Technion – Israel Institute of Technology

HW2

Numerical Methods

019003

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

November 12, 2021

Question 1

This question addresses the Gaussian elimination with and without pivoting implementation. See function *gaus_elim.m*

1. Comparing computation accuracy for toggled pivot function

In this section we have to calculate the numerical error of the Gaussian elimination with pivoting and without.

$$e = \left|\left|x^{\star} - x_{GE}\right|\right|_2$$

$e_{pivot} = 1.1574 \cdot 10^{-9}$
$e_{without\ pivot} = 2.8324 \cdot 10^{-6}$

** If the matrix consists of zeros on the diagonal variables, during the implementation, pivoting is necessary.

2. Comparing computation time for toggled pivot function

We have been asked to compare the computation time of the function with and without pivoting.

$t_{A\backslash b} = 7.9 \cdot 10^{-5}$
$t_{pivot} = 1.5040 \cdot 10^{-4}$
$t_{without\ pivot} = 1.277 \cdot 10^{-4}$

Question 2

We were asked to implement a function that computes the determinant and inverse of an input matrix using PLU decomposition, $function\ [my\_det, my\_inv]\ =\ my\_det\_and\_inv(input\_mat)$

To accomplish that, we decomposed matrix A into PLU using Matlab's lu function which uses the Doolittle method.

Calculating the determinant:

$$|A| = |P||L||U| \underbrace{=}_{triangular\ matrices} |P| \prod l_{ii} \prod u_{ii} \underbrace{=}_{doolite \rightarrow l_{ii}=1} |P| \prod u_{ii}$$

We can compute the product of u_{ii} directly but evaluating the determinant of the permutation matrix P is a little more subtle.

$$|P| = (-1)^k$$

Where k is the number of row switches in P

To calculate k we used the following algorithm which for each row, sums all the $1s$ in subsequent rows, that are to the left of the 1 in the current row. Accumulating all those sums will provide k.

<u>Noahs Algorithm:</u>

INPUT P

`OUTPUT` $(-1)^k$

```
k = 0;
n = SquareSideLength(P);
FOR index i ∈ {1 … n} 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 P to the identity matrix by switches its rows iteratively, and then summing the number of switches to find k.
Both solutions can be found in $function\ detP\ =\ detPerm(P)$, in the $funcbund$ module.

Calculating the Inverse

$$A^{-1} = U^{-1}L^{-1}P^{-1}$$

Starting easy, $P^{-1} = P^T$ as P is a permutation matrix switching the order of the vector basis.

We computed U^{-1} by solving for each column $U^{-1}(:,j)$ at a time

$$UU^{-1}(:j) = I(:,j)$$

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

Finding L^{-1} is basically the same as finding U^{-1}, only here we used forward substitution to solve for each column of $L^{-1}(:,j)$. Implementation under $funcbund.InverseL$

Unit Tests

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 3 \\ 4 & 1 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 3 & -1 \\ 1 & -2 & 1 \\ -1 & -12 & 5 \end{bmatrix} \quad C = \begin{bmatrix} 2 & 1 & 2 \\ 4 & 1 & 2 \\ 1 & 2 & 3 \end{bmatrix}$$

We implemented the script $HW2q2.m$ to check our calculations by comparing them against MATLAB's own inv and det functions.

$$e_{det} = abs\left(\frac{|A|_{us} - |A|_{Matlab}}{|A|_{Matlab}}\right) \quad e_{inv} = \frac{||A^{-1}||_{\infty,us} - ||A^{-1}||_{\infty,Matlab}}{||A^{-1}||_{\infty,Matlab}}$$

We present the results below:

A	B	C
$CondNumber = 43.5$ $e_{det} = 0$ $e_{inv} = 4.934325e - 17$	$CondNumber$ $= 77857280929788400$ $e_{det} = 0$ $e_{inv} = 3.642e - 17$	$CondNumber = 43.6$ $e_{det} = 0$ $e_{inv} = 4.934325e - 17$

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

$$B^{-1} = \begin{bmatrix} 0.1144 & -0.1716 & 0.0572 \\ -0.3431 & 0.5147 & -0.1716 \\ -0.8006 & -1.2010 & -0.4003 \end{bmatrix} \cdot 10^{16}$$

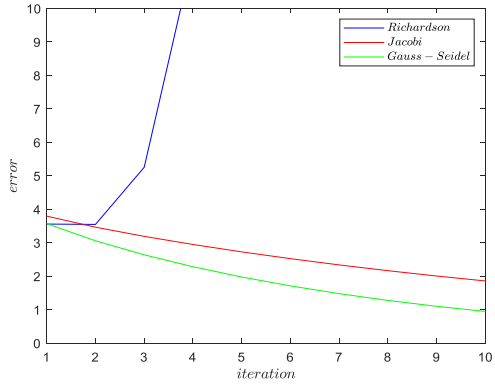
Question 3

We created a function that solves a linear system using an iterative method and 😎😎😎 without any loops 😎😎😎 by using a recursive approach:

```
function sol_itr = iterative_solver(G,c,x0,k)
if (k == 0)
    sol_itr = x0;
else
    x0 = G*x0 + c;
    sol_itr = iterative_solver(G,c,x0,k-1);
end
end
```

The results for 10 iterations with the following methods

Richardson	Jacobi	Gauss-Seidel
3.555	3.792	3.568
3.544	3.462	3.055
5.250	3.188	2.636
11.581	2.945	2.281
28.769	2.724	1.974
73.205	2.522	1.708
187.365	2.335	1.477
480.913	2.162	1.274
1236.944	2.003	1.098
3188.045	1.856	0.946



Note that the Richardson method is diverging while Jacobi and Gauss-Seidel are converging. Moreover, as we expected, the Gauss-Seidel method converging ratio is more significant than the Jacobi method.

Note:
In a different computing language, this may have been beneficial, but in MATLAB entering a new function also means creating a new workspace and allocating RAM to it. This limits the number of iterations we can use, and also hurts the computation time. A simple for loop would perform better on interpreted languages.

Question 4

In this question we were asked to implement a function $[est\_num\_iter, est\_error\,] = itr\_est(x0, G, c, threshold, k)$, and test it on $[A, b] = ODEmodel(n = 100)$, a function provided it to us.

To compute the estimated error and required iteration number we turned to the book $Numerical\ Analysis, 9^{th}$ Edition, by Richard L. Burden and J. Dougals Faires
There, in page 458 we found the equations we needed provided a <u>stable</u> system of the form:

$$\boldsymbol{x}^{(k)} = \boldsymbol{G}\boldsymbol{x}^{k-1} + \boldsymbol{c}$$

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

$$\left|\left|\boldsymbol{x} - \boldsymbol{x}^{(k)}\right|\right| \leq \frac{\left|\left|\boldsymbol{G}\right|\right|^k}{1 - \left|\left|\boldsymbol{G}\right|\right|}\left|\left|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right|\right| = est\ error\ bound$$

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

$$\boldsymbol{e} = est\ error\ bound$$

$$\frac{\left|\left|\boldsymbol{G}\right|\right|^k}{1 - \left|\left|\boldsymbol{G}\right|\right|}\left|\left|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right|\right| = \boldsymbol{e}$$

$$\left|\left|\boldsymbol{G}\right|\right|^k = \frac{\left|\left|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right|\right|\boldsymbol{e}}{1 - \left|\left|\boldsymbol{G}\right|\right|}$$

$$k = \frac{\log\left(\frac{\left|\left|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}\right|\right|\boldsymbol{e}}{1 - \left|\left|\boldsymbol{G}\right|\right|}\right)}{\log\left(\left|\left|\boldsymbol{G}\right|\right|\right)} = est\ num\ iter$$

Estimate the number of iterations for $\left|\left|\boldsymbol{x} - \boldsymbol{x}^{(k)}\right|\right| < \boldsymbol{10^{-2}}$ starting with $\boldsymbol{x}^{(0)} = \boldsymbol{0}$

We first checked the condition number for A

$$cond(A) = 3396$$

For such a large matrix, $\mathbb{R}^{n\times n}$, this is considered as 'okay'.

> <u>Wikipedia</u>*:*
> *https://en.wikipedia.org/wiki/Condition_number*
> *'As a rule of thumb, if the condition number $\kappa(A) = 10^k$ then you may lose up to k digits of accuracy on top of what would be lost to the numerical method due to loss of precision from arithmetic methods'*
>
> <u>Another reference referring to the size of the matrix:</u>
> *https://math.stackexchange.com/questions/2633861/conditional-number-growth-of-hilbert-matrix-theoretical-vs-matlab*

We used our function $funcbund.prepGausSiedel(\boldsymbol{A}, \boldsymbol{b})$ to compute the Gauss Seidel's $\boldsymbol{G}$ and $\boldsymbol{c}$ (does do by computing a splitting matrix $\boldsymbol{Q} = \boldsymbol{D} - \boldsymbol{C_L}$)

We then applied $iter\_est$ which we implemented for this question as follows:
$k = itr\_est(x0, G, c, threshold);$

$$\rightarrow k = 66656.969$$

Solve the system using k iterations and calculate $\left|\left|\boldsymbol{Ax}^k - \boldsymbol{b}\right|\right|_\infty$

We solved for $\boldsymbol{x}^{(k)}$ with funcbund.iterative_solver(G,c,x0,k)

$$\rightarrow \left|\left|\boldsymbol{Ax}^k - \boldsymbol{b}\right|\right|_\infty = 6.137130e - 16$$

Seems like we got ourselves a nice solution

Solving the system with the direct method implemented in Q1 and calculating the error $\left|\left|\boldsymbol{Ax}^k - \boldsymbol{b}\right|\right|_\infty$

calling $\text{gauss\_elim}(\mathbf{A}, \mathbf{b}, \text{true})$ to compute $\boldsymbol{x}^*$ we got:

$$\rightarrow \left|\left|\boldsymbol{Ax}^k - \boldsymbol{b}\right|\right|_\infty = 4.440892e - 16$$

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 10^{-16}. 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:

$$t_{GS} = 7.949e - 2$$

$$t_{GE} = 8.949e - 5$$

The Direct method seems to be quicker by a factor of $\times 2.5$.
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

1. Jacobi's G and $\rho(G)$ for given A,b

The Jacobi's Method's splitting matrix Q is just the diagonal of A.
As such:

$$G = D^{-1}(D - A)$$

$$c = D^{-1}b$$

$$x_{k+1} = Gx_k + c$$

See *funcbund.PrepJacobi* for the implementation.

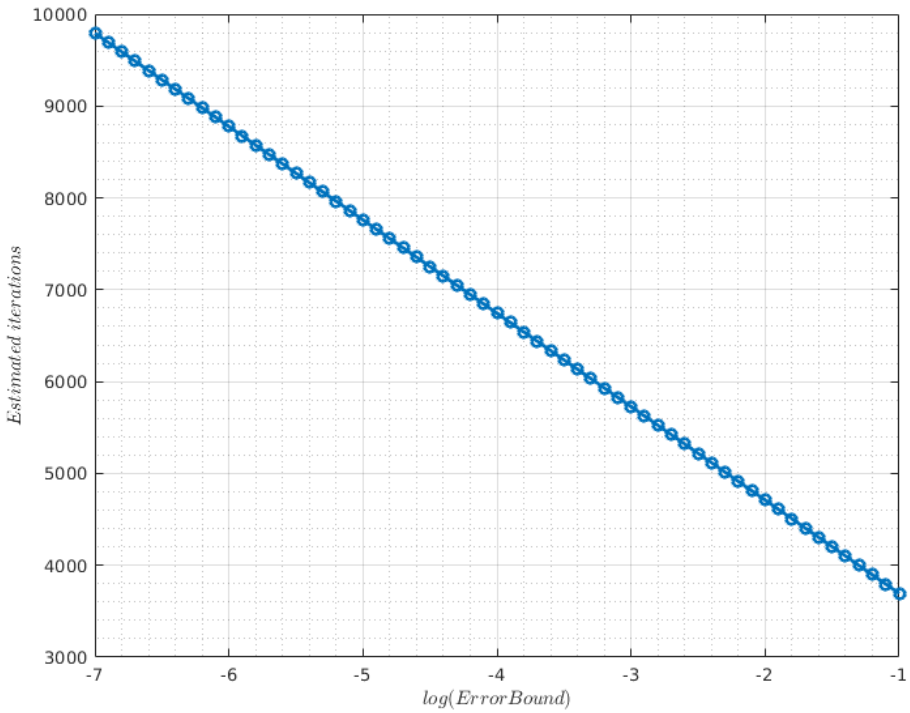
To calculate the spectral raidus $\rho(G) = \max\left(\left|\left|\lambda(G)\right|\right|_\infty\right)$ we used the following code:
max(abs(eig(G)));

$$\rightarrow \rho(G) = 0.9863 < 1$$

The condition number $cond(A)$ is 146.47.
The system is stable, and will converge.

2. Iteration amount for a given accuracy bound

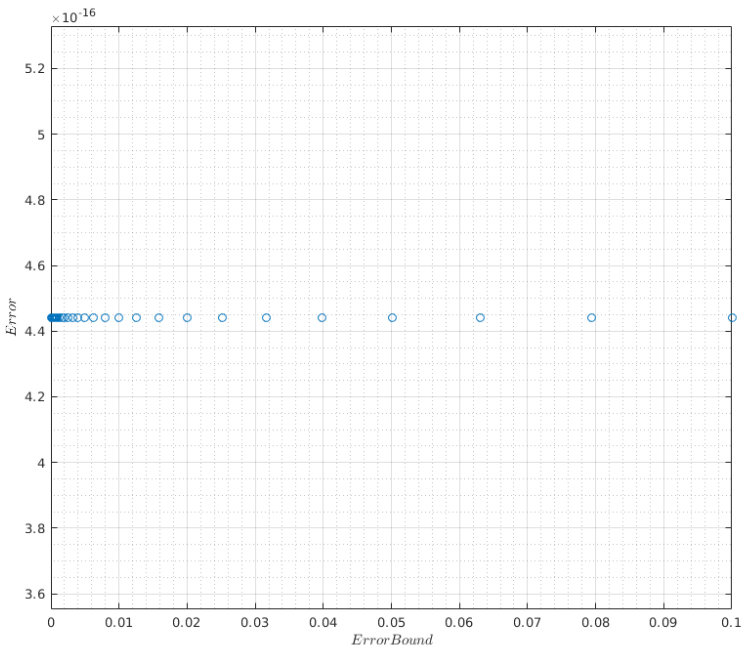
The estimation of how many iterations are needed to solve the system with a required accuracy is provided in a graph below. We used $\boldsymbol{x}^{(0)} = zeros(20,1)$



As one could expect from the derivation of the Error Boundary, there is a linear increase in the amount of iterations required to compute with respect to the logarithmic of the Error Bound. Essentially, it means that it becomes <u>increasily painful</u> as one tightenes up the boundary.

3. Actual Residual vs Error Boundary

We solved the system of equations repeatedly, each time with k iterations, to compare the residual $r = \left||Ax - b|\right|_\infty$ with the boundary estimation $BE(k)$



The upper limit is significantly far from the actual error value. Unfortunately, the scale of the results is 10^{-16} which is the maximum digits precision of Matlab (without VPA). Therefore we can only be sure that the accuracy is much better than the limit.

Question 6

In this HW we learnt how to observe, analyse and solve linear systems of equations $Ax = b$.

- When do the equations make a solid system with a robust solution? → *condition number*
- When should we solve the system with a direct method? → *Small matrices, best accuracy*
- How does one improve on the direct Gauss Elimination method? → *pivoting*
- When should we use an Iterative method? →*sparse A, lenient on accuracy, efficient*
- *When does an interative method promised to converge?* → $\rho(G) < 1$
- Which iterative method to apply and when? → *Depending on the problem, but basically: but basically $SOR > GE > Jacobi > Richardson$*
- Can the error be bounded? → $k_{iterations} = f(Upper\ Error\ Bound\ )$

We also implemented a whole sweet of functions for the task:

- Forward/Backwards subsitutation
- Inversing L,U matrices
- Det(Permutation Matrix)

Honestly, this was a nice set of questions.