Iteration Methods

Andreas Hegseth

July 9, 2024

1 problem

1) We must implement three method to solve a linear system of equation Ax=b, where b is the zero vector and that A is a tridiagonal matrix given a diagonal constants a=-4, b=1, and c=1. We will then implement Jacobian, Gauss-Siedel, and SOR iterative methods, take their norms with the help of our given lambda that equals a+2(sqrt(ac))cos[s(pi)/n+1] where s=1,..,n. Run it through our code in ten iterations then 100 iterations and see if the given data will converge a vector x in our system.

2 Mathematics

All of the iteration methods are based of Ax=b in which take the residual vector (r=b-Ax, where x is a approximation) of each iteration and then we take the norm of the residual to find the distance between each vector.I used the L2 norm to find it.That is you take the square root of the square of each iteration all summed together for each vector. Also we can look like ——x-xaprox——.

2.1 Jacobian

2) Jacobi method can find if a system of equation converge. Take Ax=b now make the matrix A=(D-L-U) where D is the diagonal matrix and L is the low triangular matrix and U is the upper triangular matrix and the matrix algebra manipulation we can get x(k)=1/D(L+U)x(k-1)+(1/D)b or for a more friendlier way to put in code *(1)* xi(k)=1/aii[nj=1, jI (-aijxj(k-1))+bi] for i=1,..,n. Then we implement however many iteration to get our desired tolerance to (10)power-10 by use of the norm.

2.2 Gauss-siedel

Gauss-sidel this method continue off the Jacobi method by incorporating it into the given equation to solve by subtracting the current sum of iteration from the previous sum of iterations*(2)* xi(k)=1/aii-SUM[i-1j=1(aijxj(k))]-SUM[nj=i+1(aijxk(k-1))+bi].

2.3 SOR

SOR just like Gauss-sidel playing of Jacobi SOR does this with Gauss-sidel. In SOR we find a Ω that is greater than 1 and then plug this omega that converges the system much faster with this given constant. The omega helps to converge much quick with each iteration show by this equation *(3)* xi(k)=(1-w)xi(k-1)+w/aii(bi-SUM(i-1j=1aijxj(k))-SUM(nj=i+1aijxj(k-1)). This method by far is the faster method but only if the weighted omega works.To find the optimal omega you need the following equation W=2/(1+sqrt[p(Tj)*P(Tj)]). Note: P(Tj) is the spectral radius of 1/D(L+U) of the deconstructed matrix in Jacobi.

3 algorithm

Jacobi I put my given inputs my parameter of a matrix in nxn where aij,1¡=I, j¡=n of the matrix, bi is 1¡=i¡=n, and XO=x(0) and our tolerance at 10e-10 and max-iterations of N.

Step 1: set k=1

Step 2: if k¡=N then set in range of i=1,..,n then set equation of Jacobi *(1)*.

```
coverged in 18 iterations as [ 8.93487595e-09  9.05129127e-09  4.35538823e-08 -3.08355084e-08
 -8.71950760e-08  3.58268153e-08  1.02489139e-07 -2.56841304e-08
 -7.07223080e-08  8.93487595e-09]
```

Figure 1: Jacobi run with 10 iterations

```
coverged in 18 iterations as [-0.00000000e+00  9.05129127e-09  0.00000000e+00 -3.08500603e-08
  0.00000000e+00  3.60305421e-08  0.00000000e+00 -2.70083547e-08
  0.00000000e+00  1.42026693e-08  0.00000000e+00 -5.26779331e-09
  0.00000000e+00  1.32422429e-09  0.00000000e+00 -2.03726813e-10
  0.00000000e+00  1.45519152e-11  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  0.00000000e+00  0.00000000e+00 -1.45519152e-11 -0.00000000e+00
  2.32830644e-10  0.00000000e+00 -1.73167791e-09 -0.00000000e+00
  7.91624188e-09  0.00000000e+00 -2.47382559e-08 -0.00000000e+00
  5.54136932e-08  0.00000000e+00 -9.00472514e-08 -0.00000000e+00
  1.02911144e-07  0.00000000e+00 -7.07514118e-08 -0.00000000e+00]
```

Figure 2: Jacobi run with 100 iterations

Step 3: set ——x-Xapprox——¡ Tol then get output
Step 4: set k=k+1 then set loop for i=1,. . . ,n where XOi=xi
Step 5: fail otherwise STOP
Gauss-Siedel input is identical to Jacobi only difference is Step 2: put in Gauss-siedel equation *(2)*.
SOR again same input except of our omega that we set as a parameter. Also put in SOR equation *(3)*.

4 results

My result confused me a little bit as I expected Jacobi was the slowest which make sense as it it is the base of Gauss and SOR which are more improve methods.Gauss and SOR converged on the same iteration which tells me that an omega in this system of equation won't effect it as much. I theorize that it is because we have a constant eigenvalues throughout the whole system. But I still believe that SOR is the fastest method.

5 conclusion

These iteration method of Ax=b are a unique and quick method to solve a linear system but like most things can only hold if given a good amount of information. The given Matrix is specific case in which

```
covered in 12 iterations as [ 7.67961126e-08  7.99606177e-08 -9.88357778e-08  7.52721705e-08
 -4.66721435e-08  2.51014361e-08 -1.19178156e-08  4.96109472e-09
 -1.73568388e-09  4.33920970e-10]
```

Figure 3: Gauss-siedel run with 10 iterations

```
coverged in 13 iterations as [-1.61684355e-09  1.32990330e-09  6.98885061e-10  8.54422269e-10
  8.06077847e-10  8.24886060e-10  8.16606060e-10  8.20381564e-10
  8.18673011e-10  8.19428981e-10  8.19103055e-10  8.19240063e-10
  8.19183805e-10  8.19206417e-10  8.19197504e-10  8.19200955e-10
  8.19199640e-10  8.19200134e-10  8.19199951e-10  8.19200018e-10
  8.19199994e-10  8.19200002e-10  8.19199999e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10  8.19200000e-10
  8.19200000e-10  8.19200000e-10  8.19200000e-10 -1.40819612e-08
  3.36017546e-08 -4.76468268e-08  5.00936148e-08 -4.27961263e-08
  3.11955151e-08 -1.99680493e-08  1.14158724e-08 -5.87716656e-09
  2.72236003e-09 -1.11817430e-09  3.88939646e-10 -9.72349116e-11]
```

Figure 4: Jacobi run with 100 iterations

```
covered in 12 iterations as [ 7.68194970e-08  7.99866844e-08 -9.88700959e-08  7.52999052e-08
 -4.66903293e-08  2.51117475e-08 -1.19229628e-08  4.96334184e-09
 -1.73650656e-09  4.34135757e-10]
```

Figure 5: SOR run with 10 iterations

```
covered in 13 iterations as [-1.61721281e-09  1.33021310e-09  6.99025595e-10  8.54615976e-10
  8.06249928e-10  8.25068371e-10  8.16783328e-10  8.20561290e-10
  8.18851573e-10  8.19608077e-10  8.19281914e-10  8.19419025e-10
  8.19362723e-10  8.19385352e-10  8.19376432e-10  8.19379886e-10
  8.19378570e-10  8.19379064e-10  8.19378882e-10  8.19378948e-10
  8.19378924e-10  8.19378933e-10  8.19378930e-10  8.19378931e-10
  8.19378930e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10  8.19378931e-10
  8.19378931e-10  8.19378931e-10  8.19378931e-10 -1.40858508e-08
  3.36118734e-08 -4.76622625e-08  5.01109483e-08 -4.28118625e-08
  3.12076552e-08 -1.99762457e-08  1.14208005e-08 -5.87982799e-09
  2.72365027e-09 -1.11872780e-09  3.89140354e-10 -9.72871314e-11]
```

Figure 6: SOR run with 10 iterations

a traditional matrix are quiet convenient but I believe given a more chaotic matrix it would more iterations to solve. The give result show me as well that I may not get the answer I was hoping for because some of the iteration would go of script from the standard norm but other than that I found coding linear systems to be fun.

6 code

You can find the code as an attachment. Note: Josef Morstatt, and Thomas Thompson help me exponentially with this code and I like you to know that When you grade theirs.