+3 for presentation

CS375 HW7

Ryan Scherbarth, University of New Mexico

September 2024

1. Jacobi Algorithms

   - (a) Write a matlab function, function x = my_jacobi(A, b, tot_it), which takes as input the matrix A, the vector $\overrightarrow{b}$, number of iterations *to_it* and outputs the Jacobi solution after *tot_it* iterations. For this part, code up the component-wise version of the Jacobi method.

```
function x = my_jacobi(A, b, tot_it)
    n = length(b);
    x = zeros(n, 1);

    D = diag(A);

    for k = 1:tot_it
        x_n = zeros(n, 1);
        for i = 1:n
            sum = 0;
            for j = 1:n
                if j ~= i
                    sum = sum + A(i, j) * x(j);
                end
            end
            x_n(i) = (b(i) - sum) / D(i);
        end
        x = x_n;
    end
end
```

- Now create another function, function x = my_vector_jacobi(A, b, tot_it), which also takes as input the matrix A, the vector $\overrightarrow{b}$, number of iterations tot_it, and outputs the Jacobi solution after tot_it iterations. For this part, code up the vector-update version of the jacobi method. Go ahead and "backslash? for D^{-1}. Verify you get the same residuals and errors using the component-wise and vector-update versions with $P = 20$ and 100 iterations. Hint: Use the diag function in Matlab to extract the main diagonal of the matrix: diag(diag(A)). You can also use triu to get the upper triangular portion and tril to get the lower triangular portion of A (with diagonals included).

```
function x = my_vector_jacobi(A, b, tot_it)
    n = length(b);
    x = zeros(n, 1);
    D = diag(diag(A));

    R = A - D;

    for k = 1:tot_it
        x = D \ (b - R * x);
    end
end
```

- (c) Which version of the Jacobi algorithm is faster? Compare the time it takes to perform 100 iterations with P=20. What do you think is causing the speed difference.

The vector update version is significantly faster. This is likely because it will be relying on matrix-vector operations as compared to the other version just using a for loops. This is likely because in the vector version we are still operating with vectors and therefore give Matlab a chance to utilize vector optimizations to speed up the calculations.

what are your times?? -5

This is compared to the first version we implemented where we broke down the entire problem into scalar multiplications, removing Matlab's ability to make any optimizations.

- (d) Now, we will investigate the error in the jacobi solution by varying the size of the linear systems, and keeping the number of Jacobi iterations fixed at 100. We compare the error in the Jacobi solution with the "true solution" obtained from the Matlab "backslash" operator. Make a table showing how the 2-norm in the relative error in the Jacobi solution; $\frac{||x_{\text{true}}-x||_2}{||x_{\text{true}}||_2}$ varies with increasing size of the linear system. Use P = 10,20,40,80,160 which corresponds to n = 100,400,1600,6400,25600. Use whichever version (component-wise or vector-update) of jacobi is faster. Does relative error increase or decrease as the system size increases.

P	n	Relative Error
10	100	1.592735e-02
20	400	3.236866e-01
40	1600	7.421344e-01
80	6400	9.253894e-01
160	25600	9.803876e-01

We see that the relative error increases as we increase the system size and leave the number of Jacobi iterations constant. This is to be expected, as the Jacobi's accuracy is relative to the total problem space and the iterations. If we were to do a much smaller matrix, our jacobi would be more accurate with the 100 iterations, but in increasing the matrix size we become less and less accurate.

2. Comparing Algorithms

- (a) Create a function, function x = my_gauss_seidel(A, b, tot_it), which also takes as input the matrix A, the vector $\overrightarrow{b}$, number of iterations tot_it and outputs the Gauss-Seidel solution after tot_it iterations. You may choose either the component-wise or vector update version.

The vector update version;

```
function x = my_gauss_seidel(A, b, tot_it)
    n = length(b);
    x = zeros(n, 1);

    L = tril(A);
    U = triu(A, 1);

    for k = 1:tot_it
        x = L \ (b - U * x);
    end
end
```

- (b) Now investigate the performance of these two (Jacobi and GS) methods. Compute the error in solutions obtained from Jacobi and GS as the number of iterations is varied for a fixed system size (use P=160). Make a single table showing the 2-norm in the relative error as the number of iterations is varied. Use tot_it = 50, 100, 200, 400, 800, 1600. This table will have three columns: Num Iterations, Error Jacobi, and Error GS. Comment on the performance of the methods in reducing the relative error.

Num Iterations	Error Jacobi	Error Gauss-Seidel
50	9.901226e-01	9.803920e-01
100	9.803876e-01	9.613118e-01
200	9.613036e-01	9.245329e-01
400	9.245182e-01	8.557666e-01
800	8.557416e-01	7.341945e-01
1600	7.341549e-01	5.412264e-01

Gauss-Seidel and Jacobi are fundamentally differnet algorithms and this is shown in this table. We see Jacobi has it's error decrease at a much slower rate than Gauss Seidel.

Gauss Seidel instantly applies the updated values into the calculation as it calculates the next value, which helps it to outperform Jacobi. By the end of our table Gauss-Seidel is around 20% more accurate than Jacobi.