МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Омский государственный технический университет»

А. М. Ласица

ИСПОЛЬЗОВАНИЕ MATLAB И GNU OCTAVE В ВЫЧИСЛИТЕЛЬНОЙ ФИЗИКЕ

Конспект лекций

В двух частях

Часть 1

Учебное текстовое электронное издание
локального распространения

Омск
Издательство ОмГТУ
2017

Сведения об издании: 1, 2

© ОмГТУ, 2017

ISBN 978-5-8149-2464-3 (ч.1)
ISBN 978-5-8149-2483-4

УДК 53:004.421
ББК 30.13+32.973
Л26

Рецензенты:

Т. А. Аронова, к. ф.-м. н., доцент кафедры физики и химии ОмГУПС;

Г. Ж. Худайбергенов, к. ф.-м. н., доцент кафедры экспериментальной физики и радиофизики физического факультета ОмГУ им. Ф. М. Достоевского

Ласица, А. М.

Л26 Использование Matlab и GNU Octave в вычислительной физике : конспект лекций : в 2 ч. / А. М. Ласица ; Минобрнауки России, ОмГТУ. – Омск : Изд-во ОмГТУ, 2017.

ISBN 978-5-8149-2483-4

Ч. 1 : ил.

ISBN 978-5-8149-2464-3

Издание содержит первую часть конспекта лекций с методическими указаниями к лабораторному практикуму по дисциплине «Вычислительная физика». Посвящено решению ряда типичных для этой дисциплины задач. Представлены условия задач, проанализированы физические процессы, проведено исследование возможных методов и алгоритмов решения. Рассмотрена численная реализация алгоритмов с помощью современных математических программных пакетов Matlab R2017 и GNU Octave 4.2.1.

Предназначено студентам всех форм обучения.

УДК 53:004.421
ББК 30.13+32.973

Рекомендовано редакционно-издательским советом Омского государственного технического университета

ISBN 978-5-8149-2464-3 (ч. 1)
ISBN 978-5-8149-2483-4

© ОмГТУ, 2017

1 электронный оптический диск

Оригинал-макет издания выполнен в Microsoft Office Word 2007/2010
с использованием возможностей Adobe Acrobat Reader.

Минимальные системные требования:

- процессор Intel Pentium 1,3 ГГц и выше;
- оперативная память 256 Мб и более;
- свободное место на жестком диске 260 Мб и более;
- операционная система Microsoft Windows XP/Vista/7/10;
- разрешение экрана 1024×768 и выше;
- акустическая система не требуется;
- дополнительные программные средства Adobe Acrobat Reader 5.0 и выше.

Редактор *М. А. Болдырева*
Компьютерная верстка *Ю. П. Шелехиной*

Сводный темплан 2017 г.
Подписано к использованию 16.06.17.
Объем 2,92 Мб.

Издательство ОмГТУ.
644050, г. Омск, пр. Мира, 11; т. 23-02-12
Эл. почта: info@omgtu.ru

ВВЕДЕНИЕ

Вычислительная физика – это направление в современной физике, занимающееся исследованием физических задач, допускающих решение с помощью численных методов. Использование численных методов продиктовано в основном двумя факторами. Во-первых, не все задачи допускают аналитическое решение. В качестве примеров можно привести задачи квантовой механики, задачи, требующие учета движения большого числа частиц, задачи расчета температурных полей в телах сложной формы и ряд других задач. Во-вторых, использование численных методов взамен аналитических может быть единственно возможным вариантом при анализе первичных экспериментальных данных, когда теоретические закономерности еще не установлены.

Одни из первых алгоритмов численного решения физических задач были разработаны Ньютоном, однако стремительное развитие численных методов решения приходится на вторую половину XX века. В это время, с одной стороны, выросла потребность в расчетах с высокой степенью точности (космонавтика, атомная энергетика), с другой – появились вычислительные машины, позволяющие автоматизировать процесс расчета.

В настоящее время можно выделить несколько тенденций в развитии вычислительной физики:

– разработка алгоритмов решения задач на основе многопроцессорных систем (параллельные вычисления). Подход позволяет значительно сократить время расчета и получить решения для задач, не решаемых ранее численными методами из-за большой длительности расчета;

– использование численных методов для получения аналитических решений. В качестве примера можно привести системы символьной математики MAPLE и MAXIMA, позволяющие получать решение в аналитическом виде;

– развитие и применение систем искусственного интеллекта.

С развитием последних двух направлений грань между аналитическими и численными методами стала стираться и в ближайшие десятилетия может исчезнуть.

Существует большое число программных продуктов, позволяющих решать задачи вычислительной физики. Часть программ распространяется

свободно, часть является проприетарными и требует покупки лицензии. Отдать предпочтение какому-либо одному пакету достаточно сложно – многие пакеты ориентированы на решение специфического круга задач. Например, SolidWorks и Salome-Meca ориентированы на создание 3D-моделей твердых тел с последующим расчетом нагрузок в них, также эти пакеты позволяют рассчитать потоки тепла, жидкостей и газов. Ansys позволяет рассчитывать динамику процессов в сложных композитных системах, Materials Studio – прогнозировать свойства материалов на основе молекулярно-атомных взаимодействий.

В первой части конспекта лекций рассмотрено применение пакета MATLAB и его свободного аналога GNU Octave для решения задач вычислительной физики. Пакеты обладают модульной структурой (что позволяет при необходимости расширять класс решаемых задач) и хорошим набором средств визуализации, упрощающим анализ получаемых решений. Общие вопросы применения MATLAB и GNU Octave рассмотрены только в необходимом минимуме, основной акцент сделан на практическое применение пакетов к решаемым задачам. Для лучшего понимания работы программ приводится листинг выполнения команд.

ЛЕКЦИЯ 1. ЗНАКОМСТВО С РАБОЧИМИ СРЕДАМИ MATLAB И GNU OCTAVE

Графический интерфейс

При запуске программы активируется ее графическое окно. Внешний вид окон представлен на рис. 1 (графическое окно MATLAB) и рис. 2 (графическое окно GNU Octave).

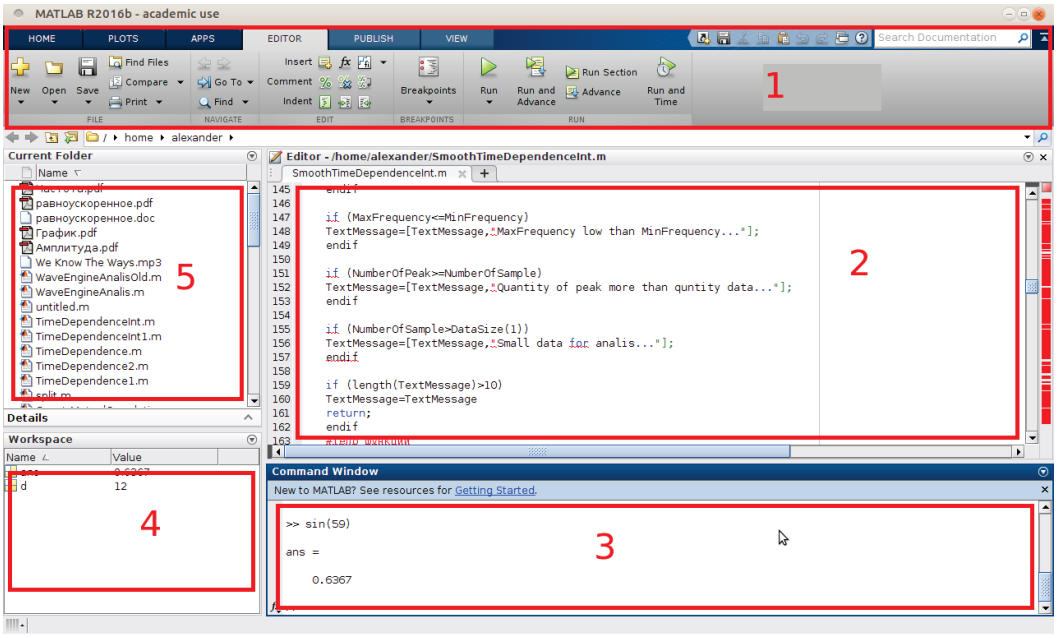


Рис. 1. Графическое окно MATLAB

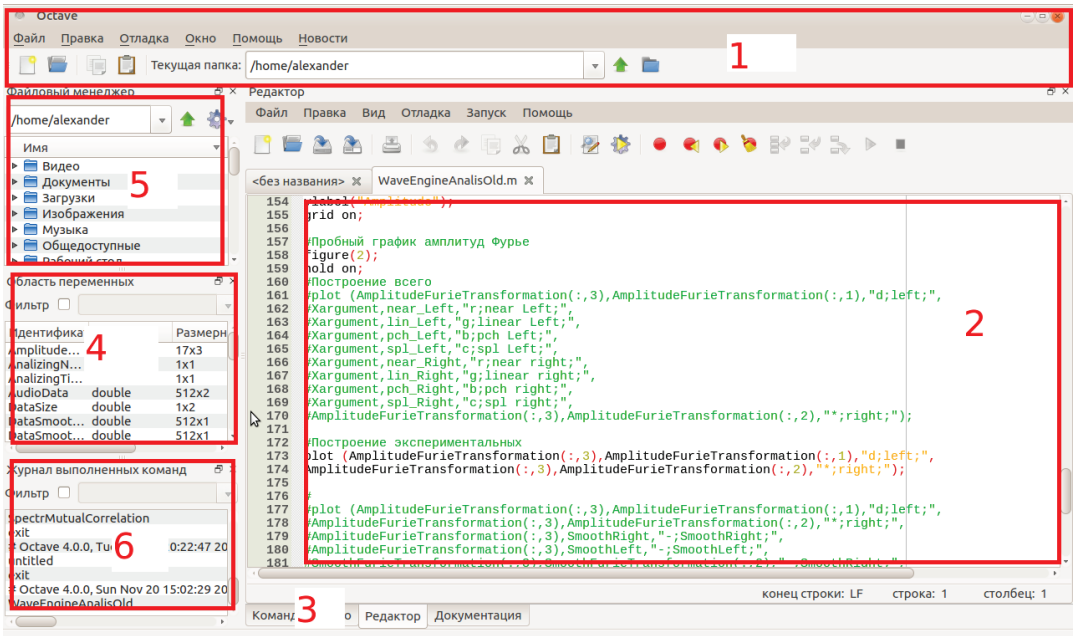


Рис. 2. Графическое окно GNU Octave

Несмотря на некоторые отличия, обе программные среды предоставляют пользователю близкие по своей структуре и функциональности интерфейсы. Перечислим основные из них:

1 – область главного меню. Позволяет получить доступ ко всем возможным командам и интерфейсам;

2 – область редактора. Позволяет вносить необходимые изменения в программы, созданные пользователем или импортированные им из внешних источников. Следует знать, что программа – это просто текстовый файл, содержащий команды для среды MATLAB или OCTAVE. Содержащиеся в файлах команды последовательно передаются на исполнение системой через командное окно;

3 – командное окно. Позволяет непосредственно ввести команду для MATLAB или OCTAVE. При этом введенная команда сразу же будет выполнена системой. Командное окно – это, пожалуй, самая главная часть взаимодействия системы и пользователя. Собственно через него и отдаются все команды системе. Графические интерфейсы представляют собой программные надстройки над командным окном, они автоматизируют процесс передачи команд системе и облегчают взаимодействие пользователя с системой. Графические интерфейсы, тем не менее, не являются обязательными – и MATLAB и GNU OCTAVE могут быть запущены в консольном режиме без графического интерфейса. Пользователю в этом случае придется непосредственно набирать в терминале все команды, что, конечно, не удобно, но позволяет значительно сократить нагрузку на процессор и использовать больший объем оперативной памяти (может быть важно при решении некоторых вычислительных задач);

4 – область переменных. В этой области пользователь может следить за значением и типом данных переменных. Такой контроль позволяет контролировать правильность расчета;

5 – файловый менеджер;

6 – журнал выполненных команд. Содержит список команд и информацию об их выполнении.

При необходимости пользователь может закрывать ненужные ему окна либо открывать новые (например, диалоговые окна для построения графиков, выполнения интерполяции и т. д.).

Основы синтаксиса языка

Основной конструкцией, с которой работают MATLAB и GNU OCTAVE, является матрица. Это не означает, что нельзя выполнять операции с обычными (или комплексными) числами, просто нужно понимать, что такому числу соответствует матрица размерности 1X1. Рассмотрим основные математические операции.

Операция присваивания «=» передает значение выражения в левой части равенства переменной, стоящей справа. Рассмотрим несколько примеров:

A=5;

Результат выполнения команды НЕ БУДЕТ отражен в командном окне, так как в конце выражения стоит оператор отключения вывода «;». При вводе данного выражения в командную строку переменной *A* будет присвоено значение 5. Тем не менее *A* с точки зрения программного интерпретатора является матрицей размером 1X1. Убедиться в этом можно, дав команду

size(A)

ans =

1 1

В результате выполнения команды будет определена размерность матрицы *A*.

Возможно выполнение операции присваивания сразу к матрицам:

B=[1,2,3]

B =

1 2 3

Команда формирует вектор-строку (матрицу размерности 1X3). При введении данных через «;» будет сформирован вектор-столбец.

C=[4;5;6]

C =

4

5

6

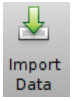
Для формирования матрицы целиком необходимо ввести команду

D=[1,2,3;4,5,6]

D =

1 2 3

4 5 6

Другим способом формирования матрицы является чтение из файла с помощью команды *load*. MATLAB предоставляет дополнительные инструменты для импорта данных из файлов, активируемые нажатием кнопки 

вкладки «HOME» главного меню.

Если значения некоторой величины меняются с равным шагом, то оператор присваивания может быть использован совместно с оператором «:», определяющим пределы изменения величины. В самом простом случае шаблон команды будет выглядеть следующим образом:

ПЕРЕМЕННАЯ=НАЧАЛЬНОЕ ЗНАЧЕНИЕ:КОНЕЧНОЕ ЗНАЧЕНИЕ;

при этом шаг изменения будет равным 1. В более развернутом варианте команда имеет вид:

ПЕРЕМЕННАЯ=НАЧАЛЬНОЕ ЗНАЧЕНИЕ:ШАГ:КОНЕЧНОЕ ЗНАЧЕНИЕ.

При этом шаг изменения можно задать самому. Рассмотрим пример:

E=1:4:17

E =

1 5 9 13 17.

Оператор «:» имеет еще одно применение – он может определять совокупность строк или столбцов в матрице. Например, команда

D(1,:)=100

D =

100 100 100

4 5 6

присвоит значение 100 ВСЕМ элементам первой строки матрицы.

Команда

D(:,2)=200

D =

100 200 100

4 200 6

присвоит значение 200 ВСЕМ элементам второго столбца матрицы.

MATLAB и GNU OCTAVE допускают переназначение переменных не только на уровне значений, но и на уровне типов данных. Например, вещественной переменной можно присвоить символьное значение – при такой операции автоматически произойдет смена ее типа. Интерпретаторы команд отслеживают только согласование размерности в выражении (умножение символьной строки на матрицу будет вызывать ошибку) [1, 2].

Пример: действие оператора присваивания:

A=[5,6;7,8]

A='строка'

B=[1,2;3,4]

*A*B*

A =

5 6

7 8

A =

строка

B =

1 2

3 4

*Error using **

Inner matrix dimensions must agree.

Про «матричность» логики нужно помнить при выполнении математических операций. Например, невозможно будет выполнить сложение или вычитание для матриц разной размерности. Требования к размерности матриц также ограничивают применение операций умножения.

Для удобства пользователей некоторые математические операции могут быть также выполнены поэлементно. Указание на поэлементное выполнение дается добавлением знака «.» к оператору.

Пример: матричное и поэлементное умножение:

```
A=[1,2;3,4]
B=A*A
C=A.*A

A =

     1     2
     3     4


B =

     7    10
    15    22


C =

     1     4
     9    16
```

В первом случае матрица *A* была умножена сама на себя (выполнено матричное умножение), во втором случае сам на себя был умножен каждый элемент матрицы. При применении тригонометрических и иных операций к матрице они также будут выполнены поэлементно [2].

Построение графиков с помощью MATLAB и GNU Octave

Для визуализации входных данных и полученных результатов MATLAB и GNU Octave располагают обширными библиотеками графических построений. Возможно построение как двумерных, так и трехмерных графиков.

Основной функцией для построения двумерных графиков служит функция *plot*. У функции несколько вариантов вызова, рассмотрим их:

– *plot(X,Y)* – в данном случае будет построен график зависимости y(x). Значения y и x берутся из матриц Y и X, которые могут быть либо вектором-столбцом, либо вектором-строкой одинаковой размерности;

– *plot(X1,Y1,...,Xn,Yn)* – будут одновременно построены несколько функциональных зависимостей y(x), при этом параметры линий на графике будут выбраны MATLAB и GNU Octave самостоятельно;

– *plot(X,Y,LineSpec), plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn)* – наиболее полный вариант вызова функции построения двумерных графиков с заданием параметров графических линий. *LineSpec* – это шаблон, с помощью которого определяется цвет линии, ее толщина, вид маркеров и другие параметры. Шаблон представляет собой взятое в апострофы название параметра, отделенное запятой от его значения. Подробное описание параметров и значений можно посмотреть в документации к MATLAB и GNU Octave, ниже приводится таблица с основными параметрами.

Таблица 1

Параметр	Возможные значения
'Color' – определяет цвет линии	'y' – желтый 'm' – магента (пурпурный) 'c' – циан (зелено-голубой) 'r' – красный 'g' – зеленый 'b' – голубой 'k' – черный [r g b] – цвет в формате RGB (параметры r, g, b лежат в пределах от 0 до 1)
'LineStyle' – вид линии	'-' – сплошная '- -' – двойная сплошная ':' – пунктирная '-.' – штрихпунктирная 'none' – без линии (будут оставлены только маркеры)

Окончание табл. 1

Параметр	Возможные значения
'LineWidth' – ширина линии	Положительные значения с шагом 0.5
'Marker' – вид маркера	'o' – круговой '+' – знак «+» 's' – квадрат 'p' – пятиугольник '^' – ориентированный вверх треугольник

Пример: построение графика *sin(x)*:

*x=0:0.1:2*pi;*

plot(x,sin(x),'Color','m','LineStyle','-.','LineWidth',4);

Выполнение данной последовательности команд приведет к построению графика синуса штрихпунктирной линией четвертой толщины цвета «магента» (рис. 3).

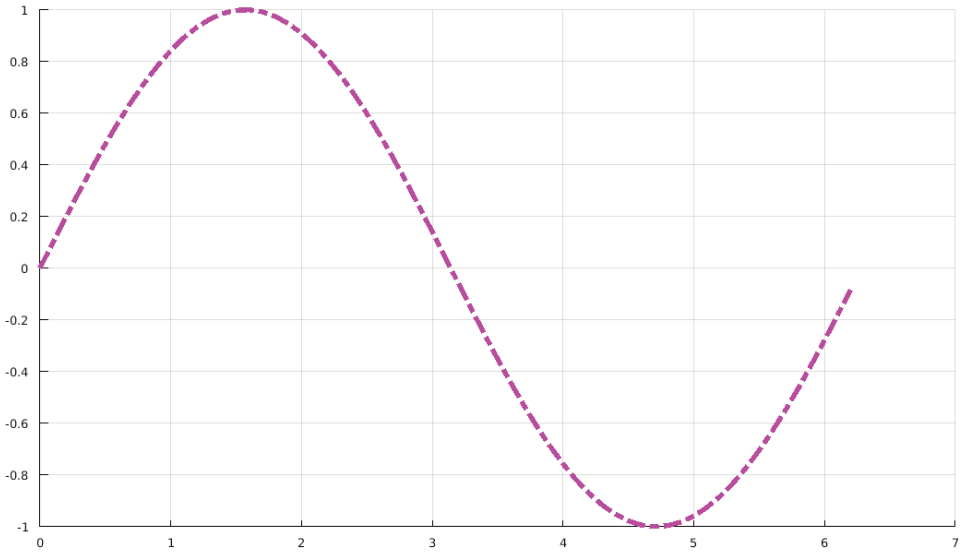


Рис. 3. График функции sin(x)

Если применить еще одну команду *plot*, то после ее выполнения график новой функции будет нарисован поверх предыдущего и затрет его. Данная особенность связана не с работой функции *plot*, а с работой графического окна. Поэтому все функции, связанные с построением графиков, применяются обычно в паре с функциями, определяющими режим работы графического окна. Некоторые из них приведены в табл. 2.

Таблица 2

Функция	Варианты использования
figure – функция запроса графического окна. После применения все изображения будут перенаправляться в запрошенное окно.	*figure()* – вызывает графическое окно, номер которому будет присвоен системой автоматически. Возможен вызов в виде *N=figure()*, в этом случае переменной N будет присвоен номер графического окна. *figure(N)* – будет создано новое окно, окну будет присвоен номер N, если такого окна не существовало. Если окно с таким номером существовало, в него будет перенаправлен графический вывод.
hold – функция, включающая и отключающая сохранение в графическом окне предыдущих графиков.	*hold on* – новые графики будут изображаться совместно с предыдущими. *hold off* – новые графики будут изображаться поверх предыдущих, затирая их.
grid – функция, включающая и отключающая отображение линий сетки.	*grid on* – включает отображение линий сетки. *grid off* – выключает отображение линий сетки.
xlabel, ylabel – функции подписывания осей.	*xlabel('НАЗВАНИЕ ОСИ X')* *ylabel('НАЗВАНИЕ ОСИ Y')*
legend – функция нанесения на график легенды (сопровождающих и поясняющих надписей).	*legend(' СТРОКА 1', ' СТРОКА 2', ...)*

Пример: построение графиков *sin(x)* и *cos(x)*:

```
x=0:0.1:2*pi;
figure(1);
hold on;
plot(x,sin(x),'Color','m','LineStyle','-.','LineWidth',4)
plot(x,cos(x),'Color','g','LineStyle','--','LineWidth',3)
legend('sin(x)','cos(x)')
grid on;
xlabel('x');
ylabel('sin(x)');
```

Выполнение программного кода приведет к построению следующего графика:

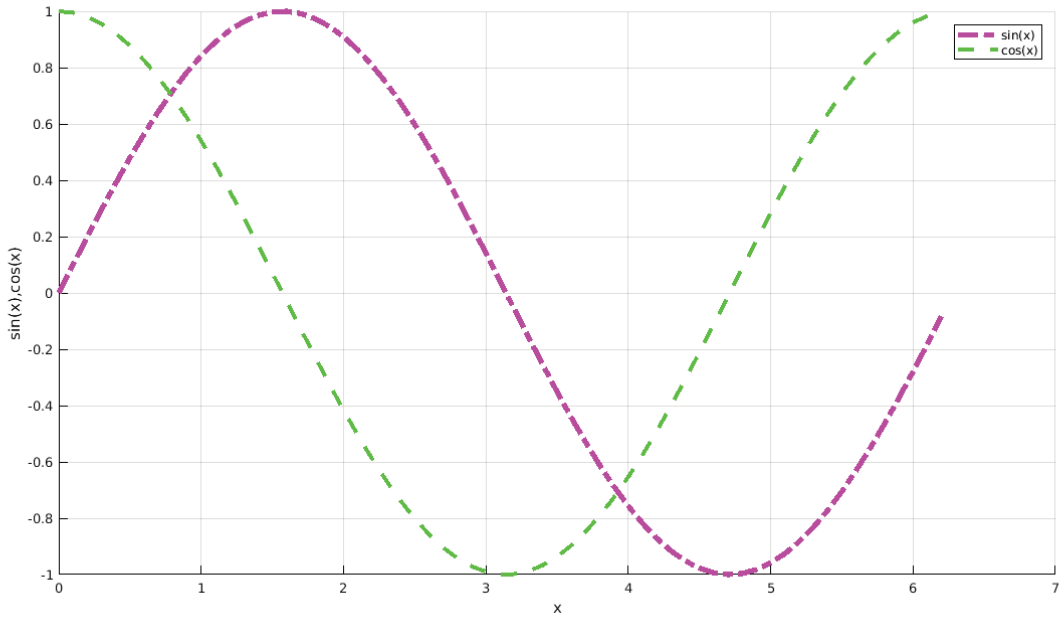


Рис. 4. Применение функции plot

Один из способов построения трехмерных графиков связан с использованием функции *surf*. Наиболее часто функция вызывается в формате *surf(X,Y, Z)* или в *surf(X, Y, Z, C)*. X и Y – векторы-строки, определяющие значения абсцисс и ординат. Z – матрица с размерностью, равной произведению размерностей матриц X и Y, задающая значения координаты z для соответствующих пар x и y. Параметр C определяет способ отображения трехмерной картинки (цвет, режим отображения кромок и т. д.).

Пример: построение трехмерной поверхности второго порядка:

```
x=-1:0.01:1;
y=-1:0.01:1;
for i=-1:0.01:1
   for j=-1:0.01:1
      z(round(i*100+101),round(j*100+101))=i^2+j^2;
   end
end
surf(x,y,z)
```

Используемые функции *round* округляют значения аргументов до целого. Результатом выполнения будет график, представленный на рис. 5.

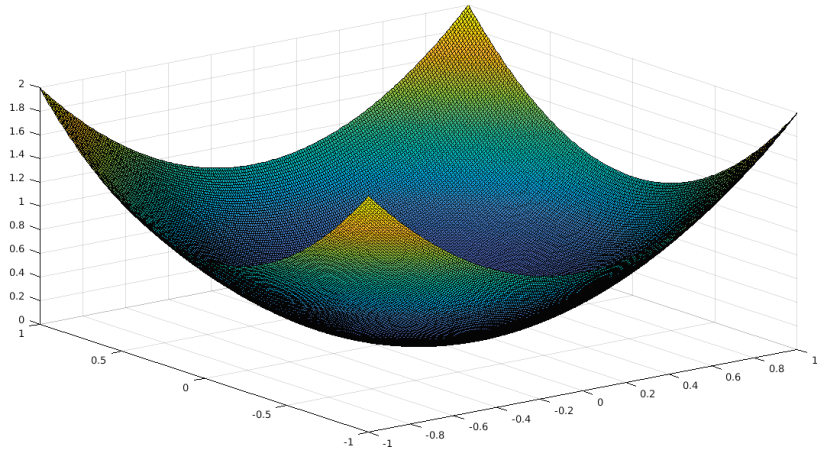


Рис. 5. Поверхность второго порядка

MATLAB и GNU Octave при формировании графического окна предоставляют набор инструментов для работы с полученным изображением. Сюда входит возможность вращения, работы с сетками, сохранение в одном из графических форматов и т. д. Отметим, что инструментарий, предоставляемый MATLAB, значительно богаче, его внешний вид приведен на рис. 6.

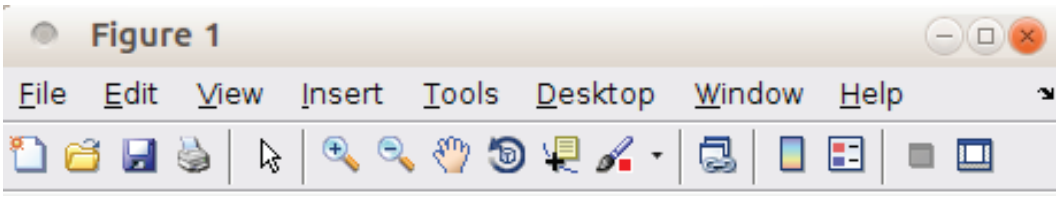


Рис. 6. Набор инструментов для работы с изображением

Перечисленные выше способы построения как двумерных, так и трехмерных графиков не являются единственно возможными. Как MATLAB, так и GNU Octave обладают богатейшими графическими библиотеками. Подробности можно посмотреть в руководствах пользователя или в справке.

Задания к лабораторному практикуму

1. По заданию преподавателя запустите MATLAB или GNU Octave.

2. Сформируйте вектор-строку, вектор-столбец, матрицу с элементами по заданию преподавателя.

3. Постройте график, отображающий одновременно три функции по выбору преподавателя. Добейтесь отображения маркеров.

4. Постройте трехмерное изображение поверхности второго порядка по заданию преподавателя. Сохраните полученное изображение в формате .jpg.

ЛЕКЦИЯ 2. ДВИЖЕНИЕ В ГРАВИТАЦИОННОМ ПОЛЕ

Движение при отсутствии сопротивления среды

Задачи о движении в гравитационном поле часто встречаются в прикладной физике (баллистика, воздухоплавание, космонавтика). Рассмотрим несколько вариантов постановки задачи и способы их решения в среде MATLAB и GNU Octave.

В самом простом случае можно пренебречь сопротивлением внешней среды и неинерциальностью системы отсчета. При этих предположениях траектория движения лежит в одной плоскости и для описания движения достаточно двух осей. Решение хорошо известно из школьного курса физики:

$$y = y_0 + V_0 \cdot sin(\alpha) \cdot t - \frac{g \cdot t^2}{2};$$

$$x = x_0 + V_0 \cdot cos(\alpha) \cdot t;$$

$$V_y = V_0 \cdot sin(\alpha) - g \cdot t;$$

$$V_x = V_0 \cdot cos(\alpha).$$

Время полета может быть найдено как положительный корень уравнения

$$y_0 + V_0 \cdot sin(\alpha) \cdot t + \frac{g \cdot t^2}{2} = 0.$$

В качестве параметров задаются x0, y0, V0, α.

Рассмотрим алгоритм решения задачи с помощью MATLAB и GNU Octave. Первое, что необходимо сделать, – это вычислить время полета, затем вычислить значения координат и скоростей за все время движения и построить графики.

Нахождение времени полета сводится к решению квадратного уравнения и выделению положительного корня. Для нахождения корней полинома используется функция *roots (c)*, принимающая в виде аргумента матрицу с коэффициентами полинома (первый коэффициент соответствует старшей степени). Функция возвращает значения корней полинома. С остальными функциями и их действием мы ознакомились на лекции 1.

Пример: движение в поле силы тяжести без учета сопротивления:

```
% Сброс всех переменных
clear all;

% Начальные условия
V0=10;
Alpha0=60;
X0=0;
Y0=0;
g=9.8;

% Поиск времени полета
% Задаем коэффициенты полинома
c=[-g/2, V0*sind(Alpha0), Y0];
% Ищем корни
FlyingTime= roots(c);
% Выделяем положительный корень
if FlyingTime(1)>0
  Time=FlyingTime(1);
else
  Time=FlyingTime(2);
end;
%Вычисляем зависимости величин от времени
t=0:(Time/100):Time;
X=X0+V0*cosd(Alpha0).*t;
Y=Y0+V0*sind(Alpha0).*t-g.*(t.^2)/2;
```

```
Vx=V0*cosd(Alpha0).*t./t;
Vy=V0*sind(Alpha0)-g.*t;
% Построение графиков
figure();
hold on;
grid on;
plot(t,X,'LineStyle','-.','LineWidth',3);
plot(t,Y,'LineStyle','--','LineWidth',3);
plot(t,Vx,'LineStyle',':','LineWidth',3);
plot(t,Vy,'LineStyle','-','LineWidth',3);
legend('X','Y','Vx','Vy');
```

Выполнение программного кода приведет к построению графика зависимости кинематических величин от времени.

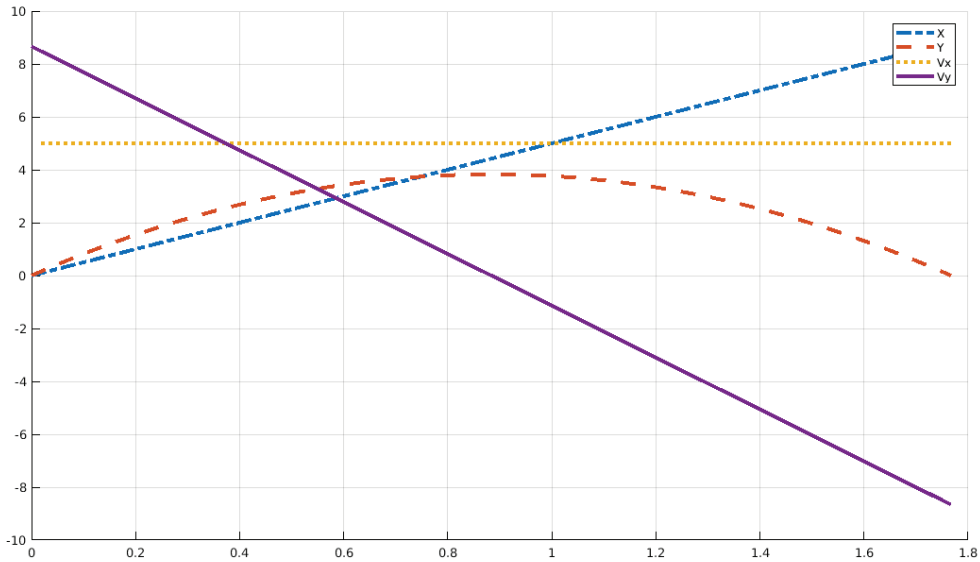


Рис. 7. Зависимость кинематических величин от времени

Движение при наличии сопротивления среды

Рассмотрим движение тела в среде с силой сопротивления, пропорциональной скорости движения.

$$\vec{F_c} = -\beta\vec{V}.$$

В этом случае уравнения движения имеют вид

$$\frac{d^2x}{dt^2} + \frac{\beta}{m} \cdot \frac{dx}{dt} = 0, \frac{d^2y}{dt^2} + \frac{\beta}{m} \cdot \frac{dy}{dt} = -g \ .$$

Представленная система обыкновенных дифференциальных уравнений второго порядка может быть проинтегрирована и решена аналитически, но возможно и численное решение.

Для численного решения обыкновенных уравнений существует много методов, наиболее известны из них метод Эйлера, метод Адамса, семейство методов Рунге – Кутты [3]. Отличие между методами заключается в разных способах конечно-разностного представления производной и соответственно разной степени точности получаемых расчетов. В MATLAB численное решение дифференциальных уравнений реализовано следующим набором функций (полный список функций можно посмотреть в руководстве):

Таблица 3

Нестационарные дифференциальные уравнения	
ode45	метод среднего порядка точности
ode23	метод низкого порядка точности
ode113	метод переменного порядка точности
Стационарные дифференциальные уравнения	
ode15s	Решение стационарных дифференциальных уравнений и дифференциально-алгебраических уравнений переменного порядка точности.
ode23s	Решение стационарных дифференциальных уравнений низкого порядка точности.
ode23t	Решение стационарных дифференциальных уравнений и дифференциально-алгебраических с использованием формулы трапеции.
ode23tb	Решение стационарных дифференциальных уравнений и дифференциально-алгебраических с использованием формулы трапеции и обратного дифференцирования.

В простейшем случае функции имеют следующий шаблон вызова:

[переменная, функция] = ode45(символьная производная функции, диапазон изменения переменной, начальные условия).

При использовании MATLAB нужно помнить, что система способна решать ОДУ только первого порядка. Поэтому уравнения более высокого порядка должны быть сначала приведены к системе уравнений первого порядка. В нашем случае

$$\frac{dX}{dt} = V_x, \qquad \frac{dV_x}{dt} = \frac{-\beta}{m} \cdot V_x = 0, \qquad \frac{dY}{dt} = V_y, \qquad \frac{dV_y}{dt} = \frac{-\beta}{m} \cdot V_y - g.$$

Рассмотрим алгоритм решения.

Пример: численное решение системы ОДУ второго порядка:

```
% Сброс всех переменных
clear all;

% Начальные условия
V0=10;
Alpha0=60;
Beta0=1;
Mass=1;
X0=0;
Y0=0;
g=9.8;

% Формируем начальные данные для системы ОДУ
Condition=[0; V0*cosd(Alpha0);0;V0*sind(Alpha0)];

% Задаем промежуток времени
tspan=[0 2];

% Решаем систему
[t,R] = ode45(@MyDiffEquation, tspan, Condition);

%Построение графиков
figure();
hold on;
plot(t,R, 'LineWidth',2);
legend ('X(t)','Vx(t)','Y(t)','Vy(t)');
```

```
figure();
hold on;
plot(R(:,1),R(:,3), 'LineWidth',2);
legend ('Y(X)');

% Формируем систему уравнений
% R(1) — соответствует x(t),
% R(2) – соответствует Vx=dx/dt.
% R(3) – соответствует y(t),
% R(4) – соответствует Vy=dy/dt.
function dRdt=MyDiffEquation(t,R)
V0=10;
Alpha0=60;
Beta=1;
Mass=1;
X0=0;
Y0=0;
g=9.8;
dRdt=[R(2); -R(2)*Beta/Mass; R(4); -R(4)*Beta/Mass-g];
end
```

Функция *MyDiffEquation(t,R)* получает в качестве аргументов названия переменной и системы дифференциальных уравнений и возвращает символьные выражения для производных функций системы. В соответствии с правилами MATLAB функции, созданные пользователем, размещаются в конце программного кода.

Сформированные выражения передаются для решения функции *ode45*. Символ @, стоящий перед первым аргументом (функцией *MyDiffEquation*), говорит о том, что будут переданы не численные значения, а аналитическая формула или алгоритм вычисления величины. Одновременно функции *ode45* передаются диапазон изменения переменной и начальные значения. В результате применения функции *ode45* формируется массив данных, столбцы которого содержат зависимость от времени для искомых функций.

В результате выполнения программы будут построены графики зависимости кинематических величин от времени и график траектории движения (рис. 8 и 9).

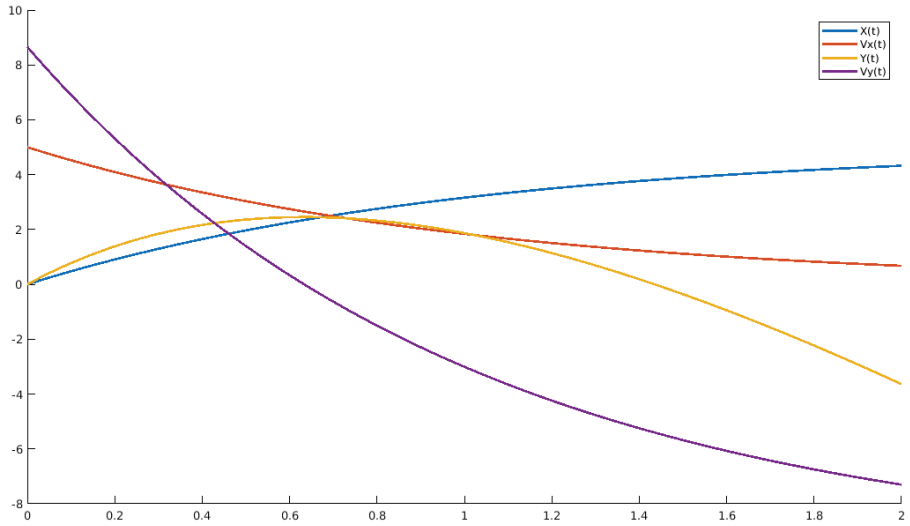


Рис. 8. Зависимость кинематических величин от времени

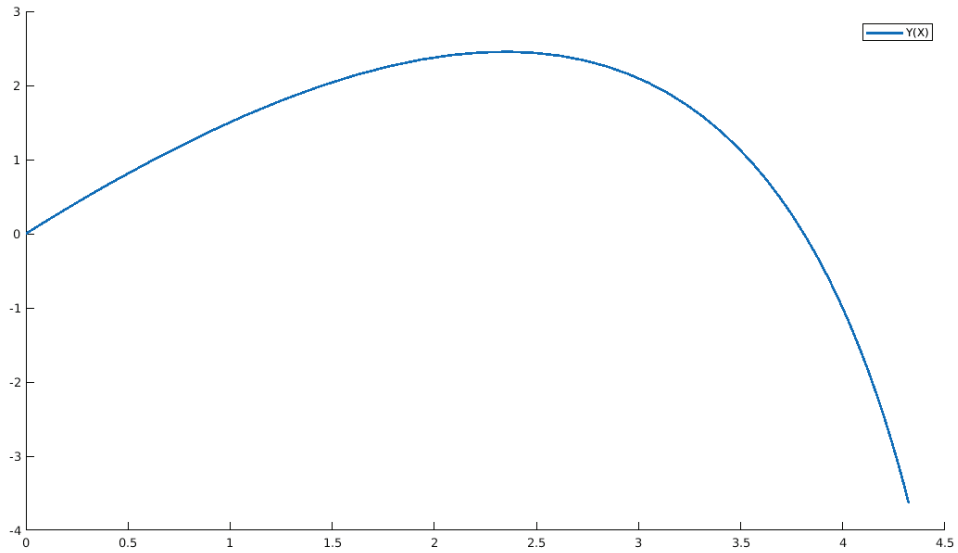


Рис. 9. Траектория движения

В GNU Octave для решения ОДУ используется функция *lsode ( )*. Шаблон вызова функции имеет вид:

[x, istate, msg]=lsode (символьная производная функции, начальные условия, диапазон изменения переменной).

Массив данных с результатами численного решения будет передан переменной *x*, переменная *istate* содержит код правильности выполнения

(по умолчанию 2), переменная *msg* содержит дополнительное сообщение системы о процессе решения. При использовании функции *lsode* GNU Octave самостоятельно подберет алгоритм решения, при желании пользователь может сделать это самостоятельно с помощью функции *lsode_options ()*. Можно определять метод решения, параметры сходимости, шаг и др. При использовании специализированного пакета *odepkg* появляется дополнительный набор специализированных функций.

Задания к лабораторному практикуму

1. По заданию преподавателя запустите MATLAB или GNU Octave.

2. Напишите программу для построения траектории движения тела, брошенного под углом к горизонту в среде без сопротивления.

3. Решите численно систему ОДУ, описывающих одномерное движение тела в среде с сопротивлением. По заданию преподавателя проведите расчет для нескольких вариантов силы сопротивления.

4. Постройте графические зависимости кинематических характеристик от времени, экспортируйте полученные графики в формат .pdf.

ЛЕКЦИЯ 3. КОЛЕБАТЕЛЬНЫЕ СИСТЕМЫ

Свободные незатухающие колебания

Исследование колебательных систем, пожалуй, самая распространенная задача прикладной вычислительной физики. С моделированием колебаний можно встретиться в самых различных областях: устойчивость механических систем к внешним нагрузкам, сейсмо- и гидролокация, радиоэлектронная аппаратура, квантовые системы и многие другие разделы прямо или косвенно связаны с рассмотрением колебательных систем. Рассмотрим методы исследования колебательных систем в среде MATLAB и GNU Octave.

Для простейших колебательных систем можно пренебречь потерями энергии, в этом случае колебания будут незатухающими. Примером таких

колебаний могут быть колебания пружинного маятника, для которого дифференциальное уравнение имеет вид:

$$\frac{d^2x}{dt^2} + \frac{k}{m} \cdot x = 0.$$

Вторым часто встречающимся примером, знакомым из курса общей физики, являются свободные незатухающие электромагнитные колебания, описываемые дифференциальным уравнением

$$\frac{d^2q}{dt^2} + \frac{1}{LC} q = 0.$$

Оба уравнения – это варианты известного дифференциального уравнения

$$\frac{d^2x}{dt^2} + {\omega_0}^2 \cdot x = 0.$$

Уравнение допускает аналитическое решение. В настоящее время многие программные математические пакеты способны получать аналитическое решение. Эталонными в этой области можно считать Maple и Maxima – программы, изначально создаваемые для символьных вычислений. MATLAB имеет в своем составе пакет Symbolic Math Toolbox для проведения символьных операций (также предоставляет возможность взаимодействия с символьным процессором программы Maple). Попытаемся найти аналитическое решение уравнения с помощью MATLAB.

Прежде всего с помощью оператора *syms* системе нужно указать, с какими объектами необходимо проводить символьные вычисления (аргументы перечисляются через пробел).

```
% Указываем системе на символьные операции с функцией x(t) и переменной
% Omega0
syms x(t) Omega0
```

Затем определяем уравнение (или систему уравнений).

```
equation = diff(x,t,2) == -Omega0^2*x;
```

В данном случае первый знак «=» указывает на то, что формируется уравнение, второй сдвоенный знак равенства «==» является частью самого уравнения. Оператор *diff* – это оператор дифференцирования, он определяет вторую производную от переменной *x* по времени *t.* Решение дифференциального уравнения осуществляется с помощью функции *dsolve.* В самом простом случае она вызывается с одним аргументом, которым является решаемое уравнение.

```
% Решение дифференциального уравнения
dsolve(equation)
```

Результатом выполнения станет
*C1*exp(Omega0*t*1i) + C2*exp(-Omega0*t*1i).*

Как видно, MATLAB нашел общее аналитическое решение системы в виде суперпозиции двух комплексных экспонент. Попытаемся уточнить решение, найдем его для случая, когда маятник запускается из крайнего положения с нулевой начальной скоростью. В этом случае начальные условия имеют вид:

$$X(0) = X_{MAX}, X'(0) = 0.$$

Дополним код соответствующими строками:

```
% Указываем системе на символьные операции с функцией x(t) и переменными
% Omega0, Xmax

syms x(t) Omega0 Xmax

%Дополнительно указываем, что переменные вещественны
assume(Omega0,'real')
assume(Xmax,'real')

%Определяем уравнение
equation = diff(x,t,2) == -Omega0^2*x;
```

```
%Добавляем начальные условия
FirstDiff = diff(x,t);
cond = [x(0)==Xmax, FirstDiff(0)==0];
% Решение дифференциального уравнения
dsolve(equation,cond )
```

После выполнения программы получим

```
ans =

Xmax*cos(Omega0*t)
```

Как видно, система великолепно справилась с аналитическим решением.

Затухающие колебания

При наличии в системе диссипативных процессов амплитуда колебаний будет уменьшаться. Физически это связано разными факторами – с наличием трения для механических систем, выделением теплоты на активном сопротивлении для электромагнитных колебаний, дисперсией в среде для волновых процессов. Однако на уровне математики большинство указанных процессов приводят к следующему дифференциальному уравнению:

$$\frac{d^2x}{dt^2} + \beta \cdot \frac{dx}{dt} + {\omega_0}^2 \cdot x = 0.$$

Небольшое изменение программного кода позволяет решить и это уравнение.

```
% Указываем системе на символьные операции с функцией x(t) и переменными
% Omega0, Xmax

syms x(t) Omega0 Xmax Beta

%Дополнительно указываем, что переменные вещественны и затухание мало
```

```
assume(Omega0,'real')
assume(Xmax,'real')
assume(in(Beta,'real')& Beta<Omega0)

%Определяем уравнение
equation = diff(x,t,2) == -Omega0^2*x-Beta*diff(x,t);

%Добавляем начальные условия
FirstDiff = diff(x,t);
cond = [x(0)==Xmax, FirstDiff(0)==0];
% Решение дифференциального уравнения
dsolve(equation,cond )
simplify(ans)
```

В результате получим аналитическое решение

ans =

*(Xmax*exp(-(t*(Beta - (Beta^2 - 4*Omega0^2)^(1/2)))/2)*(Beta + (Beta^2 - 4*Omega0^2)^(1/2)))/(2*(Beta^2 - 4*Omega0^2)^(1/2)) - (Xmax*exp(-(t*(Beta + (Beta^2 - 4*Omega0^2)^(1/2)))/2)*(Beta - (Beta^2 - 4*Omega0^2)^(1/2)))/(2*(Beta^2 — 4*Omega0^2)^(1/2)).*

Как видно, полученное компьютером решение даже после применения оператора упрощения *simplify* мало напоминает известное из курса физики

$$X = X_0 \cdot e^{-\beta \cdot t} \cdot sin(\omega \cdot t + \phi_0).$$

Пример иллюстрирует достаточно типичную ситуацию для систем компьютерной алгебры: получаемые компьютером аналитические решения не всегда понятны человеку. Тем не менее полученные выражения пригодны как для дальнейших аналитических преобразований, так и для расчета.

Непосредственное вычисление с помощью полученного выражения невозможно, так как полученный результат является символьным выражением (выражение имеет тип *«sym»*, не путать с типом *«string»*).

Перед расчетом его нужно преобразовать в функцию, что делается с помощью оператора *matlabFunction()*. Аргументом в данном случае выступает символьное выражение, возвращается указатель на функцию (*function_handle*). Полученный указатель используется как имя функции в последующих вычислениях. Модифицируем последнюю часть программного кода:

```
% Решение дифференциального уравнения
% и создание указателя на функцию
DampedOscillation = matlabFunction(dsolve(equation,cond ))
```

В результате получим:

```
DampedOscillation =

  function_handle with value:

    @(Beta,Omega0,Xmax,t)…..
```

Сообщение системы говорит, что *DampedOscillation* будет использоваться как указатель на функцию, которой нужно будет для вычисления передать четыре аргумента *DampedOscillation(Beta,Omega0,Xmax,t)*. Вместо символа троеточия в листинге будет приведено явное выражение функции (его можно увидеть в предыдущем листинге). Полученное выражение можно использовать, например, для построения графиков (график приведен на рис. 10).

```
%Задание численных значений частоты, амплитуды
%коэффициента затухания и временного диапазона
Omega0=10;
Xmax=5;
Beta=1;
t=1:0.01:10;
%Построение графиков
```

```
figure()
hold on;
grid on;
plot (t,DampedOscillation(Beta,Omega0,Xmax,t),'LineWidth',3);
xlabel('t');
ylabel('X');
legend ('Затухающие колебания');
```

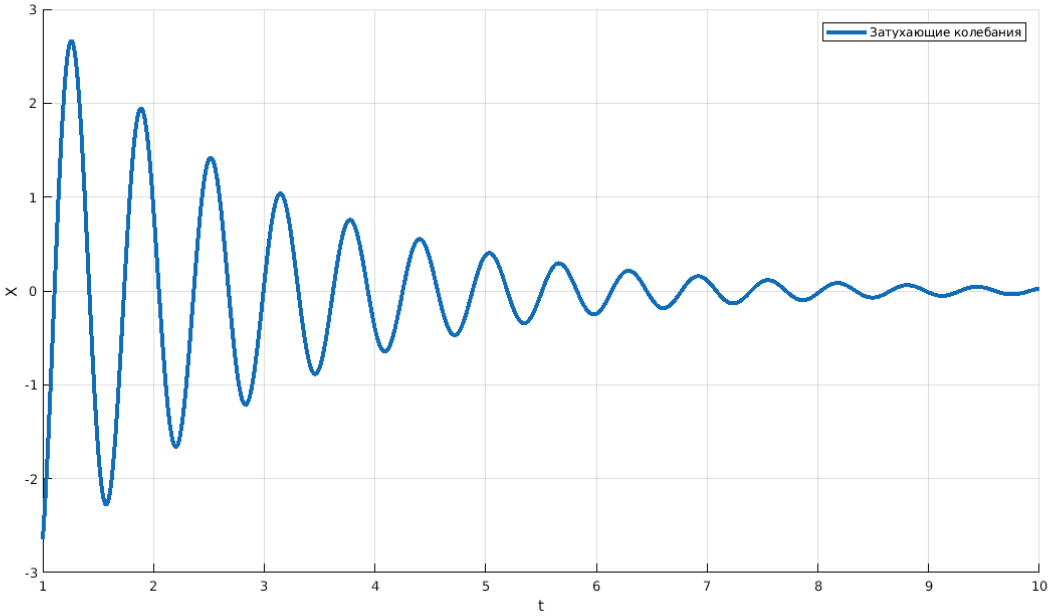


Рис. 10. Затухающие колебания

Начерченный график является классическим для затухающих колебаний. Как видно, несмотря на сложность записи решения, предлагаемого системой, оно вполне адекватно описывает колебательную систему.

При решении с использованием символьной математики важно помнить, что не все уравнения имеют аналитическое решение. В этом случае MATLAB выдаст предупреждение:

```
Warning: Explicit solution could not be found.

ans =
[ empty sym ]
```

Возможность численного решения при этом остается (см. лекцию 2).

Имитационное моделирование вынужденных колебаний с помощью SIMULINK

Рассмотрим моделирование процесса вынужденных колебаний с помощью имитационного моделирования. В MATLAB имитационное моделирование реализовано с помощью графической среды SIMULINK. GNU Octave не предоставляет пользователю такой возможности, но среди свободного программного обеспечения можно выделить пакет Scilab, обладающий схожими возможностями.

SIMULINK интегрирован в среду MATLAB и позволяет «использовать уже готовые библиотеки блоков для моделирования электросиловых, механических и гидравлических систем, а также применять развитый модельно-ориентированный подход при разработке систем управления, средств цифровой связи и устройств реального времени» [4].

На рис. 11 приведена схема для наблюдения вынужденных электрических колебаний. Рассмотрим на ее примере основные принципы работы с SIMULINK.

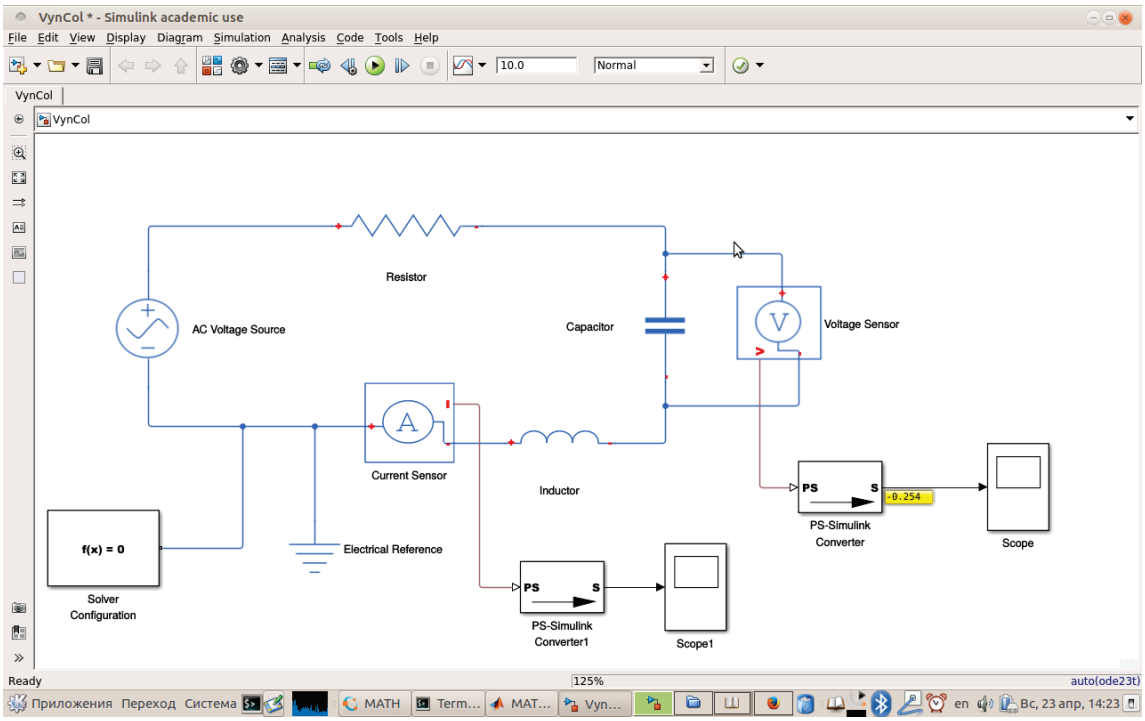


Рис. 11. Моделирование вынужденных электрических колебаний

В верхней части окна находится главное меню SIMULINK, позволяющее открывать, сохранять файлы, импортировать их в программный код

MATLAB, настраивать параметры и осуществлять запуск процесса моделирования.

В рабочей области представлена электрическая схема, сборка которой осуществляется путем добавления блоков необходимых элементов из библиотеки. Доступ к библиотеке осуществляется с помощью браузера (*Simulink Library Browser*), вызываемого нажатием кнопки в главном меню программы. На рис. 12 приведен вид браузера библиотеки.

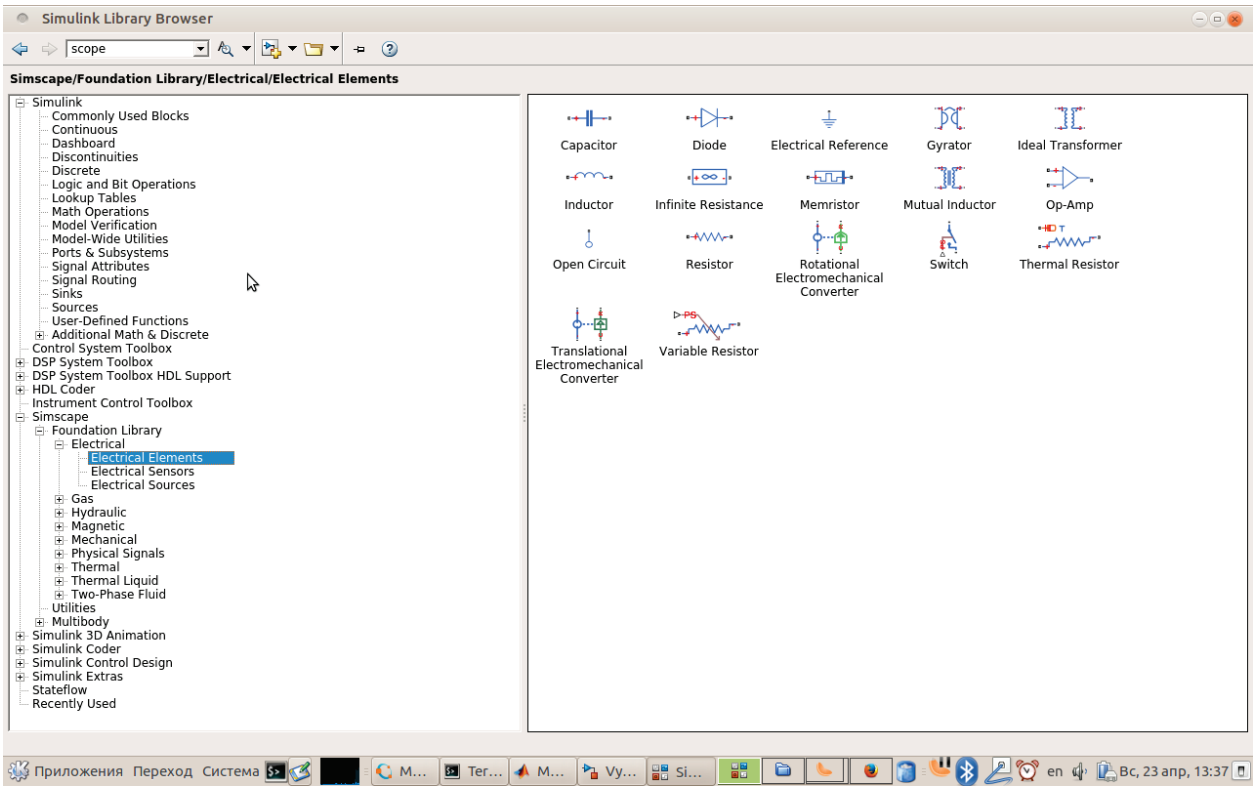


Рис. 12. Окно браузера библиотеки

Библиотека содержит набор типовых блоков для моделирования обширного класса задач: электромагнитных, механических, задач, связанных с течением газа и жидкости, температурных полей и др. Добавление блока в рабочую область осуществляется путем перетаскивания его с зажатой левой клавишей мыши. При необходимости пользователь может создавать свои блоки.

Приведенная на рис. 11 схема представляет собой последовательную RLC-цепь, в которую для контроля включены средства измерения и отображения (амперметр, вольтметр и осциллографы).

При построении электрической схемы нужно принять во внимание несколько факторов:

– в электрической схеме обязательно наличие заземления, в противном случае не понятно, где находится нулевой потенциал, и невозможно выполнить расчет;

– виртуальные электрические измерители генерируют на выходе размерный сигнал. Виртуальный осциллограф отображает зависимость от времени безразмерной величины. Для соединения измерителя и осциллографа необходимо использовать преобразователь *(converter)* размерной величины в безразмерную;

– к системе должен быть подключен решатель *(solver)*. Решатели – это наборы алгоритмов, вычисляющие динамику системы в течение некоторого времени [4].

После выполнения моделирования можно посмотреть полученный результат в окне осциллографа (рис. 13).

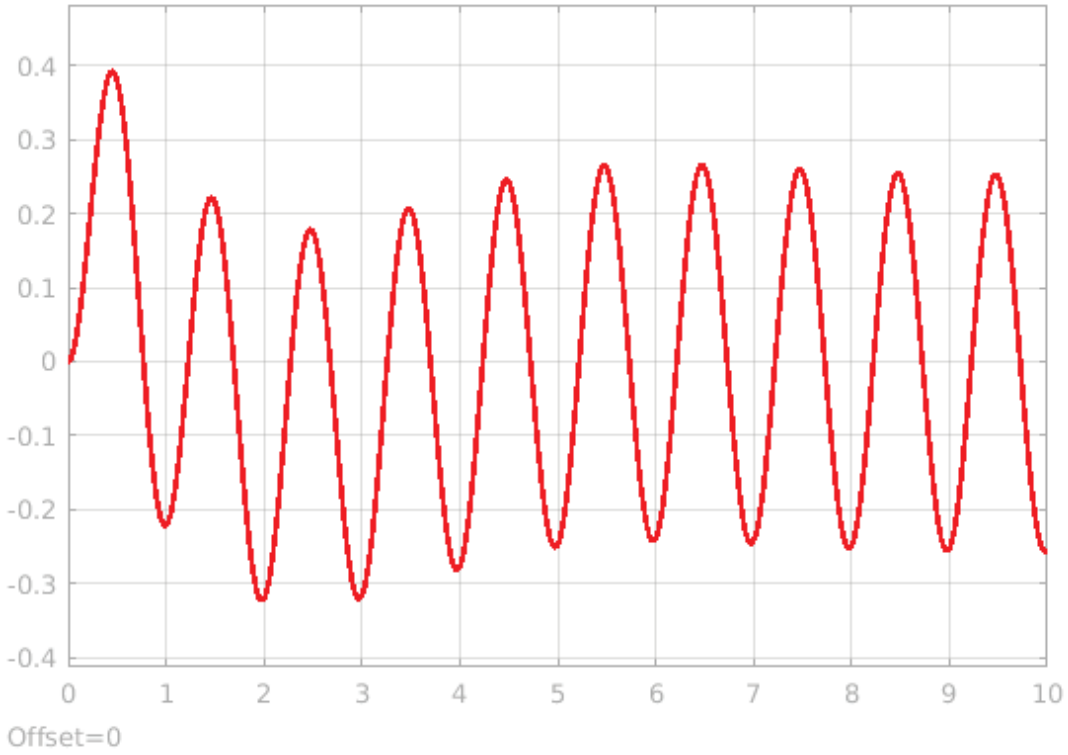


Рис. 13. Вынужденные колебания

Осциллограмма напряжения на конденсаторе показывает переходной процесс от собственных колебаний к вынужденным, длящийся около пяти секунд, и установившиеся вынужденные колебания.

Рассмотренный пример моделирования вынужденных колебаний иллюстрирует самый простой вариант использования имитационного моделирования. Однако наибольшую эффективность можно получить, используя SIMULINK для построения и анализа математической модели системы. Сборка математической модели с помощью блочного конструктора принципиально ничем не отличается от сборки электрической схемы, но возможности анализа значительно выше [5].

Нелинейные колебательные системы

Колебательная система с линейным откликом – это наиболее простая и часто встречающаяся модель, но существуют и более сложные системы, хорошо поддающиеся численному моделированию.

В 1927 г. голландский инженер Балтазар Ван дер Пол сообщил об обнаружении устойчивых релаксационных колебаний. Уравнение, описывающее осциллятор с нелинейным затуханием (уравнение Ван дер Поля), имеет вид [6]:

$$\frac{d^2x}{dt^2} + \mu \cdot (1 - x^2) \cdot \frac{dx}{dt} + x = 0.$$

Постоянная μ определяет нелинейную силу сопротивления. По своему смыслу $\mu \geq 0$, так как сила сопротивления не может быть отрицательной. При равном нулю значении μ уравнение Ван дер Поля фактически переходит в уравнение свободных незатухающих колебаний. Случай ненулевого значения более интересен, рассмотрим его численное решение по аналогии с решением задачи о движении тела в среде с сопротивлением (см. лекцию 2).

Уравнение второго порядка следует привести к системе уравнений первого порядка:

$$\frac{dX}{dt} = V_x,$$

$$\frac{dV_x}{dt} = \mu \cdot (1 - X^2) \cdot V_x - X$$

и модифицировать программный код

```
% Сброс всех переменных
clear all;
```

```
% Задаем начальные условия и коэффициент сопротивления
V0=10;
X0=0;
MyKoeffMu=0;
% Задаем промежуток времени
tspan=[0 2];

% Формируем начальные данные для системы ОДУ
Condition=[X0; V0];

% Решаем систему
[t,R] = ode45(@MyDiffEquation, tspan, Condition);

%Построение графиков
figure();
hold on;
plot(t,R(:,1), t, R(:,2),'LineWidth',2);
legend ('X(t)','Vx(t))');

figure();
hold on;
plot(R(:,1),R(:,2), 'LineWidth',2);
legend ('Vx(X)');

% Формируем систему уравнений
% R(1) — соответствует x(t),
% R(2) – соответствует Vx=dx/dt.
function dRdt=MyDiffEquation(t,R)
MyKoeffMu=0;
dRdt=[R(2); MyKoeffMu*(1-R(1)^2)*R(2)-R(1)];
end.
```

Исследуем случай без затухания (μ = 0). Выполнение программного кода приведет к построению двух графиков, представленных на рис. 14 и 15.

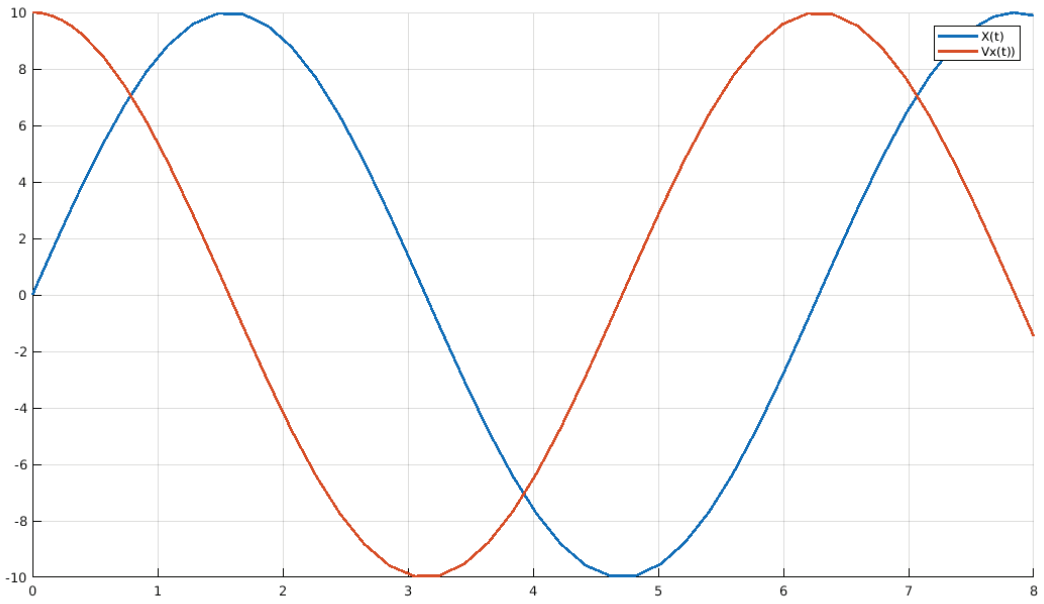


Рис. 14. Зависимости координаты и скорости от времени

На первом из них приведены зависимости координаты и скорости от времени, мы наблюдаем хорошо известные гармонические колебания. Как видно, результат численного расчета соответствует аналитическому решению, найденному ранее.

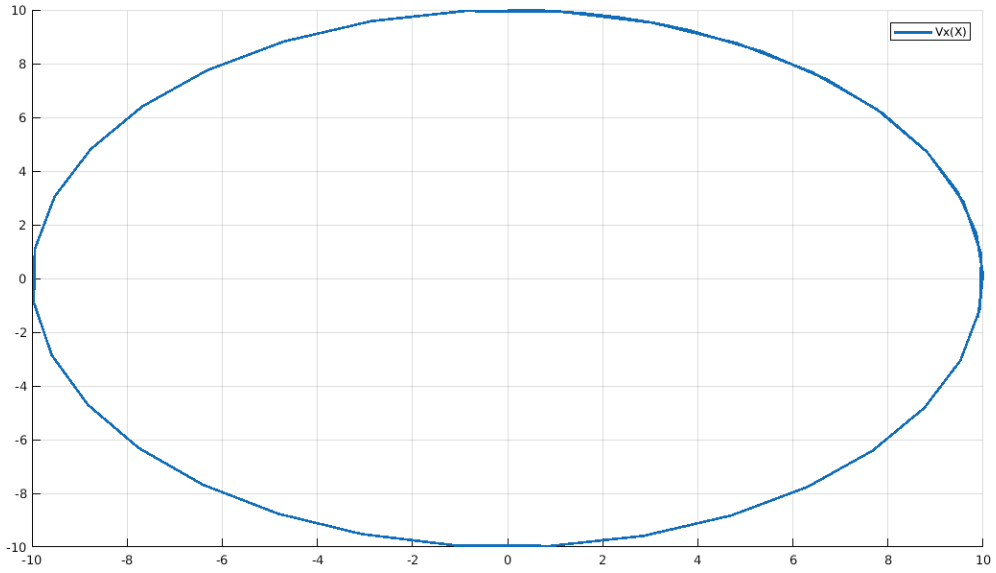


Рис. 15. Фазовая диаграмма гармонического осциллятора

Рисунок 15 представляет собой фазовую диаграмму. В общем случае для построения фазовой диаграммы выбираются две переменные, которые

однозначно определяют состояние системы второго порядка. Наиболее часто в качестве таких переменных выбираются обобщенные координаты и импульсы. Анализ фазовой диаграммы полезен, так как позволяет выявить все возможные состояния в системе, а также отслеживать динамику ее развития. Для нашего случая выбраны координата и скорость (соответствует случаю единичной массы). Видно, что фазовая диаграмма гармонического осциллятора представляет собой окружность.

Исследуем, как изменится поведение нашей системы в случае нелинейного отклика, для этого изменим значение μ на ненулевое. Зависимости скорости и координаты от времени приведены на рис. 16.

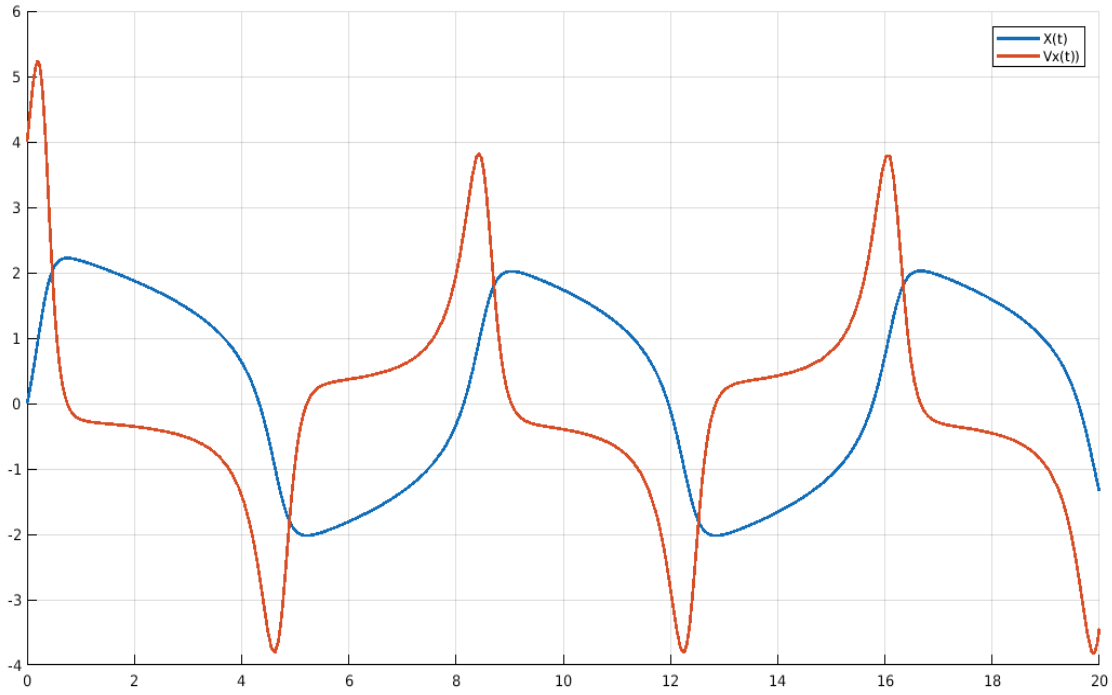


Рис. 16. Зависимость координаты и скорости от времени для осциллятора Ван дер Поля

Как видно, движение в этом случае является периодическим: и координата, и скорость с течением времени восстанавливают свои значения. Однако закон изменения величин весьма далек от гармонического, что подтверждается фазовой диаграммой на рис. 17.

Изначально движение осциллятора не является периодическим, системе требуется некоторое время для перехода к такому состоянию. Область фазовой диаграммы, отвечающая за периодическое движение, пред-

ставляет собой сложную геометрическую фигуру, что и говорит о негармоничности движения.

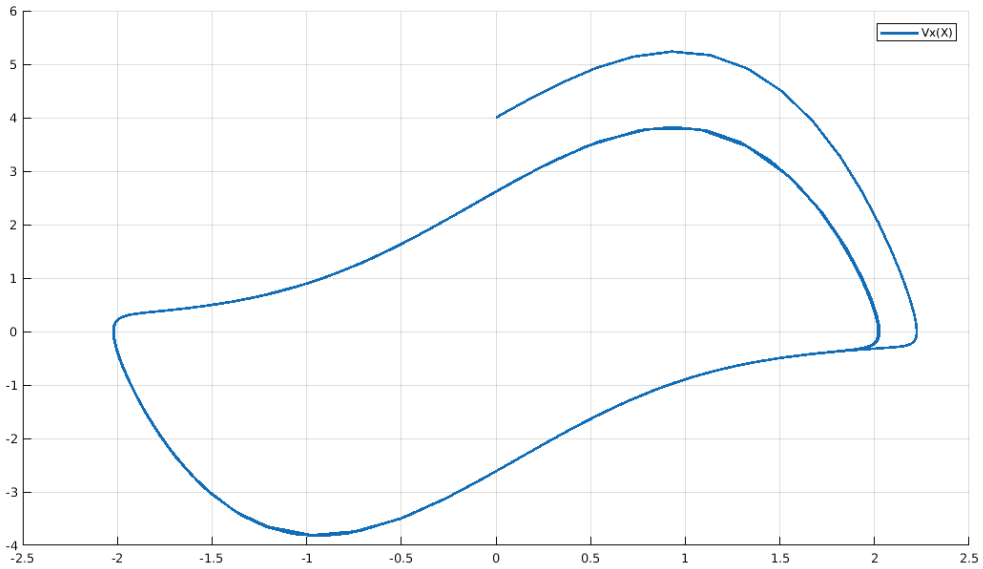


Рис. 17. Фазовая диаграмма осциллятора Ван дер Поля

Рассмотрим изменение фазовой диаграммы при изменении начальных условий движения. Для этого создадим несколько вариантов начальных значений координаты и скорости осциллятора и организуем их циклический перебор. Все результаты для сравнения будут выводиться на одну фазовую диаграмму.

```
% Задаем начальные условия и коэффициент сопротивления
V0=[4, -2, -5, 4] ;
X0=[0, 3, -1, -3];
MyKoeffMu=2;

% Задаем промежуток времени
tspan=[0 20];
% Решаем систему несколько раз с выводом графика
figure();
hold on;
grid on;
for i=1:4
```

```
    Condition=[X0(i);V0(i)]
    [t,R] = ode45(@MyDiffEquation(tspan, Condition);
    plot(R(:,1),R(:,2),'LineWidth',2);
end
```

Результат приведен на рис. 18.

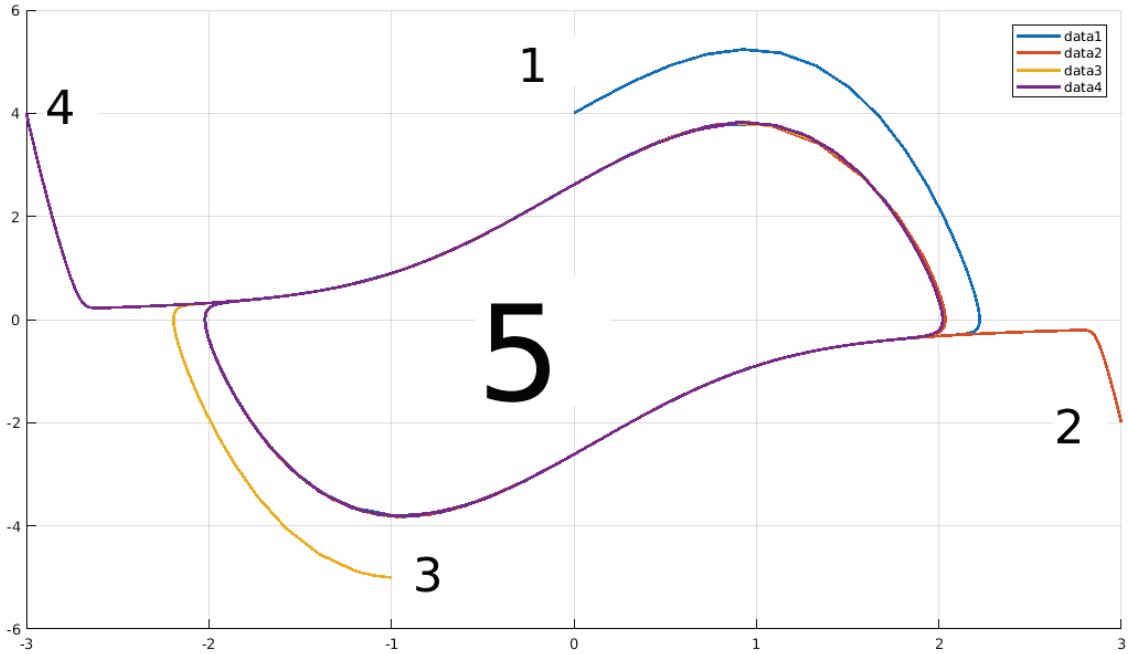


Рис. 18. Предельный цикл (1, 2, 3, 4 – начальные состояния системы, 5 – область предельного цикла)

Независимо от начального состояния системы (точки 1–4) с течением времени она приходит к установившемуся периодическому движению, получившему название «предельный цикл» (область 5 на фазовой диаграмме).

Существование предельных циклов обнаружено для многих нелинейных колебательных систем, одной из наиболее известных является аттрактор Лоренца, описываемый системой уравнений

$$\frac{dX}{dt} = \sigma \cdot (Y - X),$$

$$\frac{dY}{dt} = X \cdot (r - Z) - Y,$$

$$\frac{dZ}{dt} = X \cdot Y - b \cdot Z.$$

Подобная система встречается в приложениях газодинамики, гидродинамики и оптики. В зависимости от значений параметров σ, r и b возможны разные поведения системы (некоторые примеры приведены на рис. 19).

Необычное поведение аттрактора Лоренца связано с тем, что это одна из систем, в которых наблюдается динамический хаос, – явление, при котором поведение нелинейной системы выглядит случайным, несмотря на то что оно определяется детерминистическими законами [7]. В основе динамического хаоса – неустойчивость решения по отношению к малому изменению параметров системы вблизи особых точек. При моделировании таких систем численными методами очень желателен предварительный теоретический анализ.

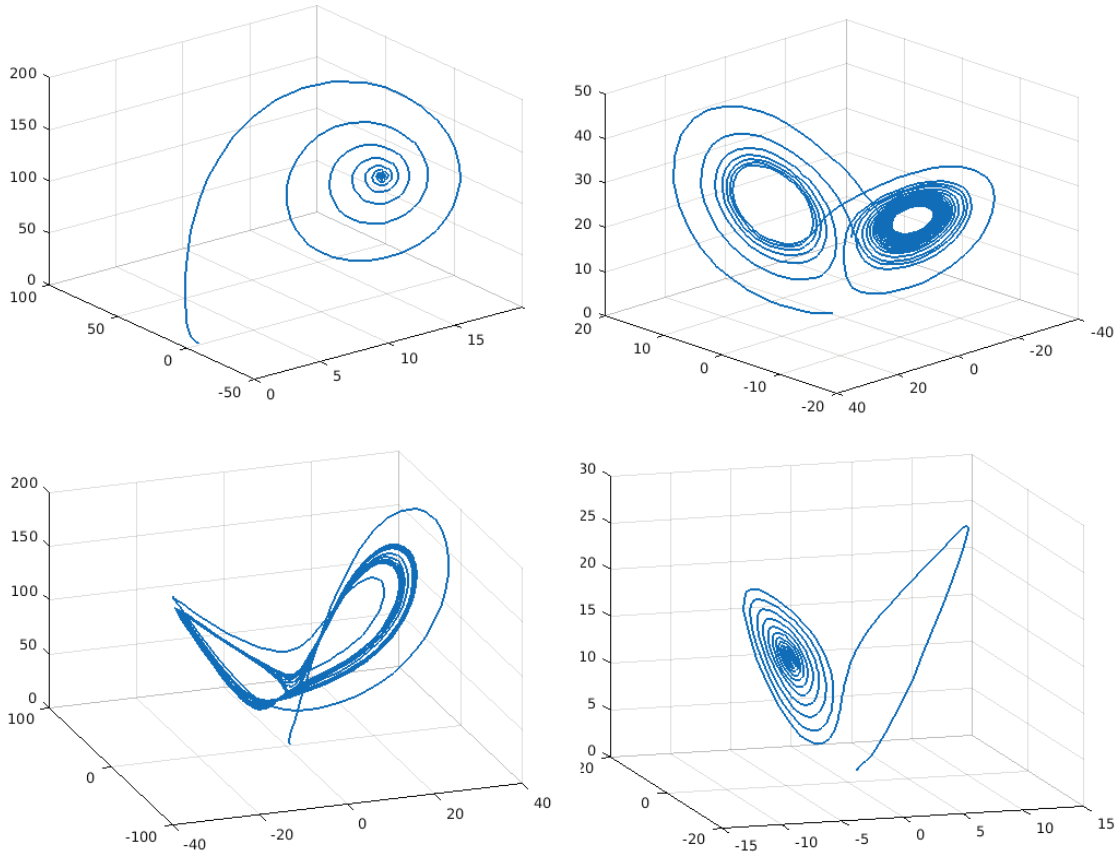


Рис. 19. Аттрактор Лоренца

Задания к лабораторному практикуму

1. Постройте график аналитического решения уравнения свободных незатухающих колебаний с начальными параметрами, заданными преподавателем.

2. Промоделируйте вынужденные колебания под действием гармонической периодический силы. Найдите численное решение системы с параметрами, заданными преподавателем.

3. Промоделируйте движение осциллятора Дуффинга – одномерного маятника, движущегося в поле.

$$U(X) = \frac{A \cdot X^2}{2} + \frac{B \cdot X^4}{4}.$$

Рассмотрите случай вынужденных колебаний под действием гармонической периодической силы. Рассмотрите случай для значений A = 2,75; B = 0,2.

4. Составьте программу для моделирования аттрактора Лоренца. Исследуйте поведение системы при различных комбинациях параметров σ, r и b.

ЗАКЛЮЧЕНИЕ

Цель настоящего издания – не только представить необходимые сведения по вычислительной физике, но и ознакомить студентов с современными методами решения задач по этой дисциплине.

В последние годы наблюдается переход от использования языков программирования к использованию специализированных программных пакетов. Если десятилетие назад исследователь самостоятельно реализовывал алгоритмы решения и конечно-разностные схемы на С, фортране или другом языке высокого уровня, то в настоящее время библиотеки программных пакетов предоставляют большой набор встроенных функций для этих целей. Тем не менее исследователь должен понимать, что постановка задачи и сведение ее к математической модели, а также выбор соответствующего алгоритма решения остаются целиком в его ведении. Для несложных задач автоматически предлагаемые алгоритмы дают достаточную точность расчетов и хорошую сходимость. В остальных случаях при использовании библиотечных функций необходимо определить параметры их работы. Подробное описание принципов работы каждой функции можно найти в документации.

Переломным моментом в современной вычислительной физике и технике стало использование имитационного моделирования. В настоящее время оно широко применяется для моделирования поведения механизмов и машин (Solidworks), физических процессов и свойств материалов (Ansys, SalomeMeca и др.). Simulink и Scilab позволяют реализовать имитационное моделирование с помощью блочного конструирования. Для задач вычислительной физики такой метод, видимо, является оптимальным: с одной стороны, позволяет достаточно наглядно представлять моделируемый процесс в целом, с другой – не слишком удаляться от его математической модели.

Еще одним важным шагом в развитии современной вычислительной физики стало широкое применение средств визуализации. Первоначально результаты численного моделирования представлялись в виде файлов, со-

держащих численные значения исследуемых величин, параметров и объектов. Несмотря на полноту представляемой информации, в таком виде она крайне неудобна для анализа и понимания. С развитием аппаратного обеспечения и разработкой OpenGL разработчики программных пакетов стали включать в их состав обширные библиотеки графических функций, что позволило представлять результаты численного моделирования в режиме реального времени, в удобной для анализа форме.

Движение в гравитационном поле и исследование колебаний – часто встречающиеся и хорошо изученные прикладные задачи, поэтому именно они выбраны для иллюстрации современных подходов и тенденций. Во второй части конспекта лекций будут разобраны методы решения других задач вычислительной физики.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Гилат, А. MATLAB. Теория и практика / А. Гилат. – М. : ДМК Пресс, 2016. – 416 с. – ISBN 978-5-97060-183-9.

2. Дьяконов, В. П. MATLAB. Полный самоучитель / В. П. Дьяконов. – М. : ДМК Пресс, 2013. – 768 с. – ISBN 978-5-97060-493-9.

3. Калиткин, Н. Н. Численные методы : учеб. пособие / Н. Н. Калиткин. – 2-е изд., испр. – СПб. : БХВ-Петербург, 2011. – 592 с. – ISBN 978-5-9775-0500-0.

4. Страница, посвященная использованию Simulink на русскоязычном сайте производителя [Электронный ресурс]. – Режим доступа: http://matlab.ru/products/simulink (дата обращения: 26.05.2017).

5. Дьяконов, В. П. Simulink. Самоучитель / В. П. Дьяконов. – М. : ДМК Пресс, 2012. – 784 с. – ISBN 978-5-97060-280-5.

6. Осциллятор Ван дер Поля [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Осциллятор_Ван_дер_Поля (дата обращения: 26.05.2017).

7. Динамический хаос [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Динамический_хаос (дата обращения: 26.05.2017).