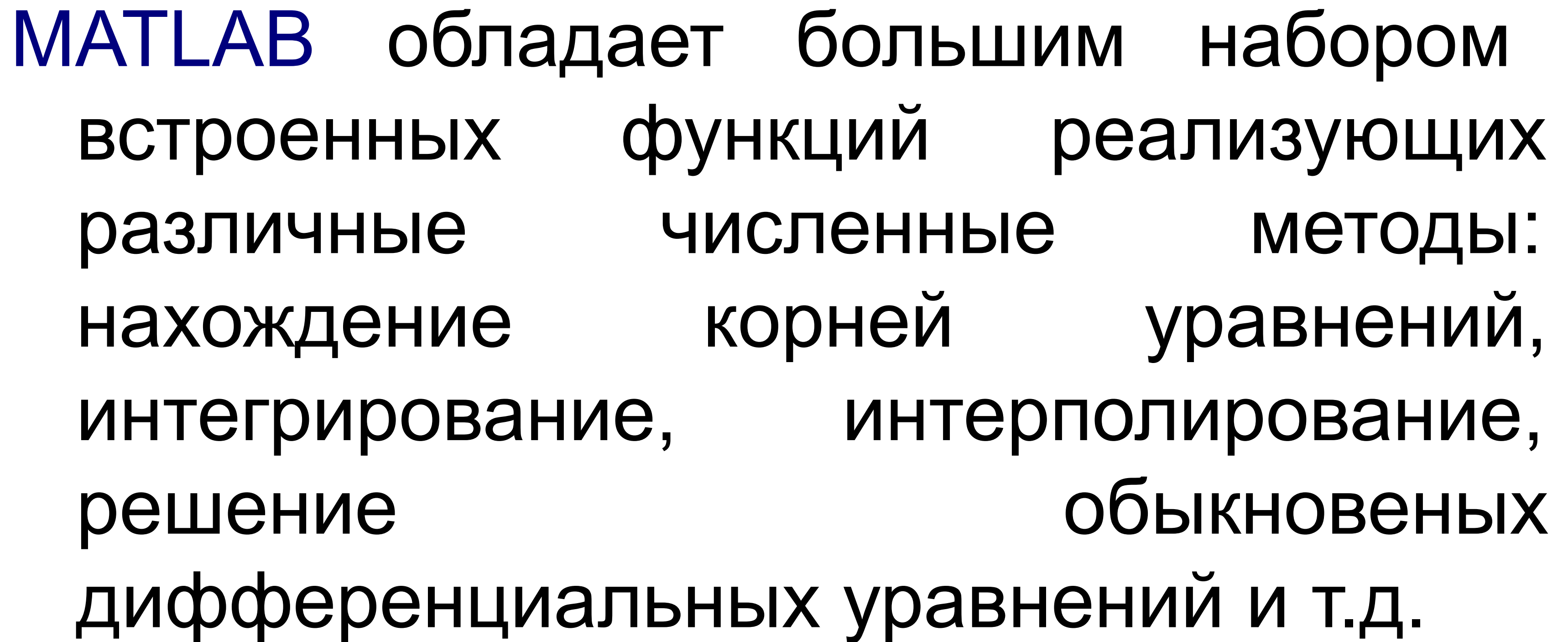
ВЫЧИСЛЕНИЯ
В MATLAB


MATLAB обладает большим набором встроенных функций реализующих различные численные методы: нахождение корней уравнений, интегрирование, интерполирование, решение обыкновеных дифференциальных уравнений и т.д.

Решение произвольных уравнений

Функция

x=fzero('myf', xo)

позволяет вычислять приближенное значение корня x уравнения

myf(x)=0, (1)

с начальным приближением к корню x_o.

Здесь myf – имя файл - функции вычисляющей левую часть уравнения.

Перед нахождением корней полезно строить график функции входящей в левую часть уравнения, используя plot, но все равно в подобных задачах удобно (нужно) создать М-файл левой части уравнений (то же касается и правых частей дифференциальных уравнений).

В этом случае можно воспользоваться функцией

```
fplot('myf', [x1,x2])
```

Пример 1

Найти корни уравнения $\sin x - x^2 \cos x = 0$ на [-5,5].

Решение

М-функция	Командная строка
function y=myf(x) y=sin(x)-x.^2.*cos(x)	fplot(‘myf’, [-5,5]) grid on x1=fzero(‘myf’,-5) x1= -4.7566 myf(x1) ans= 2.6645e-015

Аналогично находятся еще два корня, около -2 и 5. Увидеть, большее число значащих цифр приближенного решения можно задав значение формата, например

format long.

Заметим, что точность вычисления не зависит от формата вывода результата и не меньше eps:

2.220…..E-016 или ±2*10-16.

Замечание

Важной особенностью fzero является то, что она вычисляет только те корни, в которых функция меняет знак, а не касается оси абсцисс. Это происходит в связи с тем, что заложен метод деления отрезка пополам.

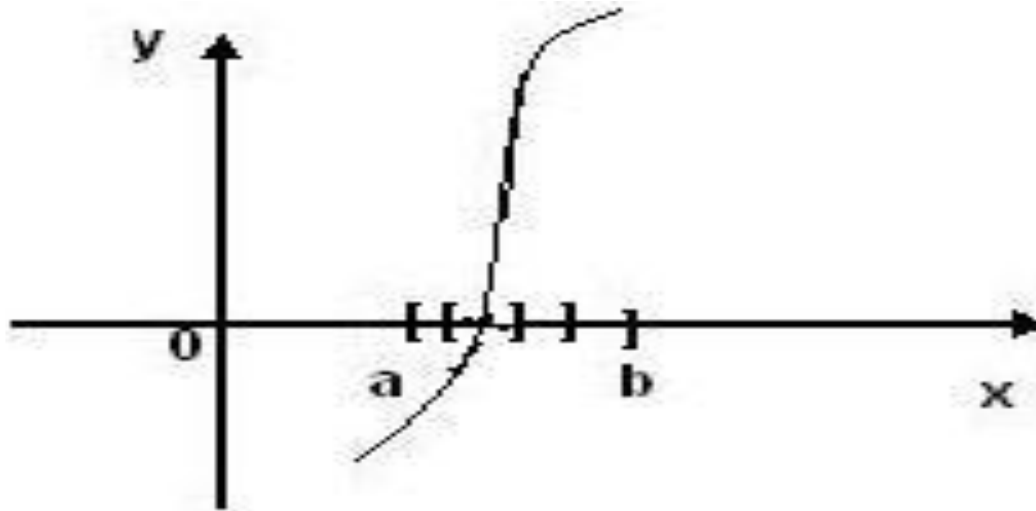


Рис.1 Метод деления отрезка пополам

Например, корень уравнения $x^2=0$ функцией fzero найти не удается.

Для нахождения корня x на интервале [a, b] уравнения (1) и значения функции myf в этом корне можно использовать следующий вид функции fzero:

[x, f]= fzero (' myf', [a, b])

Пример 2

```
> fzero('sin', [2,4])
  ans =
        3.1415…
```

Замечание

1. Если функция имеет несколько нулей, то выдается ближайший к 0;

2. На границе интервал [a,b] функция должна принимать значения разных знаков, иначе будет сообщение об ошибке NaN.

Минимизация функций

Для поиска локального минимума функции myf одной переменной на отрезке [a, b] используют

x= fminbnd ('myf', a, b)

[x, f]= fminbnd (' myf', a, b)

Замечание

1) Для нахождения локального максимума нет специальной функции и поэтому следует искать минимум функции с обратным знаком (менять знак нужно в М-функции);
2) Если есть несколько локальных min, то находиться тот, который ближе к 0.

Для нахождения локального минимума функции myfm многих переменных вблизи точки $(x_1,\ldots,x_n)$ используют

M=fminsearch('myfm',[x_1,x_2,…,x_n]);

[M, f]= fminsearch('myfm',[x_1,x_2,…,x_n]).

Здесь M=[$x_1^0, x_2^0, \ldots, x_n^0$] – вектор-строка, а f – значение.

Метод Нельдера-Мида

Для нахождение минимума используют симплекс метод Нельдера-Мида.

Он заключается в следующем: берутся 3 вершины вокруг начального положения, сравниваются значения и выбираются 2 те, где меньше значения функции, берутся опять 3 вершины и т.д. пока размеры симплекса не станут достаточно малыми.

Замечание

Для нахождения начального значения $(x_1,x_2,\ldots,x_n)$, нужно получить представление о поведении функции, например, построив линии уровня на плоскости с помощью функции

contour.

Задание дополнительных параметров

Функции

fzero, fminbnd и fminsearch

позволяют задать дополнительный параметр options, контролирующий вычислительный процесс:

options = optimset(…, вид контроля, значение,…)

Таблица 1 Значения дополнительных параметров

Вид контроля	*Значение*	*Результат*
'Display'	‘off'	Информация о вычислительном процессе не выводится.
	‘iter'	Выводится информация о каждом шаге вычислительного процесса.
	'final'	Только информация о завершении (по умолчанию).

Вид контроля	*Значение*	*Результат*
'MaxFunEvals'	Целое число	Максимальное количество вызовов (вычислений) исследуемой функции.
'MaxIter'	Целое число	Максимальное количество итераций
'TolFun'	Положительное вещественное число	Точность по функции для остановки вычисления (сравнение соседних значений функций).
'TolX'	Положительное вещественное число	Точность по аргументу функции для остановки вычислений.

Замечание

Ограничивать количество вызовов функции и число итераций имеет смысл, если есть опасение, что получить решение не удается из-за расхождения вычислительного процесса.

Пример 3

Найти минимум функции

$f(x, y) = \sin(\pi x) * \sin(\pi y)$

вблизи точки [1.4, 0.6] с точностью по функции до 1.0e-09 и выводом итераций.

Решение

M-file	Командное окно

function f=ftest(argvec)
x= argvec(1);
y= argvec(2);
f=sin (pi*x).* sin (pi*y);
	options=optimset('Display', 'iter', 'TolFun', 1.0e-09);
	[M, f]=fminsearch('ftest', [1.4, 0.6], options)

В итоге, кроме результата, выведется таблица каждая строка, которой соответствует одной итерации.

В ней будет содержаться: количество вызовов функции, текущее приближение и значение функции от него, а так же метод, применяемый при данной итерации.

Интегрирование функций

Методы интегрирования

Для вычисления определенного интеграла используются следующие функции:

1. quad('fint',a,b, Точность)

Алгоритм основан на квадратурной формуле Симпсона с автоматическим подбором шага интегрирования для достижения нужной точности:

$$s = \frac{h}{3}\sum_{k=1}^{M}(f(x_{2k-2}) + 4f(x_{2k-1}) + f(x_{2k})),$$

где x_k, k = $\overline{0,M}$ равностоящие точки на [a,b].

2. quad8('fint',a,b, Точность)

Используется для достаточно гладких функции и алгоритм основан на более точных квадратурных формулах Ньютона-Котеса.

Он требует меньше вычислений с той же точностью

3. quadl('fint',a,b, Точность)

Применяется для функций с интегрируемыми особенностями (например: $\frac{1}{\sqrt{x}}$ в нуле).

Алгоритм основан на квадратурных формулах Гаусса-Лобатто (Корни ортогонального многочлена Якоби).

Для вычисления двойного интеграла используется функция

dblquad('fint', x_{min}, x_{max}, y_{min}, y_{max}, Точность, 'Алгоритм'),

где Алгоритм – это название любого из перечисленных трех алгоритмов quad, quad8, quadl.

Пример 4

Вычислить

$$\int_0^1 \int_{-\pi}^{\pi} \left(e^x \sin^2 y + e^{-x} \cos^2 y\right) dxdy$$

по quad8 с точностью 10^{-12}.

Решение

M-file	Командное окно
function f=fint1(x,y) f=exp(x).*sin(y).^2+exp(-x). *cos(y). ^ 2;	dblquad('fint1',-pi,pi,0,1,1.0e-012, 'quad8')

Вычисление интегралов зависящих от параметров

Пример 5

Вычислить интеграл

$$\int_{-1}^{1}\left(p_1 x^2 + p_2 \sin(x)\right)dx$$

при значениях параметров $p_1=0.6$, $p_2=0.7$ по квадратурным формулам Ньютона-Котеса с автоматическим выбором шага с точностью до 10^{-5}.

Решение

M-file	Командное окно
function z=fparam(x, Par1,Par2) f= Par1.*x^2+ Par2.* sin(x);	quad('fparam',-1,1,1.0e-05,**1**,0.6,0.7) Здесь пятый параметр, равный единице, поставлен для наблюдения за процессом интегрирования, иначе он равен 0. Это обязательный параметр для интегрирования с параметром.

Интегралы с переменным верхним пределом

Пример 6

Вычислить интеграл

$$F(y) = \int_0^y e^x (\sin x - \cos x) dx$$

с точностью 10^{-6}.

Решение

M-file	Командное окно. Построим график зависимости интеграла от верхнего предела.

function f= fint(x)
f=exp(x).*(sin(x) -cos(x));
 	
function f=Fy(y)
f=quad8('fint', 0, y, 1.0e-06);
	fplot('Fy', [0, pi])

Численное решение дифференциальных уравнений

Задача Коши для дифференциального уравнения произвольного порядка имеет вид:

$$\begin{cases} y^{(n)} = f(t, y', \ldots, y^{(n-1)}) \\ y(t_0) = u_0 \\ y'(t_0) = u_1 \\ \ldots \\ y^{(n-1)}(t_0) = u_{n-1} \end{cases}$$

1. Приведение дифференциального уравнения к системе дифференциального уравнений первого порядка;
2. Написание специальной файл - функции для системы уравнений;
3. Вызов подходящего солвер (решателя);
4. Визуализация результата.

Пример 7

Задача о колебаниях под воздействием внешней силы в среде, оказывающей сопротивление колебаниям:

$y''+2y'+10y=\sin t$ - уравнение, описывающее движение.

$y(0)=1$ – кордита точки в начальный момент времени.

$y'(0)=0$ – начальная скорость.

Первый этап.

Для приведения задачи к системе дифференциальных уравнений вводим вспомогательные функции

$y_1=y$, $y_2=y'$.

Тогда система дифференциальных уравнений с начальными условиями принимает вид:

$$\begin{cases} y_1' = y_2 \\ y_2' = -2y_2 - 10y_1 + \sin t \end{cases}$$

$$\begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad (1)$$

Второй этап состоит в написании файл-функции имеющей два входных аргумента: переменную t и вектор, размер которого равен числу неизвестных функций системы.

Число и порядок аргументов фиксированы, даже если t явно не входит в систему.

Выходным аргументом файл-функции является вектор правой части системы.

Итак,

```
function F=oscil(t,y)
F=[y(2); -2*y(2)-10*y(1)+sin(t)];
```

Третий шаг.

Этот шаг состоит в решении задачи при помощи решателя или солвера (об их видах поговорим позже). Например, при помощи солвера

ode45.

Входными аргументами солверов являются:

1. Имя файл-функции в апострофах;
2. Вектор-строка с начальным и конечным значением времени наблюдения (например, $[0, t_0]$, где t_0 произвольное число);
3. Вектор-столбец начальных условий (1).

Выходных аргументов два:

1. Вектор T содержащий значение времени;

2. Матрица значений Y неизвестных функций в соответствующие моменты времени.

В первом столбце - значения первой функции, во второй и т. д.

В нашем случае: Y(:, 1) – значение функции y1,
Y(:, 2) – значение функции y2.

Как правило, размеры матрицы Y и вектора T достаточно велики, поэтому лучше сразу отобразить результат на графике.

Итак, файл программа для $0 \le t \le 15$ имеет вид.

```
Y0= [1; 0];
  [T, Y]=ode45('oscil', [0 15], Y0);
plot(T, Y(:, 1), 'r')                    - график решения
hold on
plot (T, Y (:, 2), 'k--')                - график производной
title ('Решение {\ it y}\prime\prime+2 {\it y}\ prime+10{\it y} = sin {\it y}')
xlabel ('\it t')
ylabel ('{\it y}, {\it y}\prime')
legend('координата', 'скорость', 4) – Легенда в нижний правый угол.
grid on   - сетка
hold off
```

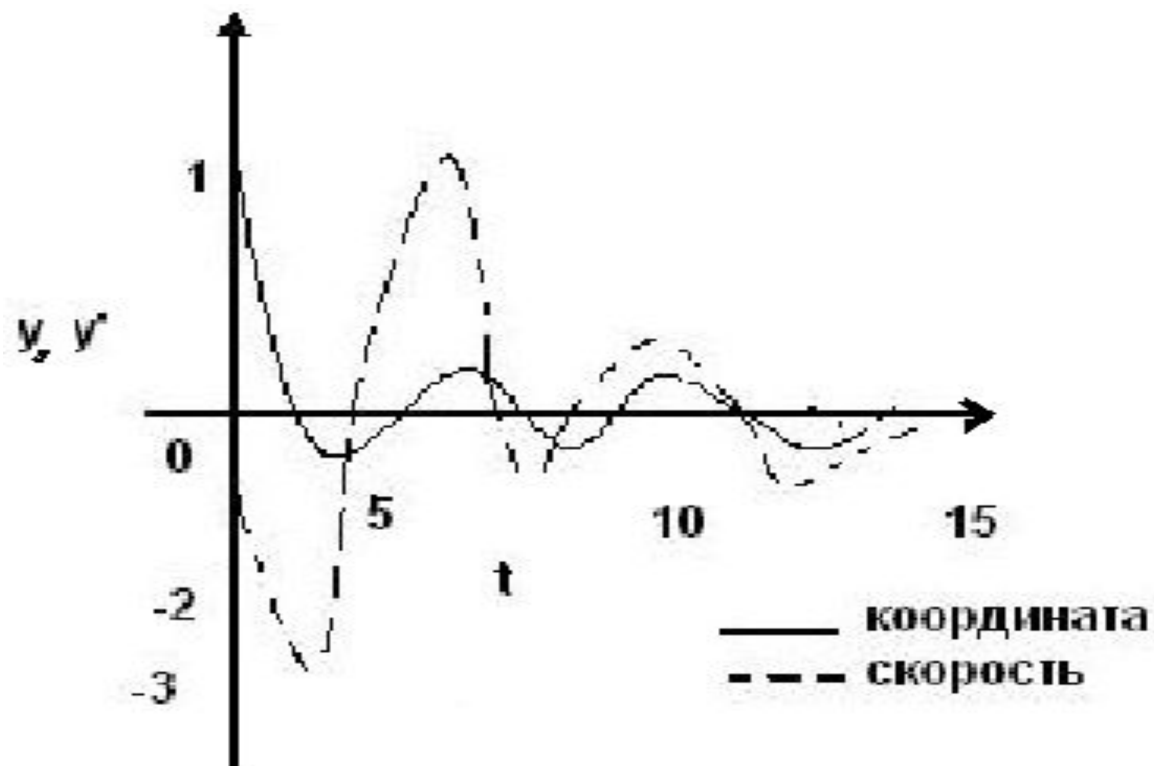


Рис.2 Решение дифференциального уравнения

Замечание

Здесь использовался солвер

ode45,

который применяет метод Рунге-Кутта четвертого порядка.

При применении солвера очень важно учитывать свойства системы дифференциальных уравнений, иначе можно получить неточный результат или затратить много времени на решение.

Жесткие дифференциальные системы

Решение уравнений Лотка-Вольтерра

Во многих приложениях встречаются так называемые «жесткие» системы дифференциальных уравнений.

Строго общепринятого определение жестокости пока не существует, и под жесткими системами обычно понимают те системы, которые плохо решаются «стандартными» численными методами.

В них часто встречается существенная разница по модулю между корнями λ характеристического уравнения.

Например

$$\begin{cases} x_1' = -\lambda_1 x_1 \\ x_2' = -\lambda_2 x_2 \end{cases}$$, причем, $\lambda_1, \lambda_2 > 0$ и $s = \frac{\lambda_1}{\lambda_2} >> 1$.

Число s называют коэффициентом жесткости системы.

Решение имеет вид:

$$\begin{cases} x_1 = e^{-\lambda_1 t} \\ x_2 = e^{-\lambda_2 t} \end{cases}$$

При моделировании физических процессов причина такой разности величин собственных чисел заключена в наличии существенно разных характерных времен процессов, описываемых данными системами ОДУ («медленного» и «быстро» времени).

Отметим, что жесткость – это качество дифференциальной системы, а не разностного метода (обычно нелинейные системы).

В качестве примера жесткой системы возьмем модель Лотка-Вольтерра борьбы за существование.

Обозначим

$y_1(t)$ – число жертв

$y_2(t)$ – число хищников

Число хищников и жертв в течение времени t изменяется по закону

$$\begin{cases} y_1' = Py_1 - py_1y_2 \\ y_2' = -Ry_2 + ry_1y_2 \end{cases} \qquad (2)$$

где P – увеличение числа жертв в отсутствие хищников

R – уменьшение числа хищников в отсутствие жертв.

Вероятность поедание хищником жертвы пропорциональна их числу y_1y_2, при этом

$-py_1y_2$ – соответствует вымиранию жертв;

ry_1y_2 – появлению хищников.

Решением системы (2) является замкнутая кривая.

Возьмем для примера

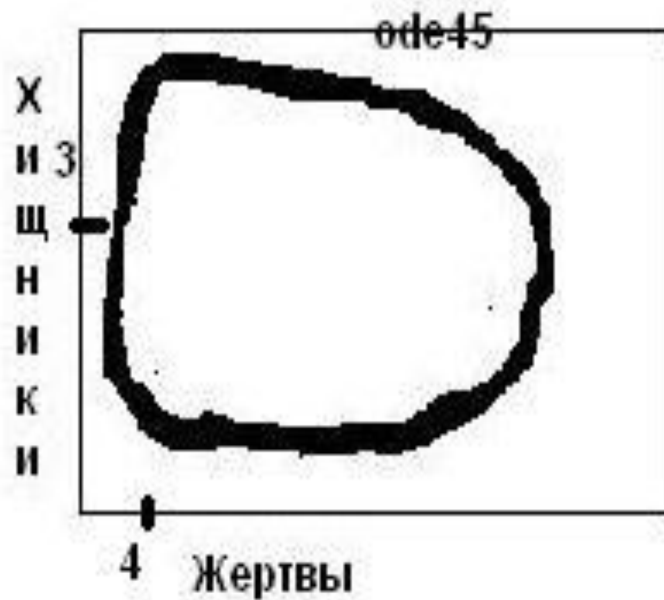
$P=3$, $R=2$, $p=r=1$

$y_1(0)=3$ – в начальный момент 3 жертвы.

$y_2(0)=4$ – в начальный момент 4 хищника.

$t \le 100$

Решая эту систему солверами ode45 и ode23s, получаем:



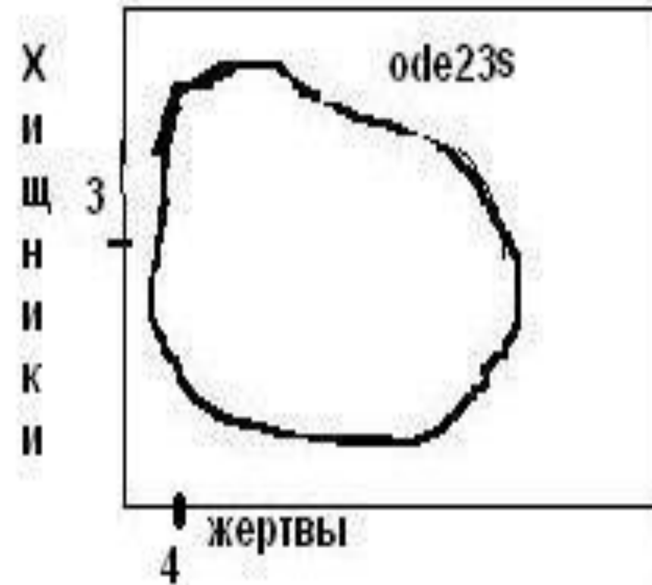


Рис.3 Решения солверами ode45 и ode23s

Вычисление, по умолчанию, в ode45 и ode23s происходит при одной точности, а приближенные решения сильно отличаются друг от друга, причем ode23s намного точнее.

Более того, для уравнения Ван-дер-Поля с большим значением параметра α:

$$y''=\alpha(1-y^2)y'-y$$

солвер ode45 не может найти значения (см. справку по MATLAB).

Управление процессом решения

Для эффективного решения дифференциальных уравнений необходимо выбрать подходящий солвер в зависимости от свойств решаемой задачи, произвести необходимые установки, обеспечивающие получение приближенного решения с требуемыми свойствами, например с заданной точностью.

Стратегия применения солверов MATLAB (см. справку по MATLAB).

- ode45 – дает хорошие результаты и основан на методе Рунге-Кутта четвертого и пятого порядков точности;

- ode23 – используется в задачах содержащих небольшую жесткость, когда требуется получить решение с невысокой степенью точности. Формулы Рунге-Кутта 2 и 3 порядка точности;
- ode113 – эффективен для нежестких систем дифференциальных уравнений, правые части которых вычисляются по сложным формулам, и решение выдается с высокой точностью. Метод переменного порядка Адамса-Бэшфорта-Милтона.

Если все попытки применения этих солверов не приводят к успеху, то возможно решаемая система является жесткой и можно применить:

- ode15s – многошаговый метод Гира, допускающий изменения порядка;
- ode23s – для решения жестких систем с невысокой точностью. Реализуется одношаговый метод Розенброка второго порядка.

- Все солверы пытаются найти решение с относительной точностью 10^{-3}.

- Для задания другой точности используются дополнительный параметр

options = odeset(…, вид контроля, значение, …)

- Его использование аналогично использованию optimset.

- Полный список параметров можно найти в справочной системе MATLAB.

Замечание

При заданных по умолчанию значениях, в частности при относительной погрешности 10^{-3}, не всегда возможно получение хорошего приближения. Например, рассмотрим систему

$$\begin{cases} y_1' = y_2 \\ y_2' = -\dfrac{1}{t^2} \end{cases}$$

на отрезке [a,100],

при начальных условиях

$$\begin{cases} y_1(a) = \ln(a) \\ y_2(a) = \dfrac{1}{a} \end{cases}$$

при a=0.001

Точное решение нашей системы будет

$$\begin{cases} y_1 = \ln(t) \\ y_2 = \dfrac{1}{t} \end{cases}$$

Если воспользуемся ode45, то для 10^{-3} получим

Рис. 4 Решение солвером ode45 с точностью 10^{-3}

Применение других солверов не улучшает ситуацию.

Выход – уменьшение относительной погрешности.

```
options=odeset('RelTol', 1.0e-04)
 [T, Y] =ode45 (‘Функция', [a, 100], Y0, options);
```

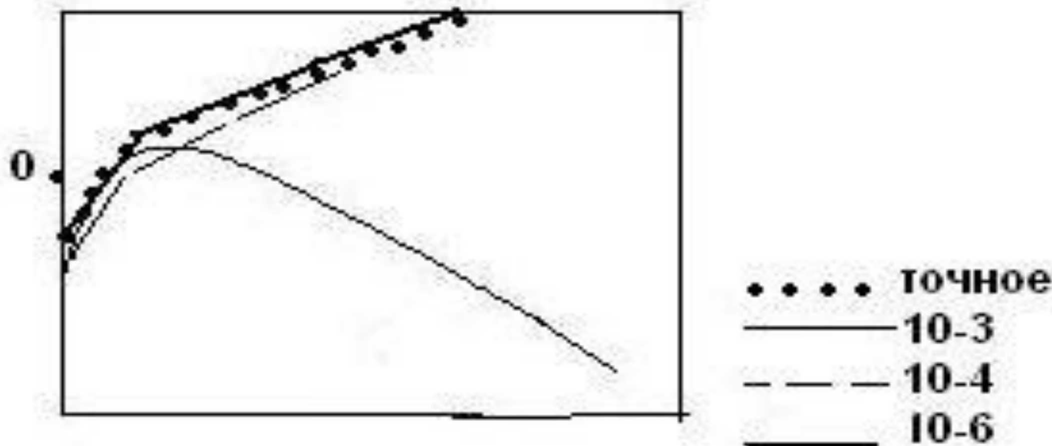


Рис. 5 Решение солвером ode45 с разной точностью

Замечание

Существует отдельный пакет для уравнений в частных производных Partial Diffential Equation ToolBOX.

Решение граничных задач

Рассмотрим граничную задачу общего вида для обыкновенного дифференциального уравнения второго порядка:

$$\begin{cases} y'' = f(x, y, y'), x \in [a, b] \\ \alpha y(a) + \beta y'(a) = A \\ \gamma y(b) + \delta y'(b) = B \end{cases}$$

где $\alpha, \beta, \gamma, \delta, A, B$ - заданные числа

Решение задачи состоит из следующих этапов:

1. Преобразование дифференциального уравнения второго порядка к системе двух уравнений первого порядка;
2. Написание файл-функции для вычисления правой части системы;
3. Написание файл-функции, определяющей граничные условия;
4. Формирование начального приближения при помощи специальной функции bvpinit;
5. Вызов солвера bvp4c для решения граничной задачи;
6. Визуализация результата.

Первые два этапа выполняются практически так же, как и при решении задачи Коши.

Для выполнения 3-го пункта граничные условия приводим к виду:

$$\alpha y_1(a) + \beta y_2(a) - A = 0 \qquad \gamma y_1(b) + \delta y_2(b) - B = 0$$

$y'_1 = y_2$, $y = y_1$

Файл-функция описывающая граничные условия должна зависеть от двух аргументов – векторов y_a, y_b и иметь структуру:

```
function g=boun (ya, yb)
g= [alpha*ya (1) +beta*ya (2)-A; gamma*yb (1)
                      +delta*yb (2)-B];
```

Здесь вместо alpha, beta, A, gamma, delta, B следует подставить заданные числа.

Выбор начального приближения может оказать влияние на решение солвером bvp4c.

MATLAB находит приближенное решение граничных задач методом конечных разностей, то есть получающиеся решение есть вектор значений неизвестных функций в точках отрезка [a,b] (в узлах сетки).

Вызов bvpinit выглядит так

initsol=bvpinit(вектор сетки на [a, b], вектор постоянных значений функций y_1 и y_2)

Пример 8

Пусть [a, b]= [0, 2*pi] и начальное приближение

$y_1=1,\ y_2=0,$

тогда

```
initsol=bvpinit ([0: pi/10:2*pi], [1 0]);
```

Замечание

Заданная сетка может быть изменена солвером в процессе решения, для обеспечения требуемой точности.

Следующим этапом мы должны вызвать солвер:

```
sol=bvp4c('Функция правой части', 'bound', initsol);
```

где структура sol имеет поля x, y, причем

- sol.x – координаты сетки полученной и возможно исправленной;
- sol.y(1, :) – соответствует значениям функции y_1 в точках сетки sol.x;
- sol.y(2, :) – соответствует значениям функции y_2 в точках сетки sol.x;

Другими словами sol.x~T, sol.y~Y.

Визуализация результата происходит аналогично задаче Коши.