Technion – Israel Institute of Technology



HW2

Numerical Methods

019003

|  |  |  |
| --- | --- | --- |
| Alon Spinner | 305184335 | alonspinner@gmail.com |
| Oren Elmakis | 311265516 | orenelmakis@gmail.com |

November 11, 2021

# Question 1

# Question 2

We were asked to implement a function that computes the determinant and inverse of an input matrix using PLU decomposition,

To accomplish that, we decomposed matrix into using Matlab’s function which uses the Doolittle method.

## Calculating the determinant:

We can compute the product of directly but evaluating the determinant of the permutation matrix is a little more subtle.

Where is the number of row switches in   
To calculate we used the following algorithm which for each row, sums all the in subsequent rows, that are to the left of the in the current row. Accumulating all those sums will provide .

Noahs Algorithm:

INPUT P

OUTPUT

k = 0;

n = SquareSideLength(P);

FOR index i do:

j = argmax(P(i,:));

k=k+sum(P(i+1:end,1:k),'all');

ENDFOR

We also implemented a more naïve algorithm which permutes to the identity matrix by switches its rows iteratively, and then summing the number of switches to find .  
Both solutions can be found in , in the module.

## Calculating the Inverse

Starting easy, as is a permutation matrix switching the order of the vector basis.

We computed by solving for each column at a time

As is an upper triangular matrix, such a system of equations is easily solved with backwards substitution. Programmatic solution can be found in

Finding is basically the same as finding , only here we used forward substitution to solve for each column of . Implementation under

## Unit Tests

We implemented the script to check our calculations by comparing them against MATLAB’s own and functions.

We present the results below:

|  |  |  |
| --- | --- | --- |
| A | B | C |
|  |  |  |

Seeing how , depict the same system of equations, it is not surprising to see similar numbers for them. As for , which has a horrible condition number, we can just assume MATLAB computes the properties in a very similar way to how we did it.

# Question 3

# Question 4

In this question we were asked to implement a function , and test it on , a function provided it to us.

To compute the estimated error and required iteration number we turned to the book Edition, by Richard L. Burden and J. Dougals Faires  
There, in page 458 we found the equations we needed provided a stable system of the form:

given , we can compute an estimated upper bound for the error

To compute the number of iterations to reach the estimated error bound we use the log function as follows:

## Estimate the number of iterations for starting with

We first checked the condition number for

For such a large matrix, , this is considered as ‘okay’.

Wikipedia*:   
https://en.wikipedia.org/wiki/Condition\_number  
‘As a rule of thumb, if the condition number then you may lose up to digits of accuracy on top of what would be lost to the numerical method due to loss of precision from arithmetic methods’*

Another reference referring to the size of the matrix: *https://math.stackexchange.com/questions/2633861/conditional-number-growth-of-hilbert-matrix-theoretical-vs-matlab*

We used our function )to compute the Gauss Seidel’s and (does do by computing a splitting matrix )

We then applied which we implemented for this question as follows:

## Solve the system using k iterations and calculate

We solved for with funcbund.iterative\_solver(G,c,x0,k)

Seems like we got ourselves a nice solution

## Solving the system with the direct method implemented in Q1 and calculating the error

calling to compute we got:

## Explaining the mismatch between the direct and iterative methods, and comparing the accuracy and computation time for both

Both the iterative method and the direct method gave a residual with an order of magnitude . As such we can call them equivalent as far as they’re their precision.  
Note: we keep in mind that the condition number is ‘okay’ and that the solution is stable

As for the computation time, we ran both computations 100 times each and averaged their tic-toc times:

The Direct method seems to be quicker by a factor of .  
We believe the cause is the number of iterations which is extremely high.   
It is ‘unfair’ to test for a lower number of iterations (solving a system of equations is something you want to do once).   
We can assure the reader that for a higher dimension problem the iterative method would have been more efficient.

# Question 5

# Question 6