Solving Systems of Equations, Errors and Explorations

David Tran and Spencer Kelly

March 9, 2024

Abstract

This lab is the 3rd in a series of 4 labs exploring various numerical methods, implementing them, and examining their tradeoffs. In this lab, we explore the PA = LU factorization method for solving systems of linear equations, as well as the Jacobi fixed-point iteration method and multivariate Newton's method. We compare the convergence of the Jacobi FPI and the PA = LU factorization method, and we compare the convergence of Newton's method for different systems of equations. We find that the Jacobi FPI is a simple and easy-to-implement method for solving systems of linear equations, but it is not the most efficient method. We also find that the PA = LU factorization method is not suited for solving systems of linear equations when the matrix is large, and that the Jacobi FPI is much faster and more reliable for this problem. We also find that Newton's method is a very fast and reliable method for solving systems of equations, and that plotting the convergence of Newton's method allows one to understand the rate of convergence and the behavior of the system of equations.

1 Introduction

2 The PA = LU factorization method for linear systems

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

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

2.3 Larger applications of PA = LU factorization

3 Iterative solution of systems of linear equations

Jacobi FPI is a method for solving systems of linear equations of the form $Ax = b$. It is a multi-dimensional analogue of the one-dimensional fixed-point iteration we explored in Lab 2. We require that A is diagonally dominant, which means that the absolute value of the diagonal element of A is greater than the sum of the absolute values of the other elements in the row. This is a sufficient condition for the convergence of the Jacobi FPI. The Jacobi FPI is given by the following formula:

$$x^{(k+1)} = D^{-1}(L + U)x^{(k)} + D^{-1}b \tag{1}$$

where D is the diagonal of A, L is the lower triangular part of A, and U is the upper triangular part of A. The Jacobi FPI is a simple and easy-to-implement method for solving systems of linear equations, but it is not the most efficient method. We will explore the convergence of the Jacobi FPI and compare it to the PA = LU factorization method.

3.1 Solving an equation for n = 100,000

We solve the following system of linear equations

$$\begin{bmatrix} 3 & -1 & 0 & 0 & 0 & \frac{1}{2} \\ -1 & 3 & -1 & 0 & \frac{1}{2} & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 3 & -1 \\ \frac{1}{2} & 0 & 0 & 0 & -1 & 3 \end{bmatrix} x = b \tag{2}$$

where $b = (5/2, 3/2, \ldots, 3/2, 1, 1, 3/2, \ldots, 5/2)^T$, where there are $n - 4$ 3/2's in the middle of the vector. Using the code in Listing 1, we solve the system of linear equations for $n = 100,000$, with the following:

```
>> n = 100000;
>> [A, b] = sparsesetup(n);
>> x = jacobi(A, b, 50);
```

which converges to $(1, \ldots, 1)^T$ after just 50 iterations.

```
% Inputs: full or sparse matrix a, r.h.s. b,
% number of Jacobi iterations, k
% Output: solution x
function x = jacobi(a,b,k)
    n=length(b); % find n
    d=diag(a); % extract diagonal of a
    r=a-diag(d); % r is the remainder
    x=zeros(n,1); % initialize vector x
    for j=1:k % loop for Jacobi iteration
        x = (b-r*x)./d;
    end % End of Jacobi iteration loop
```

Listing 1: The code for the Jacobi FPI

3.2 Comparison of PA = LU and Jacobi Iteration

Theoretically, Jacobi is guaranteed to converge for all n if A is diagonally dominant, which A is in this case. If we try $PA = LU$ decomposition for this problem, using

```
% Solve a system of equations with PA = LU
function x = solve_lu(A, b)
    [L, U, P] = lu(A);
    x = U \ (L \ (P * b));
end
```

Listing 2: The code for the LU decomposition

the function takes much longer due to its $O(n^3)$ time complexity. Even for $n = 10000$, the function takes almost a minute on an Apple M1 Pro chip. Usually for $PA = LU$, if the matrix is rank-deficient (multiple rows/columns that are linearly dependent), the LU decomposition may not be unique and thus may fail to find the correct solution. However in this case, A is not rank-deficient, so the LU decomposition should work. But, the Jacobi method is much faster and more reliable for this problem.

3.3 Why is solving such large systems important in applications?

Solving large systems of linear equations is important in applications because it is a fundamental problem in many areas of science and engineering. For example, in statstics, a linear model may depend on millions or even billions of parameters which translates to a system of equations of the same size. As a concrete example, a dimensonality-reduction technique such as principal component analysis requires solving a system with as many equations as there are data points and/or desired dimensions. Or, for example, modelling fluid dynamics similarily requires very large systems of equations due to the complexity of the problem. Data is naturally highly multi-dimensional, which makes solving large systems important in practice.

4 Implement Newton's method for multiple variables

4.1 Implementation of Newton's method for systems using vectorization

```
function [x,flag] = vectornewton(f,Df,x0,tol,maxiter)
flag = -1;
x = x0;
for i = 1:maxiter
    x_old = x;
    x = x - Df(x)\f(x);
    if norm(x - x_old) <= tol
        flag = i;
        break
    end
end
end
```

Listing 3: The code for the Newton's method

4.2 Testing

We test Newton's method on the following system of equations in Listing 4: which yields the approximate solution of $u = 1.0960, v = -1.1592, w = -0.2611$ in 7 iterations with a tolerance of 10^{-12}.

$$\begin{aligned} 2u^2 - 4u + v^2 + 3w^2 + 6w + 2 &= 0 \\ u^2 + v^2 - 2v + 2w^2 - 5 &= 0 \\ 3u^2 - 12u + v^2 + 3w^2 + 8 &= 0 \end{aligned} \tag{3}$$

In the same code, we solve

$$\begin{aligned} y^2 - x^3 &= 0 \\ x^2 + y^2 - 1 &= 0 \end{aligned} \tag{4}$$

which yields $x = 0.8260, y = 0.5636$ in 7 iterations with a tolerance of 10^{-12}.

4.3 Adding Visualization to Newton's Method

Plotting the convergence of Newton's method allows one to understand the rate of convergence and the behavior of the system of equations, since we often don't know how the system may converge or behave beforehand. For example, We can plot the convergence of Newton's method for the system of equations in Listing 4 by using the code in Listing 5, which yields the plot in Figure 1. It shows that although we don't reach the desired tolerance until iteration 7, the rate of convergence is very fast, and the solution is very close to the true solution after just 4 iterations.

5 Summary

5.1 Results

We explored 3 different methods of solving systems of equations across 2 different categories: direct substitution methods, and iterative methods. We found that the Jacobi FPI is a simple and easy-to-implement method for solving systems of linear equations, but it is not the most efficient method. We also found that the PA = LU factorization method is not suited for solving systems of linear equations when the matrix is large, and that the Jacobi FPI is much faster and more reliable for this problem. We also implemented Newton's method for systems of equations and found that it is a very fast and reliable method for solving systems of equations, and that plotting the convergence of Newton's method allows one to understand the rate of convergence and the behavior of the system of equations.

5.2 Team Work Problems and Ideas

We worked well together and were able to complete the lab in a timely manner. We were able to divide the work evenly and work on the lab together. We were able to communicate effectively and work together to solve the problems in the lab. A problem that was encountered was in the testing of the PA = LU method, where the function took much longer than expected to run. We were able to solve this problem by starting with much smaller input sizes (e.g. $n = 100$) and growing n much slower to see how the function scaled.

5.3 Future Explorations

In the future, we would like to explore other methods for solving systems of equations, such as the Gauss-Seidel method, the SOR method, and the conjugate gradient method. We would also like to explore the convergence of these methods and compare them to the methods we explored in this lab. We would also like to further explore how ill-conditioned matrices may behave with the discussed methods, and how to transform the matrices to be better conditioned.

5.4 References

6 Appendices

6.1 Code

```
clear
clc

f1 = @(x)[
    2*x(1)^2 - 4*x(1) + x(2)^2 + 3*x(3)^2 + 6*x(3) + 2;
    x(1)^2 + x(2)^2 - 2*x(2) + 2*x(3)^2 - 5;
    3*x(1)^2 - 12*x(1) + x(2)^2 + 3*x(3)^2 + 8
];

Df1 = @(x)[
    4*x(1) - 4, 2*x(2), 6*x(3) + 6;
    2*x(1), 2*x(2) - 2, 4*x(3);
    6*x(1) - 12, 2*x(2), 6*x(3)
];

[x1_soln, iters1] = vectornewton(f1, Df1, [0; 0; 0], 10^(-12), 100)


f2 = @(x)[
    x(2) - x(1)^3 ;
    x(1)^2 + x(2)^2 - 1
];
Df2 = @(x)[
    -3* x(1)^2 , 1 ;
    2*x(1) , 2* x(2)
];
[x2_soln , iters1] = vectornewton(f2 , Df2 , [ 1 ; 2 ], 10^(-12) ,10)
```

Listing 4: The code for the Newton's method test

```
function [x, flag] = vectornewtonvisualized(f, Df, x0, tol, maxiter)
    flag = -1;
    x = x0;

    % Initialize vectors to store iteration information
    iteration_numbers = zeros(maxiter, 1);
    norms_of_steps = zeros(maxiter, 1);

    for i = 1:maxiter
        x_old = x;

        % Compute the Newton step
        step = Df(x)\f(x);

        % Update the solution
        x = x - step;

        % Store iteration information
        iteration_numbers(i) = i;
        norms_of_steps(i) = norm(step);

        % Check for convergence based on the norm of the step
        if norms_of_steps(i) <= tol * (1 + norm(x))
            flag = i;
            break
        end
    end

    % Plot the convergence
    figure;
    plot(iteration_numbers(1:flag), norms_of_steps(1:flag), '-o');
    title('Convergence of Newton''s Method');
    xlabel('Iteration');
    ylabel('Norm of Newton Step');
    grid on;
end
```

Listing 5: The code for the Newton's method with visualization

6.2 Plots

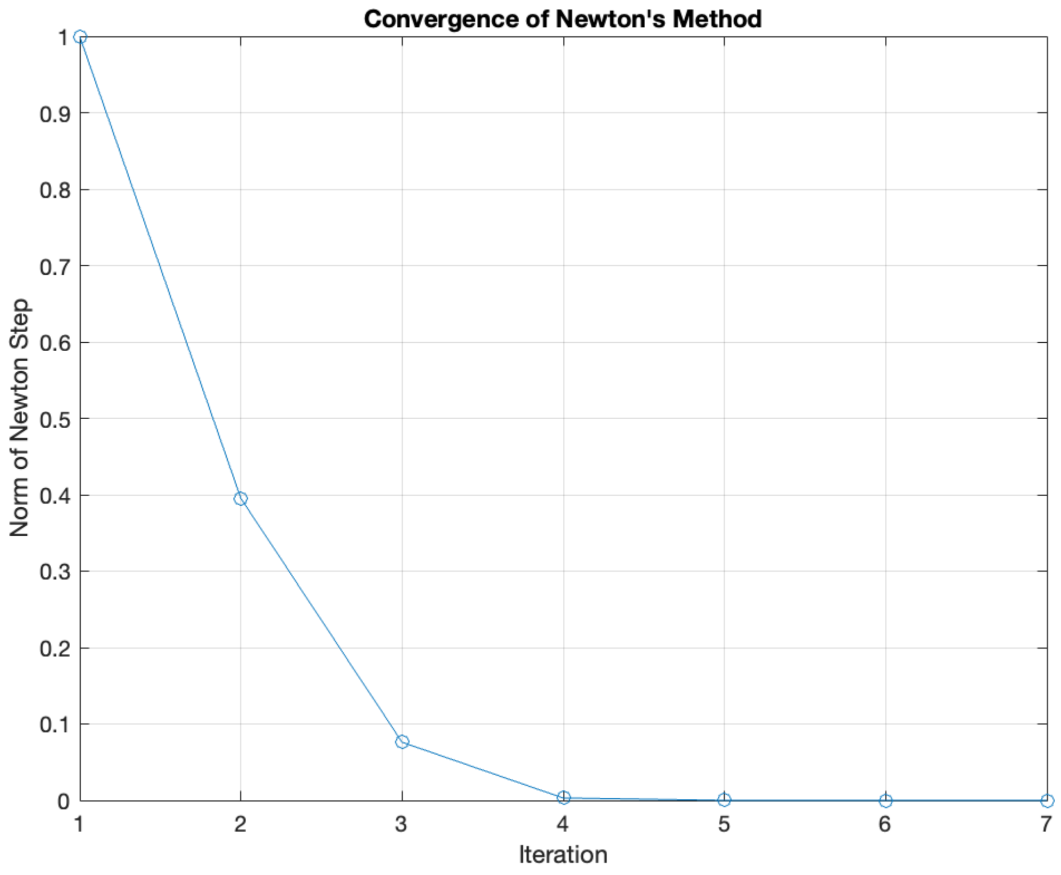

Figure 1: The convergence of Newton's method for the second system of equations in Listing 4