Homework 2 answers

① What is the complexity of using Gaussian elimination to solve $Ax = b$, where A is an $n \times n$ **tridiagonal** matrix? Is it $O(n)$? $O(n^2)$? $O(n^3)$? Justify your answer!

Answer: It is $\boxed{O(n)}$.

Here is the idea:

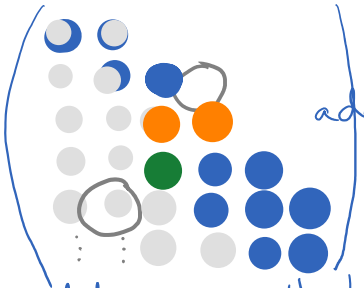
Normally (for a dense matrix), GE takes $O(n^3)$ steps, because:

- There are n rows, each of which you add to the rows below it to cancel out entries in a column.
  - To add a multiple of one row to another takes $O(n)$ operations.
  - And there are $O(n)$ rows below the row that you have to fix.

$\Rightarrow O(n \cdot n \cdot n) = O(n^3)$ steps.

In this case, for a tridiagonal matrix,

- There are still n rows.
  - But you only have to add a multiple of a row to the 1 row beneath it



  - and adding a multiple of one row to another only takes $O(1)$ time (a constant), because the row has only 2 nonzero entries.

$\Rightarrow O(n \cdot 2 \cdot 1) = O(n)$ steps overall.

Back-substitution also takes only $O(n)$ steps.

② Find the degree-three polynomial $p(x) = ax^3 + bx^2 + cx + d$ with $p(1) = 4$ $p'(-1) = 0$

$p(2) = 2 \qquad p'(2) = 6$

- First solve for p by using Gaussian elimination on a set of linear equations.
  Show your work. Please use compact matrix notation to simplify grading.
- Now enter the matrix into Matlab, Mathematica, or equivalent software, and solve the equations there. Again, show your work (commands and results).

$p'(x) = 3x^2 a + 2x b + c$

Equations
$$p(1) = a + b + c + d = 4$$
$$p(2) = 8a + 4b + 2c + d = 2$$
$$p'(-1) = 3a - 2b + c = 0$$
$$p'(2) = 12a + 4b + c = 6$$

We'll cancel out the variables in order d, c, b, a:

$$\left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 4 \\ 8 & 4 & 2 & 1 & 2 \\ 3 & -2 & 1 & 0 & 0 \\ 12 & 4 & 1 & 0 & 6 \end{array}\right) \xrightarrow{2-1} \left(\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 4 \\ 7 & 3 & 1 & 0 & -2 \\ 3 & -2 & 1 & 0 & 0 \\ 12 & 4 & 1 & 0 & 6 \end{array}\right) \begin{matrix} 3-1 \\ 4-1 \end{matrix}$$

$$\rightarrow \left(\begin{array}{cccc|c} -6 & -2 & 0 & 1 & 6 \\ 7 & 3 & 1 & 0 & -2 \\ -4 & -5 & 0 & 0 & 2 \\ 5 & 1 & 0 & 0 & 8 \end{array}\right) \begin{matrix} \\ \\ -3 \\ -5 \end{matrix} \rightarrow \left(\begin{array}{cccc|c} 4 & 0 & 0 & 1 & 22 \\ -8 & 0 & 1 & 0 & -26 \\ 21 & 0 & 0 & 0 & 42 \\ 5 & 1 & 0 & 0 & 8 \end{array}\right)$$

From the 3rd equation (3rd row), $a = 2$. Substituting into the last equation gives $b = -2$. And from the first two equations, $d = 22 - 4a = 14$, $c = -26 + 8a = -10$.

$$\Rightarrow \boxed{p(x) = 2x^3 - 2x^2 - 10x + 14}$$

Mathematica commands:

```
In[1]:= A = {{1, 1, 1, 1}, {8, 4, 2, 1}, {3, -2, 1, 0}, {12, 4, 1, 0}};
         b = {4, 2, 0, 6};
         LinearSolve[A, b]
Out[3]= {2, -2, -10, 14}
```

Matlab commands:

```
octave-3.2.3:1> A=[1,1,1,1; 8,4,2,1; 3,-2,1,0; 12,4,1,0];
octave-3.2.3:2> b=[4;2;0;6];
octave-3.2.3:3> A\b
ans =

    2.0000
   -2.0000
  -10.0000
   14.0000
```

③ By hand (not using a computer), find all solutions to the equations:

$$x_1 + x_2 + 2x_3 = 1$$
$$3x_1 + 3x_3 + 3x_4 = 6$$

$$2x_1 + 2x_2 - x_3 + x_4 = 5$$

Answer: Divide the second equation by 3 to get the system:

$$\left(\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 1 & 0 & 1 & 1 & 2 \\ 2 & 2 & -1 & 1 & 3 \end{array}\right) \xrightarrow{-1,\,-2} \left(\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & -5 & 1 & 1 \end{array}\right) \xrightarrow{\cdot(-\frac{1}{5})} \left(\begin{array}{cccc|c} 1 & 1 & 2 & 0 & 1 \\ 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 1 & -\frac{1}{5} & -\frac{1}{5} \end{array}\right)$$

$$\xrightarrow{-1} \left(\begin{array}{cccc|c} 1 & 1 & 0 & \frac{2}{5} & \frac{7}{5} \\ 0 & 1 & 0 & -\frac{4}{5} & -\frac{4}{5} \\ 0 & 0 & 1 & -\frac{1}{5} & -\frac{1}{5} \end{array}\right) \xrightarrow{-1} \left(\begin{array}{cccc|c} 1 & 0 & 0 & \frac{6}{5} & \frac{11}{5} \\ 0 & 1 & 0 & -\frac{4}{5} & -\frac{4}{5} \\ 0 & 0 & 1 & -\frac{1}{5} & -\frac{1}{5} \end{array}\right)$$

$$\Rightarrow x_3 = \tfrac{1}{5}(-1 + x_4)$$

(x_4: free variable)

$$x_2 = \tfrac{1}{5}(-4 + 4x_4)$$

$$x_1 = \tfrac{1}{5}(11 - 6x_4)$$

$$\Rightarrow \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 11/5 \\ -4/5 \\ -1/5 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} -6/5 \\ 4/5 \\ 1/5 \\ 1 \end{pmatrix}$$

is the general solution — a line!

③ Determine, by hand, which of the following systems of equations has at least one solution.

(Jargon: If a system of equations has at least one solution, we say it is "consistent." It is "inconsistent" if there are no solutions.)

(a)
$$\begin{aligned} x + 2y + z &= 2, \\ 2x + 4y &= 2, \\ 3x + 6y + z &= 4. \end{aligned}$$

(b)
$$\begin{aligned} 2x + 2y + 4z &= 0, \\ 3x + 2y + 5z &= 0, \\ 4x + 2y + 6z &= 0. \end{aligned}$$

(c)
$$\begin{aligned} x - y + z &= 1, \\ x - y - z &= 2, \\ x + y - z &= 3, \\ x + y + z &= 4. \end{aligned}$$

(d)
$$\begin{aligned} x - y + z &= 1, \\ x - y - z &= 2, \\ x + y - z &= 3, \\ x + y + z &= 2. \end{aligned}$$

(e)
$$\begin{aligned} 2w + x + 3y + 5z &= 1, \\ 4w + 4y + 8z &= 0, \\ w + x + 2y + 3z &= 0, \\ x + y + z &= 0. \end{aligned}$$

(f)
$$\begin{aligned} 2w + x + 3y + 5z &= 7, \\ 4w + 4y + 8z &= 8, \\ w + x + 2y + 3z &= 5, \\ x + y + z &= 3. \end{aligned}$$

Answer:

a) Let $w = x + 2y$. Then the equations simplify to

$$\begin{aligned} w + z &= 2 \\ 2w &= 2 \\ 3w + z &= 4 \end{aligned} \quad \Rightarrow w = 1, z = 1$$

$\Rightarrow$ There are infinitely many solutions $x = 1-2y, z=1$.

b) $(x,y,z) = (0,0,0)$ is a solution, so the equations are consistent. (Any homogeneous set of linear equations is consistent!)

c,d) Notice that the left-hand sides in (c) and (d) are the same. Also,

$$(x+y+z) = (x-y+z) - (x-y-z) + (x+y-z),$$

so for the equations

$$\begin{aligned} x-y+z &= b_1 \\ x-y-z &= b_2 \\ x+y-z &= b_3 \\ x+y+z &= b_4 \end{aligned}$$

to be consistent, it must be that $b_4 = b_1 - b_2 + b_3$.

$\Rightarrow$ (c) is inconsistent.

For (d), adding the 1st and 3rd equations gives $x=2$, whence $y=1/2$, $z=-1/2$ $\Rightarrow$ (d) is consistent!

e,f) Once again, the left-hand sides are the same. We can use the Gauss-Jordan elimination procedure to solve them simultaneously:

$$\left(\begin{array}{cccc|cc} 2 & 1 & 3 & 5 & 1 & 7 \\ 4 & 0 & 4 & 8 & 0 & 8 \\ 1 & 1 & 2 & 3 & 0 & 5 \\ 0 & 1 & 1 & 1 & 0 & 3 \end{array}\right) \xrightarrow{\text{-1, -1}} \left(\begin{array}{cccc|cc} 2 & 0 & 2 & 4 & 1 & 4 \\ 4 & 0 & 4 & 8 & 0 & 8 \\ 1 & 0 & 1 & 2 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 & 3 \end{array}\right) \xrightarrow{\text{-4, -2}} \left(\begin{array}{cccc|cc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 2 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 & 3 \end{array}\right)$$

Thus (e) is inconsistent; we have $0\cdot w + 0\cdot x + 0\cdot y + 0\cdot z = 1$.

Part (f) is consistent, with infinitely many solutions:

$$\begin{aligned} w &= 2-y-2z \\ x &= 3-y-z \end{aligned} \quad , \; y \text{ \& } z \text{ free}$$

④ By hand (not using a computer), find all polynomials

$$p(x) = a + bx + cx^2 + dx^3$$

that satisfy $p(1)=2$, $p'(3)=1$ and $p''(2)=0$.

$$\begin{aligned} 2 = p(1) &= a+b+c+d \\ 1 = p'(3) &= \quad b+6c+27d \\ 0 = p''(2) &= \quad\quad 2c+12d \end{aligned} \qquad \begin{aligned} p'(x) &= b+2cx+3dx^2 \\ p''(x) &= 2c+6dx \end{aligned}$$

$1 = p'(3) = b + 6c + 12d$ $\quad p(x) = b + 2cx + 3dx^2$

$0 = p''(2) = 2c + 12d$ $\quad p''(x) = 2c + 6dx$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 6 & 27 \\ 0 & 0 & 2 & 12 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

d is free

$c = -6d$

$b = 1 - 6c - 27d$

$= 1 + 9d$

$a = 2 - b - c - d$

$= 2 - (1+9d) - (-6d) - d$

$= 1 - 4d$

$$\Rightarrow \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} + t \begin{pmatrix} -4 \\ 9 \\ -6 \\ 1 \end{pmatrix}$$

where t is arbitrary

⑤ Use the Gauss-Jordan method to find the 4×2 matrix B satisfying

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 2 & 2 \\ 3 & 2 & 1 & 1 \\ 1 & 2 & 0 & 0 \end{pmatrix} B = \begin{pmatrix} 50 & 60 \\ 38 & 48 \\ 21 & 28 \\ 7 & 10 \end{pmatrix}$$

Do it by hand, and show your work.

Answer:

$$\left(\begin{array}{cccc|cc} 1 & 2 & 3 & 4 & 50 & 60 \\ 2 & 4 & 2 & 2 & 38 & 48 \\ 3 & 2 & 1 & 1 & 21 & 28 \\ 1 & 2 & 0 & 0 & 7 & 10 \end{array}\right) \xrightarrow{\begin{matrix} -1 \\ -2 \\ -3 \end{matrix}} \left(\begin{array}{cccc|cc} 0 & 0 & 3 & 4 & 43 & 50 \\ 0 & 0 & 2 & 2 & 24 & 28 \\ 0 & -4 & 1 & 1 & 0 & -2 \\ 1 & 2 & 0 & 0 & 7 & 10 \end{array}\right) \quad -3/2$$

Note: Here I have chosen to eliminate the first column using the last equation because the 0's in the last row make the arithmetic easier.

$$\rightarrow \left(\begin{array}{cccc|cc} 0 & 0 & 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 1 & 12 & 14 \\ 0 & 4 & -1 & -1 & 0 & 2 \\ 1 & 2 & 0 & 0 & 7 & 10 \end{array}\right) \quad +1 \rightarrow \left(\begin{array}{cccc|cc} 0 & 0 & 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 1 & 12 & 14 \\ 0 & 1 & 0 & 0 & 3 & 4 \\ 1 & 2 & 0 & 0 & 7 & 10 \end{array}\right) \quad \begin{matrix} -1 \\ -2 \end{matrix}$$

$$\rightarrow \left(\begin{array}{cccc|cc} 0 & 0 & 0 & 1 & 7 & 8 \\ 0 & 0 & 1 & 0 & 5 & 6 \\ 0 & 1 & 0 & 0 & 3 & 4 \\ 1 & 0 & 0 & 0 & 1 & 2 \end{array}\right)$$

Rearranging the rows, we have thus transformed the original equations into the equivalent equations

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}$$

$$\Rightarrow B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}$$

It is easy to check this answer, eg., in Matlab:

```
>> A = [1 2 3 4;
        2 4 2 2;
        3 2 1 1;
        1 2 0 0];
>> B = [1 2;
        3 4;
        5 6;
        7 8];
>> A * B

ans =

    50    60
    38    48
    21    28
     7    10
```

✓

⑦ By hand, compute the LU decomposition of

$$A = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & -2 & 9 \end{pmatrix}$$

In general, it is easy to see that the L, U factors of a tridiagonal matrix are themselves tridiagonal. [See problem 3.10.6.] This makes solving $Ax=b$ using the LU decomposition very fast. On the other hand, A^{-1} can be dense — as it is in this example — so naively multiplying out $A^{-1}b$ takes quadratic time, much slower than using the LU decomposition!

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & -2 & 5 & 0 \\ 0 & 0 & 22 & 8 \\ 0 & 0 & 0 & 17 \end{pmatrix}$$

3.10.6. Consider the tridiagonal matrix $\mathbf{T} = \begin{pmatrix} \beta_1 & \gamma_1 & 0 & 0 \\ \alpha_1 & \beta_2 & \gamma_2 & 0 \\ 0 & \alpha_2 & \beta_3 & \gamma_3 \\ 0 & 0 & \alpha_3 & \beta_4 \end{pmatrix}$.

(a) Assuming that $\mathbf{T}$ possesses an LU factorization, verify that it is given by

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \alpha_1/\pi_1 & 1 & 0 & 0 \\ 0 & \alpha_2/\pi_2 & 1 & 0 \\ 0 & 0 & \alpha_3/\pi_3 & 1 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} \pi_1 & \gamma_1 & 0 & 0 \\ 0 & \pi_2 & \gamma_2 & 0 \\ 0 & 0 & \pi_3 & \gamma_3 \\ 0 & 0 & 0 & \pi_4 \end{pmatrix},$$

where the π_i's are generated by the recursion formula

$$\pi_1 = \beta_1 \quad \text{and} \quad \pi_{i+1} = \beta_{i+1} - \frac{\alpha_i \gamma_i}{\pi_i}.$$

Note: This holds for tridiagonal matrices of arbitrary size thereby making the LU factors of these matrices very easy to compute.

(b) Apply the recursion formula given above to obtain the LU factorization of

$$\mathbf{T} = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

⑧ Let

$$A = \begin{pmatrix} 1 & 4 & 5 \\ 4 & 18 & 26 \\ 3 & 16 & 30 \end{pmatrix}.$$

a) Determine the LU factors of A.

(Do this, and the following parts, by hand, not computer.)

b) Use the LU factors to solve $A\vec{x} = \vec{b}$ and $A\vec{y} = \vec{c}$, where

$$\vec{b} = \begin{pmatrix} 6 \\ 0 \\ -6 \end{pmatrix}, \quad \vec{c} = \begin{pmatrix} 6 \\ 6 \\ 12 \end{pmatrix}.$$

c) Use the LU factors to determine A^{-1}.

a) $P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$, $L = \begin{pmatrix} 1 & 0 & 0 \\ 3/4 & 1 & 0 \\ 1/4 & -1/5 & 1 \end{pmatrix}$, $U = \begin{pmatrix} 4 & 18 & 26 \\ 0 & 5/2 & 21/2 \\ 0 & 0 & 3/5 \end{pmatrix}$

b) $\vec{x} = \begin{pmatrix} 110 \\ -36 \\ 8 \end{pmatrix}$, $\vec{y} = \begin{pmatrix} 112 \\ -39 \\ 10 \end{pmatrix}$

⑨ A blurry camera:

Let's pretend you have a 64×64 pixel grayscale camera. As if that isn't bad enough, the camera's sensor is blurry; some of the light from each pixel spreads into the neighboring pixels.

More precisely, let $I_{x,y}$ be the amount of light that hits pixel (x,y). Here x and y are both from 1 to 64.

The camera records

$$C_{x,y} = \frac{1}{2} \times I_{x,y} + \frac{1}{8} \times \begin{pmatrix} I_{x-1,y} + I_{x+1,y} \\ + I_{x,y-1} + I_{x,y+1} \end{pmatrix}.$$

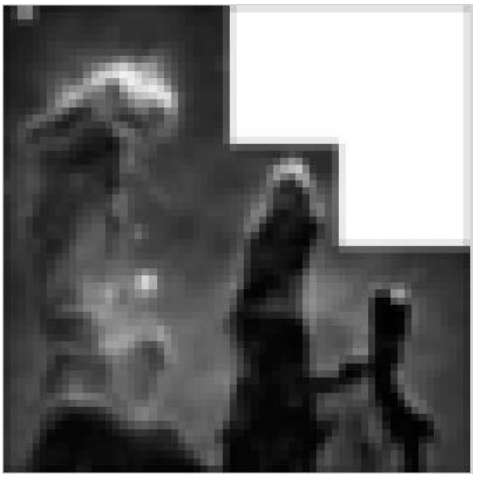
That is, it gets 1/2 times what you want, but also 1/8 times the light from the neighboring pixels.

Note: If (x,y) is on the boundary, then there will be fewer than four neighboring pixels. For example,

$$C_{1,1} = 1/2 \times I_{1,1} + 1/8 \times (I_{1,2} + I_{2,1})$$

because the corner pixel $(1,1)$ has only two neighbors.

Here's a picture that your camera took:



By setting up and solving 64^2 equations in the 64^2 variables $I_{1,1}, \ldots, I_{64,64}$, recover the correct image I.

Technical notes:

The blurred image file is available as "blurryimage.mat". To load it, run the commands

```
load('blurryimage.mat');
imshow(blurryimage);
```

← this should display the blurry image

To turn it into a vector of length n^2, you can use

```
b = reshape(blurryimage, n^2, 1);
```

where $n = 64$.

Then run the following code:

```
A = sparse(n^2, n^2);
%% fill in here commands to set up the sparse matrix A,
   each row containing an equation for one of the pixels
x = A \ b;
recoveredimage = reshape(x, n, n);
imshow(recoveredimage);
save('recoveredimage.mat', 'recoveredimage');
```

As in class, the following function for converting a pair of indices (x, y) to one index from 1 to n^2, should be helpful:

```
% returns an integer from 1 to n^2; x, y should be
  from 1 to n
function j = xytoj(x, y, n)
  j = 1 + (x-1) + n * (y-1);
end
```

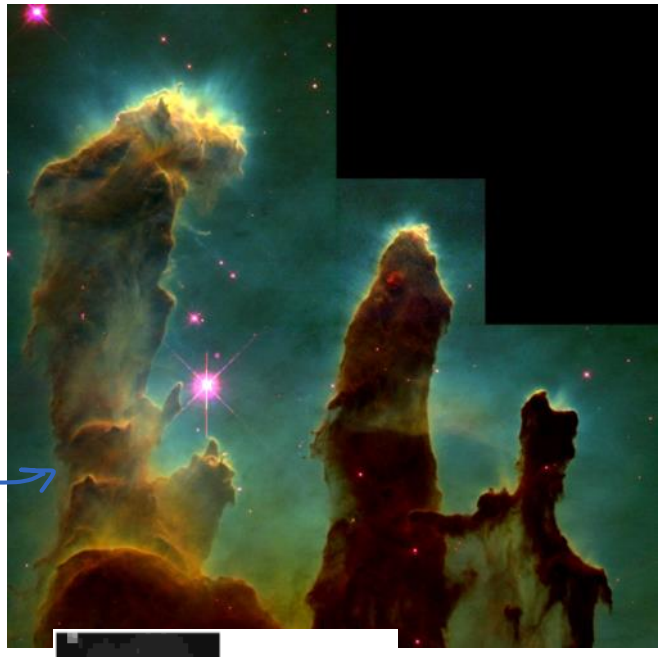
Deliverable: Print out your code and the image.

Python

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse
import scipy.sparse.linalg
import scipy.io

mat = scipy.io.loadmat('blurryimage.mat')
blurryimage = mat['blurryimage']

plt.imshow(blurryimage, cmap='gray')
plt.show()

n = blurryimage.shape[0]
b = blurryimage.flatten()

A = scipy.sparse.lil_matrix((n**2, n**2))
# fill in here commands to set up A
A = A.tocsr()

x = scipy.sparse.linalg.bicg(A, b)[0]
recoveredimage = np.reshape(x, (n,n))
plt.imshow(recoveredimage, cmap='gray')
plt.show()
```

Answer: One possible solution:

```
load('blurryimage.mat');
n = size(blurryimage, 1);
imshow(blurryimage);
b = reshape(blurryimage, n^2, 1);

A = sparse(n^2, n^2);
j = 1;
for y = 1:n
  for x = 1:n
    % the coeff. on the diagonal is 1/2
    A(j, j) = 1/2;
    % set coeff. of 1/8 looking left/right/up/down
    %  if the position is not against that boundary
    if (x > 1)  A(j, j-1) = 1/8;  end
    if (x < n)  A(j, j+1) = 1/8;  end
    if (y > 1)  A(j, j-n) = 1/8;  end
    if (y < n)  A(j, j+n) = 1/8;  end
    j = j + 1;
  end
end

x = A \ b;
recoveredimage = reshape(x, n, n);
imshow(recoveredimage);
save('recoveredimage.mat', 'recoveredimage');
```

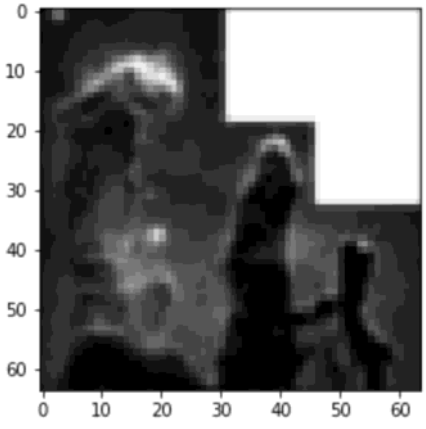


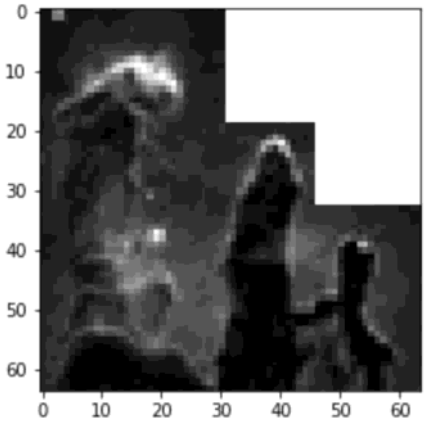
It is the Eagle Nebula!

Okay, here's what I really got:

Python

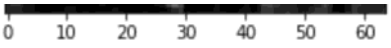
```
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse
import scipy.sparse.linalg
import scipy.io

mat = scipy.io.loadmat('blurryimage.mat')
blurryimage = mat['blurryimage']

plt.imshow(blurryimage, cmap='gray')
plt.show()

n = blurryimage.shape[0]
b = blurryimage.flatten()

A = scipy.sparse.lil_matrix((n**2, n**2))
j = 0
for y in range(n):
  for x in range(n):
    A[j,j] = 1/2
    if (x > 0):   A[j,j-1] = 1/8
    if (x < n-1): A[j,j+1] = 1/8
    if (y > 0):   A[j,j-n] = 1/8
    if (y < n-1): A[j,j+n] = 1/8
    j += 1
A = A.tocsr()

x = scipy.sparse.linalg.bicg(A, b)[0]
recoveredimage = np.reshape(x, (n,n))
plt.imshow(recoveredimage, cmap='gray')
plt.show()
```






```
import matplotlib.pyplot as plt
```

```
plt.show()
```



Alternative (easier?) approach for A:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse
import scipy.sparse.linalg
import scipy.io

mat = scipy.io.loadmat('blurryimage.mat')
blurryimage = mat['blurryimage']

plt.imshow(blurryimage, cmap='gray')
plt.show()

n = blurryimage.shape[0]
b = blurryimage.flatten()

A = scipy.sparse.diags([1/2, 1/8, 1/8, 1/8, 1/8],
                       [0, -1, 1, -n, n],
                       shape=(n**2,n**2))
# now fix the boundaries
A = A.tocsr()
A[n::n,n-1::n] = 0 # left boundary
A[n-1::n,n::n] = 0 # right boundary

x = scipy.sparse.linalg.bicg(A, b)[0]
recoveredimage = np.reshape(x, (n,n))
plt.imshow(recoveredimage, cmap='gray')
plt.show()
```

```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5), dpi=150)
plt.spy(A[:3*n,:3*n], markersize=.5)
plt.show()
```