Homework 2

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!

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

$$p(1)=4 \qquad p'(-1)=0$$
$$p(2)=2 \qquad p'(2)=6$$

- 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).

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

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

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) $\begin{aligned} 2w+x+3y+5z&=1,\\ 4w+4y+8z&=0,\\ w+x+2y+3z&=0\end{aligned}$ (f) $\begin{aligned} 2w+x+3y+5z&=7,\\ 4w+4y+8z&=8,\\ w+x+2y+3z&=5\end{aligned}$

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

(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 $p''(2) = 0$.

(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.

(7) By hand, compute the LU decomposition of
$$A = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & -22 & 9 \end{pmatrix}$$
(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!)

(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}.

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

$$C_{x,y} = \frac{1}{2} \times I_{x,y} + \frac{1}{8} \times \left(I_{x-1,y} + I_{x+1,y} + I_{x,y-1} + I_{x,y+1} \right).$$

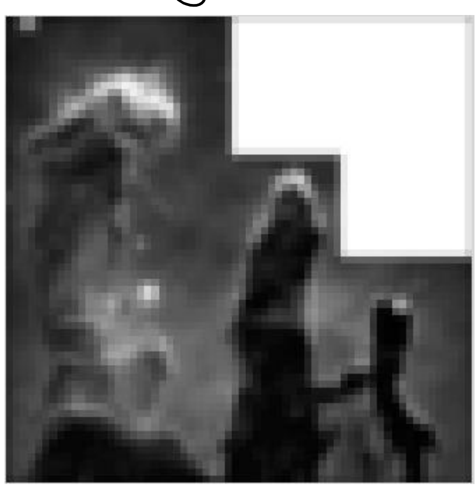
That is, it gets 1/2 times what you want, but also 1/8 times the light from the neighboring pixels.

(**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 $I_{1,1}, \dots, I_{64,64}$, 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~~ the blurry image 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');
```

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()
```

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.