Math/CS 466/666: Shifted Inverse Power Method Lab

Let A be a $n \times n$ matrix. The shifted inverse power method is an iterative way to compute the eigenvalue of A closest to a given complex number. This method is a refinement of the power method which we used to find the matrix norm $\|A\|_2$. Recall that $\|A\|_2$ is equal to the square root of the largest eigenvalue of $B = A^T A$. In this case we proved

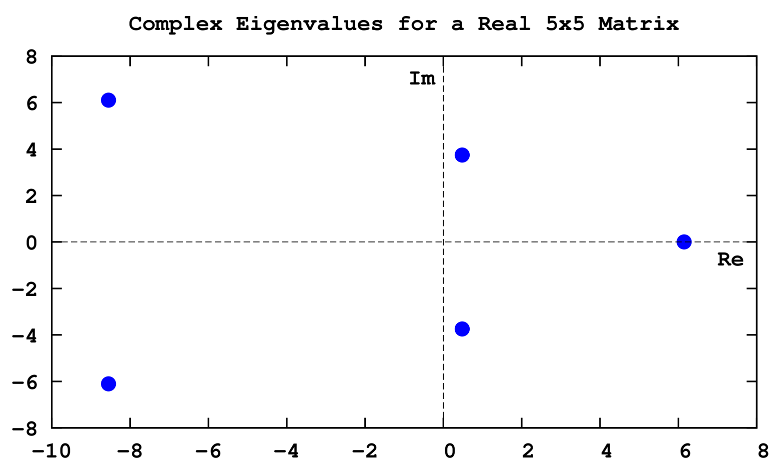
$$\max \left\{ \lambda : \lambda \text{ is the largest eigenvalue of } B \right\} = \lim_{k\to\infty} \|B^k x\| / \|B^{k-1} x\|$$

for almost every x. Note that the above computation is particularly simple because all the eigenvalues of B are real and non-negative. When working with a general $n \times n$ matrix, it may happen that the eigenvalues and corresponding eigenvectors are not real.

For example, all but one of the eigenvalues of the randomly generated matrix

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

come in complex conjugate pairs. Plotted in the complex plane these eigenvalues may be visualized as



In this computer lab you will compute some of the complex eigenvalues of the above matrix using the shifted inverse power method.

Theory

Let $\alpha \in \mathbf{C}$ be a fixed complex number closest to the eigenvalue λ that you wish to find. Given a randomly chosen vector y_0, define y_k by the recurrence

$$(A - \alpha I) y_k = y_{k-1}.$$

It follows that

$$\lambda = \alpha + \lim_{k\to\infty} \frac{\|y_{k-1}\|^2}{\overline{y_{k-1}} \cdot y_k}.$$

Any program for solving y_k will involve complex arithmetic since α is complex. In particular, the y_k are complex valued and $\overline{y_{k-1}}$ denotes the vector formed by taking complex conjugates of each of the entries in y_{k-1}. Moreover, to prevent overflow of the floating point registers it will be necessary to renormalize the vectors y_k to unit vectors after each iteration. As with the original power method $y_k/\|y_k\|$ may not converge as $k \to \infty$. This is because eigenvectors are not unique. However, the distance to the eigenspace corresponding to λ does tend to zero. Thus, at each step of the iteration we obtain a better approximation to some eigenvector of λ. Although A is real valued in the example given here, the method works equally well for any complex valued matrix.

Step 1

Let $B = A - \alpha I$. Since B is complex valued and we will be solving $By_k = y_{k-1}$ at each iteration, we need to modify the `plufact` and `plusolve` routines we wrote last month to work with complex valued matrices. In addition to changing all the variable types from `double` to `complex` you will also need to change the function call `fabs` to `cabs` in the pivoting code. Make a working directory called `invpower` and copy recent versions of `matrixlib.c` and `matrixlib.h` into that directory. Now create complex versions of `plufact` and `plusolve` called `cplufact` and `cplusolve` and add them to the `matrixlib` code library and header. Please check that the library still compiles. You may wish to create or copy a `Makefile` to do this. After you have verified that the new routines compile, please submit your current work using the command

```
$ submit -q1 invpower
```

Step 2

Create a subroutine `cdotprod` and `cvecnorm2` to compute the complex dot product and vector norms given by $\overline{x} \cdot y$ and $\|x\|_2$ by finishing the definitions

```
complex cdotprod(int n,complex x[n],complex y[n]){
    //  Put your code here.
}
double cvecnorm2(int n,complex x[n]){
    //  Put your code here.
}
```

After you have checked that your subroutines compile and work, submit your code directory using the submit command

```
$ submit -q2 invpower
```

Step 3

A good starting point for creating a shifted inverse iteration subroutine is the code for finding the matrix norm of A^{-1} from Programming Project 2. This code also appears in Part 2 Question 2 on the midterm and here with line numbers for reference:

```
1 double invmatnorm2(int n,double A[n][n]){
2     double B[n][n],*P[n],y[n],yk[n];
3     bzero(B,sizeof(double)*n*n);
```

```
4      for(int k=0;k<n;k++){                // $B = A^T A$
5          for(int i=0;i<n;i++){
6              for(int j=0;j<n;j++){
7                  B[i][j]+=A[k][i]*A[k][j];
8              }
9          }
10     }
11     PLUfact(n,B,P);
12     for(int i=0;i<n;i++){                // Choose $x \in \mathbf{R}^n$ randomly
13         y[i]=2.0*random()/RAND_MAX+1.0;  // and store $x$ in y for now
14     }
15     double q=0,qk;
16     for(int k=1;k<100*n;k++){
17         PLUsolve(n,B,P,yk,y);            // $y_k = B^{-k}x/\|B^{1-k}x\|_2$
18         qk=vecnorm2(n,yk);
19         for(int j=0;j<n;j++){
20             y[j]=yk[j]/qk;               // Overwrite y by $y_k/\|y_k\|_2$
21         }
22         if(fabs(qk-q)<5e-15*qk){         // Converge to 15 digits where
23             return sqrt(qk);             // $\|A\|_2 \approx (\|B^{-k}x\|_2/\|B^{1-k}x\|_2)^{1/2}$
24         }
25         q=qk;
26     }
27     fprintf(stderr,"invmatnorm2: Failed to converge!\n");
28     return sqrt(qk);
29 }
```

Copy this subroutine or an equivalent code into your working directory. After you have completed this step, submit your working directory using the submit command

```
$ submit -q3 invpower
```

Step 4

Modify the the `invmatnorm2` subroutine to perform the shifted inverse power method. You may do this anyway you like or write the code from scratch if you prefer. Here are some ideas about what should be changed: In line 1 change the name of the routine from `invmatnorm2` to `shiftinvpower` and change the return type to `complex`. Change the variable type of the vectors and matrices from `double` to `complex`. Add `complex alpha` to the list of arguments for the function. The code from lines 4 through 10 should be replaced by code which initializes B as $A - \alpha I$. Calls to `plufact` and `plusolve` in lines 11 and 17 should be replaced by calls to `cplufact` and `cplusolve`. Remove the square roots from lines 23 and 28 so both read as `return qk` instead. Replace `double` in line 15 by `complex`. Line 18 should be replaced by the calculation

```
qk=alpha+1.0/cdotprod(n,y,yk);
double yknorm=cvecnorm2(n,yk);
```

Finally, since `qk` no longer contains the norm of `yk`, then line 20 should be changed to

```
y[j]=yk[j]/yknorm;
```

It is possible I have missed some necessary changes. Please fix any errors or omissions that you find. We will test the resulting code in the next step. For now make sure your code complies and then submit it using

```
$ submit -q4 invpower
```

Step 5

This step tests your implementation of the shifted inverse power method using the matrix A given earlier. Choose $\alpha = 0.5 + 4i$ and check that the method converges to the correct eigenvalue $0.47880 + 3.74167i$. Please call your program `step5.c` and the executable `step5`. After debugging your code, choose a value of α in order to find the eigenvalue in the top left corner of the eigenvalue plot. Send the output of your program showing the computation of this eigenvalue to the file `result.txt` and submit it using the following commands

```
$ ./step5 >result.txt
$ cd ..
$ submit -q5 invpower
```

Step 6 Extra Credit

Modify your program to find an eigenvector corresponding to the eigenvalue found in the previous step and submit your code using

```
$ submit -q6 invpower
```