Lab3B (Version 2, Mar 3, 2024)
Solving Systems of Equations, Errors and Explorations

Lab1B, Lab2B, Lab3B, Lab4B will a series of Chapters about Numerical Analysis and its applications.

1 Title, Team Members, Abstract and Introduction (max 1 pages)

2 The PA = LU factorization method for linear systems (max 1 pgs)

2.1 Why is PA = LU needed for solving linear systems approximately?

The partial pivoting (PA = LU) method is the workhorse for solution of approximate linear systems in fixed precision.

Give a brief summary illustrated by simple examples $Ax = b$ on why PA = LU factorization is better than naive Gauss Elimination for solving approximate systems in fixed precision.

2.2 How to identify systems Ax = b for which PA=LU is not suited

Consider $n \times n$ matrices A and systems $Ax = b$ where $n < 40$. What type of square matrices A is PA = LU not suited for and how do we identify them? Give a brief example.

2.3 Give some cool larger applications of PA = LU factorization

Give some discussion and references.

3 Iterative solution of systems of linear equations (2 pgs)

In this section you will first give a brief description of Jacobi FPI for linear systems.

3.1 Read Ex 2.24 and in Ex 2.25 use Jacobi to solve the 100,000 equation version of Ex 2.24

Read Example 2.24 (disregard SOR). In Example 2.25 you will use the author's codes to set up and solve with Jacobi iteration the n = 100,000 equation version of Example 2.24 (or as large as your computer could handle).

3.2 Compare and PA = LU and Jacobi Iteration on this problem

What is the largest n for which PA=LU can solve this problem? What is largest n Jacobi iteration could handle for this problem? Discuss and compare PA=LU and Jacobi iteration for this problem.

3.3 Why is solving such large systems important in applications?

Why is solving such large linear systems important applications? Be imaginative and explore references. Give a concise discussion that is understandable to a wide audience.

4 Implement Newton's method for multiple variables and test your code (2 pgs)

In this lab you will be implementing and applying Newton's method for solving nonlinear systems of equations of the form:

$$f(x) = 0\,. \tag{1}$$

where f and x are vectors of dimension $n \geq 2$.

As you work through the lab you should use some examples which you know will work with each of the methods.

4.1 Implement Newton's method for systems using vectorization

Newton's method is a fixed point iteration (FPI) method, so here you will use your work from Lab2A on FPI to help implement Newton's method for vectors x, vector functions $f(x)$.

$$Df(x)\Delta x = -f(x) \tag{2}$$

Now we want to vectorize Newton's method, with as little use of indices as possible

$$x = x + \Delta x \tag{3}$$

We consider the system of equations from Section 2.7/ Example 2.32 of Sauer, solving:

$$v - u^3 = 0, u^2 + v^2 - 1 = 0 \tag{4}$$

in the variables u, v. You should vectorize as much as possible. To do this we define the system as a vector function

```
f = @(x) [ x(2) - x(1)^3 ; x(1)^2 + x(2)^2 - 1 ]
```

We can use the Matlab command to compute the Jacobian of the system (check):

```
syms u v ;
Jac = @(u,v) jacobian([v-u^3,u^2+v^2-1],[u,v]);
Jac(u,v)
```

This yields the output Jacobian as:

$$\begin{bmatrix} -3u^2 & 1 \\ 2u & 2v \end{bmatrix}$$

To be able to use this in the program we convert it to convenient vector form

```
Df = @(x) [-3*x(1)^2, 1 ; 2*x(1), 2*x(2)]
```

We experienced difficulty in copying the pdf file for the above commands into Matlab mainly because copying and pasting yielded extra unwanted spaces. One approach is to manually copy the expressions into your Matlab session. Another is to copy and paste these into your Matlab session and delete the unwanted spaces (which often occurred between x and (1) and x and (2)).

Following the example in the Text in Section 2.7 we initialize the vector x:

```
x = [ 1 ; 2 ]
```

In what follows dx is the vector Δx. We need to solve $Df(x)\Delta x = -f(x)$ so we can do that

```
dx =  - Df(x) \ f(x)
x = x + dx
```

and verify that you get $[1;1]$ after 1 step.

Note that we have avoided taking the very expensive inverse! Here Matlab will usually use PA = LU factoring to solve the linear system.

Note that in this Lab you will be storing values in vectors with components x_k (or in Matlab notation $x(k)$). The header below outlines what your code should take as input and output:

Now implement a program `MyVectorNewton` that takes as input $f, Df, xinitialvec, TOL, MaxIters$:

```
f   = @(x) [ x(2) - x(1)^3 ; x(1)^2 + x(2)^2 - 1 ];
Df = @(x) [-3*x(1)^2, 1 ; 2*x(1), 2*x(2)];
[xsoln , fl ]= MyVectorNewton(f, Df, [ 1 ; 2 ], 10^(-12),10);
```

Here $xsol$ is the approximate solution vector corresponding to the initial vector xinitialvec within tolerance TOL and fl is the value of the output flag for convergence/divergence and other information on the computation. It should be similar to your program for the case $n = 1$, except that a vector norm is needed instead of absolute value.

Note that your must implement your own code which use the expressions in this section and the next and in particular *must include commands based on the vector forms f and Df above*. Your program should be very similar to the one dimensional case (and not copied from the internet - we can easily check that the syntax is not similar to above).

4.2 Testing your code

To ensure that your function `mynewton` is working correctly, you should test the function on some problems which you know the solution for and that it yields the expected approximate quadratic convergence. Here you should use
2.7 Computer Problems/ 4

4.3 Challenging example or novel extended capability

You should demonstrate your code on some challenging example (e.g. bigger n) or provide some novel feature for your program and explain why it is a worthwhile feature.

5 Summary: results, team des, future res, refs (max 2 pgs)

6 Appendices

6.1 Code (max 2 pgs)

6.2 Plots (max 2 pgs)