Практическая работа № 20, ПОСТРОЕНИЕ ТРЕХМЕРНОГО ГРАФИКА ФУНКЦИИ

Постановка задачи

Разработать программу, которая выполняет следующие действия:

1. Строит на плоскости трехмерный график заданной функции.

В программе необходимо предусмотреть:

1. контроль вводимой информации.

Ограничение:

1. формула функции задается непосредственно в программе.

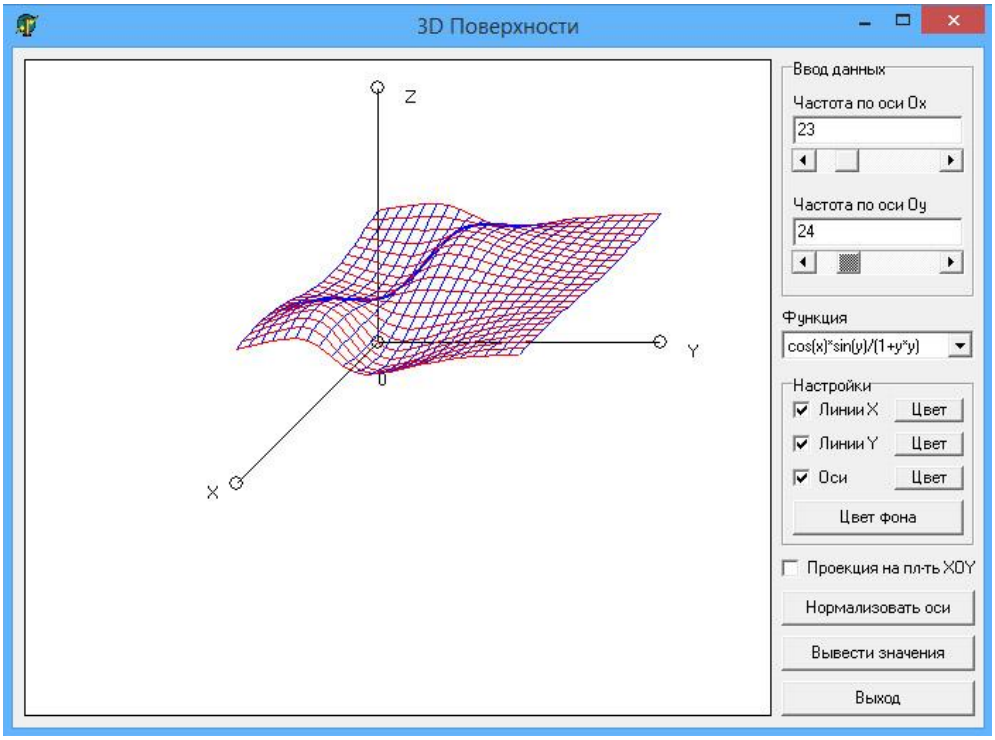


Рис.1

Новым в этой работе являются:

- представление трехмерной поверхности на плоскости.

Немного теории

График функции двух переменных Z = F(X,Y) можно представлять себе в виде некоторой поверхности в трехмерном пространстве. Для наглядного изображения поверхности используем метод сечений.

Проведем в пространстве плоскость Y = const, параллельную осям Ox и Oz. Тогда сечение данной плоскости с поверхностью функции есть некоторая кривая – график функции одной переменной X. (Правда, расположен этот график «под углом» к плоскости экрана). Изменяя значение константы с некоторым шагом, мы получим семейство таких кривых.

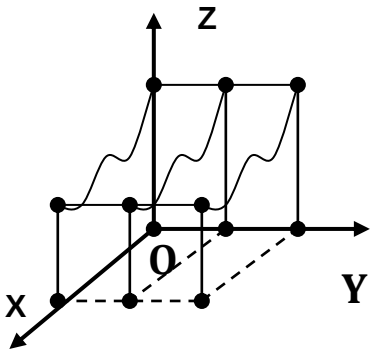


Рис.1

Построим второе семейство кривых, каждая из которых будет сечением поверхности плоскостью X = const. Наложение этих семейств кривых дают хорошее представление об исходной поверхности.

Примем, что значения X изменяются в пределах от Xmin до Xmax, Y – от Ymin до Ymax. Пределы изменения Z можно было бы вычислить точно, но для простоты определим их подбором при работе программы.

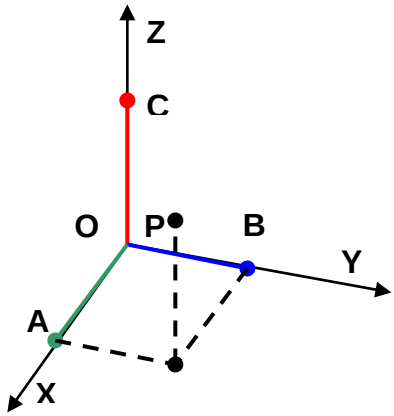


Рис.2

Основной вопрос, который нам необходимо решить: как изобразить точку P с координатами (X, Y, Z) на экране (Рис.2).

Построим сначала на экране изображения осей координат. Нам будет удобно откладывать оси OA, OB и OC от точки O, соответствующей координатам Xmin, Ymin и Zmin. Потребуем, чтобы при отображении на экран отрезок [Xmin, Xmax] накладывался на отрезок OA, отрезок [Ymin, Ymax] – на отрезок OB, а отрезок [Zmin, Zmax] – на отрезок OC.

Введем для этого коэффициенты Qa, Qb, Qc:

Qa = (X – Xmin) / (Xmax – Xmin)
Qb = (Y – Ymin) / (Ymax – Ymin)
Qc = (Z – Zmin) / (Zmax – Zmin)

Коэффициент Qa изменяется от 0 до 1, когда X «пробегает» от Xmin до Xmax, при этом точка X на экране «пробегает» вдоль отрезка OA.

Два других коэффициента связывают отрезки [Ymin, Ymax] и [Zmin, Zmax] с OB и OC соответственно.

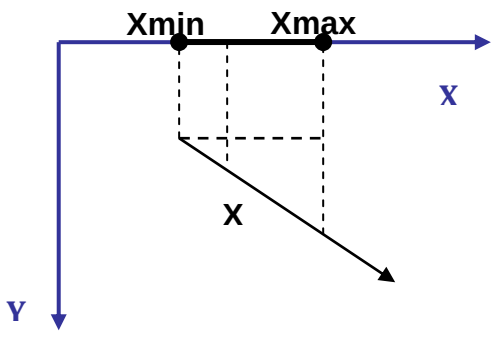
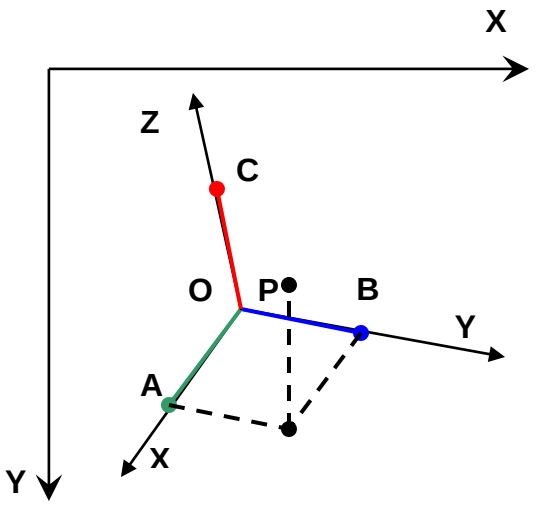


Рис.3

Обозначим через P координаты образа исходной точки (X, Y, Z) в экранной системе координат (Рис.2). Тогда

$$P = O + OX + OY + OZ = O + Qa \cdot OA + Qb \cdot OB + Qc \cdot OC$$



Для полной ясности запишем полученную формулу по координатам:

$$\begin{cases} P_x = O_x + Qa \cdot OA_x + Qb \cdot OB_x + Qc \cdot OC_x \\ P_y = O_y + Qa \cdot OA_y + Qb \cdot OB_y + Qc \cdot OC_y \end{cases}$$

План разработки программы

Поместим на форму область рисования PaintBoxl. Сам график будем рисовать как последовательность отрезков, конец каждого из которых совпадает с началом следующего. Можно использовать для этого методы MoveTo (P_x, P_y) – для получения начальной точки и LineTo (P_x, P_y) – для получения звена графика. Но для иллюстрации новых возможностей Delphi мы напишем свои методы, которые координаты точки экрана получают в виде структуры (записи) TPoint, описанной в Delphi следующим образом:

```
type TPoint = record
            X: Longint;
            Y: Longint;
               end;
```

Имена процедур оставим теми же – MoveTo и LineTo. Компилятор сумеет отличить их от «старых» процедур по количеству и типу параметров (в стандартном Паскале одинаковые имена процедур недопустимы!).

```
procedure MoveTo (A:TPoint);
  begin
PaintBoxl.Canvas.MoveTo(A.x,A.y);
  end;
procedure LineTo (A:TPoint);
  begin
PaintBoxl.Canvas.LineTo(A.x,A.y) ;
  end;
```

Однако разместить эти процедуры можно только там, где они будут иметь доступ к PaintBoxl, – например, непосредственно в процедуре рисования поверхности TForml.PaintBoxl Paint.

Напишем еще две функции, имеющие дело с TPoint, – функцию инициализации структуры и функцию получения вектора по его началу и концу.

```
function Init(x,y:Longint) : TPoint;
{функция  инициализации структуры}
var S:TPoint;
begin
  S.x:=x;
  S.y:=y;
  Init:=S;
end;
```

Вместо написания функции Init можно воспользоваться готовой, уже имеющейся в Delphi, функцией Point, выполняющей точно такие же действия.

```
function Vect(A,B:TPoint):TPoint;
{функция получения вектора по его началу и концу}
var S:TPoint;
begin
  S.x:=B.x-A.x;
  S.y:=B.y-A.y;
  Vect:=S;
end;
```

Пример использования функций:

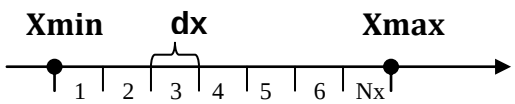
```
var A,B,V:TPoint;
begin
A := Init(100, 250) ;
B := Init(150, 200);
V := Vect(A, B);
          …
```

В результате запись V будет иметь компоненты V.x = 50, V.y = - 50.

Переменные, которые нам нужны, опишем в разделе **`var`** секции **`implementation`**:

```
var O,A,B,C,OA,OB,OC,Mouse:TPoint;
  iP,R:integer;
  Xmin,Xmax,Ymin,Ymax,Zmin,Zmax:double;
  nx,ny:word;
```

Все эти переменные, кроме Nx и Ny, уже упоминались выше. Параметр Nx задает количество шагов по оси Ox и определяет величину изменения dx, а параметр Ny – количество шагов по оси Oy и определяет величину dy.



Kx – 1, 2, 3, …, Nx

$dx = (Xmax – Xmin) / Nx$

$X = Xmin + Kx \cdot dx$

Приведем полный текст процедуры рисования поверхности:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);

    procedure MoveTo(A : TPoint);
      begin
        PaintBox1.Canvas.MoveTo(A.x, A.y);
      end;

    procedure LineTo(A : TPoint);
      begin
        PaintBox1.Canvas.LineTo(A.x, A.y);
      end;

      var x, y, dx, dy, z, Qa, Qb, Qc : double;
          P : TPoint;
          kx, ky : integer;

     begin
       OA:=Vect(O,A);
       OB:=Vect(O,B);
       OC:=Vect(O,C);
       {Вычисление шагов}
       dx:=(Xmax-Xmin)/nx;
       dy:=(Ymax-Ymin)/ny;
       with PaintBox1 do begin
         {Фон изображения}
         Canvas.Brush.Color:=clWhite;
         Canvas.Rectangle(0,0,Width, Height);
         {Рисование осей}
         Circle(O,R);
         Line(O,A); Circle(A,R);
         Line(O,B); Circle(B,R);
         Line(O,C); Circle(C,R);
       end;
       {Рисование поверхности}
       for kx:=0 to nx do begin
         x:=Xmin+kx*dx;
         for ky:=0 to ny do begin
           y:=Ymin+ky*dy;
           z:=f(x,y);
           Qa:=(x-Xmin)/(Xmax-Xmin);
           Qb:=(y-Ymin)/(Ymax-Ymin);
           Qc:=(z-Zmin)/(Zmax-Zmin);
           P.x:=round(O.x+Qa*OA.x+Qb*OB.x+Qc*OC.x);
           P.y:=round(O.y+Qa*OA.y+Qb*OB.y+Qc*OC.y);
           if y=Ymin then MoveTo(P) else LineTo(P);
         end;
       end;
```

```
    for ky:=0 to ny do begin
      y:=Ymin+ky*dy;
      for kx:=0 to nx do begin
        x:=Xmin+kx*dx;
        z:=f(x,y);
        Qa:=(x-Xmin)/(Xmax-Xmin);
        Qb:=(y-Ymin)/(Ymax-Ymin);
        Qc:=(z-Zmin)/(Zmax-Zmin);
        P.x:=round(O.x+Qa*OA.x+Qb*OB.x+Qc*OC.x);
        P.y:=round(O.y+Qa*OA.y+Qb*OB.y+Qc*OC.y);
        if x=Xmin then MoveTo(P) else LineTo(P);
      end;
    end;
  end;
```

Пояснения.

Поскольку в цикле нельзя использовать вещественный параметр, заменяем его циклом со вспомогательным параметром k по следующей схеме:

```
for  k := 0   to N do
  begin
x: = начальное_значение + k*шаг;
операторы тела цикла
  end;
```

Возможен и вариант замены с использованием цикла while:

```
x :=  началъное_значение;
while x  < конечное_значение + шаг / 2  do
begin
операторы тела цикла
x: =x  + шаг;
end;
```

Поясним, почему нельзя в этом варианте упростить условие продолжения цикла до x<= конечное_значение.

Обычно шаг вычисляется по формуле

шаг:= (конечное_значение - началъное_значение) / N

и теоретически последнее значение переменной X должно совпадать с проверяемым в условии продолжения цикла конечным значением. Но из-за ошибок округления, присущих всем арифметическим действиям с вещественными числами, это утверждение практически всегда неверно. В лучшем случае X окажется меньше – и тогда последняя итерация цикла будет выполнена. В худшем случае X окажется больше – и тогда последняя итерация цикла не будет выполнена.

Для исключения этой ошибки мы и прибавляем половину шага к конечному значению. Тем самым гарантируется выполнение последней итерации цикла и последующее завершение цикла.

Инициализацию переменных выполним при создании формы:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Xmin:=-2; Xmax:=4;
  Ymin:=-2; Ymax:=4;
  Zmin:=-2; Zmax:=2;
  nx:=SpinEdit1.Value;
```

```
    ny:=SpinEdit2.Value;
    O:=Init(100, 200);
    A:=Init(20, 300);
    B:=Init(300, 200);
    C:=Init(100, 20);
    R:=5;
  end;
```

Вычисление функции z = f(x, y) оформим в виде обычной подпрограммы-функции:

```
  function f(x,y : double) : double;
  begin
    f:=cos(x)*sin(y)/(1+y*y);
  end;
```

Изображение поверхности зависит от расположения на экране базисных точек O, A, B, C, то рисунок может получиться неудачным, с наложением частей поверхности друг на друга. Чтобы можно было рассмотреть поверхность с разных сторон, добавим в проект возможность изменения положения этих точек. Сделаем это с помощью мышки. При нажатии кнопки мыши в небольшой окрестности одной из базисных точек программа должна запомнить номер этой точки. Затем мышь (с нажатой кнопкой!) перемещается, в новое место. При отпускании кнопки в это место экрана переместится выбранная базисная точка, соответственно перестроятся оси и изображение поверхности. Чтобы сделать видимой область «захвата» базисных точек, можно нарисовать вокруг них небольшие окружности, а при попадании курсора мыши в эти области — менять форму курсора. С этого и начнем.

Введем глобальную переменную Mouse типа TPoint. В ней мы будем запоминать меняющиеся при движении мыши координаты курсора. Напишем также функцию, проверяющую попадание курсора мыши в окрестность заданной точки экрана:

```
function InCircle(A: TPoint; R : double): boolean;
begin
  if sqr(Mouse.x-A.x)+sqr(Mouse.y-A.y)<R*R then InCircle:=True
    else InCircle:=False;
end;
```

Параметр R задает «радиус» окрестности точки A. Напишем обработчик движения мыши:

```
procedure TForm1.PaintBox1MouseMove
(Sender: TObject; Shift: TShiftState; X,Y: Integer);
begin
  Mouse:=Init(X,Y);
  if InCircle(O,R) or InCircle(A,R)
or InCircle(B,R) or InCircle(C,R)
  then Cursor:=crHandPoint
  else Cursor:=crDefault;
end;
```

В обработчике события OnMouseDown определяем номер iP выбранной точки (обратите внимание, что если курсор мыши не попадает в область «захвата», то значение iP остается равным -1, во избежание ложного срабатывания при отпускании кнопки!):

```
procedure TForm1.PaintBox1MouseDown
(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
```

```
begin
  iP:=-1;
  Mouse:=Init(X,Y);
  if InCircle(O, R) then iP:=0
  else if InCircle(A, R) then iP:=1
  else if InCircle(B, R) then iP:=2
  else if InCircle(C, R) then iP:=3;
end;
```

В обработчике события OnMouseUp вычисляем новые координаты выбранной базисной точки и перерисовываем график:

```
procedure TForm1.PaintBox1MouseUp
(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
begin
  case iP of
    0: O:=Init(X,Y);
    1: A:=Init(X,Y);
    2: B:=Init(X,Y);
    3: C:=Init(X,Y);
  end;
  PaintBox1Paint(Sender);
end;
```

Задание для самостоятельного выполнения

В принципе программа готова. Но особого интереса она не вызовет по одной очевидной причине – программа только показывает «картинку» и не дает возможности ее изменять. Подумаем, чем дополнить написанную программу:

1. Предоставить пользователю возможность выбирать саму функцию. Наиболее подходящий для этого элемент управления – группа радиокнопок (или список, если вариантов выбора очень много).
2. Добавить поля ввода, в которых будут, задаваться параметры масштабирования Xmin, Xmax, Ymin, Ymax, Zmin, Zmax.
3. Регулировать «густоту» линий сетки изменением значений NX и NY.
4. Построить линии проекции на XOY.
5. Изменить цвет линий графика.
6. Восстановить положение осей и сам график после вращения его.

По материалам Г.П.Гутман «Проекты на Delphi», Информатика № 21, 2003