7. Другие задачи

Использование рекурсивного алгоритма

Большинство современных языков высокого уровня поддерживают рекурсивный алгоритм (recursive algorithm) – способ организации вычислений, при котором функция (подпрограмма) вызывает сама себя с другим аргументом. При этом функция, как элемент структуры языка программирования, возвращающая вычисленное значение по своему имени, содержит внутри себя обращение к самой себе с тем же самым именем, но с другими аргументами. Эта возможность позволяет напрямую реализовывать вычисление рекурсивно определенных объектов. Следует отметить, что любой рекурсивный алгоритм может быть реализован итерационно, причем итерационные решения предпочтительны, итерационные решения работают быстрее рекурсивных.

Рассмотрим в качестве примера рекурсивного алгоритма программу для вычисления последовательности чисел Фибоначчи. В этой последовательности при $n>1$ каждый следующий элемент есть сумма двух предыдущих:

$$F_0 = 1, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}, \tag{7.1}$$

С помощью рекурсивного алгоритма программа для вычисления чисел Фибоначчи может быть записана в виде:

```
program Fbn;
var N: integer;
function Fib(n:integer):integer;
begin
if (n>1)    then    Fib:=Fib(n −1)+Fib(n − 2);
if (n=0) or (n=1) then Fib:=1;
end;
        BEGIN
        N:=3;
        writeln(' F(',N,')= ',Fib(N));
        END.
```

Чтобы понять, как конкретно работает рекурсивный алгоритм этого примера, удобно ввести в подпрограмму Fib операторы вывода writeln(‘in…) на входе в подпрограмму, и writeln(‘out…) на выходе из неё.

Код программы	Результат (*строчки дополнительно сдвинуты, чтобы показать этапы рекурсии*)

program Fbn_1;
var N,F: integer;
function Fib(n:integer):integer;
var y:integer;
begin
writeln('in (N=',n,') ');
if (n>1) then
y:=Fib(n−1)+Fib(n−2);
if (n=0) or (n=1) then y:=1;
Fib:=y;
writeln('out ',' F(',n,')= ',y);
end;
BEGIN
N:=3;
F:=Fib(N);
writeln(' F(',N,')= ',F);
END.
	in (N=3)
	in (N=1)
	out F(1) = 1
	in (N=2)
	in (N=0)
	out F(0) = 1
	in (N=1)
	out F(1) = 1
	out F(2) = 2
	out F(3) = 3
	
	F(3) = 3

Отметим, что в графе «результат» каждая строчка дополнительно сдвинута вправо, чтобы отобразить глубину рекурсии на каждом этапе.

На первом шаге мы входим в подпрограмму с параметром N=3. Поскольку аргумент подпрограммы Fib(N) больше 1, она вызывает очередного исполнителя для вычисления нужных значений Fib(1), Fib(2). Получается серия вызовов, которые выстраиваются в ***дерево рекурсивных вызовов***. При вычислении значения $F(3)$ будут вызваны процедуры вычисления $F(1)$ и $F(2)$. В свою очередь, для вычисления последнего потребуется вычисление $F(0)$ и $F(1)$. Можно заметить, что $F(1)$ вычисляется два раза.

Если рассмотреть вычисление $F(n)$ при больших n, то повторных вычислений будет очень много. Это и есть основной недостаток рекурсии — повторные вычисления одних и тех же значений. Кроме того, с рекурсивными функциями связана одна потенциальная ошибка: дерево рекурсивных вызовов может оказаться бесконечным и компьютер «зависнет». Важно, чтобы процесс сведения задачи к набору более простых задач когда-нибудь заканчивался. Для того чтобы рекурсивный алгоритм заканчивал свою работу, необходимо, чтобы дерево рекурсивных вызовов при любых входных данных обрывалось и было конечным. В данном примере дерево рекурсивных

вызовов обрывается на $F(1)$ и $F(0)$, при вычислении которых подпрограмма подставляет конкретные значения.

Следует отметить, что числа Фибоначчи (7.1) можно вычислять и по более эффективному алгоритму (без рекурсии) или просто воспользоваться прямой формулой.

Существуют и более сложные схемы рекурсии, например: функция F1 вызывает функцию F2, а та в свою очередь вызывает F1. При этом оказывается, что описываемая первой процедура должна вызывать еще не описанную. Чтобы это было возможно, требуется использовать опережающее описание.

```
procedure F2(…); {Опережающее описание второй процедуры}
procedure F1(…); {Полное описание процедуры F1}
begin
……
F2(…);
end;
procedure F2(…); {Полное описание процедуры F2}
begin
……
F1(…);
end;
```

Опережающее описание процедуры F2 позволяет вызывать ее из процедуры F1.

Построение самоподобных (фрактальных) структур

Отличительной особенностью фракталов является наличие специфических геометрических свойств (самоподобие, самоаффинность и т.д.). Проще говоря, если изменить масштаб изображения фракталов определенное количество раз, новое изображение будет таким же, или статистически эквивалентно исходному, причем новое изображение при повторении этого преобразования ведет себя так же.

На рисунке изображены примеры типичных регулярных фракталов. Фрактал «дерево» получается путем последовательного разветвления всех веток на две новые ветки меньшего размера, начиная с центрального ствола. Треугольник Серпинского получается из равностороннего треугольника, если соединить линиями середины его сторон, а затем повторять этот алгоритм для всех получающихся малых треугольников, и так до бесконечности. При этом любая ветка

дерева является уменьшенной копией всего дерева, аналогично, любой внутренний треугольник повторяет структуру полного треугольника.

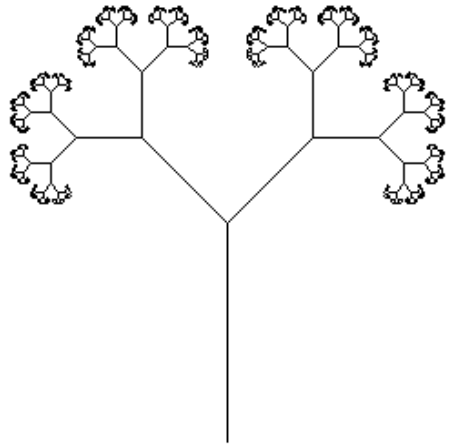
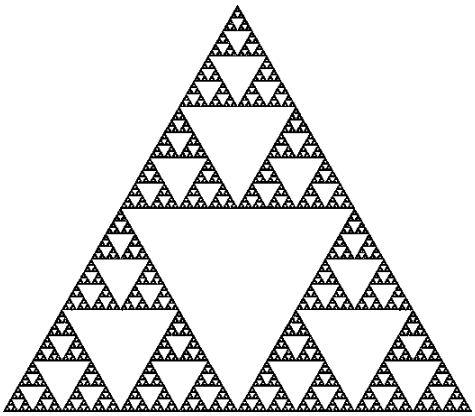
а)  б) 

Рис. 7.1. Примеры фракталов: а) дерево. б) треугольник Серпинского

Ниже приводятся коды программ для рисования этих фракталов. Обе программы имеют общую структуру – рекурсивная подпрограмма рисует базовый геометрический объект по заданным параметрам, а затем обращается сама к себе с измененными значениями этих параметров.

Рекурсивная процедура построения дерева должна рисовать одну линию (ствол до первого разветвления), а затем вызывать сама себя для рисования двух поддеревьев. Поддеревья отличаются от содержащего их дерева координатами начальной точки x1,y1, углами поворота f1,f2, уменьшенной длиной ствола L и количеством содержащихся в них разветвлений n (на одно меньше). Все эти отличия следует сделать параметрами рекурсивной процедуры.

Дерево на рисунке получено для следующих значений параметров f1=pi/4; f2=pi/4; q=0.55; и при первом вызове подпрограммы вида Tree(300,400,pi/2,100,12) Если углы поворота f1,f2 разные, дерево будет несимметричным.

Дерево	Треугольник Серпинского
program tree; uses graph,crt; var gm,gd,N:integer; const f1=pi/3; f2=pi/3; q=0.7; procedure TREE(x1,y1,F,L:real;	program triangle; Uses crt,graph; Var x1,y1,x2,y2,x3,y3,a,b,L,n: integer; procedure TR(x1,y1,x2,y2,x3,y3, N: integer); Var x12,y12,x23,y23,x31,y31: integer;

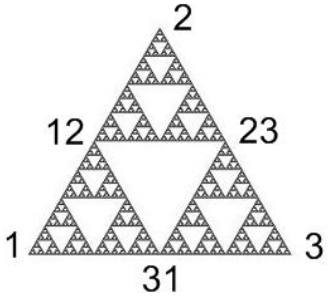
```
n:integer);
var
x2, y2: real;
begin
x2 := x1 + L*cos(F);
y2 := y1 − L*sin(F);
MoveTo(round(x1), round(y1));
LineTo(round(x2), round(y2));
if n > 1 then
begin
TREE(x2, y2, F+f1, q*L, n−1);
TREE(x2, y2, F−f2, q*L, n−1);
end;
end;

BEGIN
Initgraph(gm,gd,'');
Tree(300,400,pi/2,100,12);
readln;
END.
```

```
Begin If N=0 then EXIT;
                x12:=(x1+x2) div 2;
                y12:=(y1+y2) div 2;
                x23:=(x2+x3) div 2;
                y23:=(y2+y3) div 2;
                x31:=(x3+x1) div 2;
                y31:=(y3+y1) div 2;
MoveTo(x31,y31); LineTo(x12,y12);
                 LineTo(x23,y23);
                 LineTo(x31,y31);
TR(x1,y1,x12,y12,x31,y31, N-1);
TR(x12,y12,x2,y2,x23,y23, N-1);
TR(x31,y31,x23,y23,x3,y3, N-1);
end;
BEGIN
N:=5;  L:=500;  a:=detect;b:=detect;
InitGraph(a,b,' ');
x1:=100; y1:=450;
x2:=x1+L div 2;
y2:=y1-round(L*sqrt(3)/2);
x3:=x1+L; y3:=y1;
Moveto(x1,y1);   LineTo(x2,y2);
                 LineTo(x3,y3);
                 LineTo(x1,y1);
TR(x1,y1,x2,y2,x3,y3, N);
ReadKey;  CloseGraph; END.
```

Разберем более подробно рисование треугольника Серпинского. При вызове процедуры TR(…); в неё передаются координаты вершин треугольника, причем x_1,y_1 соответствуют левой нижней вершине, x_2,y_2 – центральной вершине, x_3,y_3 – правой нижней вершине. В первой части подпрограммы вычисляются координаты вершин центрального перевёрнутого треугольника, которые являются серединами сторон исходного, то есть,

$x_{12}:=(x_1+x_2)/2$ $x_{23}:=(x_2+x_3)/2$ $x_{31}:=(x_3+x_1)/2$

$y_{12}:=(y_1+y_2)/2$ $y_{23}:=(y_2+y_3)/2$ $y_{31}:=(y_3+y_1)/2;$



(в программе использован оператор деления целых чисел div).

После этого рисуются стороны этого треугольника. В результате образуются три новых треугольника, подобных исходному треугольнику (с центральной вершиной, направленных вверх).

На втором этапе три раза вызывается эта же процедура TRI(...), параметрами которой являются вершины этих новых треугольников, таким образом, новые треугольники аналогичным образом делятся на подобные части и т.д. Очень важно не забыть уменьшить у внутренних вызовов подпрограммы параметр N (число шагов рекурсии) на единицу, чтобы дерево рекурсивных вызовов было конечным. То есть такого рода процедуры должны выглядеть примерно так:

```
procedure TR(…, N: integer);
Begin
……….……
TR(..., N-1);
TR(..., N-1);
TR(..., N-1);
end;
```

Построение силовых линий и эквипотенциальных поверхностей

Построение эквипотенциальных поверхностей сводится к решению трансцендентных уравнений вида $\varphi(x, y, z) = \varphi_0$ и не представляет особых трудностей. В частности, в пакете MathCAD это можно реализовать как построение линий уровня для двумерного сечения этих поверхностей.

Построение силовых линий проиллюстрируем на примере электростатики. Прежде всего, следует учесть тот факт, что визуализация картины поля набором силовых линий носит качественный характер. Дело в том, что по теореме Гаусса поток поля, то есть, грубо говоря, число силовых линий, пересекающих замкнутую поверхность, прямо пропорционально заряду внутри неё. Соответственно, если окружить каждый точечный заряд такой поверхностью, число силовых линий, выходящих из заряда (и пересекающих поверхность), должно быть прямо пропорционально величине заряда. Таким образом, для системы точечных зарядов следует подбирать такой набор силовых линий, чтобы из каждого заряда выходило число линий прямо пропорциональное величине

заряда. Например, если один заряд в два раза больше другого (по модулю), то и число линий поля, начинающихся на нем, должно быть в два раза больше. Очевидно, для системы зарядов это можно сделать, только если отношения величин зарядов друг к другу являются рациональными дробями. В общем случае адекватный набор силовых линий не существует.

По определению, силовой линией векторного поля называется геометрическая линия, в каждой точке которой вектор поля сонаправлен с касательным вектором к этой линии (отложенным в той же самой точке). Этого определения достаточно, чтобы свести задачу о построении силовой линии к решению системы обыкновенных дифференциальных уравнений.

Любую непрерывную линию можно задать как зависимость радиус вектора её точек от некоторого параметра *t*. В качестве переменной *t* удобно выбрать длину этой линии, отсчитываемую от начальной точки. В произвольной декартовой системе координат имеем $\vec{r}(t) = (x, y, z)$, то есть такую линию можно задать в виде трех независимых функций *x*(*t*), *y*(*t*), *z*(*t*). Обозначим вектор касательной в произвольной точке этой линии через $\vec{v}(t) = (v_x, v_y, v_z)$. В курсе математики доказывается, что касательный вектор для любой точки этой линии определяется формулой $\vec{v}(t) = d\vec{r}/dt$, то есть

$$v_x = \frac{dx}{dt}, \qquad v_y = \frac{dy}{dt}, \qquad v_z = \frac{dz}{dt}, \tag{7.2}$$

причем модуль этого вектора всегда равен единице. Осталось связать этот вектор с вектором поля, и мы получим искомое уравнение.

Рассмотрим произвольное электрическое поле $\vec{E}(t) = (E_x, E_y, E_z)$. Из определения силовой линии непосредственно следует, что вектора $\vec{v}(t) = (v_x, v_y, v_z)$ и $\vec{E}(t) = (E_x, E_y, E_z)$ сонаправлены, таким образом, получаем:

$$\vec{v}(t) = \frac{\vec{E}}{E} = \frac{\vec{E}}{\sqrt{E_x^2 + E_y^2 + E_z^2}}, \tag{7.3}$$

где *E* – длина вектора поля.

Окончательно уравнение силовой линии произвольного векторного поля имеет вид

$$\frac{d\vec{r}(t)}{dt} = \frac{\vec{E}(x(t), y(t), z(t))}{E}. \tag{7.4}$$

Особенностью электростатики является то, что поля начинаются на зарядах (рис. 7.2). Очевидно, что точки расположения зарядов являются особыми точками данного уравнения. В этих точках значение правой части (7.4) неопределено, так как предел правой части при стремлении аргумента векторного поля к координате точечного заряда зависит от пути, по которому осуществляется этот предел. С математической точки зрения такие точки являются точками неединственности решения, где пересекаются несколько траекторий.

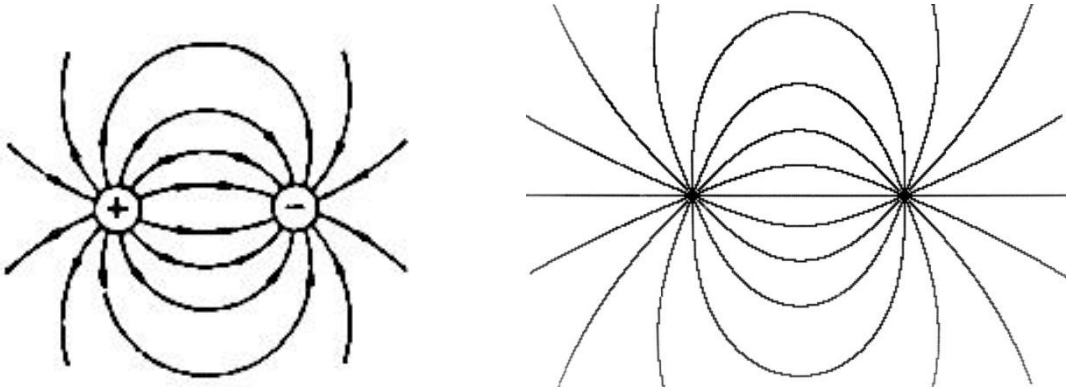


Рис. 7.2. Электрическое поле от двух точечных зарядов (диполя):
а) эскиз, б) компьютерная симуляция

С физической точки зрения это совершенно естественно, так как из зарядов выходит набор силовых линий. Таким образом, если начинать силовую линию из точки расположения заряда, то кроме начального условия $\vec{r}(0) = (x_i, y_i, z_i)$, требуется дополнительно задавать и начальное направление линии (начальное значение касательного вектора $\vec{v}(t)$). Если начало силовой линии не попадает в точку расположения заряда, то правая часть уравнения (7.4) определена однозначно, и в качестве начальных условий достаточно указать координату этого начала.

Рассмотрим случай, когда силовую линию необходимо начинать из той точки, где непосредственно находится какой-либо точечный электрический заряд. В ближайшей окрестности любого такого заряда относительный вклад в поле от остальных зарядов стремится к нулю (при стремлении к нулю размера окрестности). Соответственно, это поле асимптотически стремится к полю, имеющему вид обычного поля уединённого заряда. Следовательно, в этой окрестности уравнение (7.4) имеет вид:

$$\frac{d\vec{r}(t)}{dt} = A \cdot q_i \cdot \frac{\vec{r}(t) - \vec{r}_i}{\left|\vec{r}(t) - \vec{r}_i\right|} = A \cdot q_i \cdot \vec{n} \,. \qquad (7.5)$$

Здесь $\vec{r}_i$ – радиус вектор заряда $\vec{n}$ – единичный вектор. Это как раз тот самый вектор, который и будет задавать начальное направление силовой линии, то есть $\vec{n} = \vec{v}(0)$. Таким образом, задание конкретных направлений для набора векторов $\vec{n}$ и является дополнительными начальными условиями для набора силовых линий, «стартующих» прямо из точки расположения заряда.

Поясним вышесказанное на примере программы построения силовых линий поля двух одинаковых по величине, но разных по знаку точечных зарядов (рис. 7.2). Поместим положительный заряд в начало координат x:=0; y:=0, а отрицательный расположим в точке с координатой x=x0; y:=0 . Очевидно, абсолютная величина заряда в этом примере может быть любой. Построение силовых линий, всего их 16, начинается из положительного заряда (начала координат). Основная часть программы (цикл для построения линий) имеет вид.

```
for  j:=1 to 16 do begin
t:=0;   dt:=0.01;
df:=2*pi/16;
vx:=cos(df*j); vy:=sin(df*j);
x:=0; y:=0;
            while   t<350 do
            begin
            x:=x+vx*dt; y:=y+vy*dt;
            t:=t+abs(dt);
            R2:=sqrt(sqr(x-x0)+sqr(y));
            if R2<0.1 then
            begin
            x:=x+vx*0.2; y:=y+vy*0.2;    dt:= −dt;
            end;
            E_1(Ex,Ey,x,y);
            E:=1/sqrt(sqr(Ex)+sqr(Ey));
            vx:=Ex*E;       vy:=Ey*E;
            xx:=200+round(x); yy:=220-round(y);
            putpixel(xx,yy,12);
            end;
end;
```

Вычисление декартовых компонент вектора электрического поля Ex, Ey в точке x,y вынесено в отдельную процедуру E_1(Ex,Ey,x,y)

(написать которую не составляет труда для любого расположения точечных зарядов по принципу суперпозиции). Для решения дифференциальных уравнений силовых линий используется метод Эйлера. Индекс j нумерует силовую линию. Для каждого значения задаются свои начальные условия: начальный угол «вылета» линии df*j , начальные значения компонент касательного вектора (vx, vy), общая начальная координата x:=0; y:=0. После этого строится линия длиной 350 пикселей в цикле while.

Отметим, что построение линии продолжается, когда она проходит «сквозь» второй заряд. Это сделано с помощью условного выражения if R2<0.1 then…, где R2 – расстояние до второго заряда. Когда линия попадает в ближайшую окрестность второго заряда (R2<0.1), она скачком переходит на его «другую сторону», и при этом, самое главное, меняется знак dt. Это сделано для того, чтобы построение линии продолжалось дольше, а так как поле здесь направлено к заряду (он отрицательный), то необходимо сделать dt отрицательным, чтобы линия уходила от заряда.

Фурье-анализ поверхности твердого тела в MathCAD

Пусть поверхность образца в выбранной трехмерной системе координат описывается непрерывной функцией $Z(x,y)$, сопоставляющей каждой точке плоскости с координатами x,y значение z-координаты («высоты») соответствующей ей точки поверхности: $z = Z(x,y)$. В частности информацию о поверхности можно получить методом сканирующей зондовой микроскопии (СЗМ). Сканирование поверхности дает дискретный вариант функции $Z(x,y)$, так как высота измеряется для дискретного набора (решетки) точек. Это так называемый СЗМ кадр – двумерный массив (матрица) чисел Z_{ij}. Координаты точек поверхности связаны с этой матрицей формулами

$$x_i = x_0 + a \cdot i, \qquad y_j = y_0 + a \cdot j, \qquad Z(x_i, y_j) = z_0 + h \cdot Z_{ij}.$$

Здесь a, определяется шагом сканирования, x_0, y_0, z_0 – задают начало отсчета, h – масштабный множитель.

При 2D визуализации каждой точке поверхности $z = Z(x,y)$ ставится в соответствие цвет. Наиболее широко используются градиентные палитры, в которых раскраска изображения производится тоном определенного цвета в соответствии с высотой точки поверхности.

В учебных целях для изображения поверхности можно ограничиться монохромными рисунками – графическими файлами формата BMP. В этом значению высоты в данной точке поверхности сопоставляется один из 256 оттенков серого цвета (диапазон от белого до черного). Такой файл представляет из себя матрицу, в которой значению высоты в данной точке сопоставляется соответствующий элемент матрицы – целое число в диапазоне от 0 до 255. Минимальной высоте соответствует число 0 (черный цвет), максимальной высоте – число 255 (белый цвет).

Загрузка графического файла для математической обработки непосредственно в документ MathCAD осуществляется стандартной функцией *READBMP*. Нижеследующая строчка создает матрицу с целочисленными элементами, соответствующими каждому пикселю изображения, и размерностью, равной размерности рисунка:

Z:=*READBMP*(“disk.bmp”).

Построение изображения поверхности в рамках документа MathCAD будет осуществляться как вставка рисунка (picture). При этом следует учитывать, что такое изображение прямо соответствует расположению элементов в матрице $Z_{i,j}$. Первый индекс нумерует строки, а второй столбцы, причем элемент матрицы с индексом (0.0) соответствует левому верхнему углу рисунка. То есть мы видим как бы графическое изображение матрицы с каноническим матричным расположением элементов. Очевидно, что при переходе к двумерной декартовой системе координат первый индекс (*i*) соответствует координате *y*, а второй индекс (*j*) – координате *x*.

Таким образом, матрица СЗМ кадра, и данная матрица $Z_{i,j}$ отличаются перестановкой индексов и, кроме того, требуется отражение относительно горизонтальной оси.

Если числовые значения матрицы $Z_{i,j}$ выходят за пределы диапазона $[0, 255]$, или для улучшения контрастности можно применить преобразование, возвращающее значения элементов в пределы этого диапазона. Для этого определяется минимальный и максимальный элементы массива $Z_{i,j}$ по формулам *A*:=*min*(*Z*), *B*:=*max*(*Z*), а затем проводится преобразование

$$Z_{i,j} = 255 \cdot \frac{(Z_{i,j} - A)}{(B - A)}.$$

При необходимости можно обратить цвета на изображении с помощью простой формулы $Z_{i,j} := 255 - Z_{i,j}$.

Для улучшения качества изображения можно применить один из методов фильтрации, например, использовать сглаживающий медианный фильтр. Он хорошо убирает шумы импульсного характера, например, шумы в виде «отдельных точек», и, в то же время, сохраняет резкость границ элементов изображения.

На первом этапе медианной фильтрации для каждой точки изображения строится упорядоченный ряд, состоящий из значения высоты в этой точке и значений высот точек из ближайшей окрестности. На втором этапе, значение высоты поверхности в рассматриваемой точке заменяется медианой (средним элементом) этого ряда.

При вычислении медианы в квадратной окрестности размера $2M$ вокруг точки (i,j) можно воспользоваться функцией *submatrix*(…), которая выделяет подматрицу из ближайших соседей точки (i,j), и встроенной функцией *median*:

$F_{i,j}$:=*median*(*submatrix*(*Z, i–M, i+M, j–M, j+M*))

Матрица $F_{i,j}$ и дает сглаженное изображение.

Рассмотрим несколько способов математического анализа поверхности, основанных на дискретном преобразовании Фурье. В пакете MathCAD имеется несколько соответствующих методов, использующих алгоритм быстрого преобразования Фурье – fast Fourier transform (FFT).

В качестве объекта для применения методов Фурье-анализа выберем СЗМ изображение поверхности DVD диска размером 200×200 точек (рис. 7.3). Скан имеет размер 6×6 мкм.

Ямки (питы) на рисунке точно ложатся на почти прямолинейные отрезки, расположенные на одинаковом расстояниях друг от друга (равном шагу непрерывной спиральной дорожки). Поэтому такая структура является квазипериодической. В частности, на ней, как на дифракционной решетке, наблюдается дифракция лазерного луча, диаметр которого много больше шага спиральной дорожки.

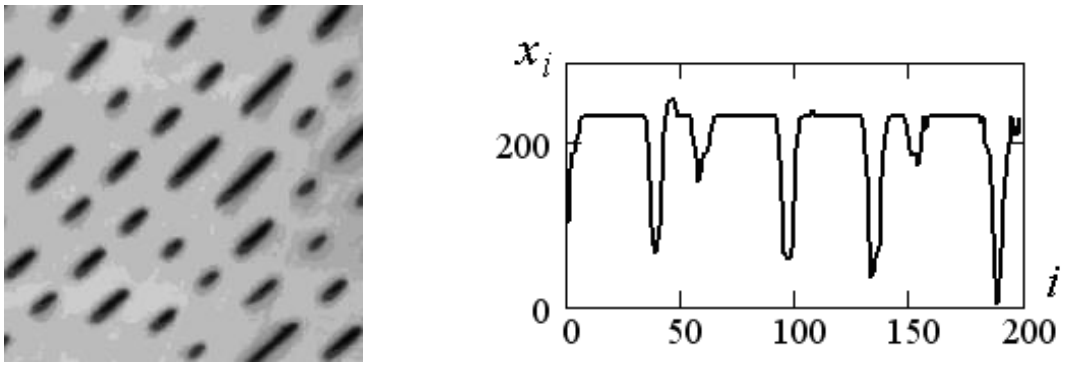


Рис. 7.3. Фрагмент поверхности DVD диска
и его сечение вдоль диагонали (i,i)

Период этой квазипериодической структуры совпадает с шагом спирали. Направление, вдоль которого присутствует периодичность, приблизительно совпадает с диагональю, проведенной из левого верхнего угла в правый нижний.

Рассмотрим сначала одномерное сечение поверхности вдоль этой диагонали и его преобразование Фурье. В данном случае i = 0, 1,..,N–1; j =0, 1,..,N–1, где N=200. Искомое сечение задается соотношением $x_i = Z_{i,i}$ и представлено на рис. 7.3.

Одномерный Фурье-образ для x_i получается, если записать выражение вида f :=$CFFT$(x). Вектор f_n, (n=0, 1,..,N–1), и является набором коэффициентов Фурье – пространственным спектром.

На рисунке 7.4 изображен модуль спектра функции x_i, то есть график зависимости $|f_n|$ от n. На этом графике пришлось обрезать нулевую составляющую сверху, так как её величина заметно больше всех остальных. На этом графике видно соотношение симметрии для коэффициентов Фурье $f_n = f^*_{N-n}$ (за исключением случая n=0).

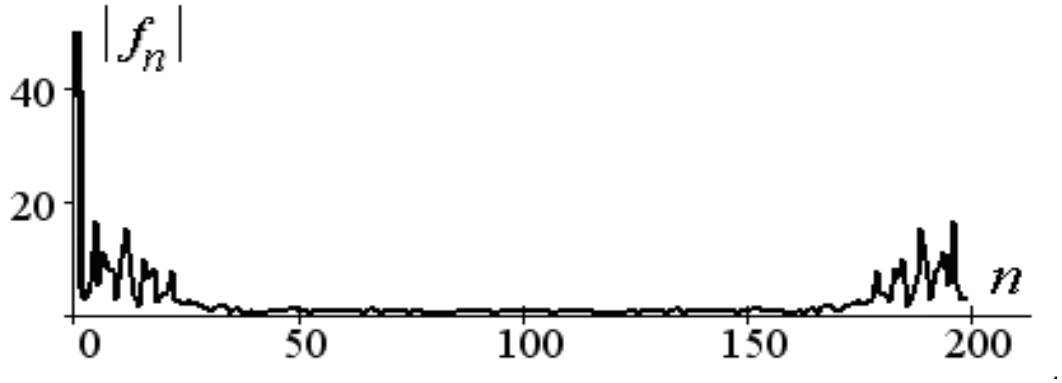


Рис 7.4. Спектр диагонального сечения поверхности с рис. 7.3

Обратное преобразование задается формулой $y=ICFFT(f)$. С точностью до погрешности вычислений $y_i=x_i$.

По виду сечения и спектра невозможно увидеть наличие квазипериодичности, связанной с постоянным шагом спирали из питов (ямок). Поэтому целесообразно применить к этому изображению двумерное преобразование Фурье.

Двумерный Фурье образ матрицы $Z_{i,j}$ в MathCAD можно получить с помощью формулы

$$F := CFFT(Z).$$

В ней используется та же самая функция *CFFT*, которая возвращает вектор, если аргументом является вектор, и возвращает матрицу, если аргумент является матрицей. Матрица двумерного пространственного спектра $F_{n,m}$ имеет размер $N \times N$, причем $n,m = 0, 1,..,N-1$. Контурный график модуля спектра (матрицы $|F_{n,m}|$) изображен на рис 7.5 (а).

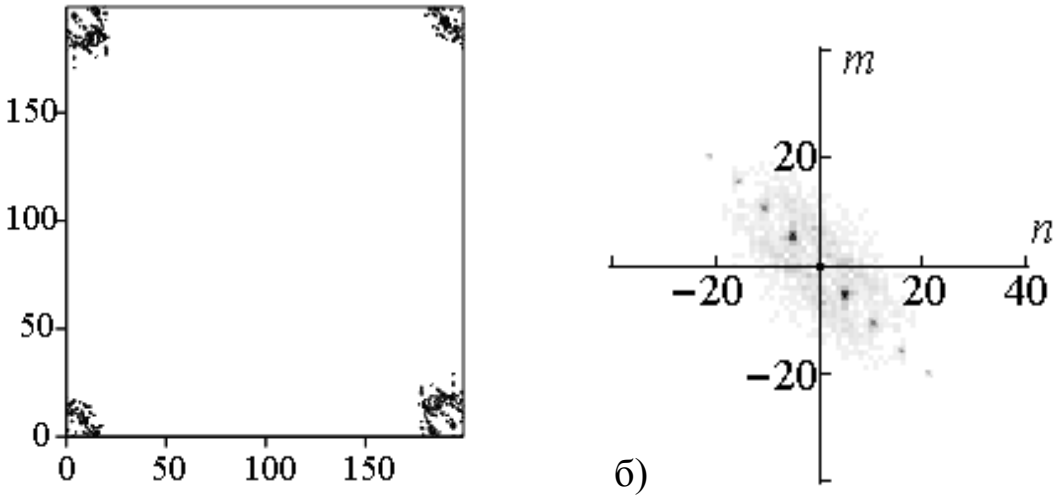


Рис. 7.5. Двумерный пространственный Фурье-спектр изображения 7.3.
а) контурный график для $|F_{n,m}|$,
б) центральная часть матрицы $|G_{n,m}|$

При использовании контурного графика вместо изображения типа «рисунок» (picture) следует учитывать изменение начала отсчета и переход от матричного к координатному порядку расположения индексов. Поэтому приходится строить контурный график не самой матрицы $|F_{n,m}|$ а матрицы $A_{n,m} = |F_{m,N-n}|$.

Из рисунка 7.5 (а) видно, что основной вклад в спектр расположен по углам матрицы $F_{n,m}$. Такое представление не очень наглядно, поэтому целесообразно преобразовать его к центрально-симметричному виду. Будем обозначать модифицированную таким

образом матрицу спектра через $G_{n,m}$. Искомое преобразование имеет вид:

$$M := round(N/2) - 1 \qquad n := 0..M \qquad m := 0..M$$

$$G_{M+n,M+m} = F_{n,m} \qquad G_{M-n,M+m} = F_{N-n,m}$$

$$G_{M-n,M-m} = F_{n,m} \qquad G_{M+n,M-m} = F_{N-n,m}$$

Центральная часть матрицы $G_{n,m}$ представлена на рис. 7.5(б). Хорошо видно, что в спектре присутствует характерная цепочка максимумов, расположенная по диагонали. Эта цепочка и соответствует наличию периодической компоненты в структуре поверхности. Расстояние между максимумами, с учетом масштабного преобразования, дает период интересующей нас квазипериодической структуры.