10.34: Numerical Methods Applied to Chemical Engineering
Prof. K. Beers
Solutions to Problem Set 4: Matrix Eigenvalue Analysis

Mark Styczynski, Ben Wang

1.(1 point) ***3.A.1*** *From Gershorgin's theorem, derive lower and upper bounds on the possible eigenvalues of the matrix* $A = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 1 \\ 3 & 1 & -1 \end{bmatrix}$

The first thing we note is that *A* is symmetric; that is, $A^T = A$. That means that *A* is also Hermetian, because $A^H = A^T$ if all values in A are real. We know that a Hermetian matrix has all real eigenvalues.

From Gershorgin's theorem, we know that the eigenvalues will be located somewhere "near" the main diagonal elements:

$$\left|\lambda - a_{kk}\right| \le \sum_{\substack{j=1, \\ j \ne k}}^{N} \left|a_{kj}\right|$$

If complex eigenvalues were possible, this would denote a circle around the diagonal elements. Since we know that the eigenvalues are purely real, that means that they can only be in intervals surrounding each diagonal element, where the half-width of the interval around some diagonal element is equal to the sum of off-diagonal elements in that row. So we look at the three rows in turn to define:

$$\left|\lambda - a_{11}\right| = \left|\lambda - 1\right| \le \left|0\right| + \left|3\right|$$

$$\left|\lambda - a_{22}\right| = \left|\lambda - 2\right| \le \left|0\right| + \left|1\right|$$

$$\left|\lambda - a_{33}\right| = \left|\lambda - (-1)\right| \le \left|3\right| + \left|1\right|$$

So that means that the eigenvalues lie in the bounds

$$[-2, 4], [1, 3], [-5, 3]$$

So, you can say that all eigenvalues must fall between the lower bound of -2 and the upper bound of 4, the lower bound of 1 and the upper bound of 3, or the lower bound of -5 and the upper bound of 3. However, that statement seems somewhat odd due to the significant overlap of the intervals.

We can instead note that the union of these intervals is actually one interval, defined as

$$[-5, 4]$$

So, all eigenvalues for *A* must fall in the above interval, making -5 the lower bound and 4 the upper bound.

Grading:
(-0.5 point): no discussion of the matrix being symmetric
(-0.25 point): missing total range of eigenvalues
(-0.25 point): mention of why the eigenvalues must be real

2. (3 points) ***3.A.2*** *Compute by hand the eigenvalues and eigenvectors of (3.269), and check your results using Matlab.*

The eigenvalues are defined as the solutions of
$A\underline{w}^{[k]} = \lambda_k \underline{w}^{[k]}$,
which can be restated as
$\left(A-\lambda_k I\right)\underline{w}^{[k]} = 0$ or $\det\left(A-\lambda_k I\right) = 0$
So we set up the matrix $\left(A-\lambda_k I\right)$, which is

$$\begin{bmatrix} 1-\lambda & 0 & 3 \\ 0 & 2-\lambda & 1 \\ 3 & 1 & -1-\lambda \end{bmatrix}$$

We take this determinant and find that

$$\begin{vmatrix} 1-\lambda & 0 & 3 \\ 0 & 2-\lambda & 1 \\ 3 & 1 & -1-\lambda \end{vmatrix} = \left[(1-\lambda)(2-\lambda)(-1-\lambda)\right] - \left[(1-\lambda)(1)(1)+(3)(2-\lambda)(3)\right] = 0$$

(I have already canceled zero terms for simplicity.) This reduces to
$\left[-\lambda^3+2\lambda^2+\lambda-2\right]-\left[1-\lambda+18-9\lambda\right] =$
$-\lambda^3+2\lambda^2+\lambda-2-1+\lambda-18+9\lambda =$
$-\lambda^3+2\lambda^2+11\lambda-21=0$
We note that there is no immediately obvious solution, and trying the integer values in our interval gives us nothing, so we go back to the good old TI-85 or Matlab or Mathematica or whatever else you prefer, and we find that the roots are
$\lambda_1 = 3.4211, \lambda_2 = -3.2880, \lambda_3 = 1.8669$
We can then use these eigenvalues to find the eigenvectors. We go back to one of our initial equations, that
$\left(A-\lambda_k I\right)\underline{w}^{[k]} = 0$
So we plug in each eigenvalue λ_k and solve for the eigenvector $\underline{w}^{[k]}$. We'll set up our augmented matrix as such:

$$\begin{bmatrix} 1-\lambda_k & 0 & 3 & 0 \\ 0 & 2-\lambda_k & 1 & 0 \\ 3 & 1 & -1-\lambda_k & 0 \end{bmatrix}$$

Since we're solving for the null space, and we know there are no repeated eigenvalues, we can just say that two of these equations are linearly independent. You can solve for this null space any way you like; I'll do the way I know.
$\left(1-\lambda_k\right)w_1^{[k]} + 3w_3 = 0$
$\left(2-\lambda_k\right)w_2^{[k]} + w_3 = 0$
I move the "free variable" over to the right side of the equation and define it as a constant, *s*. Note that you could just define it as 1… but if you were finding a null space

with dimension two, it would not be a good idea to define two variables as two numerical constants… you would lose track of which entry contained which "constant" and your eigenvectors would be hosed. So, I'm sticking with *s*.

$$\left(1-\lambda_k\right) w_1^{[k]} = -3s$$

$$\left(2-\lambda_k\right) w_2^{[k]} = -s$$

And then

$$w_1^{[k]} = \frac{-3s}{\left(1-\lambda_k\right)}$$

$$w_2^{[k]} = \frac{-s}{\left(2-\lambda_k\right)}$$

So we can say that

$$\underline{w}^{[k]} = \begin{bmatrix} w_1^{[k]} \\ w_2^{[k]} \\ w_3^{[k]} \end{bmatrix} = \begin{bmatrix} \frac{-3s}{1-\lambda_k} \\ \frac{-s}{2-\lambda_k} \\ s \end{bmatrix} = s \begin{bmatrix} \frac{-3}{1-\lambda_k} \\ \frac{-1}{2-\lambda_k} \\ 1 \end{bmatrix}$$

This means that for each valid eigenvalue, we will get a distinct eigenvector. Plugging in our valid eigenvalues, we get

$$\underline{w}^{[1]} = \begin{bmatrix} 1.2391 \\ 0.7037 \\ 1 \end{bmatrix}, \underline{w}^{[2]} = \begin{bmatrix} -0.6996 \\ -0.1891 \\ 1 \end{bmatrix}, \underline{w}^{[3]} = \begin{bmatrix} 3.4607 \\ -7.5131 \\ 1 \end{bmatrix}$$

Wonderful. These can also be normalized by dividing by their magnitudes to yield

$$\underline{w}^{[1]} = \begin{bmatrix} 0.7118 \\ 0.4042 \\ 0.5744 \end{bmatrix}, \underline{w}^{[2]} = \begin{bmatrix} -0.5665 \\ -0.1531 \\ 0.8097 \end{bmatrix}, \underline{w}^{[3]} = \begin{bmatrix} 0.4153 \\ -0.9017 \\ 0.1200 \end{bmatrix}$$

Normalizing is particularly useful because Matlab will return its eigenvectors in unit magnitude. Note that multiplying any of the eigenvectors by a constant still gives an eigenvector, meaning that our normalization is OK and it's OK if an entire vector is off by a factor of -1. Matlab tells us:

eigenvectors =

```
-0.5665   -0.4153   -0.7118
-0.1531    0.9018   -0.4042
 0.8097   -0.1200   -0.5744
```

eigenvalues =

```
-3.2880         0         0
     0    1.8669         0
     0         0    3.4211
```

Great.

Grading:
(up to -1 point): Clear writeup linking equations and concepts
(-1 point): No eigenvectors calculated by hand
(-1 point): No eigenvectors calculated by Matlab
(-0.5 point): No eigenvalues calculated by hand
(-0.5 point): No eigenvalues calculated by Matlab
(-1 point): Didn't compare to or properly reconcile hand-calculated results with those of Matlab

3. (3 points) ***3.A.3*** *Consider the following matrices,*

$$A=\begin{bmatrix} 0 & -1 & -2 & 1 \\ -1 & 2 & 0 & 4 \\ -2 & 0 & 3 & 0 \\ 1 & 4 & 0 & -1 \end{bmatrix}, B=\begin{bmatrix} 6 & 2 & 1 \\ 0 & 5 & -1 \\ -1 & 3 & 2 \end{bmatrix}, C=\begin{bmatrix} 3 & 2 \\ 1 & -1 \end{bmatrix}, D=\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) Without computing the actual eigenvalues, can you tell if any of the matrices above must have all real eigenvalues? Explain why you make this judgment.

All of the matrices are real, which is convenient. That means that if a matrix is symmetric, it is guaranteed to have all real eigenvalues; otherwise, we can make no guarantees. Clearly, only matrix *A* is symmetric, so matrix *A* has all real eigenvalues.

(b) For each of those guaranteed to have all real eigenvalues, provide an upper and lower bounds on the eigenvalues.

We can use Gershorgin's theorem to determine an upper and lower bounds on the eigenvalues for *A*. Using the same reasoning as in the first problem, namely that

$$\left|\lambda - a_{kk}\right| \le \sum_{\substack{j=1, \\ j\neq k}}^{N} \left|a_{kj}\right|$$

we can say that the bounds for the eigenvalues are:

$$[-4,4],[-3,7],[1,5],[-6,4]$$

Since these bounds have significant overlap, we can condense them to say that all eigenvalues will fall in the range

$$[-6,7]$$

(c) Show that D is unitary.

A unitary matrix meets the requirement that $D^H = D^{-1}$. Since this is a strictly real matrix, this is the same as saying $D^T = D^{-1}$. So, when we multiply out DD^T, it should be the same as DD^{-1}, which should yield the identity matrix. Actually doing the multiplication, we see that

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So D is indeed unitary.

(d) Compute by hand the eigenvalues and unit-length eigenvectors of C.

OK, we use the equations from the second problem again, namely

$$\det\left(A - \lambda_k I\right) = 0$$

So, that means

$$\begin{vmatrix} 3-\lambda & 2 \\ 1 & -1-\lambda \end{vmatrix} = (3-\lambda)(-1-\lambda) - 2 = \lambda^2 - 2\lambda - 5$$

The solution for this characteristic equation can be found using the quadratic formula,

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{2 \pm \sqrt{(-2)^2 - 4(1)(-5)}}{2}$$

$$\lambda_1 = 3.4495, \lambda_2 = -1.4495$$

Again, we know that there will be one independent equation for each eigenvalue. So we say

$$(3-\lambda_k)\, w_1^{[k]} + 2w_2^{[k]} = 0$$

$$(3-\lambda_k)\, w_1^{[k]} = -2w_2^{[k]}$$

$$(3-\lambda_k)\, w_1^{[k]} = -2s$$

$$w_1^{[k]} = \frac{-2s}{(3-\lambda_k)}$$

$$\underline{w}^{[k]} = \begin{bmatrix} \dfrac{-2s}{3-\lambda_k} \\ s \end{bmatrix} = s\begin{bmatrix} \dfrac{-2}{3-\lambda_k} \\ 1 \end{bmatrix}$$

$$\underline{w}^{[1]} = \begin{bmatrix} 4.4494 \\ 1 \end{bmatrix}, \underline{w}^{[2]} = \begin{bmatrix} -0.4495 \\ 1 \end{bmatrix}$$

$$\underline{w}^{[1]} = \begin{bmatrix} 0.9757 \\ 0.2193 \end{bmatrix}, \underline{w}^{[2]} = \begin{bmatrix} -0.4100 \\ 0.9121 \end{bmatrix}$$

And Matlab agrees with these latter normalized eigenvectors.

Grading:
Part a: 0.5 point

Part b: 0.5 point
Part c: 0.5 point
Part d: 0.5 point for eigenvalues, 0.5 point for eigenvectors, 0.5 point for normalization

4.(3 points) ***3.B.2*** *Consider the positive-definite matrix A, obtained by discretizing the Poisson equation* $-\nabla^2\varphi = f$ *in d dimensions on a hypercube grid of* N^d *points, with the following non-zero elements in each row for* $\Delta x_j = 1$,

$$A_{kk} = 2d \qquad A_{k,k\pm N^m} = -1 \qquad m = 0,1,\ldots,d-1$$

Plot as functions of N the largest and smallest eigenvalues and the condition number for d = 1,2,3. For d = 3, extend the calculation to relatively large values of N by not storing the matrix (even in sparse format) but rather by merely supplying a routine that returns $A\underline{v}$ *given an input value of* $\underline{v}$.

The actual derivation of the matrix is not asked for; I'll only give a brief discussion of it here. We have seen the one-dimensional finite difference method, and hopefully it makes sense to you. The same thing can be done for multiple dimensions by making a grid of higher dimensions. In this case, the value of each grid point is dependent not only upon its neighbors in the *x* direction, but also on the neighbors in the *y* direction and/or the *z* direction (depending on the dimensionality of the approximation). So you'll have N^d grid points in total… *N* in each direction. This means you have N^d unknowns, but also N^d equations. That means that you can form an N^d x N^d matrix to represent all of our equations and unknowns, no matter what the value of *d* is.

Now, once you have all of these points that are being represented in one matrix, you need a way to keep track of where they came from… for instance, what grid point does the 105th column in the finite difference matrix represent? You can do this by keeping track of the grid points in an orderly fashion. This is where the equations that the problem gives come in. There is one way of representing these grid points where the relevant off-diagonal elements are defined by $k \pm N^m$ for row *k*.

In addition, the Poisson equation as given to you is equivalent to (assuming Cartesian coordinates):

$$\sum_{all\,\dim} \frac{\partial^2\varphi}{\partial \dim^2} = f$$

Where the values of dim may be *x*, *y*, *z*, etc.

But you don't need to know that to solve this problem. All you need to do is get that matrix into Matlab in sparse form and calculate the eigenvectors, eigenvalues, and condition numbers for varying *N* and *d*. For *d* = 1,2, this is extremely simple and is shown below. For *d* = 3, the problem asks you to not store the matrix, but rather to be able to calculate $A\underline{v}$ given some $\underline{v}$. This is useful because Matlab (via eigs()) can use this function to iteratively determine the eigenvalues you are looking for via the power method described in class and in the text starting on page 181. That function is included after the main program.

The only real equation of relevance for this problem is the condition number:

$$\kappa = \frac{|\lambda_{max}|}{|\lambda_{min}|}$$

Given that, the program is relatively straight-forward. Note the advice that was given in an email to the class: try to use the sigma value of 'BE' when calling eigs(), and make sure you accept both the eigenvectors and eigenvalues for that call, otherwise you will get incorrect answers.

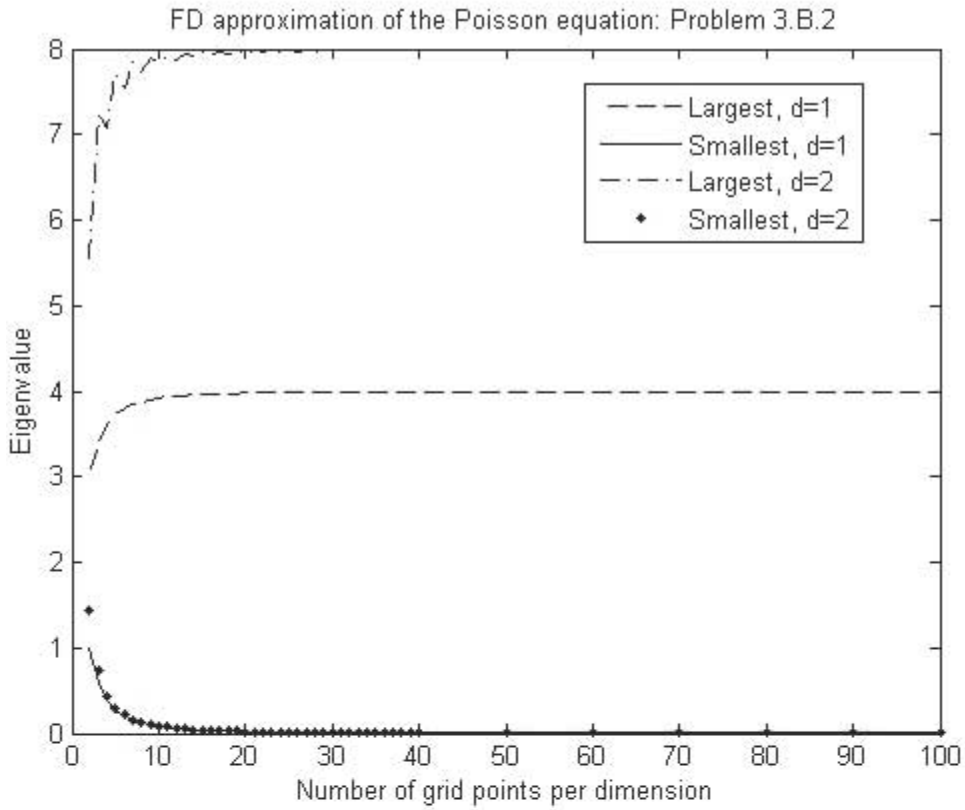
```
% Mark Styczynski
% 10.34
% Homework 4
% Problem 4, P 3.B.2

clear all; close all;

% First we set up some options that we'll pass to eigs so that it
%  knows just how simple of a matrix we are giving it.  This will
%  significantly help convergence.
opts.disp = 0;
opts.issym = 1;

% PDL> Set up values that N can take.
Nvec = [linspace(2,40,39) 50 60 70 80 90 100];

% PDL> Solve for d = 1, storing the matrix and then calculating
%  the eigenvalues using eigs()
for i=1:length(Nvec),
    N = Nvec(i)
    delPhi = spalloc(N,N,3*N);
    for k=1:N,
        if (k>1)
            delPhi(k,k-1) = -1;
        end
        delPhi(k,k) = 2;
        if (k<N)
            delPhi(k,k+1) = -1;
        end
    end

    % PDL> Store the data that we get at each stop for plotting later.
    [useless, eigMat] = eigs(delPhi,2,'BE',opts);
    dim1Small(i) = eigMat(1,1);
    dim1Large(i) = eigMat(2,2);
    dim1CondNum(i) = dim1Large(i)/dim1Small(i);
    % We keep the even ones because we like the prettier picture
    %  we get when we plot them.
    if mod(i,2) == 0
        dim1SmallEven(i/2) = eigMat(1,1);
        dim1LargeEven(i/2) = eigMat(2,2);
        dim1CondNumEven(i/2) = eigMat(2,2)/eigMat(1,1);
        NvecEven(i/2) = N;
    end
```

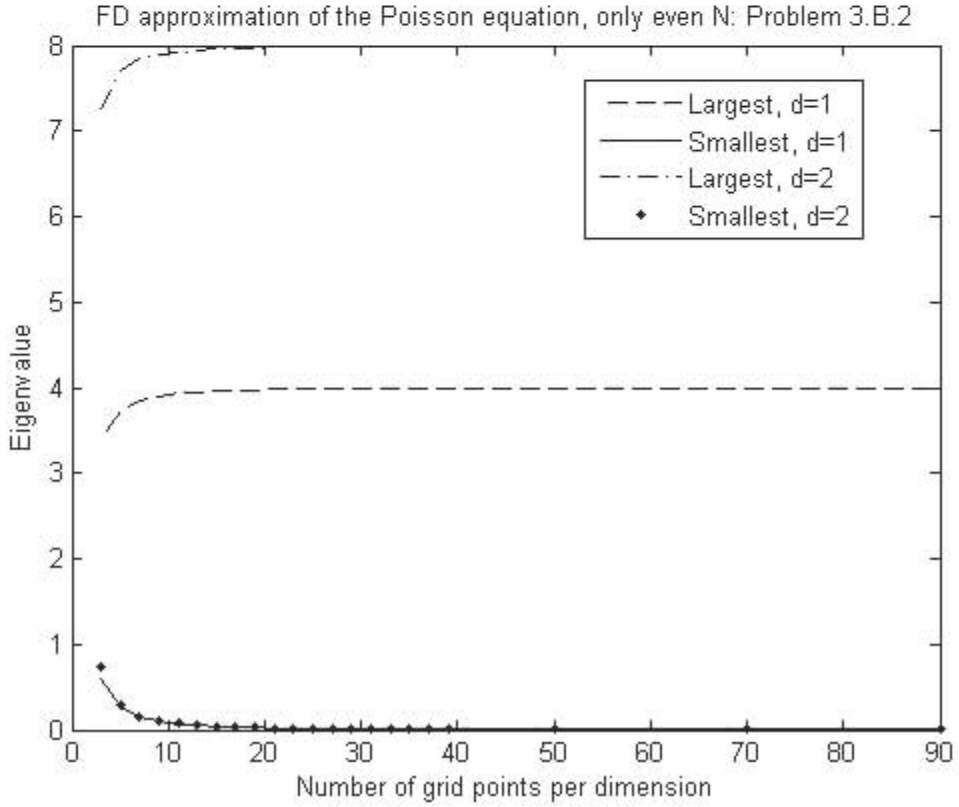
```
end

% PDL> Solve for d = 2, storing the matrix and then calculating
%  the eigenvalues using eigs()

for i=1:length(Nvec),
    N = Nvec(i)
    delPhi2D = spalloc(N^2,N^2,5*N^2);
    for k=1:N^2,
        if (k>N)
            delPhi2D(k,k-N) = -1;
        end
        if (k>1)
            delPhi2D(k,k-1) = -1;
        end
        delPhi2D(k,k) = 4;
        if (k<N^2)
            delPhi2D(k,k+1) = -1;
        end
        if (k < N^2 - N + 1)
            delPhi2D(k,k+N) = -1;
        end
    end
 % PDL> Store the data that we get at each stop for plotting later.
    [useless, eigMat] = eigs(delPhi2D,2,'BE',opts);
    dim2Large(i) = eigMat(2,2);
    dim2Small(i) = eigMat(1,1);
    dim2CondNum(i) = dim2Large(i)/dim2Small(i);
    if mod(i,2) == 0
        dim2SmallEven(i/2) = eigMat(1,1);
        dim2LargeEven(i/2) = eigMat(2,2);
        dim2CondNumEven(i/2) = eigMat(2,2)/eigMat(1,1);
    end
end

%PDL> Solve for d = 3, but this time not storing the matrix.  We'll
%  use a separate function to calculate Av given v.
Nvec2 = linspace(2,20,10);
for i=1:length(Nvec2),
    N = Nvec2(i)
    [useless, eigMat]  = eigs('marksty_calc_Av_3D',N^3,2,'BE',opts);
    dim3Small(i) = eigMat(1,1);
    dim3Large(i) = eigMat(2,2);
    dim3CondNum(i) = dim3Large(i)/dim3Small(i);
end


% PDL> Plot and label graphs.
figure(1)
plot(Nvec,dim1Large,'--')
hold on
plot(Nvec,dim1Small)
plot(Nvec,dim2Large,'-.')
plot(Nvec,dim2Small,'.')
xlabel('Number of grid points per dimension')
ylabel('Eigenvalue')
```

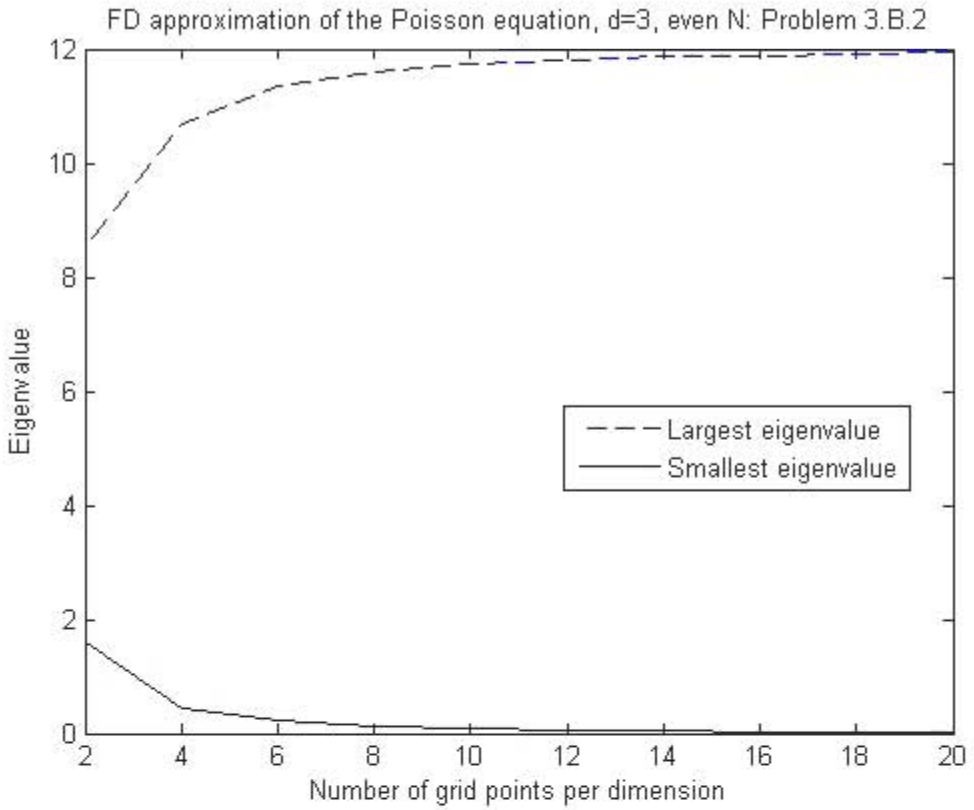
```
legend('Largest, d=1','Smallest, d=1','Largest, d=2','Smallest,
d=2',...
    'Location','Best');
title('FD approximation of the Poisson equation: Problem 3.B.2')
hold off

figure(2)
plot(NvecEven,dim1LargeEven,'--')
hold on
plot(NvecEven,dim1SmallEven)
plot(NvecEven,dim2LargeEven,'-.')
plot(NvecEven,dim2SmallEven,'.')
xlabel('Number of grid points per dimension')
ylabel('Eigenvalue')
legend('Largest, d=1','Smallest, d=1','Largest, d=2','Smallest,
d=2',...
    'Location','Best');
title('FD approximation of the Poisson equation, only even N: Problem
3.B.2')
hold off

figure(3)
plot(Nvec2,dim3Large,'--')
hold on
plot(Nvec2,dim3Small)
xlabel('Number of grid points per dimension')
ylabel('Eigenvalue')
legend('Largest eigenvalue','Smallest eigenvalue','Location','Best');
title('FD approximation of the Poisson equation, d=3, even N: Problem
3.B.2')
hold off

figure(4)
plot(Nvec,dim1CondNum)
hold on
plot(Nvec,dim2CondNum,'--')
xlabel('Number of grid points per dimension')
ylabel('Condition number')
legend('d = 1','d = 2','Location','Best')
title('Condition numbers for a large range of N for the Poisson
equation: Problem 3.B.2')
hold off

figure(5)
plot(Nvec(1:19),dim1CondNum(1:19))
hold on
plot(Nvec(1:19),dim2CondNum(1:19),'--')
plot(Nvec2,dim3CondNum,'-.')
xlabel('Number of grid points per dimension')
ylabel('Condition number')
legend('d = 1','d = 2','d = 3','Location','Best')
title('Condition numbers for small N for the Poisson equation: Problem
3.B.2')
hold off

function Av = marksty_calc_Av_3D(v)
```

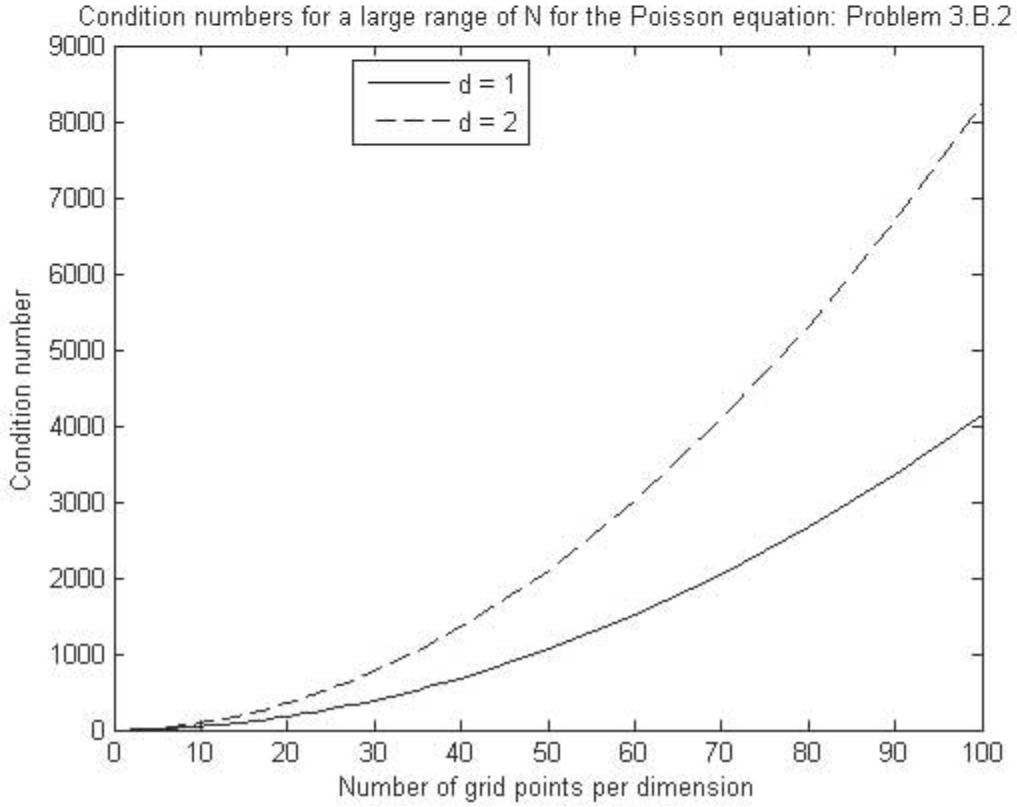
```
% Input: any one-dimensional vector v
% Output: a vector Av the same size as v

% PDL> Calculate values of N, N^2, and N^3.
vecSize = length(v);
N = round(vecSize^(1/3));
Nsq = round(N^2);
Ncu = vecSize;

% PDL> Figure out all of the special cases... subtract values
%  as appropriate.

% We note that we would like to just implement the equation as shown
%  in the text, but in some cases that would entail using values
%  outside of the vector bounds.  (For example, if we're on row
%  1, then we can't use the value from 1-N.)  So, we just make
%  sure we're doing something legitimate every step along the way.
for k=1:vecSize
    Av(k) = 0;
    if k > Nsq
        Av(k) = Av(k) - v(k - Nsq);
    end
    if k > N
        Av(k) = Av(k) - v(k - N);
    end
    if k > 1
        Av(k) = Av(k) - v(k - 1);
    end
    Av(k) = Av(k) + 6*v(k);
    if k < Ncu
        Av(k) = Av(k) - v(k + 1);
    end
    if (k < Ncu - N + 1)
        Av(k) = Av(k) - v(k + N);
    end
    if (k < Ncu - Nsq + 1)
        Av(k) = Av(k) - v(k + Nsq);
    end
end
```

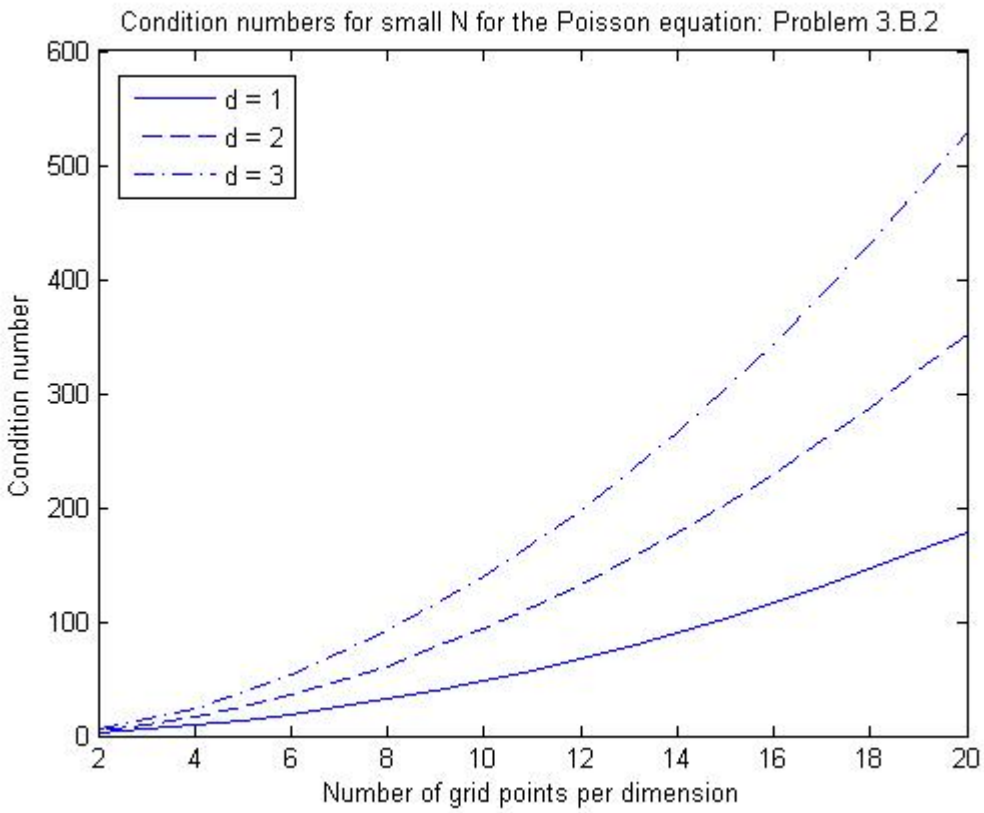
So here are the graphs. We note that for higher dimensions, the highest eigenvalue displays a “sawtooth”-type pattern when both odd and even values are used for low values of *N*. This is somewhat intuitive, as the switch between odd and even number of grid points in such a coarse approximation may significantly change the character of the solution. (They are representing significantly different points.) On the other hand, steadily increasing in only even values or only odd values of *N* will only be a refinement of the system and won’t drastically change anything. Both graphs are correct; they are both included for completeness.


FD approximation of the Poisson equation: Problem 3.B.2
Largest, d=1
Smallest, d=1
Largest, d=2
Smallest, d=2
Eigenvalue
Number of grid points per dimension
0
1
2
3
4
5
6
7
8
0
10
20
30
40
50
60
70
80
90
100


FD approximation of the Poisson equation, only even N: Problem 3.B.2
Largest, d=1
Smallest, d=1
Largest, d=2
Smallest, d=2
Eigenvalue
Number of grid points per dimension
0
1
2
3
4
5
6
7
8
0
10
20
30
40
50
60
70
80
90


FD approximation of the Poisson equation, d=3, even N: Problem 3.B.2
Eigenvalue
Number of grid points per dimension
Largest eigenvalue
Smallest eigenvalue
0
2
4
6
8
10
12
14
16
18
20


Condition numbers for a large range of N for the Poisson equation: Problem 3.B.2
Condition number
Number of grid points per dimension
d = 1
d = 2
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
0
10
20
30
40
50
60
70
80
90
100



Grading:
1 point each for the cases of d=1,2,3
(-0.25 point): Not calculating and plotting condition number
Credit was not given for d=3 if you stored the matrix A… you had to make a function that returned Av in order to get credit.