```
    Brush.Style := bsClear; // область вывода текста не закрашивается

    TextOut(10, 10, 'Borland Delphi 6');
  end;
```

После вывода текста методом TextOut указатель вывода (карандаш) перемещается в правый верхний угол области вывода текста.

Иногда требуется вывести какой-либо текст после сообщения, длина которого во время разработки программы неизвестна. Например, это может быть слово "руб." после значения числа, записанного прописью. В этом случае необходимо знать координаты правой границы уже выведенного текста. Координаты правой границы текста, выведенного методом TextOut, можно получить, обратившись к свойству PenPos.

Следующий фрагмент программы демонстрирует возможность вывода строки текста при помощи двух инструкций TextOut.

```
with Form1.Canvas do
  begin
    TextOut(10, 10, 'Borland ');
    TextOut(PenPos.X, PenPos.Y, 'Delphi 6');
  end;
```

Методы вычерчивания графических примитивов

Любая картинка, чертеж, схема могут рассматриваться как совокупность графических *примитивов*: точек, линий, окружностей, дуг и др. Таким образом, для того чтобы на экране появилась нужная картинка, программа должна обеспечить вычерчивание (вывод) графических примитивов, составляющих эту картинку.

Вычерчивание графических примитивов на поверхности компонента (формы или области вывода иллюстрации) осуществляется применением соответствующих методов к свойству Canvas этого компонента.

Линия

Вычерчивание прямой линии осуществляет метод LineTo, инструкция вызова которого в общем виде выглядит следующим образом:

```
Компонент.Canvas.LineTo(x, y)
```

Метод LineTo вычерчивает прямую линию от текущей позиции карандаша в точку с координатами, указанными при вызове метода.

Начальную точку линии можно задать, переместив карандаш в нужную точку графической поверхности. Сделать это можно при помощи метода MoveTo, указав в качестве параметров координаты нового положения карандаша.

Вид линии (цвет, толщина и стиль) определяется значениями свойств объекта Pen графической поверхности, на которой вычерчивается линия.

Довольно часто результаты расчетов удобно представить в виде графика. Для большей информативности и наглядности графики изображают на фоне координатных осей и оцифрованной сетки. В листинге 10.2 приведен текст программы, которая на поверхность формы выводит координатные оси и оцифрованную сетку (рис. 10.4).

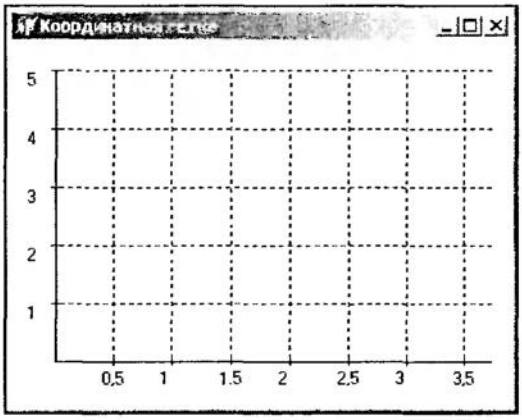

Рис. 10.4. Форма приложения **Координатная сетка**

Листинг 10.2. Оси координат и оцифрованная сетка

```
unit grid_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```
var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormPaint(Sender: TObject);
var
 x0,y0:integer; // координаты начала координатных осей
 dx,dy:integer; // шаг координатной сетки (в пикселах)
 h,w:integer;   // высота и ширина области вывода координатной сетки
 x,y:integer;

 lx,ly:real;    // метки (оцифровка) линий сетки по X и Y
 dlx,dly:real;  // шаг меток (оцифровки) линий сетки по X и Y
 cross:integer; // счетчик неоцифрованных линий сетки
 dcross:integer;// количество неоцифрованных линий между оцифрованными
begin
 x0:=30; y0:=220; // оси начинаются в точке (40,250)
 dx:=40; dy:=40;  // шаг координатной сетки 40 пикселов
 dcross:=1;       // помечать линии сетки X: 1 - каждую;
                  //                         2 - через одну;
                  //                         3 - через две;
 dlx:=0.5;        // шаг меток оси X
 dly:=1.0;        // шаг меток оси Y, метками будут: 1, 2, 3 и т. д.

 h:=200;
 w:=300;

 with form1.Canvas do
 begin
   cross:=dcross;
   MoveTo(x0,y0); LineTo(x0,y0-h);  // ось X
   MoveTo(x0,y0); LineTo(x0+w,y0);  // ось Y

   // засечки, сетка и оцифровка по оси X
   x:=x0+dx;
   lx:=dlx;
   repeat
     MoveTo(x,y0-3);LineTo(x,y0+3);  // засечка
     cross:=cross-1;
     if cross = 0 then   // оцифровка
     begin
       TextOut(x-8,y0+5,FloatToStr(lx));
       cross:=dcross;
     end;
```

```
      Pen.Style:=psDot;
      MoveTo(x,y0-3);LineTo(x,y0-h);  // линия сетки
      Pen.Style:=psSolid;
      lx:=lx+dlx;
      x:=x+dx;
    until (x>x0+w);

    // засечки, сетка и оцифровка по оси Y
    y:=y0-dy;
    ly:=dly;
    repeat
      MoveTo(x0-3,y);LineTo(x0+3,y);     // засечка
      TextOut(x0-20,y,FloatToStr(ly));  // оцифровка
      Pen.Style:=psDot;
      MoveTo(x0+3,y); LineTo(x0+w,y);     // линия сетки
      Pen.Style:=psSolid;
      y:=y-dy;
      ly:=ly+dly;
    until (y<y0-h);
  end;
end;

end.
```

Особенность приведенной программы заключается в том, что она позволяет задавать шаг сетки и оцифровку. Кроме того, программа дает возможность оцифровывать не каждую линию сетки оси X, а через одну, две, три и т. д. Сделано это для того, чтобы предотвратить возможные наложения изображений чисел оцифровки друг на друга в случае, если эти числа состоят из нескольких цифр.

Ломаная линия

Метод `Polyline` вычерчивает ломаную линию. В качестве параметра метод получает массив типа `TPoint`. Каждый элемент массива представляет собой запись, поля `(x,y)` которой содержат координаты точки перегиба ломаной. Метод `Polyline` вычерчивает ломаную линию, последовательно соединяя прямыми точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д.

В качестве примера использования метода `Polyline` в листинге 10.3 приведена процедура, которая выводит график изменения некоторой величины. Предполагается, что исходные данные находятся в доступном процедуре массиве `Data` (тип `Integer`).

Листинг 10.3. Построение графика с использованием метода Polyline

```
procedure TForm1.Button1Click(Sender: TObject);

var
   gr: array[1..50] of TPoint; // график – ломаная линия
   x0,y0: integer;  // координаты точки начала координат
   dx,dy: integer;  // шаг координатной сетки по осям X и Y

   i: integer;

begin
   x0 := 10; y0 := 200;
   dx :=5; dy := 5;
   // заполним массив gr
   for i:=1 to 50 do
   begin
      gr[i].x := x0 + (i-1)*dx;
      gr[i].y := y0 – Data[i]*dy;
   end;

   // строим график
   with form1.Canvas do
   begin
     MoveTo(x0,y0); LineTo(x0,10);     // ось Y
     MoveTo(x0,yo); LineTo(200,y0);    // ось X
     Polyline(gr);                     // график
   end;
end;
```

Метод Polyline можно использовать для вычерчивания замкнутых контуров. Для этого надо, чтобы первый и последний элементы массива содержали координаты одной и той же точки. В качестве примера использования метода Polyline для вычерчивания замкнутого контура в листинге 10.4 приведена программа, которая на поверхности диалогового окна, в точке нажатия левой кнопки мыши, вычерчивает контур пятиконечной звезды (рис. 10.5).

Листинг 10.4

```
unit Stars_;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;
```

```
type
  TForm1 = class(TForm)
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure StarLine(x0,y0,r: integer; Canvas: TCanvas);
    // x0,y0 - координаты центра звезды
    // r - радиус звезды
var
    p : array[1..11] of TPoint; // массив координат лучей и впадин
    a: integer;    // угол между осью OX и прямой, соединяющей
                   // центр звезды и конец луча или впадину
    i: integer;
begin
    a := 18; // строим от правого гор. луча
    for i:=1 to 10 do
       begin
          if (i mod 2 = 0) then
             begin // впадина
               p[i].x := x0+Round(r/2*cos(a*2*pi/360));
               p[i].y:=y0-Round(r/2*sin(a*2*pi/360));
             end
          else
             begin // луч
               p[i].x:=x0+Round(r*cos(a*2*pi/360));
               p[i].y:=y0-Round(r*sin(a*2*pi/360));
             end;
          a := a+36;

       end;

    p[11].X := p[1].X; // чтобы замкнуть контур звезды
    p[11].Y := p[1].Y;
```

```
    Canvas.Polyline(p); // начертить звезду
end;

// нажатие кнопки мыши
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
                               Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft  // нажата левая кнопка?
     then Form1.Canvas.Pen.Color := clRed
     else Form1.Canvas.Pen.Color := clGreen;
  StarLine(x, y, 30,Form1.Canvas);
end;

end.
```

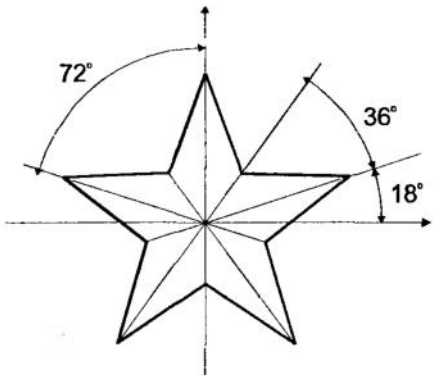

Рис. 10.5. Звезда

Звезду вычерчивает процедура `StarLine`, которая в качестве параметров получает координаты центра звезды и холст, на котором звезда должна быть выведена. Сначала вычисляются координаты концов и впадин звезды, которые записываются в массив `p`. Затем этот массив передается в качестве параметра методу `Polyline`.

Примечание

Обратите внимание, что размер массива `p` на единицу больше, чем количество концов и впадин звезды, и что значения первого и последнего элементов массива совпадают.

Окружность и эллипс

Метод `Ellipse` вычерчивает эллипс или окружность, в зависимости от значений параметров. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Объект.Canvas.Ellipse(x1,y1, x2,y2)
```

где:

Объект — имя объекта (компонента), на поверхности которого выполняется вычерчивание;

x1, *y1*, *x2*, *y2* — координаты прямоугольника, внутри которого вычерчивается эллипс или, если прямоугольник является квадратом, окружность (рис. 10.6).

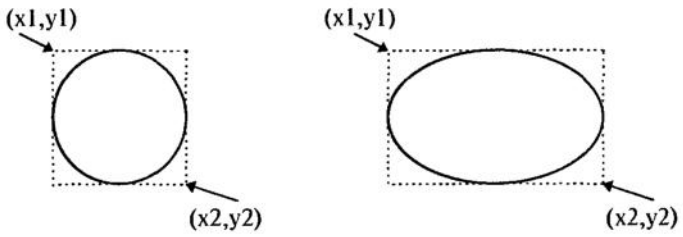

Рис. 10.6. Значения параметров метода `Ellipse` определяют вид геометрической фигуры

Цвет, толщина и стиль линии эллипса определяются значениями свойства `Pen`, а цвет и стиль заливки области внутри эллипса — значениями свойства `Brush` поверхности (`Canvas`), на которую выполняется вывод.

Дуга

Вычерчивание дуги выполняет метод `Arc`, инструкция вызова которого в общем виде выглядит следующим образом:

```
Объект.Canvas.Arc(x1,y1,x2,y2,x3,y3,x4,y4)
```

где:

`x1`, `y1`, `x2`, `y2` — параметры, определяющие эллипс (окружность), частью которого является вычерчиваемая дуга;

`x3`, `y3` — параметры, определяющие начальную точку дуги;

`x4`, `y4` — параметры, определяющие конечную точку дуги.

Начальная (конечная) точка — это точка пересечения границы эллипса и прямой, проведенной из центра эллипса в точку с координатами `x3` и `y3` (`x4`, `y4`). Дуга вычерчивается против часовой стрелки от начальной точки к конечной (рис. 10.7).

Цвет, толщина и стиль линии, которой вычерчивается дуга, определяются значениями свойства `Pen` поверхности (`Canvas`), на которую выполняется вывод.

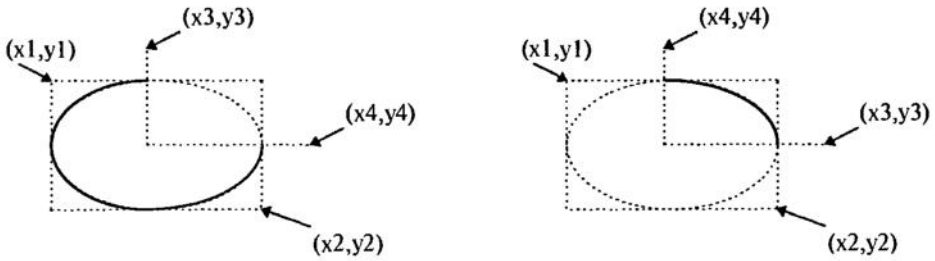

Рис. 10.7. Значения параметров метода Arc определяют дугу как часть эллипса (окружности)

Прямоугольник

Прямоугольник вычерчивается методом Rectangle, инструкция вызова которого в общем виде выглядит следующим образом:

```
Объект.Canvas.Rectangle(x1,y1,x2,y2)
```

где:

Объект — имя объекта (компонента), на поверхности которого выполняется вычерчивание;

x1, y1 и x2, y2 — координаты левого верхнего и правого нижнего углов прямоугольника.

Метод RoundRec тоже вычерчивает прямоугольник, но со скругленными углами. Инструкция вызова метода RoundRec выглядит так:

```
Объект.Canvas.RoundRec(x1,y1,x2,y2,x3,y3)
```

где:

x1, y1, x2, y2 — параметры, определяющие положение углов прямоугольника, в который вписывается прямоугольник со скругленными углами;

x3 и y3 — размер эллипса, одна четверть которого используется для вычерчивания скругленного угла (рис. 10.8).

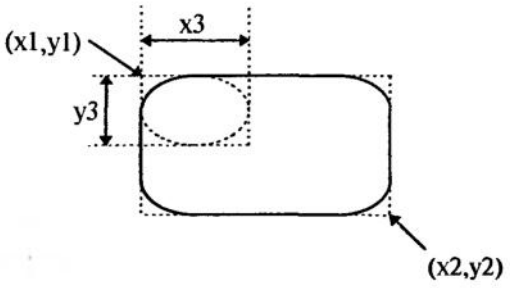

Рис. 10.8. Метод RoundRec вычерчивает прямоугольник со скругленными углами

Вид линии контура (цвет, ширина и стиль) определяется значениями свойства Pen, а цвет и стиль заливки области внутри прямоугольника — значениями свойства Brush поверхности (Canvas), на которой прямоугольник вычерчивается.

Есть еще два метода, которые вычерчивают прямоугольник, используя в качестве инструмента только кисть (Brush). Метод FillRect вычерчивает закрашенный прямоугольник, а метод FrameRect — только контур. У каждого из этих методов лишь один параметр — структура типа TRect. Поля структуры TRect содержат координаты прямоугольной области, они могут быть заполнены при помощи функции Rect.

Ниже в качестве примера использования методов FillRect и FrameRect приведена процедура, которая на поверхности формы вычерчивает прямоугольник с красной заливкой и прямоугольник с зеленым контуром.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   r1, r2: TRect; // координаты углов прямоугольников

begin
    // заполнение полей структуры
    // зададим координаты углов прямоугольников
    r1 := Rect(20,20,60,40);
    r2 := Rect(10,10,40,50);

    with form1.Canvas do
      begin
        Brush.Color := clRed;
        FillRect(r1);          // закрашенный прямоугольник
        Brush.Color := clGreen;
        FrameRect(r2);         // только граница прямоугольника
      end;
end;
```

Многоугольник

Метод Polygon вычерчивает многоугольник. В качестве параметра метод получает массив типа TPoint. Каждый элемент массива представляет собой запись, поля (x,y) которой содержат координаты одной вершины многоугольника. Метод Polygon вычерчивает многоугольник, последовательно соединяя прямыми линиями точки, координаты которых находятся в массиве: первую со второй, вторую с третьей, третью с четвертой и т. д. Затем соединяются последняя и первая точки.

Цвет и стиль границы многоугольника определяются значениями свойства Pen, а цвет и стиль заливки области, ограниченной линией границы, — зна-

чениями свойства Brush, причем область закрашивается с использованием текущего цвета и стиля кисти.

Ниже приведена процедура, которая, используя метод Polygon, вычерчивает треугольник:

```
procedure TForm1.Button2Click(Sender: TObject);
var
   pol: array[1..3] of TPoint; // координаты точек треугольника
begin
   pol[1].x := 10;
   pol[1].y := 50;
   pol[2].x := 40;
   pol[2].y := 10;
   pol[3].x := 70;
   pol[3].y := 50;
   Form1.Canvas.Polygon(pol);
end;
```

Сектор

Метод Pie вычерчивает сектор эллипса или круга. Инструкция вызова метода в общем виде выглядит следующим образом:

```
Объект.Canvas.Pie(x1,y1,x2,y2,x3,y3,x4,y4)
```

где:

x1, y1, x2, y2 — параметры, определяющие эллипс (окружность), частью которого является сектор;

x3, y3, x4, y4 — параметры, определяющие координаты конечных точек прямых, являющихся границами сектора.

Начальные точки прямых совпадают с центром эллипса (окружности). Сектор вырезается против часовой стрелки от прямой, заданной точкой с координатами (x3, y3), к прямой, заданной точкой с координатами (x4, y4) (рис. 10.9).

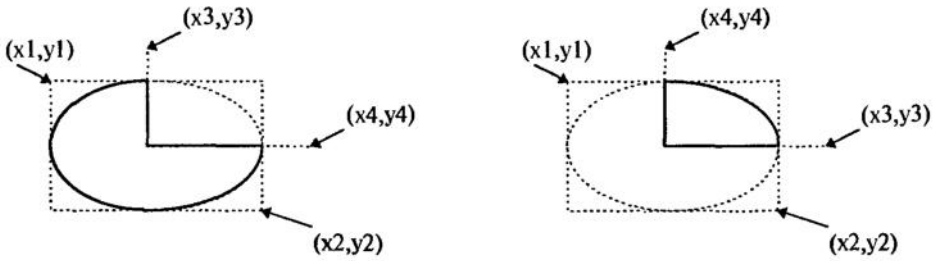

Рис. 10.9. Значения параметров метода Pie определяют сектор как часть эллипса (окружности)

Точка

Поверхности, на которую программа может осуществлять вывод графики, соответствует объект `Canvas`. Свойство `Pixels`, представляющее собой двумерный массив типа `TColor`, содержит информацию о цвете каждой точки графической поверхности. Используя свойство `Pixels`, можно задать требуемый цвет для любой точки графической поверхности, т. е. "нарисовать" точку. Например, инструкция

```
Form1.Canvas.Pixels[10,10]:=clRed
```

окрашивает точку поверхности формы в красный цвет.

Размерность массива `Pixels` определяется размером графической поверхности. Размер графической поверхности формы (рабочей области, которую также называют *клиентской*) задается значениями свойств `ClientWidth` и `ClientHeight`, а размер графической поверхности компонента `Image` — значениями свойств `Width` и `Height`. Левой верхней точке рабочей области формы соответствует элемент `Pixels[0,0]`, а правой нижней — `Pixels[ClientWidth - 1,ClientHeight - 1]`.

Свойство `Pixels` можно использовать для построения графиков. График строится, как правило, на основе вычислений по формуле. Границы диапазона изменения аргумента функции являются исходными данными. Диапазон изменения значения функции может быть вычислен. На основании этих данных можно вычислить масштаб, позволяющий построить график таким образом, чтобы он занимал всю область формы, предназначенную для вывода графика.

Например, если некоторая функция `f(x)` может принимать значения от нуля до 1000, и для вывода ее графика используется область формы высотой в 250 пикселов, то масштаб оси `Y` вычисляется по формуле: `m = 250/1000`. Таким образом, значению `f(x) = 1000` будет соответствовать точка области с координатой `Y = 0 (Y = 250 - f(x)× m = 250 - 1000×(250/1000))`, а значению `f(x) = 70` — с координатой `Y = 233 (Y = 250 - 70×(250/1000))`.

Примечание

Обратите внимание на то, что точное значение выражения `250 - 70× ×(250/1000)` равно `232,5`. Но т. к. индексом свойства `Pixels`, которое используется для вывода точки на поверхность `Canvas`, может быть только целое значение, то число 232,5 округляется к ближайшему целому, которым является число 233.

Следующая программа, ее текст приведен в листинге 10.5, используя свойство `Pixels`, выводит график функции $y=2\sin(x)e^{x/5}$. Для построения графика используется вся доступная область формы, причем если во время работы программы пользователь изменит размер окна, то график будет выведен заново с учетом реальных размеров окна.

Листинг 10.5. График функции

```
unit grfunc_;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs;
type
  TForm1 = class(TForm)
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

// Функция, график которой надо построить
 Function f(x:real):real;
 begin
    f:=2*Sin(x)*exp(x/5);
 end;

// строит график функции
procedure GrOfFunc;
var
 x1,x2:real;     // границы изменения аргумента функции
 y1,y2:real;     // границы изменения значения функции
 x:real;         // аргумент функции
 y:real;         // значение функции в точке x
 dx:real;        // приращение аргумента
 l,b:integer;    // левый нижний угол области вывода графика
 w,h:integer;    // ширина и высота области вывода графика
 mx,my:real;     // масштаб по осям X и Y
 x0,y0:integer; // точка - начало координат
```

```
begin
 // область вывода графика
 l:=10;                       // X - координата левого верхнего угла
 b:=Form1.ClientHeight-20;    // Y - координата левого верхнего угла
 h:=Form1.ClientHeight-40;    // высота
 w:=Form1.Width-40;           // ширина

 x1:=0;      // нижняя граница диапазона аргумента
 x2:=25;     // верхняя граница диапазона аргумента
 dx:=0.01;   // шаг аргумента

 // найдем максимальное и минимальное значения
 //  функции на отрезке [x1,x2]
 y1:=f(x1); // минимум
 y2:=f(x1); // максимум
 x:=x1;
 repeat
   y := f(x);
   if y < y1 then y1:=y;
   if y > y2 then y2:=y;
   x:=x+dx;
 until (x >= x2);

 // вычислим масштаб
 my:=h/abs(y2-y1);  // масштаб по оси Y
 mx:=w/abs(x2-x1);  // масштаб по оси X

 // оси
 x0:=l;
 y0:=b-Abs(Round(y1*my));

 with form1.Canvas do
 begin
   // оси
   MoveTo(l,b);LineTo(l,b-h);
   MoveTo(x0,y0);LineTo(x0+w,y0);
   TextOut(l+5,b-h,FloatToStrF(y2,ffGeneral,6,3));
   TextOut(l+5,b,FloatToStrF(y1,ffGeneral,6,3));
   // построение графика
   x:=x1;
   repeat
     y:=f(x);
     Pixels[x0+Round(x*mx),y0-Round(y*my)]:=clRed;
     x:=x+dx;
   until (x >= x2);
 end;
end;
```

```
procedure TForm1.FormPaint(Sender: TObject);
begin
  GrOfFunc;
end;

// изменился размер окна программы
procedure TForm1.FormResize(Sender: TObject);
begin
  // очистить форму
  form1.Canvas.FillRect(Rect(0,0,ClientWidth,ClientHeight));
  // построить график
  GrOfFunc;
end;

end.
```

Основную работу выполняет процедура `GrOfFunc`, которая сначала вычисляет максимальное (`y2`) и минимальное (`y1`) значения функции на отрезке `[x1,x2]`. Затем, используя информацию о ширине (`Form1.ClientWidth - 40`) и высоте (`Form1.ClientHeight - 40`) области вывода графика, вычисляет масштаб по осям `X` (`mx`) и `Y` (`my`).

Высота и ширина области вывода графика определяется размерами рабочей (клиентской) области формы, т. е. без учета области заголовка и границ. После вычисления масштаба процедура вычисляет координату `Y` горизонтальной оси (`y0`) и вычерчивает координатные оси графика. Затем выполняется непосредственное построение графика (рис. 10.10).

Вызов процедуры `GrOfFunc` выполняют процедуры обработки событий `OnPaint` и `OnFormResize`. Процедура `TForm1.FormPaint` обеспечивает вычерчивание графика после появления формы на экране в результате запуска программы, а также после появления формы во время работы программы, например, в результате удаления или перемещения других окон, полностью или частично перекрывающих окно программы. Процедура `TForm1.FormResize` обеспечивает вычерчивание графика после изменения размера формы.

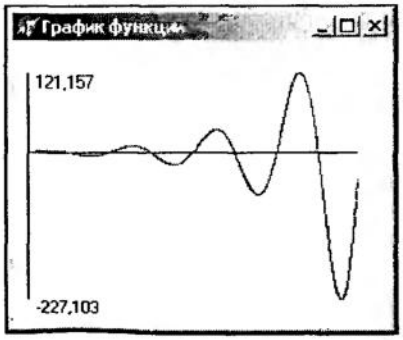

Рис. 10.10. График, построенный процедурой `GrOfFunc`