M Tech (Computer Science) Dissertation Series

STUDY ON GEOMETRIC MODELS OF MEDICAL IMAGES

BY

PARAMARTHA DUTTA

Under the supervision of

PROF. D. DUTTAMAZUMDER

&

DR. S. E. SHARMA

INDIAN STATISTICAL INSTUTUTE
203, B.R. Road, Kol - 108

CONTENTS

Acknowledgement

It is with great pleasure and experience on my part I express my profound regard to Prof. D. Dutta Majumder under whose supervision the project has been carried out. It was a sheer joy and excitement of learning things while working under his great optimism, all-out help and watchful eyes lined with critical appreciation.

This would remain incomplete without mentioning the names of Dr.S.E.Sharma and Dr.S.Banerjee as well as Mr.Subhasis Pal of the CSSC of my Institute and Mr. Palash Sarkar of my class who throughout cooperated with me with all their sincereity and seriousness.

Finally, I thank all my teachers and friends for their help and cooperation during my dissertation work.

Abstract

'GEOMETRIC MODEL OF MEDICAL IMAGES' can be thought of as composed of broadly two phases.

(I) Image reconstruction part based on phantoms.

(II) Simulation of three dimensional medical objects starting from the reconstructed images.

In course of our following discussion we have mainly concentrated on the first phase of our modelling i,e the problem regarding image reconstruction of phantoms. This problem we have carried out using different methodologies. Broadly speaking here we have adopted three methods -

(a) Convolution BackProjection (CBP);

(b) Positron Emmision Tomography(PET);

(c) Magnetic Resonance (MRI);

Let us discuss the (a) and (b) one by one. In our discussion we did not incorporate (c) though some works have been done on it in reality.

Introduction

We have covered two types of methods in our image reconstruction problem. Deterministic and Stochastic. In our discussion in the following chapters we have discussed two procedures one by one. Convolution BackProjection is the primary topic of our discussion of deterministic approach whereas we make study on Positron Emission Tomography as the part of Stochastic approach. In Convolution BackProjection we give an algorithm and its corresponding program implemented in the computer. In this algorithm our input is elemental objects described in terms of their locations, orientations, size and densities respectively. Its output is an image which we have displayed in our work. On the other hand, regarding stochastic approach we use Positron Emission Tomography applied in circular ring geometry environment. Here in this approach our basic problem lies with the estimation of the pixel densities. Since our image prcessing problem needs a lot of space and time we will discuss about a more efficient algorithm. We then try to show the relative performance of the above two procedures.All our programs we have written in pascal.

Chapter 1
Convolution BackProjection (CBP)

CBP is a deterministic method of image reconstruction. The theoretical development of this method owes a lot to reconstruction of mathematically described objects (they are called phantoms) from computer simulated projection data. A program capable of simulating data collection has been found to be rather complex. In recent times many systems have been developed to meet this end. For example SNARK77 is one such system. As evident from the problem posed itself is that the data simulation part is quite a real problem and as such there is no concrete rule established as yet. So applying on a particular data set one phantom is developed. Here our assumption is that the pixel density at a particular point (x, y) is the relative linear attenuation of the tissue concerned at that point under a given energy level.

Creation of a Mathematical Phantom:

We now discuss how we have created a test phantom.(a test phantom is nothing but a picture on which we wish to test reconstruction algorithms or data collection methods). Basically a phantom is put together by superimposing a number of elemental objects. In our simulation we have used ellipse, rectangle, triangle as our elemental objects. In other simulations, there can be other geometric figures taken as elemental objects. These elemental objects are placed at desired positions in the coordinate system, with suitable orientation each having their own densities. The density of an elemental object may be negative. The density of a picture at a particular point is the sum of the densities of the elemental objects within which that point lies. Now we will describe how we have obtained the density estimate within a particular pixel. Basically what we do here is we subdivide a particular pixel into a numbar of grids (say k X k finer squares, where k is user specified each square constituting a grid). The densities of these grids of each pixel are calculated on the basis of above procedure and once they are found, the average of the densities are taken over the grids within a particular pixel. This average value of the density is assigned as the density of that pixel. In this way all the densities are calculated of all the pixels. That is how we have developed the image. Given below is the program written to derive the image.

```
program phantom (input, output);

label   1;

const   maxinten = 320;
        obj_no = 20; {* maximum number of objects*}
maxpic = 150; {* maximum number of pixels in a row*}
maxgrid = 20;
pi = 3.141592624;

type object = record
xx : real; {* location *}
yy : real; {* location *}
uu : real; {* axis*}
vv : real; {* axis*}
th : real; {* inclination*}
class : integer; {* represents the type of object *}
density : real; {* density of a particular object*}
end;

var     list : array [1..obj_no] of object; {* array of objects*}
delta : array [-160..160, -160..160,1..obj_no] of integer;
ind1, ind2, i,l, j, k,index, inten, nobj :integer;
d : array [-75..75, -75..75] of integer;
dense : array [-75..75, -75..75] of real;
picsize, level : integer;
filvar1, filvar2 : text;
filname1, filname2 : varying [15] of char;
answer : varying [1] of char;

procedure del_cal(temp1, temp2 : integer; factor1, factor2 : real);

{* table is created to find the pixel densities in future *}

var row, col : integer;
denseval : real;
```

```
function ellipse (x, y: integer; cx, cy, u,v, ang :real):boolean;

{* defines ellipse as an elemental object *}

var x_prm, y_prm, temp, dis :real;

begin {* ellipse*}

x_prm := x - cx;
y_prm := y - cy;
temp := x_prm;
x_prm := x_prm * cos(ang) + y_prm * sin(ang);
y_prm := y_prm * cos(ang) - temp * sin(ang);
x_prm := (x_prm * x_prm)/(u * u);
y_prm := (y_prm * y_prm)/(v * v);
dis := x_prm + y_prm - 1;
if dis <= 0 then
ellipse := true
else
ellipse := false;
end; {* ellipse*}

function rectangle(x, y: integer; cx,cy, u,v, ang :real):boolean;

{* defines rectangle as an elemental object *}

var x_prm, y_prm, temp, d1, d2 :real;

begin {* rectangle*}
x_prm := x - cx;
y_prm := y - cy;
temp := x_prm;
x_prm := x_prm * cos(ang) + y_prm * sin(ang);
y_prm := y_prm * cos(ang) - temp * sin(ang);
d1 :=abs(x_prm) - u ;
d2 := abs(y_prm) - v ;
if ((d1 <= 0) and (d2 <= 0)) then
```

```
rectangle := true
else
rectangle := false;
end;{* rectangle*}

function triangle(x, y: integer; cx, cy, u, v, ang :real):boolean;

{* defines triangle as an elemental object *}

var x_prm, y_prm, temp, d1, d2, d3, ht :real;

begin{* triangle*}
x_prm := x - cx;
y_prm := y - cy;
temp := x_prm;
x_prm := x_prm * cos(ang) + y_prm * sin(ang);
y_prm := y_prm * cos(ang) - temp * sin(ang);
ht := v*(u - abs(x_prm))/u;
if ((abs(x_prm) <= u) and (y_prm > 0) and (y_prm <= ht))
then triangle := true
else triangle := false;
end;{* triangle*}


begin {* del_cal*}
for row := temp1 * k to temp1 * k + k - 1 do
for col := temp2 * k to temp2 * k + k - 1 do
for i := 1 to nobj do
begin
if list[i].class = 1 then
begin
if ellipse(row, col, list[i].xx * factor1, list[i].yy * factor1,
 list[i].uu * factor1, list[i].vv * factor1,
list[i].th * factor2)  then
delta[row, col, i] := 1
else
delta[row, col, i] := 0;
```

```
end
else
if list[i].class = 2 then
begin
if rectangle(row, col, list[i].xx * factor1,
list[i].yy * factor1, list[i].uu * factor1,
list[i].vv * factor1, list[i].th * factor2)  then
delta[row, col , i] := 1
else
delta[row, col, i] := 0;
end
else
if list[i].class = 3 then
begin
if triangle(row, col, list[i].xx * factor1,list[i].yy * factor1,
 list[i].uu * factor1, list[i].vv * factor1,
list[i].th * factor2)  then
delta[row, col , i] := 1
else
delta[row, col, i] := 0;

end;
end;
denseval := 0;
for row := temp1 * k to temp1 * k + k - 1 do
for col := temp2 * k to temp2 * k + k - 1 do
for j := 1 to nobj do
denseval := denseval +
delta[row,col, j] * list[j].density;

dense[temp1, temp2] := denseval / inten;
end;{* del_cal*}


procedure rounding;

{* maps the densiies obtained into gray values with levels being user-speci
```

```
var max, min,range : real;

begin
max := 0;
min := 0;
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
begin
if dense[ind1, ind2] > max then
max := dense[ind1,ind2];
if dense[ind1,ind2] < min then
min := dense[ind1 , ind2];
end;
range := max - min;
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
d[ind1,ind2] :=round(level * ((dense[ind1,ind2] - min)/range));
end;

procedure probability;

{* calculates acceptance probability of a particular pixel as emission pixe

var     filvar3 : text;
filname3 : varying [15] of char;
temp : real;

begin
writeln ('what is the name of probality output file');
readln (filname3);
open (filvar3, filname3, new);
rewrite (filvar3);
temp := 0;
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
temp := temp + d[ind1, ind2];
```

```
if temp <> 0 then
begin
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
begin
dense[ind1, ind2] := d[ind1, ind2] / temp;
writeln (filvar3, dense[ind1, ind2]);
  end;
end
else
writeln ('all gray values are zero');
close (filvar3);
end;

procedure plot;

{*graph plotting *}

label   22;

var     nn : integer;
filvar5 :text;

begin
22: writeln ('which row do you want to plot, maximum is', picsize);
readln (nn);
if nn > picsize then
begin
writeln ('no such row exists, do you want to repeat');
readln (answer);
if ((answer = 'y') or (answer = 'Y')) then
goto 22;
end
else
begin
open (filvar2, filname2, old);
reset (filvar2);
```

```
for j := -trunc(picsize/2) to trunc(picsize/2) - 1 do
readln (filvar2, d[nn - trunc(picsize/2) - 1, j]);
open (filvar5, 'graph.dat', new);
rewrite (filvar5);
for j := -trunc(picsize/2) to trunc(picsize/2) - 1 do
writeln (filvar5, j + trunc(picsize/2) + 1
, d[nn - trunc(picsize/2) - 1, j]);
close (filvar5);
close (filvar2);
end;
end;{plot}

begin {*main*}

1: writeln ('give the gridsize you want');
readln (k);
inten := k * k;
writeln ('give the number of pixels in a row');
readln (picsize);
writeln ('what is the level of the picture you want');
readln (level);
writeln ('number of objects?');
readln (nobj);
writeln ('give the input filename');
readln (filname1);
writeln ('give the output filename');
readln (filname2);
open (filvar1,filname1,old);
reset (filvar1);
for index := 1 to nobj do
readln (filvar1,list[index].xx, list[index].yy, list[index].uu,
list[index].vv, list[index].th, list[index].class,
list[index].density);
open (filvar2, filname2, new);
rewrite (filvar2);
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
```

```
if filname1 = 'test3.dat' then
del_cal(ind1, ind2, picsize/ ( k * 2), pi/180)
else
del_cal(ind1, ind2, (picsize * k)/2, 1);
        rounding;
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
writeln (filvar2, d[ind1, ind2]);
close (filvar2);
writeln ('do you want the probability measurement');
readln (answer);
if ((answer = 'y') or (answer = 'Y')) then
begin
open (filvar2, filname2, old);
reset (filvar2);
for ind1 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
for ind2 := -trunc(picsize/2) to trunc(picsize/2) - 1 do
  readln (filvar2, d[ind1, ind2]);
probability;
end;
close (filvar1);
writeln ('do you want graph plot');
readln (answer);
if ((answer = 'y') or (answer = 'Y')) then
plot;
writeln ('do you want another figure');
readln (answer);
if ((answer = 'y') or.(answer = 'Y')) then
goto 1;
end.
```

The above program can be applied on different input files to get images accordingly. As for example we have used table1 to the figure in fig1, whereas by applying on data of table2 we get fig2. In table1, there are 18 elemental objects and in table2 there are 9 elemental objects.

Table 1

object no.	origin		radius		orientation	object type	density
	x_0	y_0	semi-major	semi-minor			
1	0.15	0	0.15	0.08	1.57	1	0.4
2	0.15	0	0.15	0.08	3.14	1	-0.4
3	0.15	0	0.15	0.08	-0.785	1	0.4
4	0.15	0	0.15	0.08	-0.785	1	-0.4
5	-0.55	0	0.15	0.08	0.785	1	0.3
6	-0.55	0	0.15	0.08	3.14	1	-0.3
7	-0.55	0	0.15	0.08	-0.785	1	0.3
8	-0.55	0	0.15	0.08	1.57	1	-0.3
9	0	0	0.95	0.80	0	1	0.45
10	0	0	0.85	0.72	0	1	-0.45
11	0.25	0.34	0.35	0.15	0.785	1	0.35
12	0.25	-0.34	0.35	0.15	-0.785	1	0.35
13	-0.25	0.34	0.28	0.009	-0.392	2	-0.5
14	-0.16	-0.16	0.02	0.19	1.57	3	0.33
15	-0.1	-0.16	0.22	0.019	0	2	-0.533
16	-0.04	-0.16	0.02	0.19	-1.57	3	0.33
17	0.45	0	0.103	0.106	0	3	0.5
18	0.45	0	0.103	0.106	3.14	3	0.5

Table 2

object no.	origin		radius		orientation	object type	density
	x_0	y_0	semi-major	semi-minor			
1	0	0	0.69	0.92	0	1	0.1
2	0	-0.018	0.66	0.87	0	1	0.9
3	0	0.35	0.21	0.25	0	1	1
4	0.35	0	0.11	0.31	-0.314	1	-0.7
5	-0.35	0	0.16	0.41	0.314	1	-0.5
6	0	-0.1	0.046	0.046	0	1	0.5
7	-0.08	-0.605	0.046	0.023	0	1	0.5
8	0.06	-0.605	0.023	0.046	0	1	0.5
9	0.5	-0.5	0.0375	0.125	-0.524	1	0.5

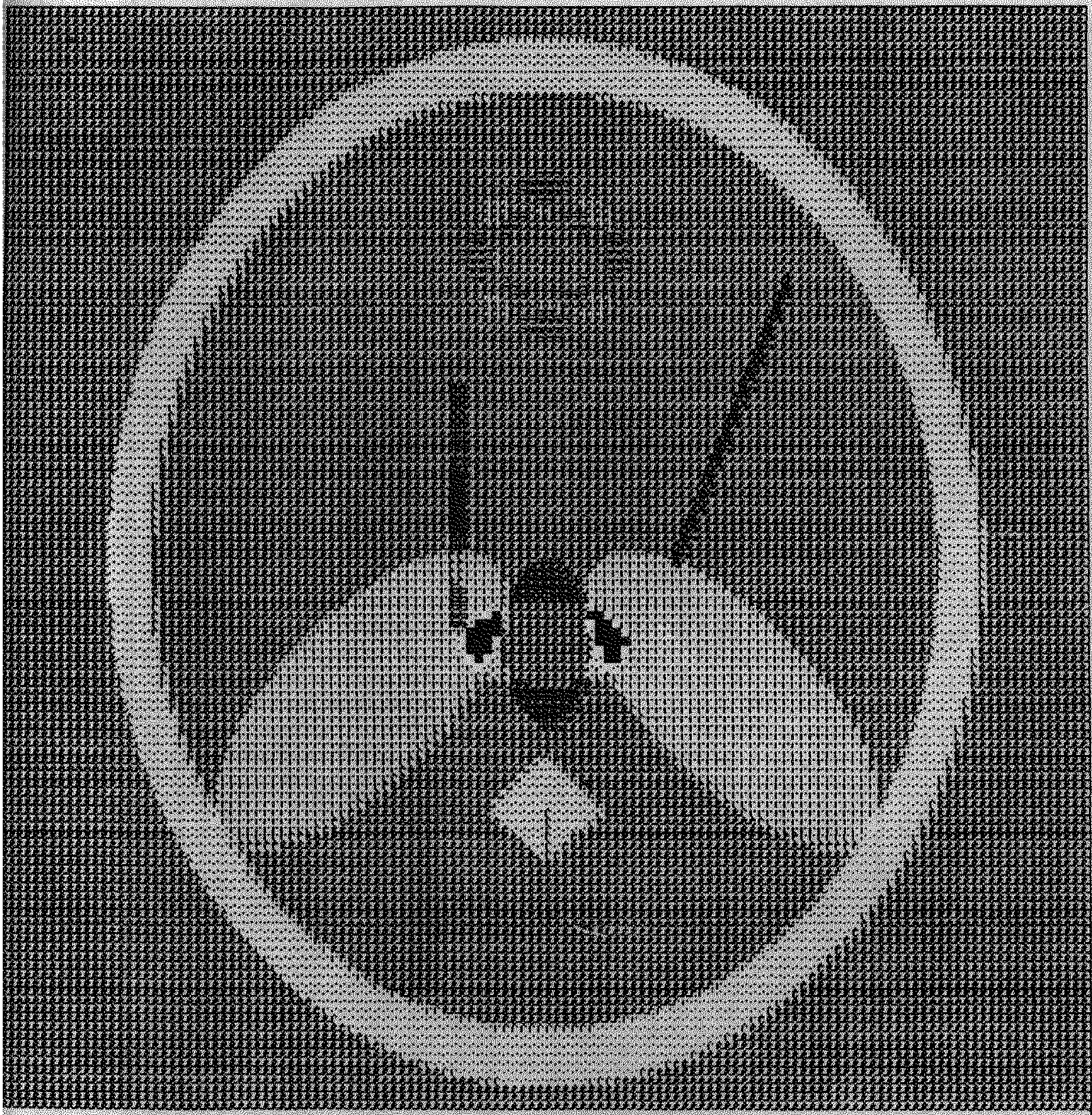


Figure 1

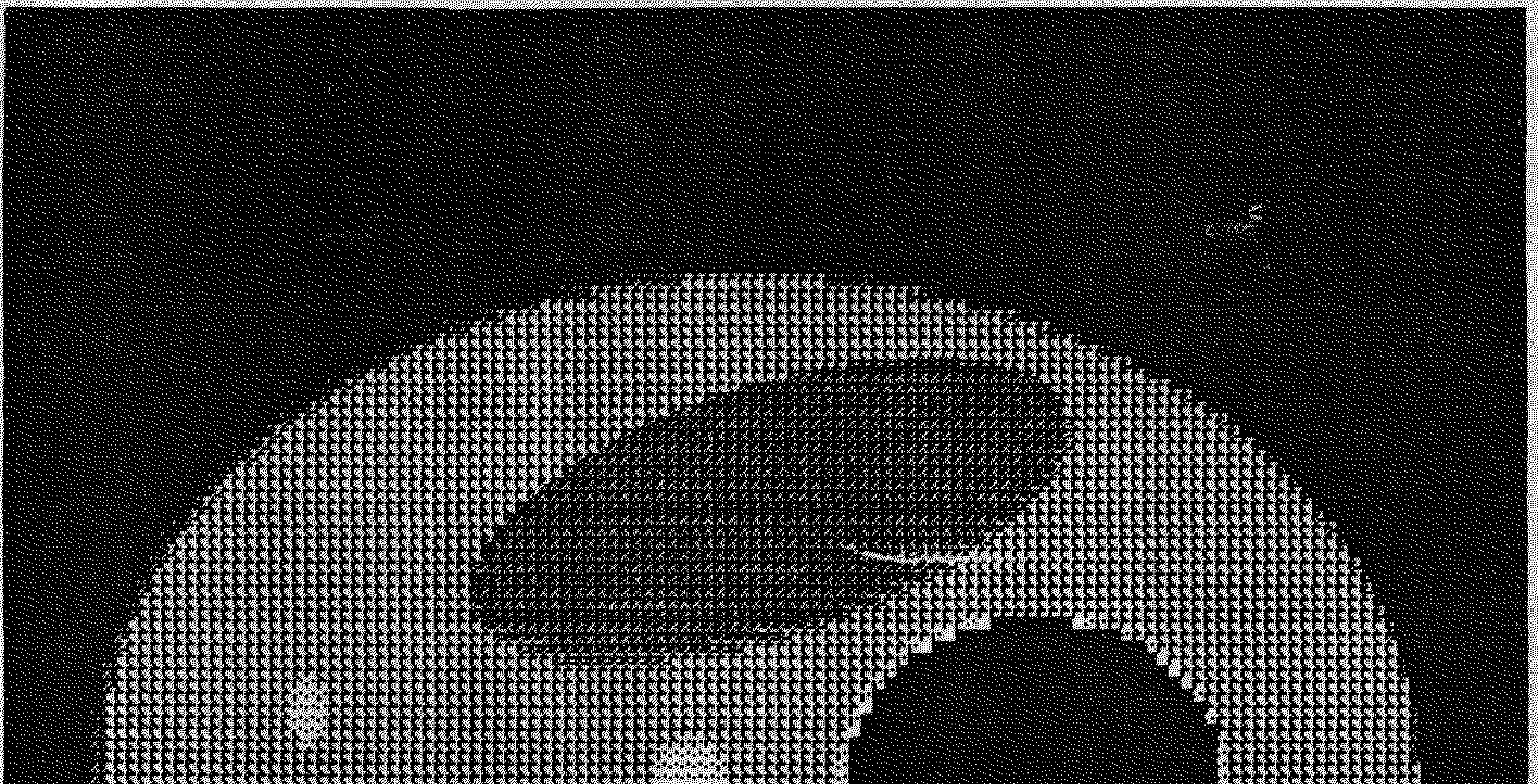


Figure 2

In the above algorithm, we find its time complexity to be $O((nk)^2m)$ where

m : number of elemental objects,

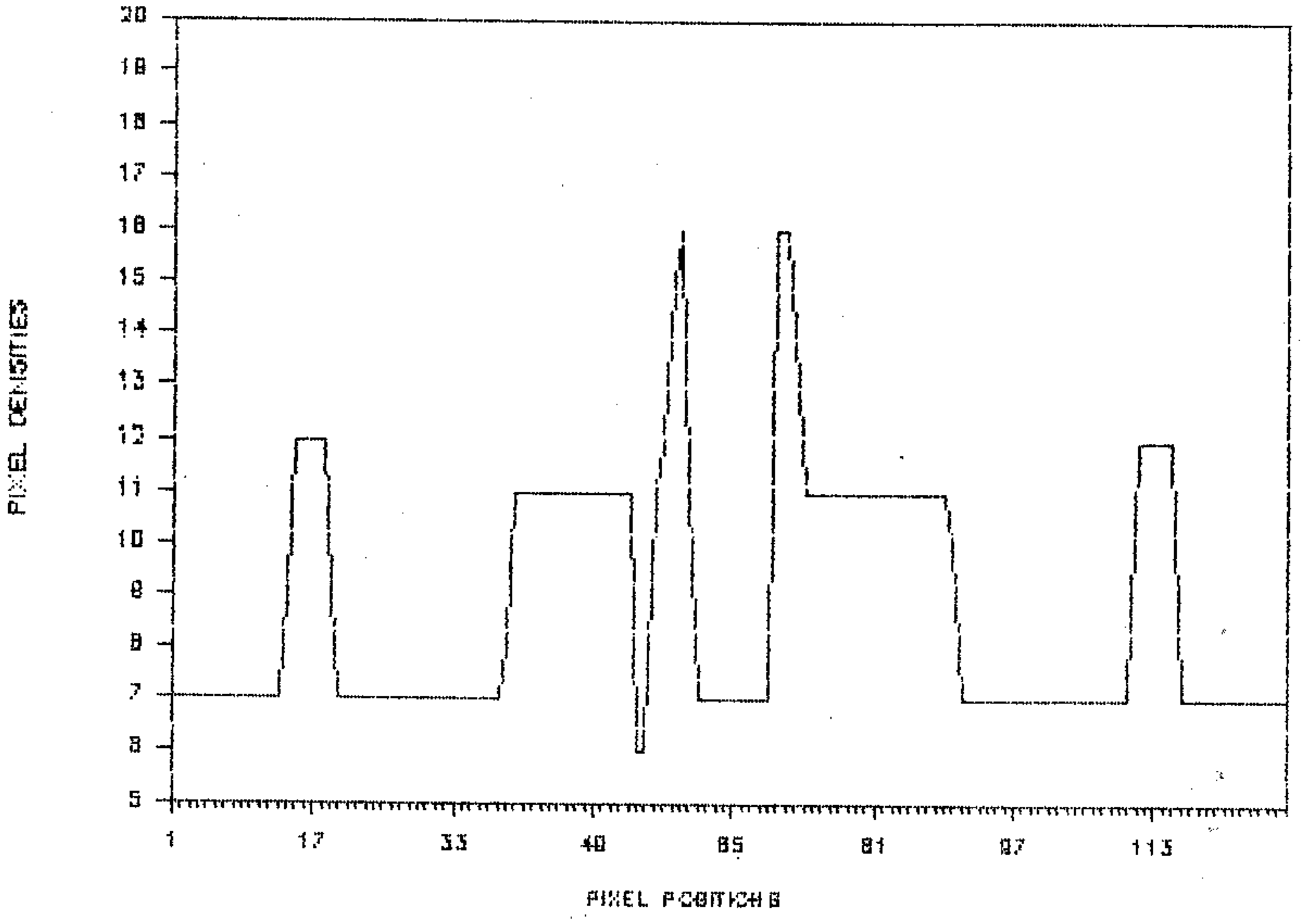
n : number of pixels in a row,

k : number of grids within a pixel, specified by the user

and its space complexity is $O((nk)^2)$

In the above program there is also provision of plotting the graph of densities of pixels lying in a particular row or coloumn. The output are as follows with input taken from table1 and table2 respectively. The graph of the 75th column of figure 2 and 70 th row of figure 1 are attached.

GRAPH OF 70TH COLUMN OF FIGURE 2



Chapter 2
Positron Emmision Tomography (PET)

Unlike CBP , the physical processes concerning PET is assumed to be stochastic in nature. Before we go into details of the physical processes concerning PET and their stochastic behaviour whatsoever let us discuss a bit on how the PET works.

(1)Mechanism of PET :

PET isotopes are characterised by the emission of a positron on decay. The emitted positron undergoes annihilation reaction with an atomic electron in its immediate vicinity producing two gamma ray photons. These photons fly off the point of annihilation in opposite directions along a line with a completely random (stochastically speaking it is uniformly distributed in space) orientation. While passing through the body some of the photons will be attenuated, while some penetrate through the body . In the PET measurement system those pairs of annihilation photons which travel along the plane of detectors in the system (geometrically all the detectors lie in one plane) as well as avoid absorption in the body of the patient are however detected by a pair of detector elements. Such an observation in respect of a photon pair constitutes a tube. In our pracical PET experiment, the number of photon counts observed by all such tubes in a finite duration of time is our measurement data. We will attach a diagram. Later on after make an elaboration on this in 'Simulation in PET system'.

(2)Mathematics behind the PET :

In PET, the decay of radionuclides are modelled as a spatial poisson point process with mean intensity function $\{\lambda(s); s\epsilon O,\}$ where O is the object space.

Assumptions regarding λ :

(a) We assume that the value of λ at a particular pixel is proportional to the concentration of the radionuclide at that pixel.

(b) As the decay of radionuclides are spontaneous in nature so the emisssions occuring in two different pixels are independent of one another.

With the above assumptios, our objective is to estimate the λ s statistically. So mathematically, we model the emission process X in the complete set of pixels as a spatially independent poisson process with the joint probability density function given by,

$$P(X = x) = \prod_{i=1}^{n} \frac{e^{-\lambda_i} \lambda_i^{x_i}}{x_i!}$$

where $x = \{x_i; i = 1, ..., n\}$ is the emission vector, x_i being the number of particles emitted and λ_i being the mean parameter of the poisson process in pixel i. Since the mean of the emission process is assumed to be proportional to the concentration of radionuclides , an estimate of the intensity parameter $\lambda = \{\lambda_i; i = 1, ..., n\}$ is also an estimate of the radionuclide concentrations in the whole object.

So far as the measurement process in PET is concerned , we have $y = \{y_j; j = 1, ..., m\}$ be the observed measurement data where y_j is the number of coincidence photons observed in tube j . Emissions ocurring in a pixel pixel i is assumed to be modelled as a Poisson random variable λ_i. A particle emitted in pixel i is detected in tube j with a probability p_{ij}. This amounts to considering that the number of particles which originate in pixel i and is detected in tube j is also a Poisson random variable with mean $\lambda_i p_{ij}$. Thus the measurements Y_j, j = 1, ..., m are independent Poisson variables with means given by

$$E(Y_j) = \sum_{i=1}^{n} \lambda_i p_{ij}, j = 1, ..., m$$

Hence the process y follows the Joint probability density function given by

$$P(Y = y) = \prod_{j=1}^{m} \frac{e^{-\sum_{i=1}^{n} \lambda_i p_{ij}} (-\sum_{i=1}^{n} \lambda_i p_{ij})^{y_j}}{y_j!}$$

The deterministic image reconstruction methods are limited by the suboptimal use of the statistical informations available from the models for the physical processes.

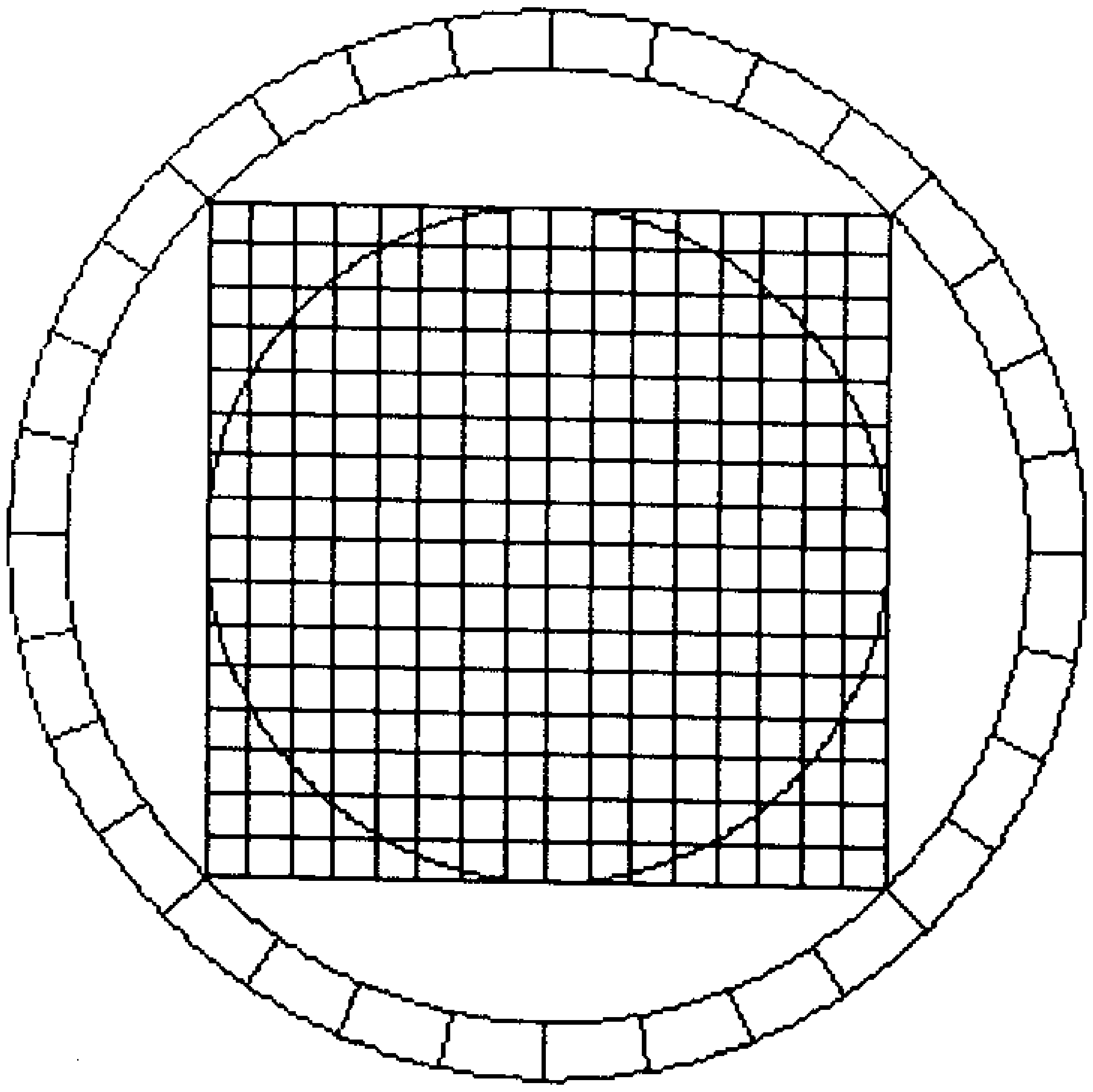
(3)Simulation of PET system :

In the previous discussion, we have had a little discussion on the physical and mathematical fundamentals of the PET. In our discussions so far we did observe that most of the physical processes involved in this imaging technique is stochastic in nature and hence the estimation of spatial concentrations of radionuclides basically boils down to a stochastic estimation. Using the necessary statistical rudimentaries different image reconstruction algorithms could be developed. At this stage, for evaluation of performance of these algorithms a PET system is simulated in computer. Now we will discuss about the simulation part mentioned above. To go into this let us first describe about its measurement geometry and corresponding pixel-detector probabilities. Basically we use circular ring geometry as our measurement system. The program for deriving the circular ring geometry is attached below.

```
program circle(input,output);
const pi=3.14159;
var x, cx,y,p,th,q,aa,bb,rr1, rr2 ,length : real;
    xx1,xx2, xx3, xx4, yy1, yy2, yy3, yy4 : real;
i,j,n, detno, picno : integer;


procedure grstrt(i,j:integer);external;
procedure draw(i,j:real);external;
procedure move(i,j:real);external;
procedure grstop;external;

procedure circle(a, b, r :real);
begin
  move(a+r,b);
  for i:= 1 to n do
  begin
    th := 2*pi/n*i;
    x:=a+r*cos(th);
    y:=b+r*sin(th);
    draw(x,y);
  end;
end;

procedure line (x1, y1, x2, y2 : real);

begin
move(x1, y1);
draw(x2, y2);
end;

function min (x, y :real): real;

begin
if x < y then
min := x
else
min := y;
end;

begin {main}
          grstrt(4209,1);

          writeln ('give origin and radii.');
          readln (aa, bb, rr1, rr2);
  writeln('give n.');
  readln(n);
  writeln ('give the number of detectors');
  readln (detno);
  writeln ('what is the number of pixels in a row');
  readln (picno);
circle(aa, bb, rr1);
circle(aa, bb, rr2);
cx := 2 * pi / detno;
for i := 1 to detno do
line(aa + rr1 * cos(cx * i), bb + rr1 * sin(cx * i),
aa + rr2 * cos(cx * i), bb + rr2 * sin(cx * i));
for i := 0 to 3 do
```

```
begin
j := 2 * i + 1;
      line(aa + min(rr1, rr2) * cos(j * pi/4),
bb + min(rr1, rr2) * sin(j * pi/4),
aa + min(rr1, rr2) * cos((j + 2) * pi/4),
bb + min(rr1, rr2) * sin((j + 2) * pi/4));


end;
xx1 := aa + min(rr1, rr2) * cos(3 * pi/4);
xx2 := aa + min(rr1, rr2) * cos(pi/4);
yy1 := bb + min(rr1, rr2) * sin(3 * pi/4);
yy2 := bb + min(rr1, rr2) * sin(pi/4);
xx3 := aa + min(rr1, rr2) * cos(5 * pi/4);
xx4 := aa + min(rr1, rr2) * cos(7 * pi/4);
yy3 := bb + min(rr1, rr2) * sin(5 * pi/4);
yy4 := bb + min(rr1, rr2) * sin(7 * pi/4);
for i := 1 to picno - 1 do
begin
line(xx1 + (xx2 - xx1) * i/picno,yy1 + (yy2 - yy1) * i/picno,
xx3 + (xx4 - xx3) * i/picno,yy3 + (yy4 - yy3) * i/picno);
line(xx1 + (xx3 - xx1) * i/picno,yy1 + (yy3 - yy1) * i/picno,
xx2 + (xx4 - xx2) * i/picno,yy2 + (yy4 - yy2) * i/picno);
end;
   grstop;
end.{main}
```

As a result of the above program we get the following diagram -

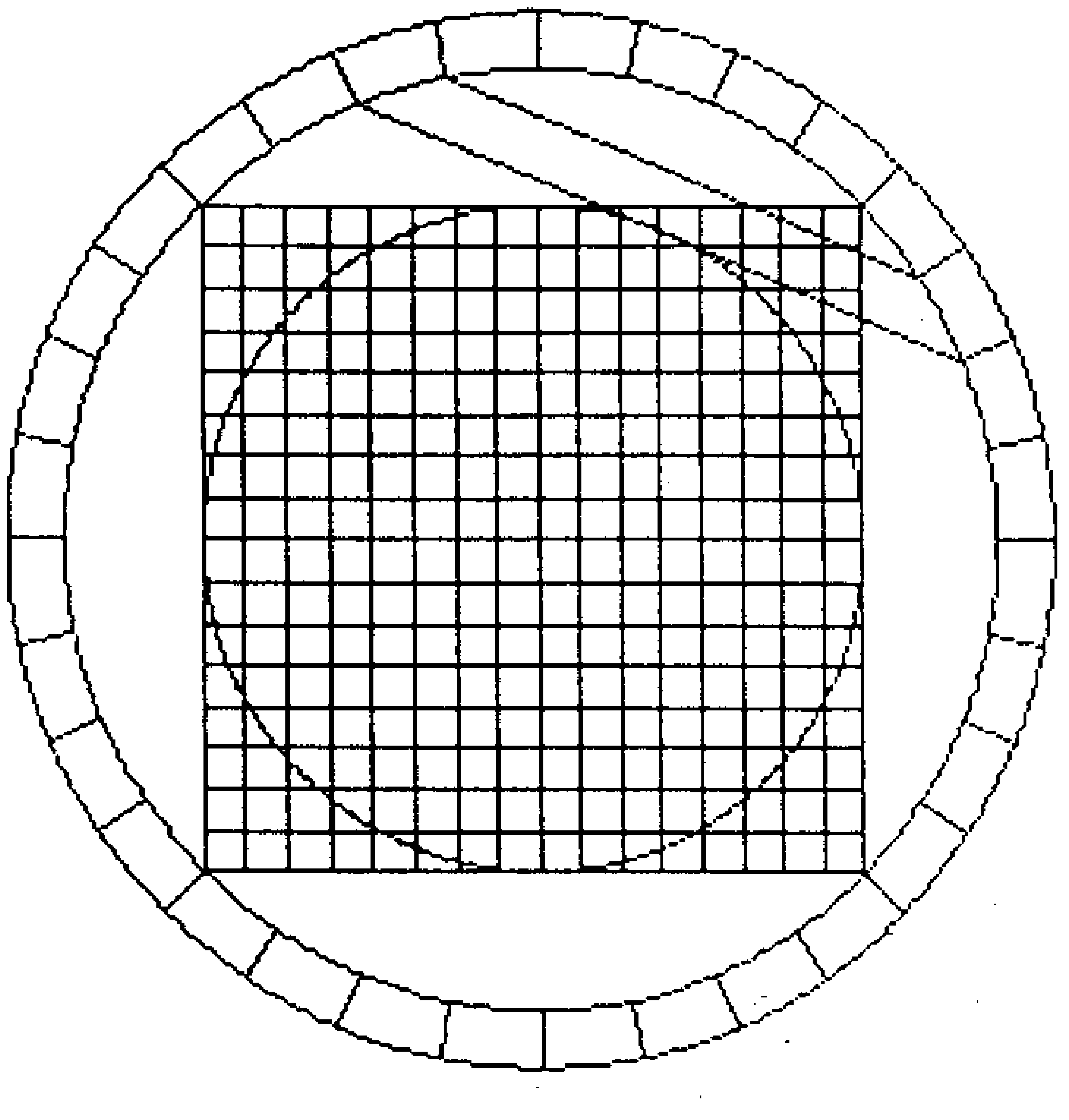


(4)Geometry behind the PET simulation :

The schematic figure is as shown in the figure. In this there are a number of detectors and there are a number of pixels in the picture frame. The detectors are kept inside the circular ring as shown. Within that circular ring the pixel frame is inscribed. The pixel frame is partitioned into small pixels which has been schematically shown in our reference figure. However, in the pixel frame we have our circular object space (the largest possible circle completely inscribed within the square pixel frame). Our assumption is that the phantom to be simulated lies geometrically inside the object space.

In PET, a positron emission and its subsequent annihilation with an electron produces two gamma ray photons which travel in the opposite direction as we have already stated. These photons are detected in time coincidence by a pair of detector elements situated in the outer circular ring. This pair of detectors constitute a tube a term which we have used a number of times in our discussion. The example of a tube is also evident from our reference picture.

An annihilation event ocurring inside a pixel has a finite probability for its detection in a given tube. Though this probability depends on many factors, however, so far our theoretical consideration of our simulation procedure is concerned we shall only consider the geometric factor. An annihilation event ocurred in pixel i is detected in detector j with a probability p_{ij}. The diagramatic representation of the above geometry is as follows



(5)Mathematical Phantom using PET :

The mathematical phantom we use is made up of eighteen elemental objects having different sizes , orientations and density values, viz table1. The characteristics of the objects are given in the above table. Each of these elemental objects are superimposed to obtain the phantom .

For simulating measurement data a Monte Carlo procedure is used. In this , each emission is simulated as follows. First a random point is chosen in the phantom. The concentration of radionuclide at the sampled point is assumed to be proportional to the density value of the phantom at that point. This point is taken as an emission point with probability proportional to the density value. For each of the accepted emission point, a randomly oriented (between 0 and π) line is selected and the detectors which this line intersect is found. The emission point corrosponds to the annihilation point and the random line corrosponds to the direction in which the pair of annihilation photons travel. The detectors which this line intersect are assumed to detect this annihilation event and the tube corrosponding to this pair of detectors is incremented. In this way, all the emissions are simulated and counted in the respective tubes, which is used as the measurement data for image reconstruction. We have used the following program to simulate this mathematical phantom -

```
program phantom (input, output);


label   1;

const   pi = 3.14259;
inf = 10000;

var     x_co, y_co : array[1..2] of real;
temp, angle1, angle2, th, r, cx, cy : real;
det : array[0..16] of real;
p : array [0..290, 0..290] of real;
tube : array [0..16, 0..16] of real;
tube_count : array [0..290] of integer;
emno,index,picno,start,final,i,j,n,row, col,detno, ind1, ind2 : integer;
```

```
filvar1, filvar2 : text;
filname1, filname2 : varying [15] of char;

function rnunf : real; external;

function tan(x: real): real;

begin
if cos(x) <> 0 then
tan := sin(x) / cos(x)
else
tan := inf;
end;{tan}

function discrim : real;

var     temp1, temp2 : real;

function sec(x :real): real;
begin
if cos(x) <> 0 then
sec := 1/cos(x) .
else
sec := inf;
end;{sec}

begin
n := picno;
temp1 := (((n + 1)/2 - row) + ((n +1)/2 - col) * tan(th)) ** 2;
temp2 := sec(th) * sec(th) * ((n + 1)/2 ** 2 - 2 * ((n + 1)/2) * row
 + ((n + 1)/2 - col) ** 2 - r ** 2);
discrim := temp1 - temp2;
end;{discrim}


function circle(x, y, cx, cy, r : real): boolean;
```

```
var     x_prm, y_prm, dis : real;

begin
x_prm := x - cx;
y_prm := y - cy;
x_prm := x_prm ** 2;
y_prm := y_prm ** 2;
dis := x_prm + y_prm - r ** 2;
if dis <= 0 then
circle := true
else
circle := false;
end;{circle}

procedure normalise;

begin
for ind1 := 1 to picno * picno  do
begin
temp := 0;
for ind2 := 1 to detno * detno do
        begin
temp := temp + p[ind1, ind2];
        end;
for ind2 := 1 to detno * detno do
if temp <> 0 then
p[ind1, ind2] := p[ind1, ind2] / temp
else
p[ ind1, ind2] := 0;
end;
end;{normalise}


begin {main}
writeln ('what is the number of pixels in a row');
readln (picno);
writeln ('how many detectors are there');
```

```
readln (detno);
temp := (2 * pi)/detno;
for i := 0 to detno do
det[i] := temp * i;
writeln ('how many emissions do you want');
readln (emno);
for index := 1 to emno do
begin
1: row := trunc(rnunf * picno) ;
col := trunc(rnunf * picno) ;
th := rnunf * pi;
if circle(row, col, picno/2, picno/2, picno/2) then
begin
if discrim > 0 then
begin
cx := picno/2;
cy := cx;
x_co[1] := row + cos(th) * ((cx - row) + (cy - col) * tan(th)
+ sqrt(discrim));
y_co[1] := col + sin(th) * ((cx - row) + (cy - col) * tan(th)
+ sqrt(discrim));
x_co[2] := row + cos(th) * ((cx - row) + (cy - col) * tan(th)
- sqrt(discrim));
y_co[2] := col + sin(th) * ((cx - row) + (cy - col) * tan(th)
- sqrt(discrim));
end
else
goto 1;
angle1 := arctan(y_co[1] / x_co[1]);
angle2 := arctan(y_co[2] / x_co[2]);
for i := 0 to detno do
begin
if ((det[i] < angle1) and (det[i + 1] > angle1)) then
start := i;
if ((det[i] < angle2) and (det[i + 1] > angle2)) then
final := i;
end;
```

```
for i := 1 to picno * picno do
for j := 1 to detno * detno do
p[i, j] := 0;
for i := 1 to detno do
for j := 1 to detno do
tube[i, j] := 0;
for i := 1 to detno * detno do
tube_count[i] := 0;
ind1 := (row - 1) * picno + col;
ind2 := (start - 1) * detno + final;
p[ ind1, ind2] := p[ind1, ind2] + 1;
tube[start, final] := tube[start, final] + 1;
tube[final, start] := tube[final, start] + 1;
tube_count[ind2] := tube_count[ind2] + 1;
end;
end;
normalise;
writeln ('give the filename of your tubecount');
readln (filname1);
open (filvar1, filname1, new);
rewrite (filvar1);
for i := 1 to detno * detno do
writeln (filvar1, tube_count[i]);
writeln ('give the filename of your pixel_detector probability');
readln (filname2);
open (filvar2, filname2, new);
rewrite (filvar2);
for i := 1 to picno * picno do
for j := 1 to detno * detno do
writeln (filvar2, p[i, j]);
close (filvar1);
close (filvar2);
end.
```

In the above procedure what we adopted is that for each emission we choose a point randomly in the object and it is accepted as an emission point

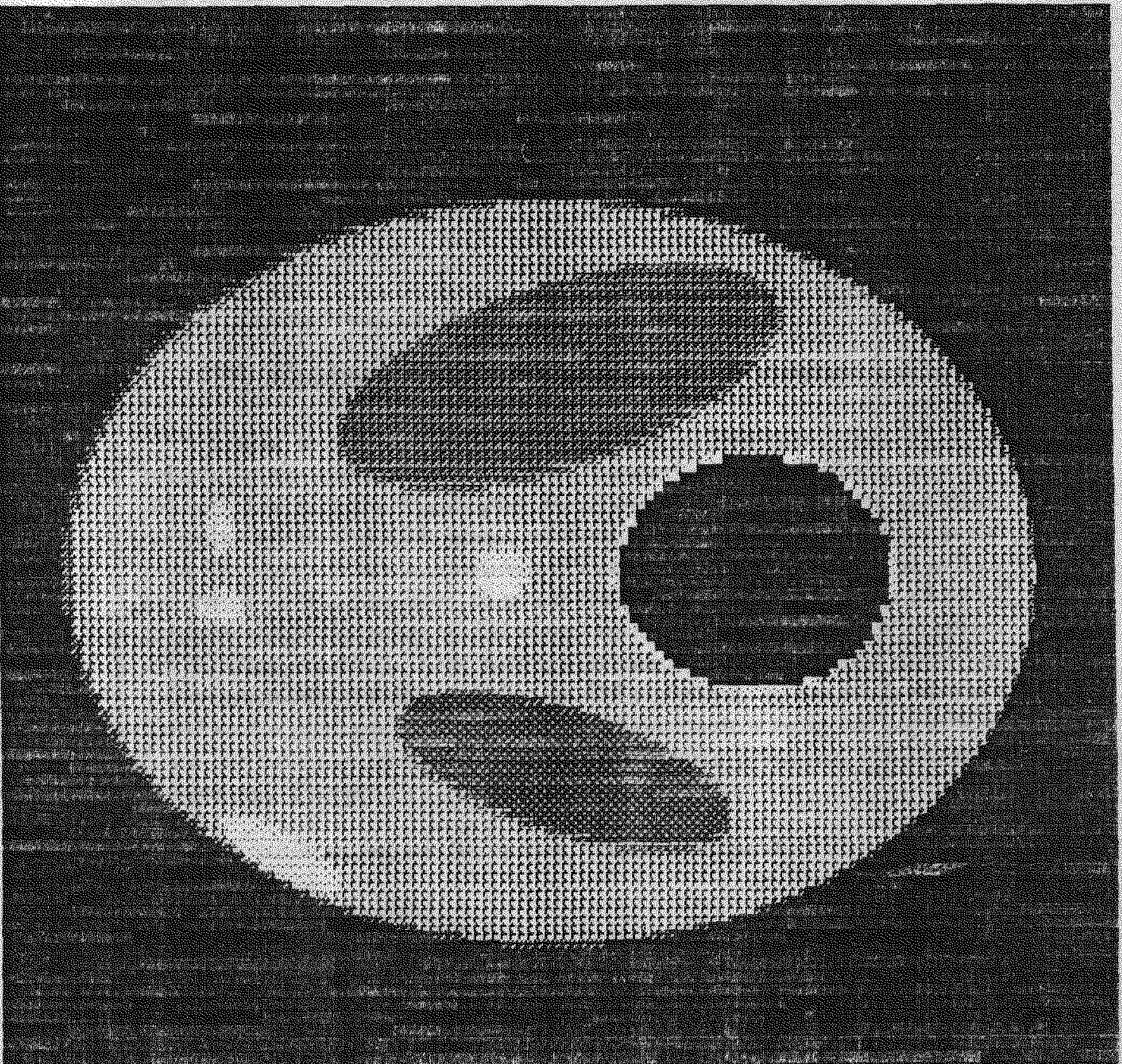


Figure 2

with probability proportional to the true object value at that point. For each accepted emission point a randomly oriented line is selected and the detectors which this line intersect is found . The tube corresponding to these two detectors is found and the corresponding to this tube is incremented by one. In this way a number of emissions are simulated as per the specification of the user.

In the above algorithm, we find its time complexity to be $O((nd)^2e)$

where d : number of detectors present,

n : number of pixels in a row,

e : number of emissions

and its space complexity is $O((nd)^2)$

(6) Estimation of mean pixel densities :

We have already seen that the image reconstruction problem, which is essentially the estimation of the mean parameters of the emission process. We will deal with the Expectation Maximisation (EM) estimation approach to emiss- ion tomographic image reconstruction.However we will get shortly that EM algo- rithm is computationally excessive space consuming as well as its convergence is very slow. In recent times there has been positive effort to improve the performance of EM algorithm by making certain modifications on it. They constitute Expectation Maximisation Search (EMS) algorithm , Overrelaxation procedure applied on EM algorithm. In our study we have developed a program to estimate the mean parameters using EM algorithm. Later we have also incorporated the Overrelaxation procedure as a program implement. This procedures are incorpo- rated below.

```
program expetation_maximization (input, output);

const   maxpic = 4100;
maxdet = 128;

type    arr = array [1..maxpic] of real;
```

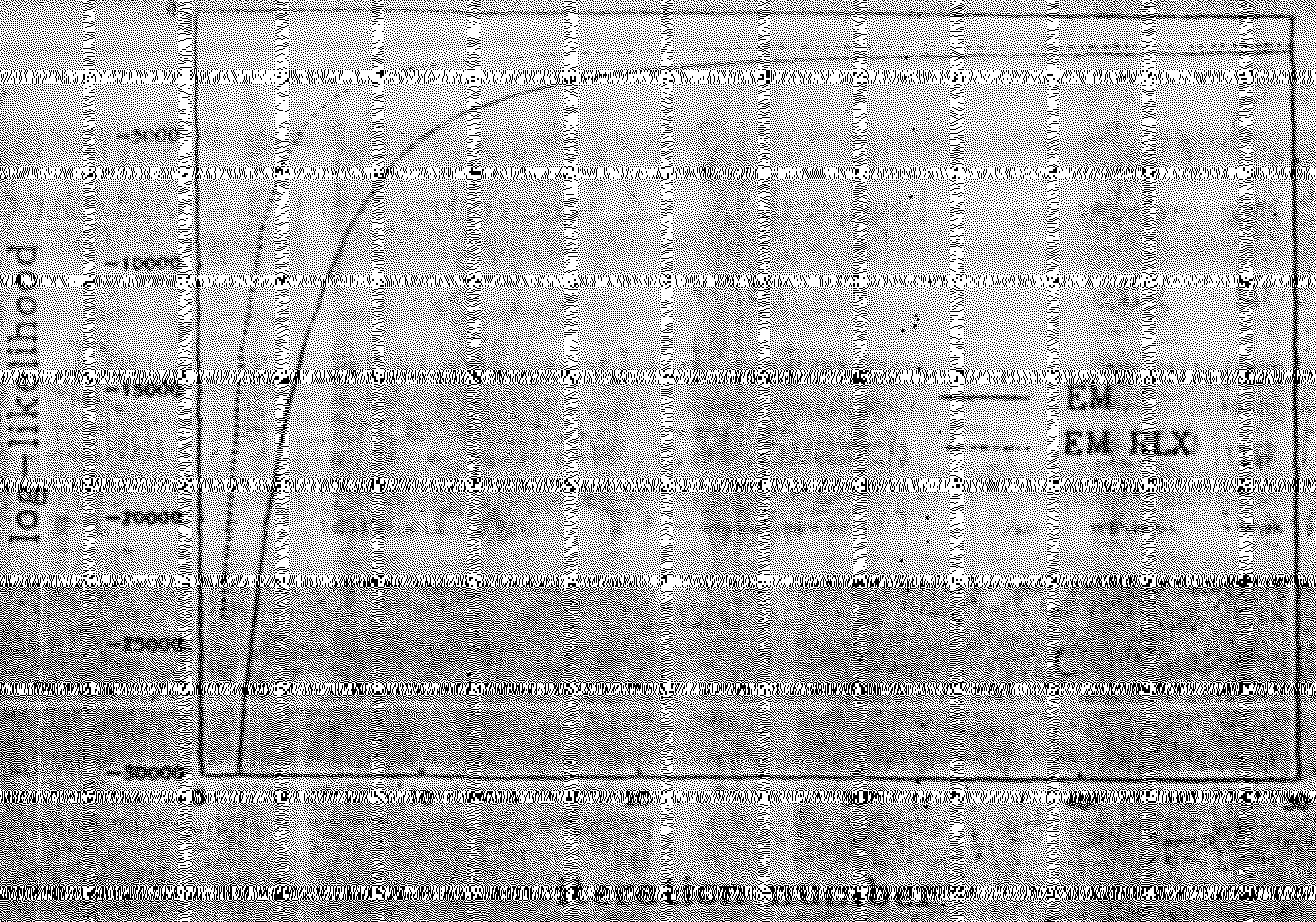
```
var     n, relx, count, i, j, detno, picno : integer;
temp, temp1, eps, parm, loc : real;
yyt : array [1..maxdet] of real;
yy : array [1..maxdet] of integer;
lam, a : arr;
p : array [1..maxpic, 1..maxdet] of real;
filvar1, filvar2, filvar3, filvar4, filvar5 : text;
infil1, infil2, infil3, infil4, infil5 : varying [15] of char;
response : varying[1] of char;

function  meu : real;

var     mumax , min :real;
m : array [1.. maxpic] of real;

begin
mumax := 256;
min := 0;
if a[i] < 0 then
m[i] := - (lam[i] / a[i])
else
m[i] := mumax;
for i := 1 to picno do
if m[i] < min then
min := m[i];
meu := min;
end;{*meu*}

procedure iteration(flag  : boolean);

begin
if count <= relx then
begin
count := count + 1;
for j := 1 to detno do
begin
temp := 0;
```

```
for i := 1 to picno do
temp := temp + lam[i] * p[i, j];
if temp <> 0 then
yyt[j] := (yy[j] - temp)/ temp;
end;
for i := 1 to picno do
begin
temp := 0;
temp1 := 0;
for j := 1 to detno do
begin
temp1 := temp1 + p[i, j];
temp := temp + yyt[j] * p[i, j];
end;
if temp1 <> 0 then
begin
a[i] := temp * lam[i]/ temp1;
if flag = true then
lam[i] := lam[i] + meu * a[i]
else
lam[i] := lam[i] + a[i];
end;
end;
end;
end; {* iteration *}

function convergence (xx : arr): boolean;

begin
for i := 1 to picno do
if xx[i] < eps then
convergence := true
else
convergence := false;
end;

procedure log_likelihood;
```

```
var     x : real ;


begin
x := 0;
temp := 0;
for j := 1 to detno do
begin
x := x + temp;
temp1 := 0;
for i := 1 to yy[j] do
temp1 := temp1 + ln(i);
temp := 0;
for i := 1 to picno do
temp := temp + lam[i] * p[i, j];
temp := -temp + ln(temp) * yy[j] + temp1;
end;
x := x + temp;
loc := x;
end;{*log_likelihood*}

procedure loop1(flag2 :boolean);

begin
repeat
iteration(flag2);
log_likelihood;
writeln (filvar5, count, loc);
until convergence(a);
end;{*loop1*}

procedure loop2 (flag1 :boolean);

begin
repeat
iteration(flag1);
```

```
until convergence(a);
end;{*loop2*}

begin {* main*}
writeln ('what is the number of pixels');
readln (picno);
writeln ('what is the number of detectors');
readln (detno);
writeln ('what is the probability input filename');
readln (infil1);
open (filvar1, infil1, old);
reset (filvar1);
for i := 1 to picno do
for j := 1 to detno do
readln (filvar1, p[i, j]);
writeln ('what is the mean_intensity filename');
readln (infil2);
open (filvar2, infil2, old);
reset (filvar2);
for i := 1 to picno do
readln (filvar2, lam[i]);
writeln ('what is the detector input filename');
readln (infil3);
open (filvar3, infil3, old);
reset (filvar3);
for j := 1 to detno do
readln (filvar3, yy[j]);
writeln ('what is the level of precision you want');
readln (eps);
count := 0;
writeln ('do you want log_likelihood function calculation(y/n)');
readln (response);
if ((response = 'y') or (response = 'Y')) then
begin
writeln ('what is the output filename of log likelihood');
readln (infil4);
open (filvar4, infil4, new);
```

```
rewrite (filvar4);
repeat
iteration(false);
log_likelihood;
writeln (filvar4, count, loc);
until convergence(a);
end
else
repeat
iteration(false);
until convergence(a);
writeln (' convergence in', count, 'iterations');
writeln (' do you want  overrelaxation method implementation');
readln (response);
if ((response = 'y') or (response = 'Y'))   then
begin
writeln ('how many intervals do you choose');
readln (n);
count := 0;
writeln ('do you want log_likelihood function calculation(y/n)');
readln (response);
if ((response = 'y') or (response = 'Y')) then
begin
writeln ('what is the output filename of log likelihood');
readln (infil5);
open (filvar5, infil5, new);
rewrite (filvar5);
loop1 (true);
end
else
loop2 (true);
writeln (' convergence in', count, 'iterations');
end;
close (filvar1);
close (filvar2);
close (filvar3);
close (filvar4);
```


log-likelihood
0
-5000
-10000
-15000
-20000
-25000
-30000
0
10
20
30
40
50
iteration number
EM
EM RLX

```
close (filvar5);
end.
```

(7) Basics of MLE :

We have already defined λ_i i = 1,...., n as well as y_j j = 1,...., m. Moreover we know what are p_{ij}. Our likelihood function is given by,

$$L(\lambda) = \prod_{j=1}^{m} \frac{e^{-\sum_{i=1}^{n} \lambda_i p_{ij}}(-\sum_{i=1}^{n} \lambda_i p_{ij})^{y_j}}{y_j!} \qquad (i)$$

The MLE($\hat{\lambda}^{ML}$), if it exists, is obtained by maximising the likelihood function $L(\lambda)$ or equivalently the log-likelihood function given by $\log(L(\lambda)) = l(\lambda)$. from equation (i) we have,

$$l(\lambda) = \sum_{j=1}^{M}\{-\sum_{i=1}^{N} \lambda_i p_{ij} + y_j log(\sum_{i=1}^{N} \lambda_i p_{ij}) - log(y_j!)\} \qquad (ii)$$

To maximise l(λ) we differentiate it with respect to λ and set the result to zero. We get that

$$\hat{\lambda}_i = \frac{\hat{\lambda}_i}{\sum_{j=1}^{m} p_{ij}} \sum_{j=1}^{m} \frac{y_j p_{ij}}{\sum_{k=1}^{n} \hat{\lambda}_k p_{kj}}, i = 1, ..., n \qquad (iii)$$

It can be mathematically checked that the hessian matrix consisting of the second order derivatives of l(λ) corresponding to the solutions obtained in (iii) of λ comes out to be negative definite. Hence we can at once conclude that the hypersurface l(λ) with respect to its axes λ_i -s is concave in nature and as such an MLE exists. Since the estimates obtained in (iii) are iterative in nature we can set the equations in (iii) iteratively as follows

$$\hat{\lambda}_i^{k+1} = \frac{\hat{\lambda}_i^{k}}{\sum_{j=1}^{m} p_{ij}} \sum_{j=1}^{m} \frac{y_j p_{ij}}{\sum_{l=1}^{n} \hat{\lambda}_l^{k} p_{lj}}, i = 1, ..., n \qquad (iv)$$

The EM algorithm given by equation (iii) has certain interesting pro- perties though we do not go into their proofs.

(I) The iterates have nondecreasing log-likelihood values. It means that

$$l(\hat{\lambda}^{k+1}) \geq l(\hat{\lambda}^{k}) \qquad (v)$$

This property is called the monotonicity property of EM algorithm.
(II) As the log-likelihood function is concave the equality in equation (v) means that the iterates have already converged to the required MLE solution. In that case we can say that

$$l(\hat{\lambda}^k) = l(\hat{\lambda}^{ML})$$

(III) The EM estimates have self normalising properties. This can be mathematically attributed as follows from equation (iv)

$$\sum_{i=1}^{n} \hat{\lambda}_i^{k+1} = \frac{\sum_j y_j}{\sum_j p_{ij}} \qquad (vi)$$

Since the right hand side is independent of k, the iteration number, therefore

$$\sum_{i=1}^{n} \hat{\lambda}_i^{k+1} = \sum_{i=1}^{n} \hat{\lambda}_i^{k} \qquad (vii)$$

Moreover, if $\sum_j p_{ij} = 1$ whose physical interpretation is that each emission ocurring in pixel i is detectable in one of the detectors then from equation (v) it is evident that

$$\sum_{i=1}^{n} \hat{\lambda}_i^k = \sum_{j=1}^{m} y_j \qquad (viii)$$

(8)Improvement of EM algorithm :

The EM iteration step in equation (iii) in the previous section can be rewritten in the additive form as follows

$$\hat{\lambda}_i^{k+1} = \hat{\lambda}_i^k + \frac{\hat{\lambda}_i^k}{\sum_{j=1}^{m} p_{ij}} \sum_{j=1}^{m} \frac{y_j - \sum_{l=1}^{n} \hat{\lambda}_l^k p_{lj}}{\sum_{l=1}^{n} \hat{\lambda}_l^k p_{lj}} p_{ij}, i = 1, \ldots, n \qquad (ix)$$

Vectorially it can be represented as follows

$$\hat{\lambda}^{k+1} = \hat{\lambda}^k + \Delta(\hat{\lambda}^k) \qquad (x)$$

where

$$\Delta(\hat{\lambda}_i^k) = \frac{\hat{\lambda}_i^k}{\sum_{j=1}^{m} p_{ij}} \sum_{j=1}^{m} \frac{y_j - \sum_{l=1}^{n} \hat{\lambda}_l^k p_{lj}}{\sum_{l=1}^{n} \hat{\lambda}_l^k p_{lj}} p_{ij}, i = 1, \ldots, n \qquad (xi)$$

$\Delta(\hat{\lambda}_i^k)$ is the i-th component of $\Delta(\hat{\lambda}^k)$ and $\hat{\lambda}_i^k$ is the i-th component of $\hat{\lambda}^k$. It is here we introduce the method of overrelaxation where we substitute the second part of (x) with $\mu^k\Delta(\hat{\lambda}^k)$ so that one can regulate the conver- gence rate of the algorithm. In our program, we could find how did we choose our μ^k.

Conclusion

There is a lot of scope which we did not cover in our present discussion. For example in improving the EM algorithm we only concentrated on overrelaxation method. There are other methods as well. e,g EMS algorithm vector extrapolated maximum likelihood estimators etc. In future they need a lot of attention. Moreover we mainly concentrated on CBP and PET. However, what we did not cover at all is MRI. Recently a lot of work has been being done in this area. That is another angle where we can concentrate. What remained uncovered as yet is the application of Bayesian approach in different methods. In this field also there is a lot of scope left.

Reference

1. Andrews, H.C. and Hunt, B.R. Digital Image Restoration.
2. Brooks, R.A. and Weiss, G.H. Interpolation problem in image reconstruction.
3. D.Duttamajumder and S.Banerjee Mathematical techniques in Bio-Medical imaging.
4. Budinger,T.F., Gullberg, G.T., and Huesman, R.H Emission Comoputed Tomography in 'Image Reconstruction from Projections': Implementation and Application.
5. Herman, G.T Image Reconstruction from Projections: Implementation and Apllications.
6. Rajeevan, N. Ph.D thesis, Indian Institute of Science, Bangalore, India,1991.