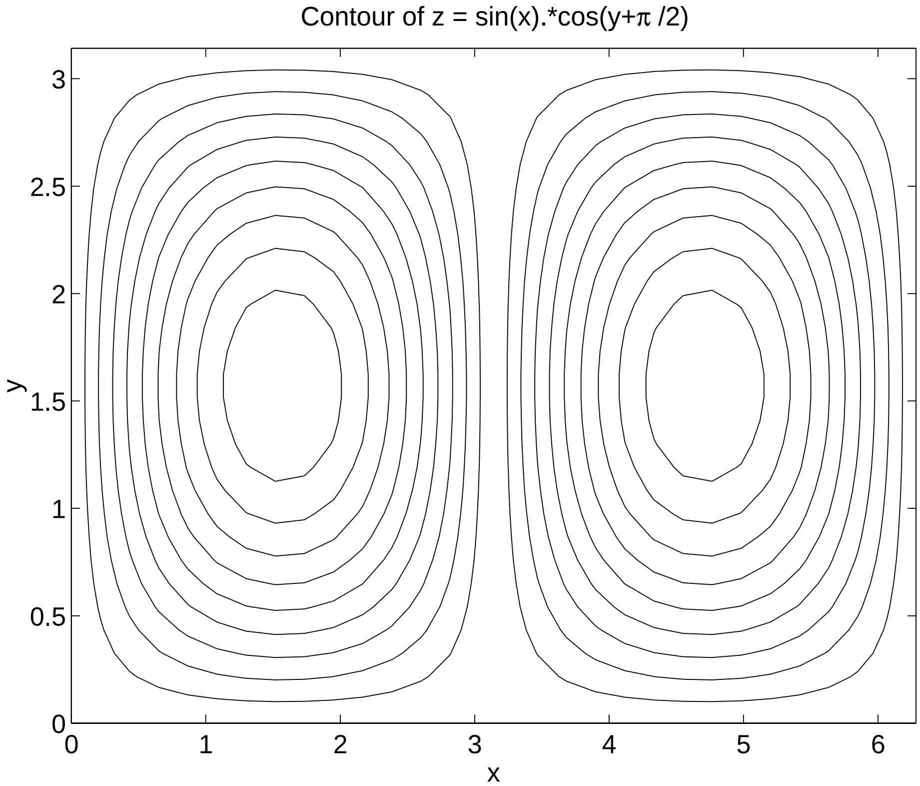


Рис. 6. Пример построения изолиний функции $z(x, y) = \sin(x) \cdot \cos(y + \pi/2)$.

В качестве примера рассмотрим построение изолиний функции $f(x) = \sin(x) \cdot \cos(y + \pi/2)$.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Пример построения изолиний функции
%        sin(x)*cos(y+pi/2)
% Пример взят из [4]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Вычисление равномерной по x и y сетки
 [x,y]=meshgrid(linespace(0,2*pi,30),linespace(0,pi,30));
% Расчет значений функции в узлах сетки
% Использование '.*' означает почленное умножение
 z=sin(x).*cos(y+pi/2);
% В следующей строке строятся изолинии
% со значениями в интервале от -1 до 1
% с шагом 0.1 исключая значение 0
 contour(x,y,z,[-1:0.1:-0.1 0.1:0.1:1]);
xlabel('x');
```

```
ylabel('y');
title('Contour of z = sin(x)*cos(y+ ρi/2)');
```

В результате выполнения этой программы будет выведен рис. 6. Если у вас цветной дисплей, то линии будут разноцветные, а порядок их раскраски будет такой же, как и при выводе нескольких линий с помощью функции **plot**. К сожалению, на полученном таким образом графике нельзя будет установить, какое значение относится к какой линии, и являются ли явно видные области экстремума впадинами или выступами. Для того чтобы пометить требуемые линии их значениями, используется функция **clabel**, а для того чтобы увидеть выступы или впадины, можно просто использовать функцию **contour3** вместо функции **contour**.

Функция **clabel** может быть использована совместно с приведенным выше примером. Необходимо вместо обращения к функции **contour** вставить следующие строки

```
c=contour(x,y,z,[-1:0.1:-0.1 0.1:0.1:1]);
 clabel(c);
```

при этом все изолинии будут помечены, но эти цифровые метки-значения будут размещены произвольным образом. Для того чтобы были помечены не все изолинии, достаточно модифицировать приведенные выше две строки следующим образом:

```
c=contour(x,y,z,[-1:0.1:-0.1 0.1:0.1:1]);
 clabel(c,[-1:.2:1]);
```

В этом случае будут помечены только те линии, значения которых определены вторым аргументом функции **clabel**. Если же вы хотите расставить значения на изолиниях вручную, то второе обращение к функции **clabel** необходимо видоизменить на **clabel(c,'manual')**. После вывода рисунка значения не будут проставлены, но появится крест вместо обычного указателя мыши. Щелкните левой кнопкой мыши около той изолинии, значение которой вы хотите вывести, и в том месте, где вам это представляется удобным. Когда вы пометите нужное вам количество изолиний, нажмите клавишу **Return**. Существуют еще возможности по изменению цвета выводимых изолиний, изменению размера шрифта, которым подписываются значения и т.д., но все они требуют работы со свойствами графических объектов и со знанием основ дескрипторной графики, о чем будет рассказано в разделе 8.4. Для тех, кому нравятся изолинии, выполненные в стиле географических карт, т.е. определенный диапазон значений заливается одним цветом, следующий — другим и т.д., существует функция **contourf** с теми же аргументами, что и функция **contour**.

8.4. Дескрипторная графика (графика низкого уровня)

Этот и следующие разделы будут посвящены графике низкого уровня, которую на русском языке принято называть *дескрипторной*. В них будут описаны средства решения целого ряда задач, которые не могут быть решены с помощью описанных в разделах 8.1- 8.3 функций высокого уровня.

8.4.1. Графические объекты и их иерархия

Прежде всего познакомимся с понятием *графический объект*. Графические объекты - это те базисные объекты, из которых на экране возникает изображение. Даже самый элементарный график состоит из нескольких графических объектов. Это окно, в котором выводится график, линии, оси, метки на осях и т.д. Все графические функции высокого уровня, описанные ранее, создавали такие графические объекты как линии (**line**), оси (**axes**) и др. Все эти объекты являются строительными блоками, из которых **MATLAB** создает различные изображения.

Существуют графические команды низкого уровня, которые создают 10 типов графических объектов в дополнение к корневому объекту, который создается автоматически при входе в **MATLAB**. Эти команды и создаваемые ими объекты перечислены в таблице.

Объект	Команда низкого уровня
Figure(рисунок)	figure
Axes (оси)	axes
Line(линия)	line(x,y) или line(x,y,z)
Patch (заплата)	patch(x,y,c) или patch(x,y,z,c)
Surface (поверхность)	surface(X,Y,Z,C), surface(X,Y,Z) surface(Z,C), surface(Z)
Image (картинка)	image(C) или image(x,y,C)
Light (освещение)	light('Prop_name','Prop_value',...)
Text (текст)	text(x,y,text_str) или text(x,y,z,text_str)
User Interface Control (интерактивное управление)	uicontrol
User Interface Menu (меню)	uimenu

Переменные **x,y** и **z**, используемые в качестве аргументов команды **line**, являются векторами или матрицами одинаковой длины, и они задают последователь-

ность координат кривой (или кривых - если переменные матрицы). В случае матриц каждая кривая определяется столбцами соответствующих матриц.

Переменные **x,y** и **z**, используемые в качестве аргументов команды **patch**, являются координатами вершин соответствующего многоугольника (заплатки). Переменная **c** используется для задания цвета заливки многоугольника.

При создании поверхности (команда **surf**) **X,Y** и **Z** являются матрицами, описывающими координаты четырехугольников, которыми аппроксимируется поверхность. Переменная **C** определяет цвет.

Размещение текстового объекта (строка **text_str**) определяется переменными **x,y** и **z**. По умолчанию первая буква строки прижимается левым краем и центрируется по вертикали к точке **x,y** или **x,y,z**.

Команда **light** сама не создает какого-либо нового изображения, а только определяет освещение объектов **patch** и/или **surf**. Объекты (и соответствующие команды) взаимодействия с пользователем будут описаны в соответствующем месте.

Графические объекты системы **MATLAB** являются не только физическими объектами, но и объектами в смысле объектно-ориентированного программирования. В связи с этим они связаны иерархической структурой типа родители-дети. Каждый объект-наследник не может появиться без появления соответствующего предка. Например, линия (**line**) не может быть нарисована перед тем, как будут созданы оси (**axes**) и рисунок (**figure**). К счастью, вы не должны сами с помощью команд низкого уровня создавать всю последовательность объектов. Как только вы создадите кривую, например с помощью команды **plot(1:10)**, соответствующие рисунок и оси будут созданы автоматически. С другой стороны, вам не возбраняется написать последовательность низкоуровневых команд типа

```
figure;
axes;
plot(1:10);
```

Если после создания рисунка вы закроете его, например с помощью команды **close**, то при этом исчезнут и дети, т.е. оси и линии. То же самое происходит при использовании команды **delete** (удалить). Иерархическая структура объектов показана на рис. 7. Основным или корневым объектом **root** является экран. Он генерируется автоматически при вхождении в систему **MATLAB**. Рисунок (**figure**) является следующим в иерархии объектом, и он либо создается специально одноименной командой, либо генерируется при вызове высокоуровневых команд, описанных ранее. При необходимости можно создавать много рисунков, ограничением является только количество памяти у вашего компьютера. Все последующие (после **figure**)

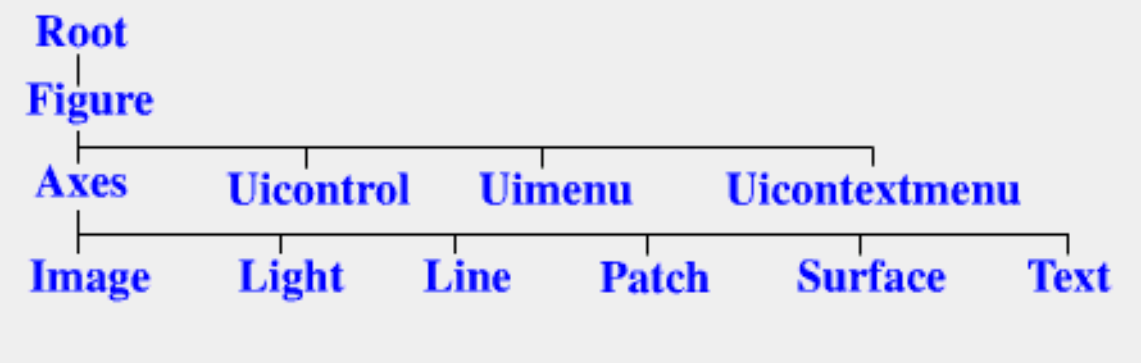


Рис. 7. Иерархическая структура графических объектов

команды осуществляют вывод в *текущее окно*. Простейшим способом сделать тот или иной рисунок текущим используется команда **figure(num)**, где **num** - это номер рисунка, который выводится вверху соответствующего окна.

Рисунок (**figure**) имеет четыре наследника. Три из них - **uicontrol**, **uimenu** и **Uicontextmenu** являются объектами, предназначенными для создания интерфейса пользователя, и о них мы поговорим позже. Объект оси (**axes**) определяет область вывода детей данного объекта на рисунке. Таких областей может быть несколько, каждая со своими наследниками (т.е. линиями, текстами, поверхностями и т.д.). Это аналогично использованию команды высокого уровня **subplot**, но обладает более гибкими возможностями.

8.4.2. Дескрипторы и работа с ними

Для того чтобы манипулировать упомянутыми выше, а также другими объектами в системе **MATLAB**, используются *дескрипторы графических объектов* (**graphics object handles**). Эти дескрипторы можно воспринимать как метки, ярлычки соответствующих объектов. Их нельзя изменять. Они создаются и уничтожаются вместе с объектами, но каждый объект однозначно определяется своим дескриптором. Дескриптор экрана (**root**) и дескриптор рисунка (**figure**) имеют целые значения, все остальные дескрипторы имеют значения вещественные.

Все функции низкого уровня и большинство функций высокого уровня возвращают значение дескриптора, если они вызываются с выходным аргументом, т.е. в виде **graph_handl=function_name**.

Например, мы можем создать такие объекты, как рисунок, оси и линию, и сохранить значения их дескрипторов в переменных **fig_han, axes_han** и **line_han** с помощью последовательности операторов

```
fig_han = figure;
```

```
axes_han = axes;
line_han = plot(exp(-([-3:3].^2)))
```

Если при вызове, например, функции высокого уровня **plot** создается несколько однотипных объектов (несколько линий), то возвращаемый дескриптор является вектор-столбцом. Хотя, как мы знаем, объекты-предки, не созданные явно заранее, создаются в процессе создания объекта-наследника, тем не менее, используя только третью строку вышеприведенного примера, мы создадим рисунок и оси, но их дескрипторы не узнаем во время создания самих объектов. Как узнать дескриптор объекта после его создания, будет описано далее.

Не все функции высокого уровня при обращении к ним в виде **h=f()** возвращают дескрипторы соответствующих объектов. Не возвращают дескрипторы следующие функции:

bar	compass	errorbar	feather
fplot	hist	polar	rose
stairs	quiver	sphere	cylinder

Если вы не сохранили дескриптор рисунка, осей или других созданных объектов, их дескриптор можно получить с помощью ряда команд. Например, дескриптор текущего рисунка можно получить с помощью команды **gcf**(get current figure), дескриптор текущих осей (т.е. тех осей, в которых сейчас будет происходить рисование) можно получить с помощью команды **gca**(get current axes). Если на рисунке несколько осей, то мы получим дескриптор той области рисования, которая была создана последней или внутри которой последним создавался какой-либо объект. Существует на рисунке еще объект, который называется *текущим*. Это такой объект, который был последним создан, с которым последним производили манипуляции или на котором последним щелкали мышкой. Дескриптор такого объекта можно получить с помощью команды **gco**(get current object).

Если тем или иным способом вы сохранили дескрипторы объектов, то вы можете делать текущим тот или иной объект (т.е. переходить от объекта к объекту) с помощью команд **figure(fig_handl)** и/или **axes(axes_handl)**.

Дескрипторы, или функции, возвращающие дескрипторы (**gcf, gca** и др.), могут использоваться в качестве аргументов функций, которые изменяют или устанавливают те или иные свойства объектов. Это прежде всего описанные далее функции **get** и **set**.

Одним из самых популярных способов узнать и/или использовать дескриптор объекта является обращение к функции **findobj**. Если эту функцию использовать без параметров, то в результате она выдаст список всех дескрипторов объекта **Root**

и всех его наследников. При использовании функции в виде **H=findobj(hl)**, где **hl** - дескриптор какого-нибудь созданного объекта, будет возвращен и занесен в массив **H** список дескрипторов данного объекта и всех его наследников. При использовании же этой функции в виде **h=findobj('Имя_Свойства','Значение_Свойства')** будет получен список всех дескрипторов объектов, у которых свойство с именем **'Имя_Свойства'** имеет значение **'Значение_Свойства'**. Если таких объектов окажется несколько, то будет получен вектор дескрипторов.

8.5. Свойства графических объектов и работа с ними. Функции get и set

Каждый из объектов, упомянутых выше, имеет набор свойств, которые ассоциируются только с ним. В этих свойствах заключается вся информация, которая необходима для того, чтобы этот объект был выведен на экран. В момент создания объекта те свойства, которые вы не определили, принимают свои значения по умолчанию. Всякое свойство имеет имя ('Property_Name') и значение, в качестве которого может быть строка, вектор или матрица. Для того чтобы работать со свойствами объектов, вовсе не обязательно помнить все их имена и их возможные значения. Не обязательно также постоянно консультироваться со справочной системой или с руководством (даже если оно у вас имеется). Для определения имен свойств и возможных значений этих свойств, а также для изменения их значений имеются две функции **get** и **set**.

Для того чтобы получить полный список имен свойств некоторого графического объекта с дескриптором **h**, достаточно ввести команду **get(h)**. Следует при этом отметить, что этот объект должен быть создан, пусть даже формально. Например, если вы хотите определить, какие свойства имеет объект **line**, то достаточно определить дескриптор линии командой **hl=line(1,1)** или даже просто **hl=line**, после чего можно будет с помощью команды **get(hl)** определить список имен свойств, а также присвоенные им на данный момент значения. Большинство свойств имеет вполне понятные имена (конечно, для знающего английский язык). Например, легко увидеть, что у объекта **line**(линия) имеются свойства[7]

[7] **В правой колонке приводится перевод имен свойств.**

Color = [0 0 0]	Цвет = [0 0 0]
EraseMode = normal	Режим стирания = normal
LineStyle = -	Стиль линии = -
LineWidth = [0.5]	Ширина линии = [0.5]
Marker = none	Маркер = none
MarkerSize = [6]	Размер маркера = [6]
MarkerEdgeColor = auto	Цвет контура маркера = auto
MarkerFaceColor = none	Цвет заливки маркера = none
XData = [0 1]	XДанные = [0 1]
YData = [0 1]	YДанные = [0 1]
ZData = []	ZДанные = []
...	...

Ряд свойств может принимать произвольные значения, а другие свойства — только определенные. Если необходимо узнать, какие значения может принимать то или иное свойство, то используется оператор **set(hl)**, где **hl** - дескриптор соответствующего, интересующего вас объекта. В результате получаем на экране информацию, подобную приведенной далее

```
Color
EraseMode: [ {normal} | background | xor | none ]
LineStyle: [ {-} | -- | : | -. | none ]
LineWidth
Marker: [ + | o | * | . | x | square | diamond | v | ^ | >]
MarkerSize
MarkerEdgeColor: [ none | {auto} ] -or- a ColorSpec.
MarkerFaceColor: [ {none} | auto ] -or- a ColorSpec.
....
```

Данные, приведенные в квадратных скобках и отделенные друг от друга вертикальной чертой, являются вариантами возможных значений, а значения в фигурных скобках являются значениями по умолчанию.

Для получения значения свойства какого-либо объекта и возможного присвоения этого значения переменной используется оператор **get** в форме **x=get(hl, 'Property_Name')**. Такая запись приведет к тому, что переменной **x** будет присвоено значение свойства **'Property_Name'**. Например, если в предыдущем примере записать оператор **x=get(hl, 'Color')**, то будет создан вектор **x= [0 0 0]** - это запись цвета линии в RGB-формате. Для присвоения соответствующему свойству нового значения используется оператор **set** в виде **set(hl, 'Prop_Name',**

'Prop_Value'); Например, для смены цвета линии (объект **line** с дескриптором **hl**) используется оператор **set(hl,'Color','red')**, или, что эквивалентно, **set(hl,'Color',[1 0 0])**.

8.6. Движущиеся графики (анимация)

Для вывода на экран движущихся объектов в **MATLAB** имеется ряд возможностей. Одна из них - расчет последовательных состояний объекта и сохранение соответствующего изображения в специальном формате с последующей прокруткой этого мультфильма. Такие возможности нами в этом руководстве рассматриваться не будут. Интересующиеся могут познакомиться с описанием функций **movie**.

Нас будет интересовать возможность получения на экране движущихся объектов "на лету", т.е. в процессе счета. Частично эту проблему решают такие функции, как **comet** и **comet3**. Эти функции позволяют получить квази-анимацию, т.е вывести и показать в движении траекторию одной или нескольких частиц (**comet** - на плоскости, а **comet3** — в трехмерном пространстве), но после того, как эти траектории будут насчитаны. Это бывает довольно полезно, но это лишает нас возможности оперативно вмешиваться в описываемый процесс (менять какие-либо параметры расчетной модели).

Можно попытаться, например, при отрисовке изменяющейся кривой (или любого другого меняющего свое положение объекта) вызывать на каждом шаге расчета функцию **plot**, **line** или **histo**, но поскольку все функции высокого уровня при обращении к ним сами заново создают объект **axes**(оси) и сами выбирают масштаб, то при таком подходе вы увидите постоянно мигающее изображение с изменяющимся масштабом осей. Попробуйте сами набрать и исполнить программу, приведенную ниже.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Это пример неудачной анимации при выводе бегущей волны %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all
t=0:0.1:100;
x=0:0.3:30;
k=1.3; w=0.9;
n=length(t);
y=cos(k*x-w*t(1))+cos(x-t(1));
figure
```

```
plot(x,y);
for i=2:n;
  y=cos(k*x-w*t(i))+cos(x-t(i));
  plot(x,y);
  drawnow
end;
```

Это вовсе не то, что подразумевается под анимацией. Дело в том, что постоянное создание-удаление объектов приводит к миганию картины и к медленному ее обновлению. Для получения «правильной» анимации в процессе счета необходимо перейти к использованию графических функций низкого уровня и не использовать механизм создания-удаления объекта для изображения его движения, а создав однажды объект менять только его свойства (координаты в нашем случае).

Рассмотрим, как это выглядит практически на том же примере — движение пакета волн.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Это пример "правильной">> анимации при выводе пакета волн %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all
% Задание значений параметров
t=0:0.1:100;
x=0:0.3:30;
k=1.3; w=0.9;
n=length(t);
% Расчет формы волны при t(1)=0
y=cos(k*x-w*t(1))+cos(x-t(1));
% Вычисление границ рисунка
axlim = [min(x) max(x) min(y) max(y)];
figure
% Сохранение дескриптора выводимого объекта
lh = line(x,y);
% Задание цвета кривой и выбор границ
set(lh,'color','r');
axis(axlim);
% Выбор режима стирания
set(lh,'erasemode','xor');
% Начало цикла анимации
```

```
 for i = 2:n
% Пересчет координат кривой
   y=cos(k*x-w*t(i))+cos(x-t(i));
% Самый главный момент - замена координат
   set(lh,'XData',x,'YData',y);
drawnow;
 end
```

В приведенной выше программе есть три узловых момента, обеспечивающих качественную анимацию.

1. Во-первых, это подготовка системы координат и создание дескриптора выводимого объекта с присвоением свойству **'EraseMode'** (режим стирания) значения **'xor'**. Обратите внимание на этот параметр. Дело в том, что значение этого свойства по умолчанию- **'none'** и при этом выведенный ранее объект не стирается, а остается на экране. Возможно осуществление анимации и с помощью режима стирания **'background'**, причем анимация будет происходить быстрее. Различия между этими двумя режимами лучше всего посмотреть введя дополнительно перед циклом анимации команду **grid on** (нарисовать координатную сетку). При режиме **xor** сетка остается после «прохождения» волны по экрану, а при использовании режима стирания **background** сетка затирается там, где прошла волна.

2. Фактически перерисовка объекта происходит при изменении значений свойств объекта **'XData'** и **'YData'** (возможно, **'XData'**, **'YData'** и **'ZData'**, если это объект в трехмерном пространстве).

3. Важным фактором непрерывного обновления картинки является команда **drawnow**, которая обеспечивает постоянный вывод обновленных значений на экран. Такой принудительный вывод обеспечивают также функции **pause** и **getframe**. Возможен (и иногда более интересен) другой способ принудительного вывода на экран. Если после создания дескриптора объекта вы выполните принудительный вывод этого объекта на экран с помощью команды **pause** до входа в цикл анимации, то при вхождении в цикл команда **drawnow** уже не нужна, анимация будет происходить до исчерпания цикла или до принудительного прекращения программы.

9. Разработка графического интерфейса пользователя

При разработке прикладных программ представляется полезным создание *графического интерфейса пользователя*. Фактически, это создание среды расчета задач определенного класса без программирования со стороны пользователя. Как правило, такие интерфейсы имеет смысл разрабатывать для задач с несколькими параметрами если предполагается неоднократное решение подобных задач. В таком случае целесообразно разработать графический интерфейс, который помогает пользователю получать результаты решения задачи (как правило в графическом виде) при определенном выборе параметров. Такой интерфейс может быть также удобен при создании учебных задач, потому что обучающийся в таком случае основное внимание тратит не на программирование или решение задачи, а на подбор требуемых параметров, анализ и осмысление получающихся результатов.

Из приведенного выше краткого введения понятно, что обязательными элементами графического интерфейса при решении научных и/или учебных задач должны быть:

1. Одно или несколько окон для вывода графических результатов расчета.
2. Несколько редактируемых окон, с помощью которых задаются и/или изменяются значения параметров задачи.
3. Управляющие кнопки, которые позволяют запускать и останавливать процесс расчета, перерисовывать результаты, выходить из задачи.
4. Поясняющие надписи (статический текст).

Конечно, возможны и другие элементы управления, такие как прокручиваемые списки, радио-кнопки для выбора одного из многих вариантов и т.д., но в настоящем пособии мы рассмотрим подробно только перечисленные в списке четыре типа. На рис. 8 показан простейший интерфейс, созданный для исследования биений, образующихся при сложении двух гармонических колебаний с близкими частотами. Как видно из рисунка, все вышеперечисленные элементы в нем присутствуют.

Для создания такого интерфейса можно воспользоваться функциями графического вывода, а также специальной функцией, разработанной для интерактивного взаимодействия пользователя с рисунком. Эта функция называется **uicontrol**. Но для упрощения работы и создания однотипных элементов интерфейса в системе **MATLAB** имеется специальная программа, которая позволяет на уровне визуального программирования, почти без написания кода создать требуемые элементы.

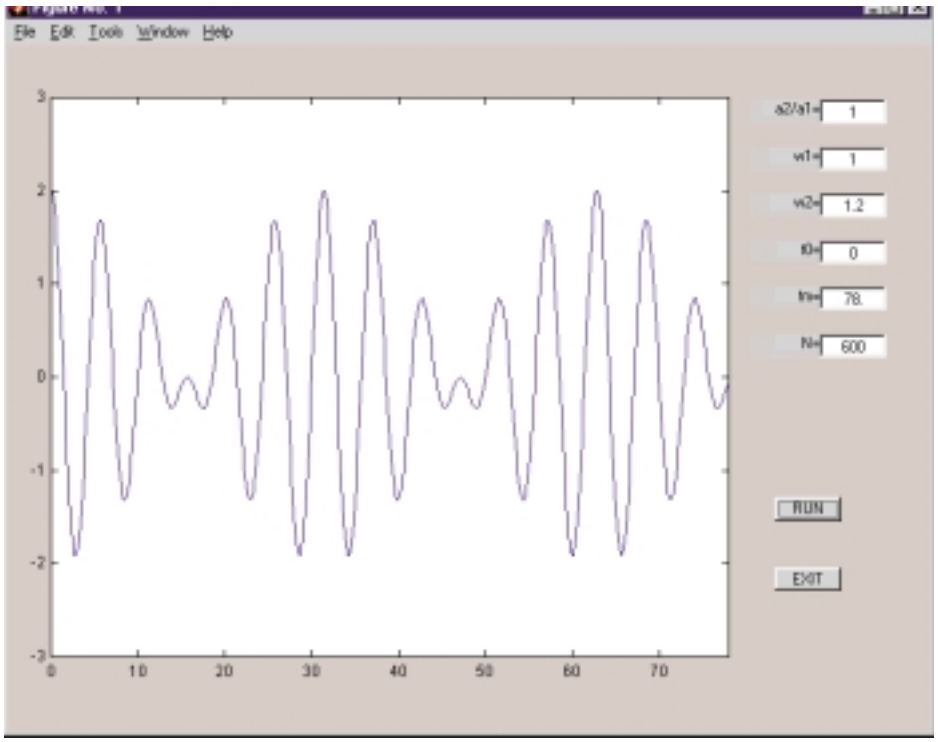


Рис. 8. Простейший графический интерфейс пользователя для решения задачи “Биения”

9.1. Создание внешнего вида интерфейса

В этом параграфе мы рассмотрим использование **MATLAB** для разработки внешнего вида графического интерфейса (**GUI**-GraphicsUserInterface) с использованием средств графического (визуального) программирования. Для вызова визуального редактора необходимо в командном окне **MATLAB** набрать команду **guide**. По истечении определенного времени, определяемого быстродействием вашего компьютера, появятся два новых окна, показанные на рис.9. Одно из них — панель управления (**Control Panel**, на рисунке слева) и форма или область рисования (**Figure**, на рисунке справа). Эти окна могут перекрываться, но мы для ясности изложения расположили их рядом. Показанная выше картинка появится на экране в том случае, если перед вызовом **guide** отсутствует какой-либо открытый рисунок. В случае же если функция **guide** вызывается после отрисовки какого-либо рисунка, то он открывается вместо пустого. Мы же рассмотрим создание графического интерфейса с самого начала.

Перед созданием графического интерфейса желательно “разработать проект” того, что вы хотите иметь в качестве интерфейса. Мы рассмотрим пример вывода трех разных сигналов в трех подокнах, что в терминах графики высокого уровня определялось бы операторами subplot(3,1,k), где k - номер графика. Кроме того,

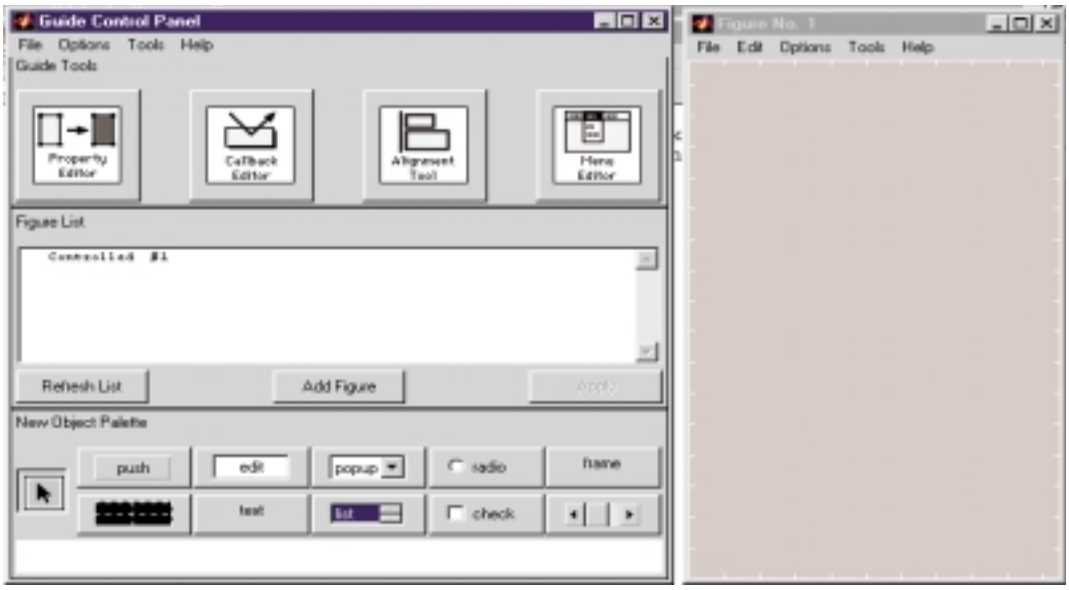


Рис. 9. Общий вид визуального графического редактора и окна редактирования

справа от собственно подокон с графиками мы хотим иметь три редактируемых поля, в которых можно осуществлять ввод/редактирование числовых значений трех переменных. Пусть эти переменные принимают произвольные значения.

В данном изложении мы не будем оснащать наши редактируемые окна проверкой, удовлетворяют ли введенные значения каким-либо условиям, хотя такое возможно. Назовем эти переменные N, R, C. В данном примере имеется в виду расчет тока в RC-цепи при подаче на зажимы сигнала с номером N, а R и C - сопротивление и емкость в цепи (подробное описание задачи см. в параграфе 10 основного текста пособия).

Наш интерфейс должен позволить менять значения N, R, и C, получая в трех расположенных друг над другом подокнах сигнал (напряжение, подаваемое на зажимы), производную от сигнала и напряжение на сопротивлении U_r. Помимо окон для вывода графиков и редактируемых окон ввода необходимо поместить на панель интерфейса еще две кнопки - **RUN** - запуск программы на счет и **EXIT** - прекращение работы и удаление графического окна.

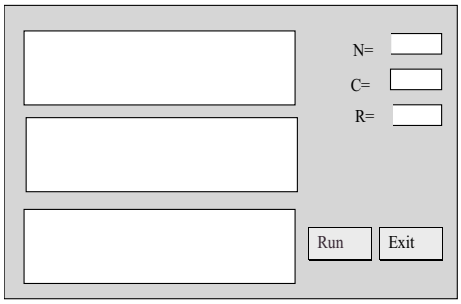
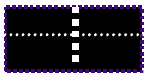


Рис. 10. Вид интерфейса

На первом этапе необходимо разработать внешний вид интерфейса. Пусть по нашему предварительному экскизу интерфейс должен выглядеть приблизительно следующим образом (рис. 10). Элементами этого интерфейса являются три окна

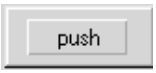
вывода графиков (**axes** в терминах панели управления), три статических надписи N,R,C (**text**), три окна ввода/редактирования данных (**edit**) и две кнопки (**push**).



Для создания подокон, в которые будут выводиться графики, используется кнопка, показанная справа на рисунке (окно и оси). Щелкнув по этому элементу на панели управления и переведя мышь на панель рисунка, необходимо поместить крест, котрый будет на кончике мыши, в то место, где должен находиться левый верхний угол первого подокна. Нажав и удерживая левую кнопку мыши, необходимо вытянуть получающийся прямоугольник до нужных размеров. После этого процедура построения двух других окон повторяется аналогично.

Для создания редактируемых окон ввода используется кнопка **edit**, показанная справа. Используется она так же, как при создании подокон с осями. Сначала появляется мышь, нагруженная крестиком, с помощью которой строится прямоугольник ввода.

Надписи на панели рисунка создаются с помощью кнопки **text**, которая переносится и выравнивается аналогично вышеописанному. Для того чтобы внутри области статического текста появилась какая-либо надпись, необходима работа с редактором свойств, который вызывается либо при помощи кнопки **Property editor**, либо с помощью двойного нажатия левой кнопкой мыши на соответствующем объекте на панели рисунка.

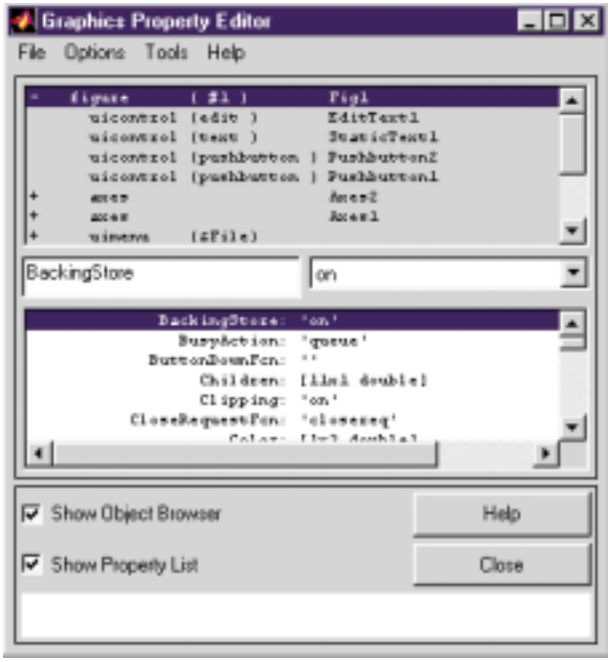


Для создания и размещения кнопок используется панель с надписью **Push**. Способ размещения кнопки и выбора ее размера полностью совпадает с методом, описанным выше для окна редактирования и окна статического текста.

Построенные таким образом окна вывода и редактирования, окна статического текста и кнопки, а также другие объекты можно выровнять и установить определенные промежутки между ними с помощью панели выравнивания (**Alignment Tools**).

Для этого необходимо на панели управления щелкнуть по соответствующей кнопке, и появится панель выравнивания. Для задания ряда объектов, с которыми будут выполняться какие-либо действия, необходимо их выделить, щелкая по каждому из них при нажатой клавише **Shift**. Выделенные объекты отмечаются черными точками вокруг соответствующих объектов. При необходимости изменить размер какого-либо объекта (кнопки, окна и т.д.) необходимо щелкнуть по этому объекту с помощью левой кнопки мыши и с помощью мыши изменить требуемый размер так же, как и размер любого окна Windows.

При работе с графическими объектами на панели рисунка желательно раскрыть (с помощью соответствующей кнопки окна Windows) панель рисунка на весь экран и выбрать размер всех объектов на панели рисунка.



Поскольку по умолчанию размеры всех объектов задаются в пикселях, это может привести к тому, что при изменении размера окна вывода кнопки и/или окна могут наехать друг на друга. Для избежания такого нежелательного явления необходимо задать единицы размера всех объектов в безразмерных переменных — долях размера окна. Этот размер называется **normalized**. Для этого необходимо вызвать редактор свойст щелкнув на панели управления на кнопке с надписью **Property editor**. Выделив в верхнем окне редактора свойств все введенные нами объекты (с помощью удерживаемой клавиши **Ctrl** и выделения объектов левой кнопкой мыши), в нижнем окне редактора свойств находим свойство **Units** (единицы измерения) и выбираем его щелчком левой кнопки мыши. После этого в средней части редактора в специальном окне слева появится свойство **Units**, а справа - окно выбора значений, в котором имеется раскрывающийся список с допустимыми значениями свойства. Для свойства **Units** необходимо выбрать значение **normalized**. Анологично надо задать значение свойства **FontUnits** — единицы измерения размера шрифтов. Это обеспечивает изменение размера шрифта при изменении размера окна.

Для размещения надписей на кнопках и в области статического текста необходимо выделить соответствующий объект (либо двойным щелчком прямо в области рисунка, либо в верхнем окне редактора свойств) и в нижнем окне редактора свойств найти свойство **String**, и после его выделения вписать между кавычками требуемый текст (например, 'Пуск' на соответствующей кнопке). Для задания надписей над каждым из окон вывода необходимо выделить соответствующее окно и вызвать редактор свойств[8], в нижнем окне которого надо найти свойство **Title**.

[8] **Редактор свойств можно вызвать не только с помощью кнопки на панели управления, но и двойным щелчком на соответствующем объекте.**

Поскольку это свойство само является объектом, то, выполнив двойной щелчок на этом свойстве, мы раскроем его свойства. После этого следует найти свойство **String**, задать его значение (соответствующий текст, например 'Номер сигнала') в среднем окне, а, найдя свойство **FontUnits**, задать его значение **normalized**.

Следует еще отметить свойство **Tag**, которое имеют все объекты. Особенно важно знать и/или задать это свойство для тех объектов, к которым потом в программе придется обращаться. Например, если на рисунке задано несколько окон вывода, то для вывода графика в требуемое окно проще всего его будет идентифицировать окно с помощью свойства **Tag**. Это свойство, как правило, имеет по умолчанию вполне осмысленное значение (**Axes1**, например), но при желании его можно задать так, как вам нравится.

После разработки внешнего вида графического интерфейса с помощью визуального редактора необходимо сохранить это в виде m-файла. При нажатии на панели управления пункта меню **File/Close control panel** появится запрос на сохранение созданного интерфейса, и в случае положительного ответа будут созданы два файла – **NAME.M** и **NAME.MAT** (где **NAME** – заданное вами имя). Первый файл – это соответствующий текст программы, реализующий разработанный интерфейс, а второй – набор данных для него. При необходимости впоследствии внести изменения во внешний вид графического интерфейса нужно в командном окне **MATLAB** запустить файл **NAME.M**, а потом с помощью команды **guide(gcf)** вызвать визуальный редактор с этим же файлом. Следует полностью выполнить этап разработки внешнего вида графического интерфейса, поскольку дальнейшая модификация текста программы **NAME.M** будет производиться вручную, путем изменения и/или дополнения текста программы определенными операторами. Если Вы после этого решите снова вызвать **GUIDE** – визуальный редактор формы, то все ваши изменения, внесенные вручную, при попытке сохранить доработанный интерфейс, пропадут. Поэтому сначала разрабатывайте внешний вид своего графического интерфейса, а потом его дорабатывайте.

9.2. Способы взаимодействия графического интерфейса с функциями пользователя

Созданный по описанной выше схеме **GUI** не обладает никакой функциональностью – он пока ничего не может делать за исключением стандартных функций Windows - менять размеры окна и закрывать окно. Теперь необходимо научить его выполнять требуемые действия при нажатии соответствующих кнопок и редактировании данных. Основным средством взаимодействия графического интерфейса с функциями, выполняющими требуемые действия, является задание свойств кно-

пок и редактируемых окон, которое называется **CallBack**. Значение (типа строка), которое присваивается этому свойству, и есть имя функции, которая вызывается при активации соответствующего объекта. Эта активация происходит при нажатии мышкой на соответствующую кнопку (если это свойство кнопки) и при нажатии клавиши **Enter** либо любой другой кнопки (если это свойство редактируемого окна).

Для задания этих свойств можно использовать визуальный редактор **GUIDE** (при вызванном рисунке) или с помощью стандартного редактора m-файлов дописать соответствующие параметры функции **uicontrol** при обращении к ней в функции **NAME.M**, которую создал **GUIDE**. Например для того, чтобы задать имя функции, вызываемой при нажатии кнопки **RUN**, необходимо вызвать соответствующий рисунок-интерфейс (запустить **NAME.M**) и после этого вызвать **GUIDE**. Выделив соответствующий элемент (кнопку **RUN**) и вызвав редактор свойств, необходимо в списке свойств выделить свойство **CallBack** и в строке задания свойств в кавычках вписать имя функции (например, **Func_Run**), которая будет вызываться при нажатии этой кнопки. Аналогично задаются функции и для других кнопок и редактируемых окон. Тогда соответствующее обращение к функции **uicontrol** для кнопки **RUN** будет выглядеть следующим образом:

```
h1 = uicontrol('Parent',h0, ...
  'Units','normalized', ...
  'FontUnits','normalized', ...
  'Callback','Func_Run', ...
  'FontSize',0.45, ...
  'ListboxTop',0, ...
  'Position',[0.722 0.131 0.1 0.0542], ...
  'String','Run', ...
  'Tag','Run');
```

При ручной правке функции **NAME.M** необходимо добавить в аргументы соответствующей функции **uicontrol** строку вида **'CallBack','Func_run', ...**.

9.2.1. Общая структура функции NAME.M графического интерфейса

После создания внешнего вида интерфейса и описанной его доработки M-файл имеет, как правило, следующую структуру.

1. Один или несколько операторов определения окна вывода, которые имеют вид

```
h1 = axes('Parent',h0, ...
   'FontUnits','normalized', ...
   'CameraUpVector',[0 1 0], ...
   'CameraUpVectorMode','manual', ...
   'Color',[1 1 1], ...
   'ColorOrder',mat7, ...
   'FontSize',0.0877, ...
   'Position',[0.061 0.048 0.63 0.24], ...
   'Tag','Axes3', ...
   'XColor',[0 0 0], ...
   'YColor',[0 0 0], ...
   'ZColor',[0 0 0]);
```

Среди всех свойств этого объекта — окна вывода — для нас наибольший интерес представляет свойство **'Tag'**, ибо по имени свойства (в данном примере **'Axes3'**) впоследствие можно будет направлять вывод результатов расчета в это окно.

2. Каждый оператор определения окна выбора сопровождается целым рядом операторов, определяющих разметку осей, надписи на осях и т.д. Это операторы вида

```
h2 = text('Parent',h1, ...
   'Color',[0 0 0], ...
   'HandleVisibility','off', ...
   'HorizontalAlignment','center', ...
   'Position',[-0.0648 0.487 9.160], ...
   'Rotation',90, ...
   'Tag','Axes2Text3', ...
   'VerticalAlignment','baseline');
set(get(h2,'Parent'),'YLabel',h2);
```

Первый оператор ответственен за создание текста — подписи под осью. В приведенном примере сам текст отсутствует — у этого объекта свойству 'String' никакое значение не присвоено, а, значит, под данной осью нет никакого текста. Второй оператор указывает на то, что этот текст является наследником объекта **'YLabel'** и что текст, если бы он был, размещался бы под осью Y.

3. Остальные операторы — операторы **uicontrol**, определяющие свойства таких объектов, как редактируемые окна, статический текст и кнопки. Типичный оператор, определяющий редактируемое окно, имеет вид

```
h1 = uicontrol('Parent',h0, ...
   'Units','normalized', ...
   'FontUnits','normalized', ...
   'BackgroundColor',[1 1 1], ...
   'Callback','Func_Edit', ...
   'FontSize',0.260, ...
   'ListboxTop',0, ...
   'Position',[0.814 0.889 0.120 0.0583], ...
   'String',num2str(N), ...
   'Style','edit', ...
   'Tag','EditN');
```

Следует обратить внимание на такие свойства этого объекта, как **'Callback'** — его значение (в данном примере **'Func_Edit'**) определяет имя процедуры, которая осуществляет редактирование значения, введенного в соответствующее окно. Свойство **'Tag'**, имеющее в данном примере значение **'EditN'**, определяет возможность найти этот объект по этому свойству. Свойство **'Style'**, имеющее значение **'edit'**, указывает на то, что это редактируемое окно. Значение свойства **'String'** определяет строку, которая будет выводиться в редактируемом окне. В данном примере для вывода используется функция **num2str**, которая переводит значение числовой переменной (в данном примере **N**) в строковое представление.

4. Оператор, определяющий кнопку, как правило, имеет вид

```
h1 = uicontrol('Parent',h0, ...
   'Units','normalized', ...
   'FontUnits','normalized', ...
   'Callback','Func_Run', ...
   'FontSize',0.45, ...
   'ListboxTop',0, ...
   'Position',[0.722 0.13125 0.1 0.0542], ...
   'Style','pushbutton', ...
   'String','Run', ...
   'Tag','Run');
```

Из существенных для нас свойств этого объекта отметим свойство **'Callback'**, значение которого (в данном примере **'Func_Run'**) определяет имя функции, которая будет вызываться при нажатии мышкой на эту кнопку. Текст, который выводится на кнопке, определяется значением свойства **'String'**, а свойство **'Tag'** (значение которого можно выбрать по своему усмотрению) определяет возможность впоследствии обратиться к этому объекту.

Начальные значения, которые необходимо задать для занесения начальных значений в интерфейсную форму, помещаются в начале функции **NAME.M** в виде

```
N=1;
R=1;    % Задание начальных значений
C=1;
info.N=N;
info.R=R; % Задание начальных значений в записи INFO.
info.C=C;
```

В операторах **uicontrol**, которые ответственны за редактируемые окна (признаком этого является строка **'Style','edit',...**), необходимо свойству **'String'** присвоить значение **num2str(N)**, а вместо формального, одинакового дескриптора **h1** использовать в этих строках дескриптор **info.sN**, так что соответствующее обращение будет иметь вид

```
info.sN = uicontrol('Parent',h0, ...
  'Units','normalized', ...
  'FontUnits','normalized', ...
  'BackgroundColor',[1 1 1], ...
  'Callback','Func_Edit', ...
  'FontSize',0.26, ...
  'ListboxTop',0, ...
  'Position',[0.814 0.89 0.12 0.058], ...
  'String',num2str(N), ...
  'Style','edit', ...
  'Tag','EditN');
```

В конце функции **NAME.M** необходимо вставить строку **Set(h0,'UserData',info);**. Это сохранение всех данных из записи **info** в общем хранилище данных - **UserData**.

9.2.2. Функционирование графического интерфейса

Принципиальная схема взаимодействия модулей разработанной программы показана на рис. 11. Головная программа **DRIVE** вызывает основную графическую

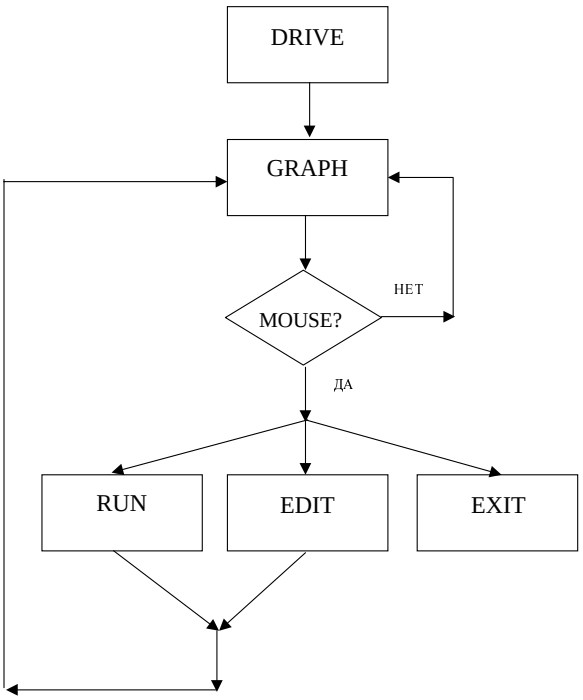


Рис. 11. Принципиальная схема функционирования графического интерфейса

программу **NAME.M**, которая при первом исполнении создает графический интерфейсный экран с кнопками, окнами, поясняющим текстом и т.д. После этого система переходит в режим ожидания событий - нажатия мышкой в интерфейсном окне. Если это нажатие происходит в редактируемом окне, после чего с помощью клавиатуры изменяется значение в этом окне и нажимается клавиша **Enter**, то начинается выполнение процедуры **Func_Edit**, которая и обновляет соответствующие данные в записи **info**, после чего заносит обновленные значения в **UserData**. Если же происходит нажатие на кнопку **Run**, то обновляются значения редактируемых окон и выполняется функция **Run**.

9.2.3. Разработка функции Run

Функция выполнения основного действия (кнопка **RUN**) пишется на основании ранее имевшейся программы расчета R-C цепочки. Первая группа добавляемых операторов - извлечение новых значений задаваемых переменных

```
 h0=gcf;
info=get(h0,'UserData');
N=info.N;
R=info.R;
C=info.C;
```

```
.....................
% Основной расчетный алгоритм

% Построение насчитанных графиков в окне
% Первое окно - рисуем сигнал
 hax1=finobj('Tag','Axes1'); % Определение дескриптора осей этого окна
 subplot(hax1);              % Рисуем подрисунок в этом окне
 cla;                        % Стирание предыдущего изображения в этом окне
 hl1=line(t,E);              % Определить линию с дескриптором hl1
 set(hl1,'color','r',...     % Обновить данные для линии hl1 цветом 'r'
   'parent',hax1);           % в осях hax1
 title('Сигнал E(t)');       % добавить надпись к этому окну
.........................
```

И так далее для каждого окна.

9.2.4. Разработка функции Exit

Самая простая процедура - это процедура отключения программы и стирания рисунка

```
 function f=Func_Exit()
delete(gcf);
```

После выполнения этой процедуры удаляются все графические объекты в текущем окне и закрывается графическое окно.

9.2.5. Разработка функции Edit

Функция редактирования значений в окнах ввода.

```
 h0=gcf;
% Считывание данных из общего блока всего рисунка.
 info=get(h0,'UserData');
% Определение дескриптора редактируемого окна
 hedN=findobj(h0,'Tag','EditN'
% Сканирование строки с преобразованием к формату e
 newN=sscanf(get(hedN,'string'),'%e');
 if ~isempty(newN)
   info.N=newN;
end;
..............
```

Дескриптор соответствующего редактируемого окна в данном примере находится с помощью функции **findobj** - нахождение объекта по известному значению его свойства (в данном случае - свойству **Tag**). Можно эти дескрипторы передавать из процедуры **NAME.M** с помощью оператора **global** или через свойство **UserData**. В конце всей функции после выполнения группы операторов, подобных приведенным, для каждого редактируемого окна необходимо результаты обновления **info** присвоить свойству **'UserData'** с помощью оператора

set(h0,'UserData',info); % Занесение новых данных из info в UserData.

Глеб Леонидович Коткин
Валерий Семенович Черкасский

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ

Учебное пособие

Подписано в печать | Формат $60 \times 84/16$
Офсетная печать. | Уч.-изд.
Заказ № | Тираж 300 экз. | Цена

Редакционно-издательский отдел Новосибирского университета;
участок оперативной полиграфии НГУ; 630090, Новосибирск-90,
ул. Пирогова, 2.