1. In this problem we use Monte Carlo integration to approximate the area of a circle of radius, \( R = 1 \).

A. For method 1, the idea is to conduct a binomial experiment using the random number generator:

1. Let \( \Gamma \) be a set that contains \( \Omega \) and has known area (or volume in 3-dimensions), \( A \).
2. Generate \( n \) randomly distribution points \( z_i \) of element of \( \Gamma \).
3. Let \( n' \) be the number of these points that lie in \( \Omega \) and then estimate the area of \( \Omega \) by \( R_n = n'/n \cdot A \).

This method is binomially distributed with the probability of success, \( p \), equal to the ratio of the areas, \( p = \Omega/\Gamma \). Recall that the variance or error in the estimate is given by \( \sigma^2 = np(1-p) \), and therefore the method converges to \( p \) at a rate determined by \( \mu/\sigma = np/\sqrt(np(1-p)) = \sqrt(np/(1-p)) \). Convergence is at a rate proportional to \( \sqrt{n} \).

B. For method 2, the general approach can be stated as: \( I = \int_{\Omega} f(x) \, dx = \langle f(x) \rangle = \int_{d\Omega} \), where \( \Omega \) is the interval \([-R, R]\) for the specific case of the circle problem. The algorithm can be summarized as:

1. Generate \( n \) points \( z_i \) that are a subset of \( \Omega \).
2. The average value of \( f \) in the region \( \Omega \) is approximated by,
   \[
   \mu_n = \tfrac{1}{n} \sum_{i=1}^{n} f(z_i) .
   \]
3. The estimate of the value of the integral is then \( I_n = \mu_n \int_{d\Omega} \).

Notice that for method B the expectation value of \( I \) is the integral’s true value and since this is a random process we expect the estimates of the integral, \( I_n \), to be Gaussian distributed about the true value \( I \) with variance, \( \sigma^2 = \int_{\Omega} (f(x) - I)^2 \, dx \).

Notice that for Method 2 the expectation value of \( I \) is the true value of the integral, not just the average value of the ratio of areas. Also, since this is a random process we expect the estimates of the integral, \( I_n \), to be distributed about the true value of \( I \) with variance, \( \sigma^2 = \int_{\Omega} (f(x) - I)^2 \, dx \). Moreover, since we are making \( n \) independent samples of the mean, the estimate is Gaussian distributed by the Central Limit Theorem. Convergence is therefore, \( \mu/\sigma = \sqrt{(n) \langle f(x) \rangle}/\sigma \). Again, we converge at a rate proportional to root \( n \). But the original estimate is closer to the true value of the integral so Method 2 is more accurate at lower number of trials.

<table>
<thead>
<tr>
<th>( n )</th>
<th>Method 1</th>
<th>Method 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.1246</td>
<td>0.0126</td>
</tr>
<tr>
<td>100</td>
<td>0.0854</td>
<td>0.0387</td>
</tr>
<tr>
<td>100,000</td>
<td>0.0014</td>
<td>8.6e-004</td>
</tr>
</tbody>
</table>
2. Problem 3,
   A. ART solution (from class):
   % Direct algebraic reconstruction of test image, I:
   >> I

   I =

   0   100
   200   50

   >> A

   A =

   1     0     1     0
   0     1     0     1
   0     0     1     1
   1     1     0     0

   >> g

   g =

   200
   150
   250
   100

   >> f = A\g
   Warning: Matrix is singular to working precision.

   f =

   -50
   150
   250
% The data are incomplete for 2 projections. With 3 projections (2 orthogonal and 1 diagonal), then
A =
    1  0  1  0
    0  1  0  1
    0  0  1  1
    1  1  0  0
    1  0  0  1 % This row added for diagonal projection at phi = 135 degrees

g =
    200
    150
    250
    100
    50

>> f = A\g

f =
    0.0000
    100.0000
    200.0000
    50.0000

Needed 5 equations (i.e. projection values) to solve for 4 unknowns (i.e. the pixel values) for the
direct solution.

B. E-M Matlab function: This is inherently iterative by definition. I’ve written a simple function to
iterate using 2 orthogonal or 2 orthogonal plus 1 diagonal projection. The EM algorithm does not
obtain the correct result even for 100 iterations when only 2 orthogonal projections are used. For 3
projections, the result is nearly exact for 100 iterations and very close for 10 iterations.

function [f_est,e] = expmax(N,P,f,I,Np)
% Expectation Maximization
% [f_est] = expmax(N,P,f,I)
% N = number of pixels in image
% P = number of detector bins
% Np = number of projections (1 to 3) only orthogonal or diagonal pojection
% angles allowed.
% f = true image
% I = number of iterations to perform
E-M solution for 2 orthogonal projections:

\[
\begin{bmatrix}
57.1429 & -57.1429 \\
-57.1429 & 57.1429
\end{bmatrix}
\]

E-M solution for 3 projections (2 orthogonal and 1 diagonal):

\[
\begin{bmatrix}
2.5340 & -2.5340 \\
-0.5841 & 0.5841
\end{bmatrix}
\]

\[
\begin{bmatrix}
2.5340 & 97.4660 \\
199.4159 & 50.5841
\end{bmatrix}
\]

Absolute error for 100 iterations with 3 projections:

\[
\begin{bmatrix}
0.2390 & -0.2390 \\
-0.0484 & 0.0484
\end{bmatrix}
\]

Absolute error for 100 iterations with 2 projections:

\[
\begin{bmatrix}
57.1429 & -57.1429 \\
-57.1429 & 57.1429
\end{bmatrix}
\]
function [f_est,e] = expmax(N,P,f,I,Np)
% Expectation Maximization
% [f_est] = expmax(N,P,f,I)
% N = number of pixels in image
% P = number of detector bins
% Np = number of projections (1 to 3) only orthogonal or diagonal projection
% angles allowed.
% f = true image
% I = number of iterations to perform
% 
% g1 = sum(f);
g2 = sum(f');
g3 = sum(diag(f));
base = zeros(N);
f_est{1,1} = ones(N);
for k = 1:I,
    %Projection subframe 1
    f1 = sum(f_est{k,1});
r1 = g1./f1;
    base(:,1) = r1(1);
    base(:,2) = r1(2);
    f_est{k,2} = f_est{k,1}.*base;
    %Projection subframe 2
    base = zeros(N);
    f2 = sum(f_est{k,2}');
r2 = g2./f2;
    base(1,:) = r2(1);
    base(2,:) = r2(2);
    if(Np>2)
        f_est{k,3} = f_est{k,2}.*base;
        %Projection subframe 3
        base = zeros(N);
        f3 = sum(diag(f_est{k,3}));
r3 = g3./f3;
        base(1,1) = r3(1);
        base(2,2) = r3(1);
        base(2,1) = r2(2);
        base(1,2) = r2(1);
        f_est{k+1,1} = f_est{k,3}.*base;
    else
        f_est{k+1,1} = f_est{k,2}.*base;
    end
end

e = f_est{1,Np}-f;
3. Basic operation of the ordered subsets concept:
MP/BME 574 Application 4, Solutions
A. Comparison of OSEM performance with increasing undersampling factor:
B. Comparison of OSEM performance with interpolation method (I used 90 projections with 3 subsets of 30 projections and 10 iterations):

NN interpolation (recon time 56 seconds):

Linear interpolation (recon time 67 seconds):

There is a textured interpolation artifact that is clearly visible in both images. The linear interpolation result is slightly smoother and mitigates the severity of this artifact. For further comparison, I’ve added the cubic spline interpolation result below which differs very little from the linear interpolation result. Note the factor of 3 increase in computation time:

Spline interpolation (recon time 208 seconds):

By the way, I added the following to the script to convert the image to integer data type and scale to avoid quantization noise:

```matlab
osem_final = 1000.*im_osem_current;
osem_final = uint8(osem_final);
figure; imshow(osem_final,[]);colorbar;truesize; title('EM reconstruction');brighten(0.1)
```
By the way, I added the following to the script to suppress streak artifact emanating from amplified noise at the corners of the image:

```matlab
mask = ones(256);
N = 256;
R = N/2;
xx = 1:256;
yy = 1:256;
[XX,YY] = meshgrid(xx,yy);
RR = sqrt((XX-N/2).^2+(YY-N/2).^2);
mask(find(RR(:)>R))=0;
figure; imshow(mask,[]);
```