Санкт-Петербургский политехнический университет Петра Великого

Институт металлургии, машиностроения и транспорта

Кафедра «Компьютерные технологии в машиностроении»

Петраш В.И.

Вычислительная система MathCAD

Учебное пособие

Часть 2

Санкт-Петербург
2017

УДК 519.673

В.И. Петраш. Вычислительная система MathCAD. Учебное пособие. Часть 2. / Санкт-Петербургский политехнический университет Петра Великого, 2017 – 23 с.

Учебное пособие соответствует государственному образовательному стандарту дисциплины «Информатика» (раздел «Программное обеспечение и технологии программирования») направления подготовки бакалавров машиностроительного направления.

В учебном пособии рассматриваются различные способы решения обыкновенных дифференциальных уравнений.

Рассмотрено решение задачи нахождения экстремума функции одного и нескольких переменных.

Рассмотрены способы обработки экспериментальных данных путем решения задачи интерполяции.

Описываются операторы для составления программ и особенности их работы.

Представлены практические примеры.

Предназначено для студентов Института металлургии, машиностроения и транспорта, изучающих дисциплины «Информатика» и «Вычислительная математика».

Оглавление

ГЛАВА 1 РЕШЕНИЕ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

1.1 Решение ОДУ первого порядка

Дифференциальное уравнение первого порядка может по определению содержать, помимо самой искомой функции $y(x)$, только ее первую производную $y'(x)$. В подавляющем большинстве случаев дифференциальное уравнение можно записать в стандартной форме (форме Коши):

$$y'(x) = f(y(x), x)$$

и только с такой формой умеет работать вычислительный процессор.

Правильная с математической точки зрения постановка соответствующей задачи Коши для ОДУ первого порядка должна, помимо самого уравнения, содержать одно начальное условие — значение функции $y(x_0)$ в некоторой точке x_0. Требуется явно определить функцию $y(x)$ на интервале от x_0 до x_1.

Для численного интегрирования одного ОДУ имеется выбор — либо использовать вычислительный блок **Given**/**odesoive**, либо встроенные функции. Первый путь предпочтительнее из соображений наглядности представления задачи и результатов, а второй дает пользователю больше рычагов воздействия на параметры численного метода.

1.1.1 Вычислительный блок Given/odesoive

Вычислительный блок для решения одного ОДУ, реализующий численный метод Рунге-Кутта, состоит из трех частей:

- **Given** — ключевое слово;
- ОДУ и начальное условие, записанное с помощью логических операторов, причем начальное условие должно быть в форме $y(x_0) = b$ (со знаком символьного равенства);
- **odesolve**(***x*** , ***x₁***) — встроенная функция для решения ОДУ относительно переменной ***x*** на интервале (***x₀***, ***x₁***).

Допустимо, и даже часто предпочтительнее, задание функции **Odesolve** (**t**, $\mathbf{t_1}$, **step**) с тремя параметрами, где **step**— внутренний параметр численного метода, определяющий количество шагов, в которых метод Рунге - Кутта, будет рассчитывать решение дифференциального уравнения. Чем больше **step**, тем с лучшей точностью будет получен результат, но тем больше времени будет затрачено на его поиск.

Результатом применения блока **Given**/**odesoive** является функция $y(x)$, определенная на интервале от x_0 до x_1. Следует воспользоваться обычными

средствами, чтобы построить ее график или получить значение функции в какой-либо точке указанного интервала.

Внимание! Уравнение и начальные условия записываются с помощью символьного знака равенства.

На рисунке 1.1 показано решение уравнения $y' + y = x\cos x$ на промежутке $[0;12\pi]$.

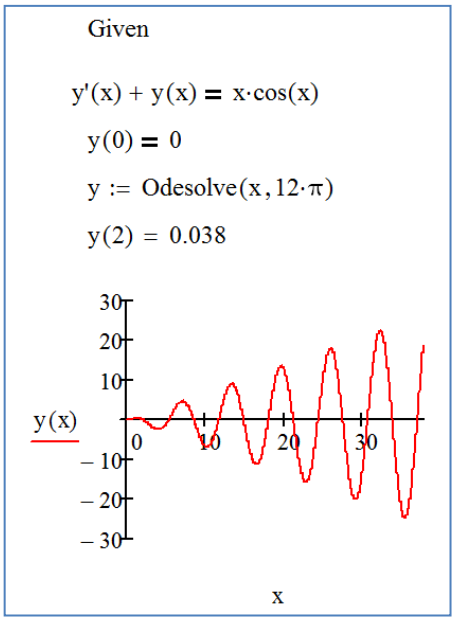

Рис. 1.1

1.1.2 Встроенная функция rkfixed

Эта функция использует для поиска решения метод Рунге-Кутта четвертого порядка.

В результате решения получается матрица, имеющая два столбца. Первый столбец содержит точки, в которых ищется решение уравнения. Второй столбец содержит значения найденного решения в соответствующих точках.

Обращение к функции **rkfixed** выглядит следующим образом.

rkfixed(■, ■, ■, ■, ■)

В первом поле для ввода указывается переменная, для которой ищется решение.

Во втором и третьем полях указываются граничные точки интервала, на котором ищется решение.

В четвертом поле указывается число точек (не считая начальной точки), в которых ищется приближенное решение. При помощи этого аргумента определяется число строк в матрице, возвращаемой функцией.

В последнем поле указывается функция, возвращающая значение в виде вектора, содержащего первые производные.

В качестве примера рассмотрим решение уравнения, решенного выше. Число точек поиска решения – 100.

Чтобы решить уравнение с помощью встроенной функции, надо в левой части оставить только производную, а все остальные слагаемые перенести в правую часть. То есть записать уравнение в следующем виде: $\frac{dy}{dx} = -y + x\cos x$.

После этого необходимо набрать следующие команды (рисунок 1.2).

```
y0 := 0

A(x,y) := −y0 + x·cos(x)

Rez := rkfixed(y,0,12·π,100,A)
```

Рис.1.2

Если после этого набрать команду «**Rez**=», появится матрица, содержащая значения ***x*** и ***y*** (рисунок 1.3).

Rez =

	0	1
0	0	0
1	0.377	0.061
2	0.754	0.191
3	1.131	0.3
4	1.508	0.301
5	1.885	0.129
6	2.262	-0.235
7	2.639	...

Рис. 1.3

Результаты интегрирования удобнее представлять в виде графика. По оси абсцисс откладывается нулевой столбец матрицы, а по оси ординат – первый столбец матрицы (рисунок 1.4).

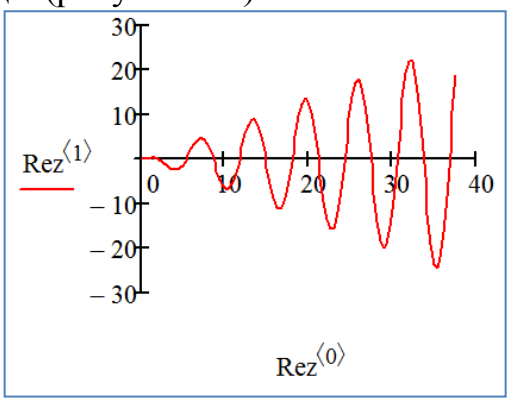

Рис. 1.4

1.2 Решение ОДУ второго порядка

1.2.1 Вычислительный блок Given/odesoive

На рисунке 1.5 показано решение уравнения $y'' = -y' + 2y$ с начальными условиями $y(0) = 1$ и $y'(0) = 3$ на промежутке $[0;2]$.

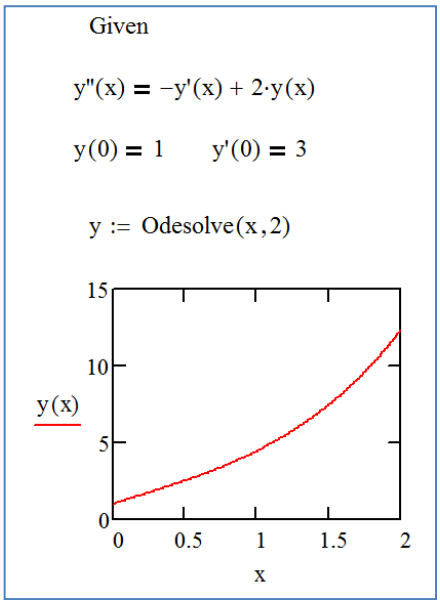

Рис. 1.5

1.2.2 Встроенная функция rkfixed

Отличия от способа решения уравнения первого порядка с помощью встроенной функции состоят в следующем.

Вектор начальных условий ***y*** теперь состоит из двух элементов: значения функции и значения ее первой производной в начальной точке.

Функция $\boldsymbol{A(x,y)}$ является теперь вектором с двумя элементами:

$$\boldsymbol{A(x,y)} = \begin{pmatrix} y'(x) \\ y''(x) \end{pmatrix}$$

Матрица, полученная в результате решения, содержит теперь три столбца: первый столбец содержит значения ***x***, в которых ищется решение, второй столбец содержит значения ***y***, третий столбец содержит значения первой производной.

На рисунке 1.6 показано решение уравнения из п. 1.2.1.

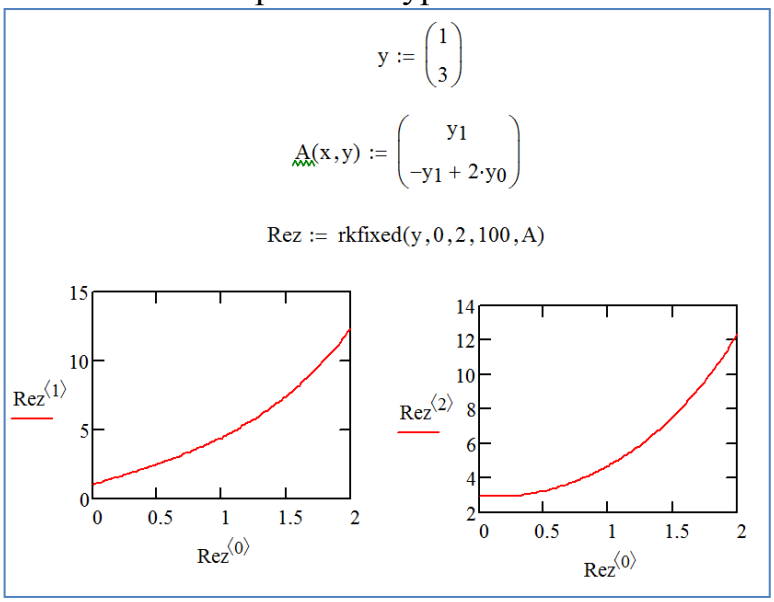

Рис. 1.6

1.3 Решение систем ОДУ первого порядка

Методика решения системы обыкновенных дифференциальных уравнений очень похожа на методику решения дифференциального уравнения второго порядка.

Чтобы решить систему ОДУ первого порядка необходимо:

- Определить вектор, содержащий начальные значения для каждой неизвестной функции.
- Определить функцию, возвращающую значение в виде вектора из ***n*** элементов, которые содержат первые производные каждой из неизвестных функций.
- Выбрать точки, в которых нужно найти приближенное решение.

Передать всю эту информацию в функцию **rkfixed**. Она вернет матрицу, чей первый столбец содержит точки, в которых ищется приближенное решение, а остальные столбцы содержат значения найденных приближенных решений в соответствующих точках.

На рисунке 1.7 показано решение системы ОДУ первого порядка

$$\begin{cases} \dfrac{dy}{dx} = 2x - 3y - 4z \\ \dfrac{dz}{dx} = x + y + z \end{cases}$$

с начальными условиями: $y(0) = 0$ и $z(0) = 0$ на промежутке $x \in [0;10]$.

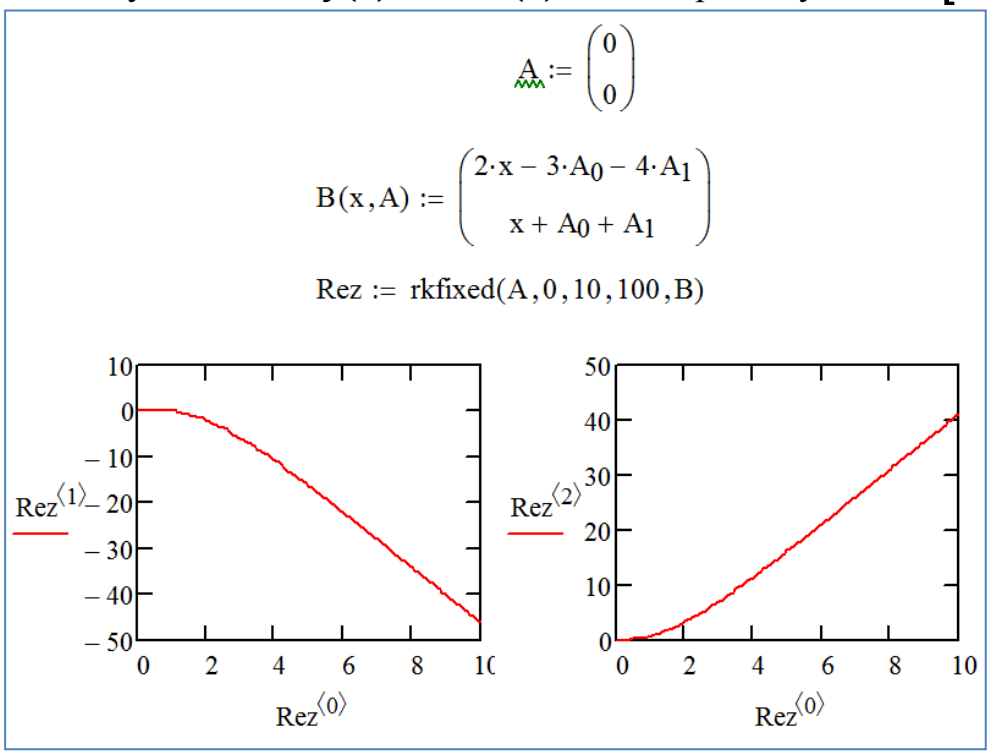

Рис. 1.7

ГЛАВА 2 ПОИСК ЭКСТРЕМУМА ФУНКЦИИ

Задачи поиска экстремума функции означают нахождение ее максимума (наибольшего значения) или минимума (наименьшего значения) в некоторой области определения ее аргументов. Ограничения значений аргументов, задающих эту область, как и прочие дополнительные условия, должны быть определены в виде системы неравенств и (или) уравнений. В таком случае говорят о задаче на условный экстремум.

Для решения задач поиска максимума и минимума имеются встроенные функции **Minimize** и **Maximize**. Все они используют градиентные численные методы.

2.1 Экстремум функции одной переменной

Поиск экстремума функции включает в себя задачи нахождения локального и глобального экстремума. Последние называют еще задачами оптимизации. Рассмотрим функцию $f(x) = x^4 + 5x^3 - 10x$, заданную на промежутке [-5;2] (рисунок 2.1).

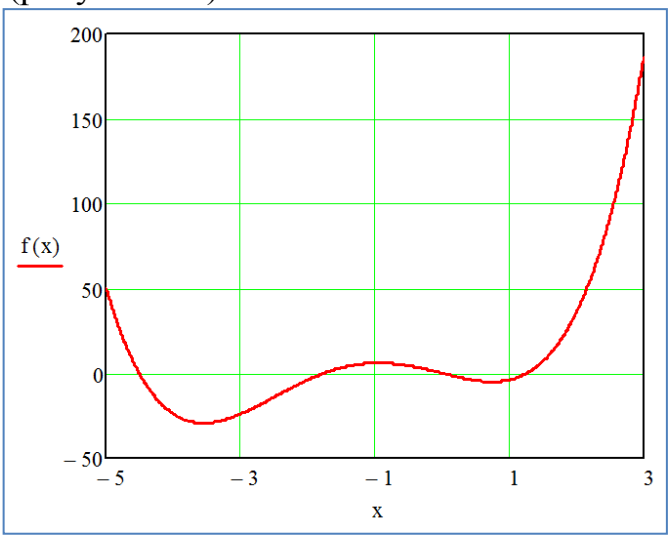

Рис. 2.1

Она имеет: глобальный максимум на правой границе интервала; локальный максимум на левой границе интервала; глобальный минимум, локальный максимум, локальный минимум внутри интервала.

В **MathCAD** с помощью встроенных функций решается только задача поиска локального экстремума. Чтобы найти глобальный максимум (или минимум), требуется либо сначала вычислить все их локальные значения и потом выбрать из них наибольший (наименьший), либо предварительно просканировать с некоторым шагом рассматриваемую область, чтобы выделить из нее подобласть наибольших (наименьших) значений функции и осуществить поиск глобального экстремума, уже находясь в его окрестности. Последний путь таит в себе некоторую опасность уйти в зону другого локального экстремума, но часто может быть предпочтительнее из соображений экономии времени.

Для поиска локальных экстремумов имеются две встроенные функции, которые могут применяться как в пределах вычислительного блока, так и автономно.

Minimize$(f, x_1,..., x_m)$ — вектор значений аргументов, при которых функция f достигает минимума;

Maximize$(f, x_1,..., x_m)$ — вектор значений аргументов, при которых функция f достигает максимума;

$f(x_1,..., x_m)$ — функция;

$x_1,..., x_m$ — аргументы, по которым производится минимизация (максимизация).

Всем аргументам функции $f(x_1,..., x_m)$ предварительно следует присвоить некоторые значения, причем для тех переменных, по которым производится минимизация, они будут восприниматься как начальные приближения.

На рисунке 2.2 показаны результаты поиска минимума функции на заданном промежутке при разных начальных значениях аргумента.

$f(x) := x^4 + 5 \cdot x^3 - 10 \cdot x$

$x := -1$

$\text{Minimize}(f, x) = -3.552$

$x := 1$

$\text{Minimize}(f, x) = 0.746$

Рис. 2.2

На рисунке 2.3 показаны результаты поиска максимума функции на заданном промежутке при разных начальных значениях аргумента.

$f(x) := x^4 + 5 \cdot x^3 - 10 \cdot x$

$x := -1$

$\text{Maximize}(f, x) = -0.944$

$x := 10$

$\text{Maximize}(f, x) =$

Рис. 2.3

Как видно из примеров, существенное влияние на результат оказывает выбор начального приближения, в зависимости от чего в качестве ответа выдаются различные локальные экстремумы. В последнем случае численный метод вообще не справляется с задачей (функция выделена красным цветом), поскольку начальное приближение выбрано далеко от области локального максимума, и поиск решения уходит в сторону увеличения ***f (x)***.

2.2 Условный экстремум

В задачах на условный экстремум функции минимизации и максимизации должны быть включены в вычислительный блок, т. е. им должно предшествовать ключевое слово **Given**. В промежутке между **Given** и функцией поиска экстремума с помощью булевых операторов записываются логические выражения (неравенства, уравнения), задающие ограничения на значения аргументов минимизируемой функции. На рисунке 2.4 показаны примеры поиска условного экстремума на различных интервалах, определенных неравенствами.

$f(x) := x^4 + 5 \cdot x^3 - 10 \cdot x$ $x := 1$ Given $-5 < x < -2$ $\text{Minimize}(f, x) = -3.552$

$f(x) := x^4 + 5 \cdot x^3 - 10 \cdot x$ $x := 1$ Given $x > 0$ $\text{Minimize}(f, x) = 0.746$

$f(x) := x^4 + 5 \cdot x^3 - 10 \cdot x$ $x := -10$ Given $-3 < x < 0$ $\text{Maximize}(f, x) = -0.944$

Рис. 2.4

2.3 Экстремум функции многих переменных

Программа может находить экстремум функции, содержащей до трех неизвестных при наличии или отсутствии дополнительных условий. Для нахождения экстремума строится целевая функция, которую надо минимизировать или максимизировать.

Вычисление экстремума функции многих переменных не несет принципиальных особенностей по сравнению с функциями одной переменной.

На рисунке 2.5 показано нахождение минимума функции двух переменных без дополнительных условий.

$x := 10 \qquad y := 1$

$z(x, y) := (x - 1.5)^2 + 2 \cdot (y - 2.5)^2$

$\text{Minimize}(z, x, y) = \begin{pmatrix} 1.5 \\ 2.5 \end{pmatrix}$

Рис. 2.5

На рисунке 2.6 показано нахождение минимума функции двух переменных с дополнительными условиями.

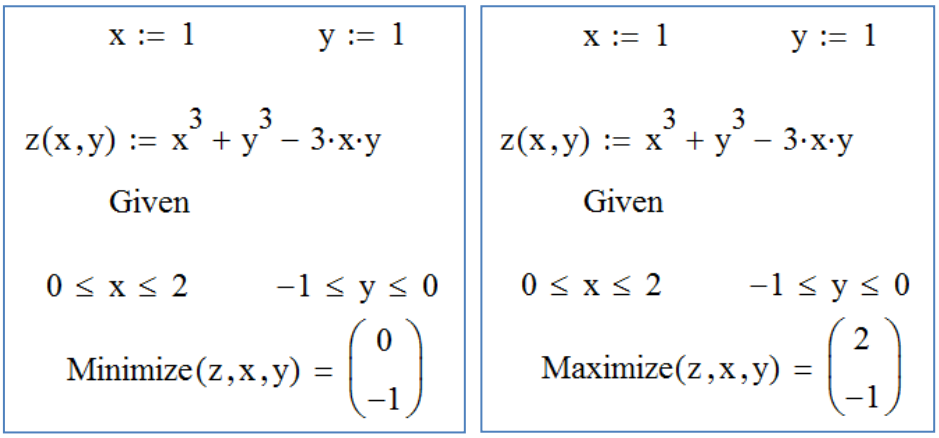

Рис. 2.6

2.4 Линейное программирование

Задачи поиска условного экстремума функции многих переменных часто встречаются в экономических расчетах для минимизации издержек, финансовых рисков, максимизации прибыли и т. п. Целый класс экономических задач оптимизации описывается системами линейных уравнений и неравенств. Они называются задачами линейного программирования.

Основной задачей линейного программирования является нахождение чисел $x_1, x_2, \ldots, x_n$, для которых целевая функция

$$f(x) = \sum_{i=1}^{n} c_i \cdot x_i$$

достигает наибольшего или наименьшего значения при наличии следующих ограничений:

$$\sum_{i=1}^{n} a_{i,j} \cdot x_i = b_j \,,\; j = 1,2,\ldots,m$$

$$x_i \geq 0 \,,\; i = 1,2,\ldots,n$$

где c_i , $a_{i,j}$, b_j - заданные числа.

В качестве примера рассмотрим типичную транспортную задачу.

Пусть имеется n предприятий-изготовителей, выпускающих продукцию в количестве $b_0, \ldots, b_{n-1}$ тонн. Эту продукцию требуется доставить m потребителям в количестве $a_0, \ldots, a_{m-1}$ тонн каждому. Известна стоимость перевозки продукции от i-го производителя к j-му потребителю $c_{i,j}$.

Необходимо составить оптимальное распределение соответствующего товаропотока $x_{i,j}$ с точки зрения минимизации транспортных расходов.

На рисунке 2.7 показан пример решения транспортной задачи для двух заводов-изготовителей и трех потребителей.

$$a := \begin{pmatrix} 145 \\ 210 \\ 160 \end{pmatrix} \qquad b := \begin{pmatrix} 237 \\ 278 \end{pmatrix}$$

$$m := rows(a) \qquad n := rows(b)$$

$$c := \begin{pmatrix} 11.5 & 7 & 12 \\ 6.2 & 10 & 9 \end{pmatrix}$$

$$f(x) := \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \left(c_{i,j} \cdot x_{i,j}\right)$$

$$x_{n-1,m-1} := 0$$

Given

$$x_{0,0} + x_{1,0} = a_0 \qquad x_{0,0} \geq 0 \qquad x_{1,0} \geq 0$$

$$x_{0,1} + x_{1,1} = a_1 \qquad x_{0,1} \geq 0 \qquad x_{1,1} \geq 0$$

$$x_{0.2} + x_{1.2} = a_2 \qquad x_{0,2} \geq 0 \qquad x_{1,2} \geq 0$$

$$x_{0,0} + x_{0,1} + x_{0,2} = b_0$$

$$x_{1,0} + x_{1,1} + x_{1,2} = b_1$$

$$rez := Minimize(f, x)$$

$$rez = \begin{pmatrix} 0 & 210 & 27 \\ 145 & 0 & 133 \end{pmatrix}$$

Рис. 2.7

Условия, выражающие неотрицательность товаропотока, и равенства, задающие сумму произведенной каждым предприятием продукции и сумму заказов каждого потребителя, находятся после ключевого слова Given. Решение, присвоенное матричной переменной **rez**, выведено в последней строке вместе с соответствующей суммой затрат.

В строке, предшествующей ключевому слову **Given**, определяются нулевые начальные значения для x созданием нулевого элемента матрицы $x_{n-1,m-1}$.

ГЛАВА 3 ОБРАБОТКА ДАННЫХ

Когда имеется выборка экспериментальных данных, то она, чаще всего, представляется в виде массива, состоящего из пар чисел (x_i, y_i). Поэтому возникает задача аппроксимации дискретной зависимости $y(x)$ непрерывной функцией $f(x)$. Функция $f(x)$, в зависимости от специфики задачи, может отвечать различным требованиям:

– $f(x)$ должна проходить через точки (x_i, y_i), т.е. $f(x_i) = y_i$, $i = 1 \ldots n$. В этом случае говорят об интерполяции данных функцией $f(x)$ во внутренних точках между x_i, или экстраполяции за пределами интервала, содержащего все x_i;

– $f(x)$ должна некоторым образом (например, в виде определенной аналитической зависимости) приближать $y(x_i)$, не обязательно проходя через точки (x_i, y_i). Такова постановка задачи регрессии, которую во многих случаях также можно назвать сглаживанием данных;

– $f(x)$ должна приближать экспериментальную зависимость $y(x_i)$, учитывая к тому же, что данные (x_i, y_i) получены с некоторой погрешностью, выражающей шумовую компоненту измерений. При этом функция $f(x)$, с помощью того или иного алгоритма, уменьшает погрешность, присутствующую в данных (x_i, y_i). Такого типа задачи называют задачами фильтрации. Сглаживание — частный случай фильтрации.

Для построения интерполяции-экстраполяции в **MathCAD** имеются несколько встроенных функций, позволяющих "соединить" точки выборки данных (x_i, y_i) кривой разной степени гладкости. По определению интерполяция означает построение функции $A(X)$, аппроксимирующей зависимость $y(x)$ в промежуточных точках (между точками x_i). Поэтому интерполяцию еще по-другому называют аппроксимацией. В точках x_i значения интерполяционной функции должны совпадать с исходными данными, т.е. $A(x_i) = y(x_i)$.

3.1 Линейная интерполяция

Самый простой вид интерполяции — линейная, которая представляет искомую зависимость $A(X)$ в виде ломаной линии. Интерполирующая функция $A(X)$ состоит из отрезков прямых, соединяющих точки.

Для построения линейной интерполяции служит встроенная функция **linterp(x,y,t),** где: **x** — вектор действительных данных аргумента, **y** — вектор действительных данных значений функции того же размера, **t** — значение аргумента, при котором вычисляется значение интерполирующей функции.

Элементы вектора **x** должны быть определены в порядке возрастания.

На рисунке 3.1 показан пример линейной интерполяции при использовании встроенной функции **linterp** для набора дискретных данных и их совмещенный график.

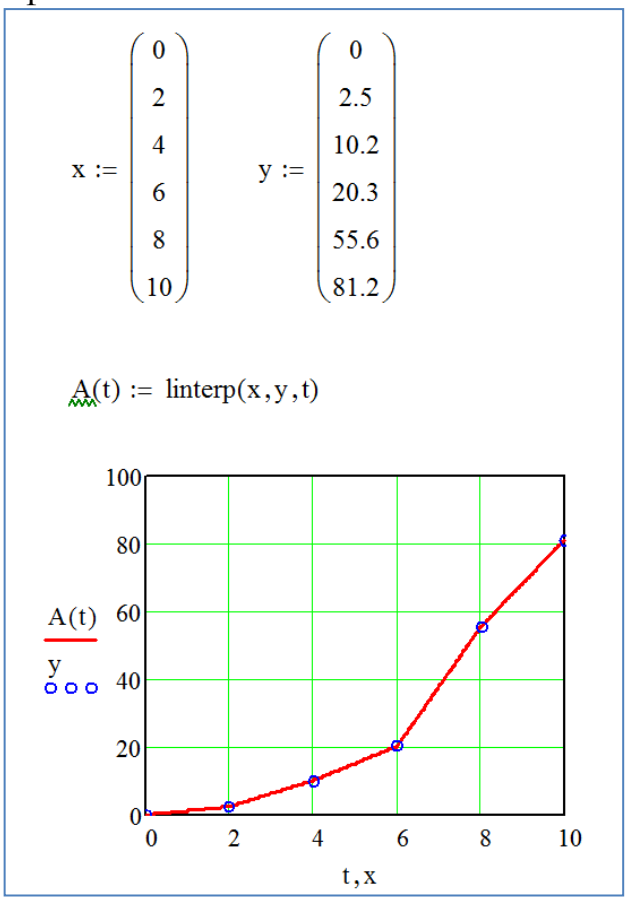

Рис. 3.1

Обратите внимание, что функция **A(t)** на графике имеет аргумент **t**, а не **x**. Это означает, что функция **A(t)** вычисляется не только при значениях аргумента (т. е. в шести точках), а при гораздо большем числе аргументов в интервале [0;10], что автоматически обеспечивает **MathCAD**.

3.2 Кубическая сплайн-интерполяция

В большинстве случаев желательно соединить экспериментальные точки не ломаной линией, а гладкой кривой. Лучше всего для этих целей подходит интерполяция кубическими сплайнами, т. е. отрезками кубических парабол.

Для этой цели используется встроенная функция **interp(s,x,y,t)**, где: **s** — вектор вторых производных, созданный одной из сопутствующих функций **cspline**, **pspline** или **lspline, x** — вектор действительных данных аргумента, элементы которого расположены в порядке возрастания, **y** — вектор действительных данных значений функции того же размера, **t** — значение аргумента, при котором вычисляется интерполирующая функция.

Сплайн-интерполяция в **MathCAD** реализована чуть сложнее линейной. Перед применением функции **interp** необходимо предварительно определить первый из ее аргументов — векторную переменную **s**. Делается это при помощи одной из трех встроенных функций: **lspline(x,y)** — вектор

значений коэффициентов линейного сплайна, **pspline(x,y)** — вектор значений коэффициентов квадратичного сплайна, **cspline(x,y)** — вектор значений коэффициентов кубического сплайна, **x**, **y** — векторы данных.

На рисунке 3.2 показан пример интерполяции исходных данных кубическим сплайном и их совмещенный график.

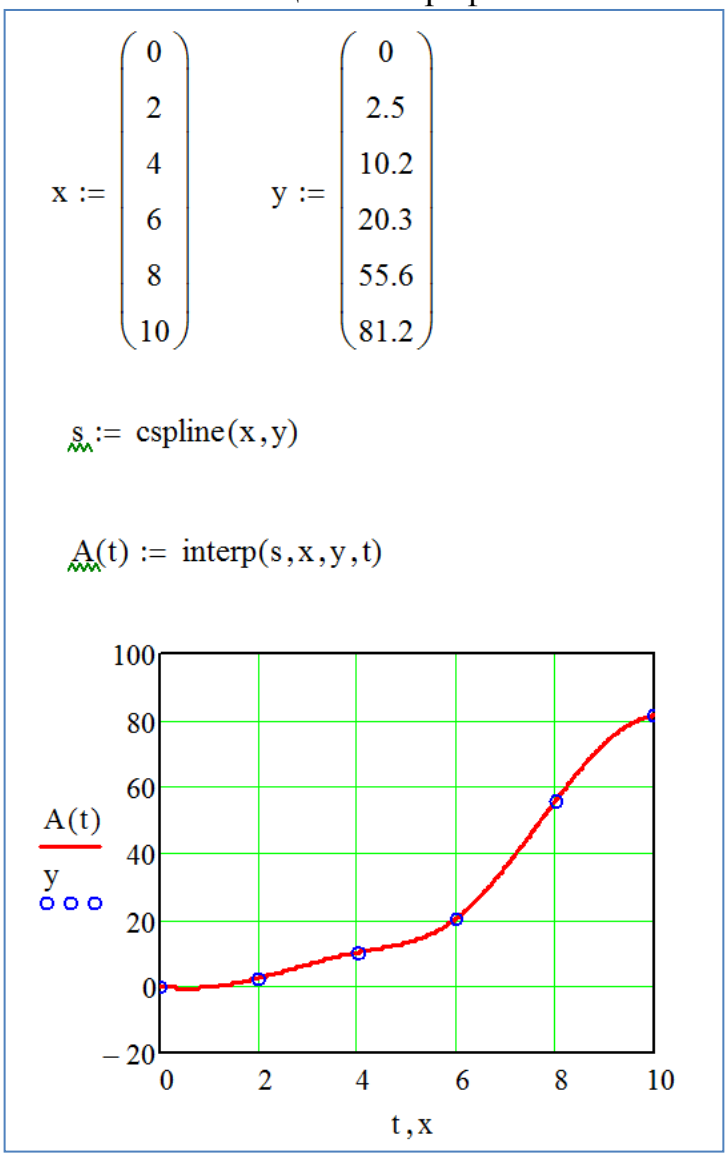

Рис. 3.2

Смысл сплайн-интерполяции заключается в том, что в промежутках между точками осуществляется аппроксимация в виде зависимости $A(t) = at^3 + bt^2 + ct + d$. Коэффициенты ***a***, ***b***, ***c***, ***d*** рассчитываются независимо для каждого промежутка, исходя из значений ***y*** в соседних точках. Этот процесс скрыт от пользователя, поскольку смысл задачи интерполяции состоит в выдаче значения ***A(t)*** в любой точке ***t***.

Выбор конкретной функции сплайновых коэффициентов влияет на интерполяцию вблизи конечных точек интервала.

ГЛАВА 4 ПРОГРАММИРОВАНИЕ

Программа – это выражение содержащее более одного утверждения.

Основными инструментами работы в **MathCAD** являются математические выражения, переменные и функции.

Программирование в **MathCAD** имеет ряд существенных преимуществ, которые в ряде случаев делают документ более простым и читаемым:

– возможность применения циклов и условных операторов дает большую гибкость построения выражений;

– простота создания функций и переменных, требующих нескольких простых шагов;

– возможность создания функций, содержащих закрытый для остального документа код, включая преимущества использования локальных переменных и обработку исключительных ситуаций (ошибок).

Для вставки программного кода в документы в **MathCAD** имеется специальная панель инструментов **Программирование** (рисунок 4.1). Большинство кнопок этой панели выполнено в виде текстового представления операторов программирования, поэтому их смысл легко понятен.

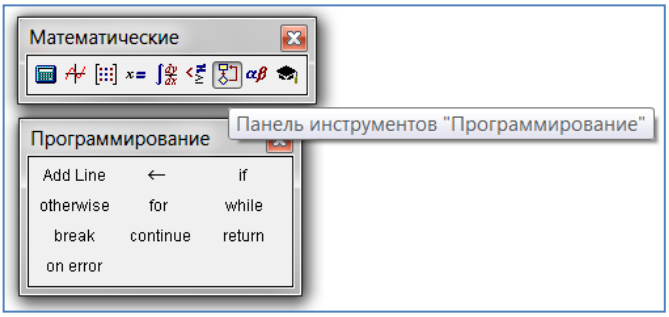

Рис. 4.1

Программный модуль обозначается в **MathCAD вертикальной чертой**, справа от которой последовательно записываются операторы языка программирования.

4.1. Создание программы (Add Line)

Чтобы создать программный модуль:

– Ввести имя функции и знак присваивания.

– Нажать на панели **Программирование** кнопку **Add Line** (Добавить линию). Появляется вертикальная черта и два местозаполнителя.

– Если приблизительно известно, сколько строк кода будет содержать программа, можно создать нужное количество линий повторным нажатием кнопки **Add Line** (при этом каждый раз добавляется один местозаполнитель) соответствующее число раз. На рисунке 4.2 показан результат трехкратного нажатия.

– В появившиеся местозаполнители ввести желаемый программный код, используя программные операторы.

После того как программный модуль полностью определен и ни один местозаполнитель не остался пустым, функция может использоваться обычным образом, как в численных, так и в символьных расчетах.

Не вводите с клавиатуры имена операторов. Для их вставки пользуйтесь панелью Программирование.

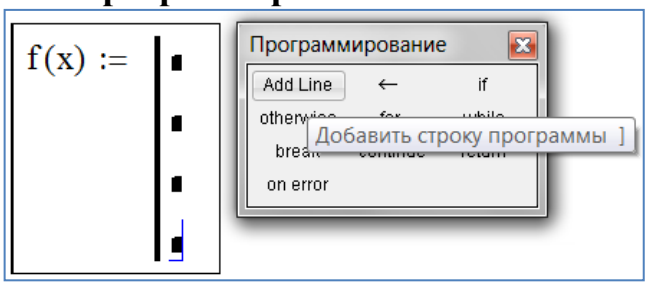

Рис. 4.2

Вставить строку программного кода в уже созданную программу можно в любой момент с помощью той же самой кнопки **Add Line**. Для этого следует предварительно поместить на нужное место внутри программного модуля курсор ввода.

Основной принцип создания программных модулей заключается в правильном расположении строк кода. Ориентироваться в их действии довольно легко, т. к. фрагменты кода одного уровня сгруппированы в программе с помощью вертикальных линий.

4.2 Оператор локального определения (←)

Язык программирования позволяет создавать внутри программных модулей локальные переменные, которые "не видны" извне, из других частей документа. Присваивание значения переменной производится с помощью оператора **Локальное определение**, который вставляется нажатием кнопки с изображением стрелки **←.**

Ни оператор присваивания :=, ни оператор вывода = в пределах программ не применяются.

Локальное присваивание иллюстрируется примером на рисунке 4.3. Переменная **z** существует только внутри программы, выделенной вертикальной чертой. Из других мест документа получить ее значение невозможно.

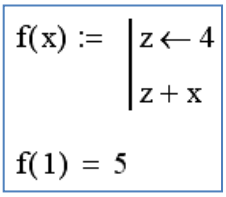

Рис. 4.3

4.3 Условный оператор (if, otherwise)

В условном операторе **if** сначала проверяется логическое выражение (условие) справа от него. Если оно истинно, выполняется выражение слева от

оператора **if**. Если оно ложно, выполнение программы продолжается переходом к следующей строке.

Оператор **otherwise** используется совместно с оператором **if** и указывает на выражение, которое будет выполняться, если проверяемое условие не выполняется.

На рисунке 4.4 показано задание $f(x)=|x-2|$ с помощью оператора **if** и примеры обращения к ней при разных значениях переменной.

$$f(x) := \begin{cases} x-2 & \text{if } x \geq 2 \\ 2-x & \text{if } x < 2 \end{cases}$$

$f(1) = 1 \qquad f(5) = 3$

$f(2) = 0$

Рис. 4.4

На рисунке 4.5 пример построения графика кусочно-заданной функции

$$f(x)=\begin{cases} \dfrac{8}{x} & \textit{если } x \le -2 \\ x^3+4 & \textit{если } -2 < x \le 0 \\ \dfrac{4}{x^2+1} & \textit{если } x > 0 \end{cases}$$ на промежутке $x \in [-15;15]$.

$$f(x) := \begin{cases} \dfrac{8}{x} & \text{if } x \le -2 \\ x^3+4 & \text{if } -2 < x \le 0 \\ \dfrac{4}{x^2+1} & \text{otherwise} \end{cases}$$

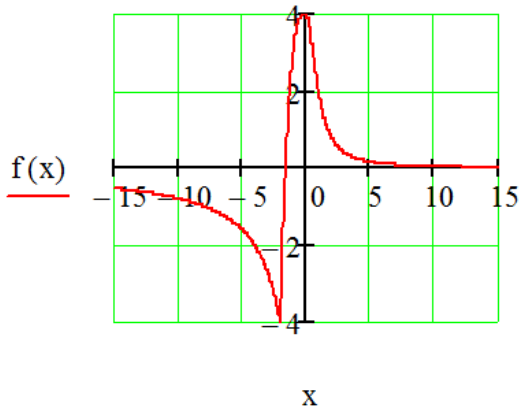

Рис. 4.5

4.4 Операторы цикла (for, while)

В языке программирования имеются два оператора цикла: **for** и **while**. Первый из них дает возможность организовать цикл по некоторой переменной, заставляя ее пробегать заданный диапазон значений. Второй создает цикл с выходом из него по некоторому логическому условию.

На рисунке 4.6 показана вставка в программу оператора цикла **for**.

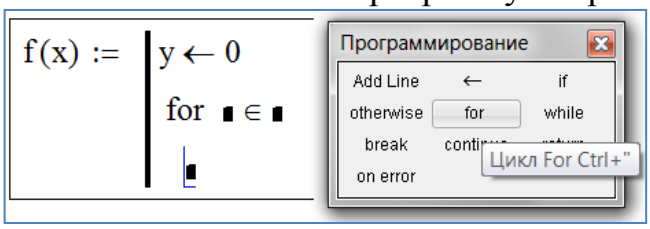

Рис. 4.6

Диапазон значений переменной в условии цикла **for** можно задать как с помощью дискретной переменной, так и с помощью вектора.

На рисунке 4.7 показан оператор цикла **for** с дискретной переменной.

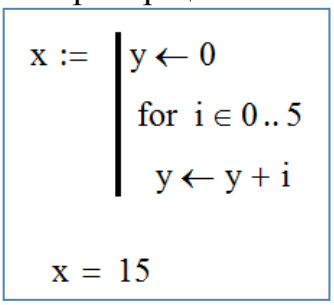

Рис. 4.7

На рисунке 4.8 показан оператор цикла **for** с вектором.

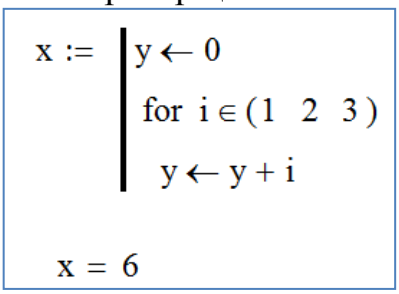

Рис. 4.8

На рисунке 4.9 показан оператор цикла **while**.

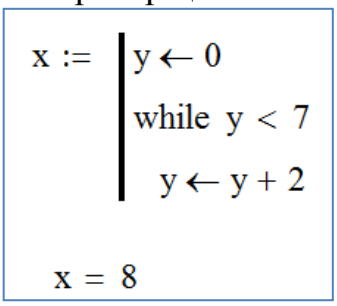

Рис. 4.9

Иногда необходимо досрочно завершить цикл, т. е. не по условию в его заголовке, а в некоторой строке в теле цикла. Для этого предназначен оператор **break**.

На рисунке 4.10 показан пример использования оператора **break** внутри цикла **for**.

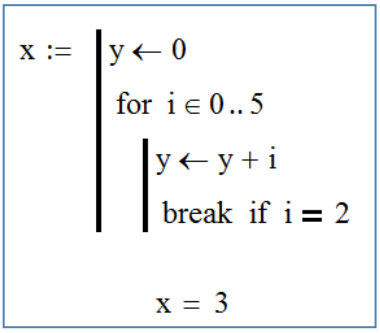

Рис. 4.10

На рисунке 4.11 показан пример использования оператора **break** внутри цикла **while**.

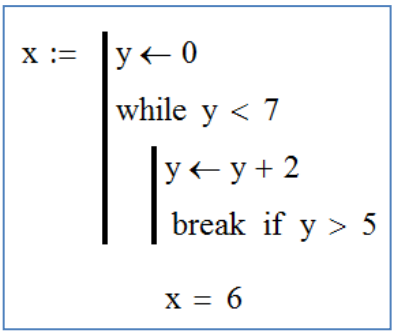

Рис. 4.11

Чтобы четче обозначить границы завершения тела цикла, в его конце может использоваться дополнительная строка с оператором **continue**.

4.5 Возврат значения (return)

Если для определения переменной или функции применяется программный модуль, то его строки исполняются последовательно при вычислении в документе этой переменной или функции. Соответственно, по мере выполнения программы рассчитываемый результат претерпевает изменения. В качестве окончательного результата выдается последнее присвоенное значение. Чтобы подчеркнуть возврат программным модулем определенного значения, можно взять за правило делать это в последней строке программного модуля.

На рисунке 4.12 возврат значения обозначен явно в последней строке программы.

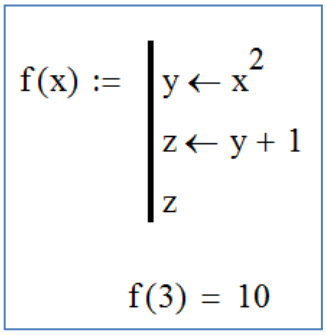

Рис. 4.12

Вместе с тем, можно прервать выполнение программы в любой ее точке (например, с помощью условного оператора) и выдать некоторое значение, применив оператор **return**. В этом случае при выполнении указанного условия значение, введенное в местозаполнитель после **return**, возвращается в качестве результата, а никакой другой код больше не выполняется (рисунок 4.13).

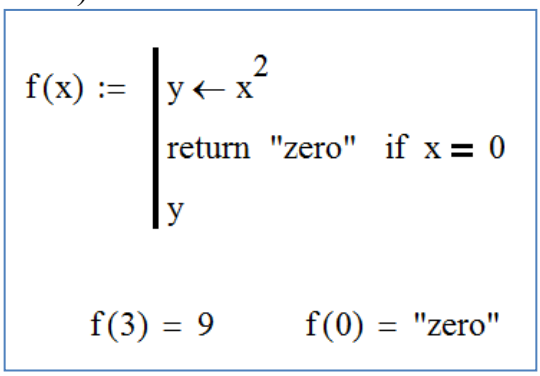

Рис. 4.13

4.6 Перехват ошибок (on error)

Программирование позволяет осуществлять дополнительную обработку ошибок. Если пользователь предполагает, что выполнение кода в каком-либо месте программного модуля способно вызвать ошибку (например, деление на ноль), то эту ошибку можно перехватить с помощью оператора **on error**. Чтобы вставить его в программу, надо поместить линии ввода в ней в нужное положение и нажать кнопку с именем оператора **on error**. В результате появится строка с двумя местозаполнителями и оператором **on error** посередине (рисунок 4.14).

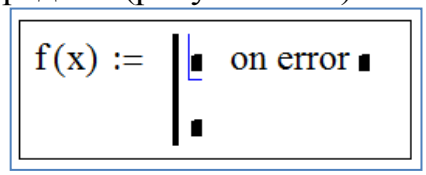

Рис. 4.14

В правом местозаполнителе следует ввести выражение, которое должно выполняться в данной строке программы. В левом — выражение, которое будет выполнено вместо правого выражения, если при выполнении последнего возникнет ошибка.

На рисунке 4.15 показан перехват ошибки деления на ноль.

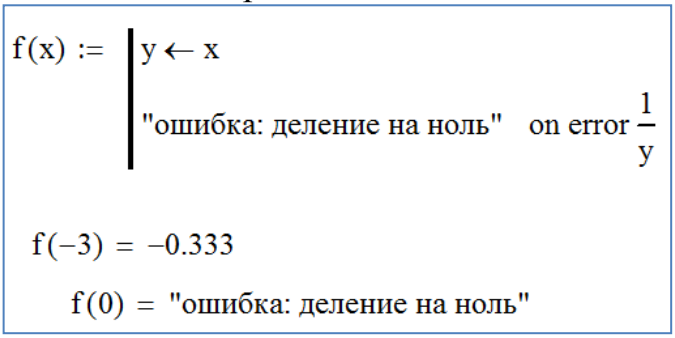

Рис. 4.15

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) **В.Ф. Очков.** MathCAD 14 для студентов и инженеров: русская версия. С-Пб.: BHV, 2009.

2) **В.А. Охорзин.** Прикладная математика в системе MathCAD Учебное пособие. 3-е изд. С-Пб.: Лань, 2009, 352с.

3) **Д.Гурский** Вычисления в MathCAD 12. С-Пб: Питер, 2006, 544с.

4) **Д.В. Кирьянов.** Самоучитель MathCAD 13. С-Пб: БХВ-Петербург, 2006, 528 с.

5) **А.М. Половко, И.В. Ганичев.** MathCAD для студента. С-Пб: БХВ-Петербург, 2006.