NUMERICAL METHODS 2

Mini-project Task 2 - Report

By: Noman Noor

(Student ID: 302343)

Introduction

❖ The goal and the approach used to:

The goal of the task is to calculate/approximate of the condition number of a symmetric positive-definite matrix. The condition number of a matrix is equal to its biggest eigenvalue divided by its smallest eigenvalue (i.e. condition number=$\lambda_{\max}/\lambda_{\min}$). $\lambda_{\max}$ and $\lambda_{\min}$ are approximated to the specified degree of accuracy (refer to Usage section) using the Power Method and Inverse Power Method respectively. In order to maintain some degree of efficiency, the Inverse Power Method uses the lower and upper triangular matrices found using Doolittle's LU Factorization algorithm instead of relying on calculating the inverse of the matrix. More specifics and details about the algorithms is provided alongside their implementation in the source code (in the form of comments).

❖ Usage:

$$\mathbf{NM2Test2(A, AcceptableErrPM, AcceptableErrIPM)}$$

Where A is the input matrix and AcceptableErrPM and AcceptableErrIPM are the maximum acceptable errors. Note that the testing function "*CheckNM2Test2*" may be called by simply typing its name in the console.

Numerical Tests

❖ For a matrix with Eigenvalues 2 and 6 (generated *by-hand*):

Input matrix: $\begin{bmatrix} 5.2 & -1.6 \\ -1.6 & 2.8 \end{bmatrix}$ *Eigenvalues = {2, 6}*. [Verified with WolframAlpha (which is analytical) so these values can be trusted]. **So, clearly, actual condition number is: 6/2=3.**

How is such a matrix was generated: This matrix was generated by-hand using several key facts about creating Symmetric Positive Definite matrices with specific Eigenvalues (as shown on the Lab classes). The exact formula used is:

$\begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 \\ 0 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 \\ 2 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{26}{5} & -\frac{8}{5} \\ -\frac{8}{5} & \frac{14}{5} \end{pmatrix}$ and then, multiplying and dividing all the elements by 2, we get: $\begin{pmatrix} \frac{54}{10} & -\frac{16}{10} \\ -\frac{16}{10} & \frac{28}{10} \end{pmatrix}$, which is our input matrix.

Analyzing the approximations by NM2Test2 for various acceptable error combinations:

Power Method Acceptable Error	Inverse Power Method Acceptable Error	**Actual Condition Number**	**Approximated Condition Number**	**Error in Condition Number (abs)**
2.2204e-16 (eps)	0.01	3	**3.0124**	0.0349
0.01	2.2204e-16 (eps)		**2.9986**	0.0014
0.0001	0.0001		**3.0001**	0.0001
2.2204e-16 (eps)	2.2204e-16 (eps)		**3**	0

❖ For matrices generated with my function GenSymmPosDefWithEigs(EIGVALS):

How is such a matrix was generated: The testing function *CheckNM2Test2*'s sub-function "*GenSymmPosDefWithEigs(EIGVALS)*" generates a Symmetric Positive-definite matrix with the eigenvalues provided in EIGVALS. The function was implemented by me and mainly relies on the MATLAB function "*sprandsym()*" to generate such a Matrix.

Traits of such a matrix: Since we know the Eigenvalues, it's easier to find expected values of the condition number.

Note: *For this section, the acceptable error for both Inverse Power Method and Power Method is set to "eps" (2.2204e-16), which is the Floating-point relative accuracy in double precision.*

Eigenvalues	Expected Condition Number	Approximated Condition Number
[1,2,3,4,5]	$\frac{5}{1} = 5$	5
[1,2,3,4,5,6,7,8,9,10]	$\frac{10}{1}$=10	10
[33,54,66,98,99] *(Matrix semi-randomly generated on each run, eigenvalues remain the same.)*	$\frac{99}{33}$=3	3.0000

Now, comparing this function to the MATLAB default function on a RANDOM SYMMETRIC-POSITIVE DEFINITE MATRIX: (Note that the dimensions of the matrix are also randomized between 2-11).

Input Matrix B	Cond(B)	NM2Test2(B,-1,-1)
0.6626 0.0115 0.0202 0.0115 0.1907 0.2372 0.0202 0.2372 0.4735	11.9902	11.9902

On each execution, a different matrix is generated. So, we'll look at the one that appeared during my time executing the function for data collection

Conclusion

By now, we can see that our function works and approximates the expected values very well depending on the acceptable errors specified in the parameters. We can also change the level of accuracy to obtain the solution faster or more accurately depending on out needs. We also notice that our function can be just as accurate as the default MATLAB implementation of Cond() function, depending on the provided value.