NumComp - Fall 2022
Project #4

Due –

1. Generate a right-hand-side b of all ones of appropriate size.
2. Solve $Ax = b$ with a generic linear solver (eg, `numpy.linalg.solve` or Matlab's `backslash`). Call the resulting vector *truth*. This is the vector against which you will compute the error. Run a timing study with the generic linear solver.
3. Write a function that solves $Ax = b$ using either the LU decomposition or the Cholesky factorization, depending on whether the matrix is symmetric or not.
4. Write a function that solves $Ax = b$ using the Jacobi method. Run a timing study with your function.
5. Write a function that solves $Ax = b$ using the Gauss-Seidel method. Run a timing study with your function.

For parts 3-5, report the relative error compared to the *truth* that you computed in part 2.

Here are some questions that might help you think about this problem.

- Look up "diagonally dominant." It relates to the convergence of the iterative methods.
- Remember your paramedic training. Which methods work better for which matrices? Why do you think that is? And what do you mean by "work better"?
- Try interpreting the results from the iterative methods in light of the theory we know about fixed point methods.

- Results

	Method	backslash	luchol	jacobi	gs
A1 10	Iterations	N/A	N/A	10	10
	Time	0.001288	N/A	0.001036	0.001593
	Ave Error	N/A	0	0	0
A2 30	Iterations	N/A	N/A	MAX	798
	Time	0.011253	N/A	0.028582	0.00636
	Ave Error	N/A	0	NaN	3.88E-15
A3 400	Iterations	N/A	N/A	MAX	MAX
	Time	0.003617	N/A	3.183682	12.05529
	Ave Error	N/A	NaN	NaN	NaN
A4 50	Iterations	N/A	N/A	MAX	MAX
	Time	0.002613	N/A	0.044686	0.055024
	Ave Error	N/A	0	NaN	NaN
A5 625	Iterations	N/A	N/A	4368	2218
	Time	0.007237	N/A	1.32252	2.588112
	Ave Error	N/A	0	3.27E-13	2.10E-13

- Iterative methods are given all the same arguments of 1e-13 tolerance, 10000 iterations max, and initial guess.
- All methods performed fairly similarly for the 10x10 matrix, with no error for any method
- GS method was able to solve a system that jacobi method was not. The resulting value from both convergence calculations $max((L+D)^-1 * U)$ and $max(D^-1(L+U))$. Here are the results for each matrix

  $M1 = 2.5484 \quad M2 = 0.4164 \quad M3 = N/A \quad M4 = 7.3888 \quad M5 = 0$

- Despite the requirement that result be less than 1, both GS and Jacobi were able to solve Matrix1 which had the convergence function result of 2.5484. Strangely, Matrix2 with the convergence function value of .4164 was unable to be solved by my Jacobi method, but was solved by my GS method. Matrix2 was also solved by GS about twice as fast as the backslash method.

- Matrcies 2 and 5 were the only symmetric ones, which both were solved by GS, while only 5 was solved by Jacobi. It appears that the Jacobi method is more reliant on matrix symmetry and diagonal proportion.
- For matrix5, which all methods could solve, both iterative methods took a significant while longer than the backslash method. On top of that, my GS implementation takes about twice as long to solve M5 than my Jacobi implementation. In line with lecture, GS took less iterations and converged faster, but had more heavy calculations per, while Jacobi took more iterations with lighter calculations per round.
- It seems the ratio of the matrix diagonal to its upper / lower triangles is incredibly important, as well as its sparseness.

real quick look at this ratemyprof of this dude's class I had the displeasure to drop: https://www.ratemyprofessors.com/professor?tid=1702778

He inspired me to actually go and get diagnosed with ADHD!

- Code

```
function oA = gs(iA, b, nit, tol, x0)

    k = 0;
   D = diag(diag(iA));
   L = tril(iA,-1);
   U = triu(iA,1);
    fx = iA*x0 -b;
   while (k < nit)

        %disp('iteration: '); disp(k);
        ci = b - U*x0;
        xi = (D+L)\ci;
        cerror = norm(xi-x0);
        if (fx == 0 | cerror < tol)
            break;
        end
        x0 = xi;
        k = k + 1;
        fx = iA*x0 - b;

   end
   disp('iteration: '); disp(k);
   oA = xi;

end
```

jacobi2.m

```
function oA = jacobi2(iA, b, nit, tol, x0)
%your lecture code and this dudes code for e
%https://sites.google.com/site/numericosspj/

k = 0;
   LU = triu(iA,1) + tril(iA,-1);
   D = diag(diag(iA));
    fx = iA*x0 -b;
   while (k < nit)

        %disp('iteration: '); disp(k);
        ci = b - LU*x0;
        xi = D\ci;
        cerror = norm(xi-x0);
        if (fx == 0 | cerror < tol)
            break;
        end
        x0 = xi;
        k = k + 1;
        fx = iA*x0 - b;

   end
   disp('iteration: '); disp(k);
   oA = xi;

end
```

```
function oA = luchol(iA, b)
%pretty much copy paste from matlab docs for t
if issymmetric(iA)
    disp('is sym, using chol');
    R = chol(iA);
    oA = R\(R'\b);
else
    disp('not sym, using lu');
    [L, U, P] = lu(iA);
    y = L\(P*b);
    oA = U\y;

end

end
```

```
function oA = lumax(iA, opt)
%LUMAX Summary of this function goes here
%   Detailed explanation goes here
    L = tril(iA,-1);
   U = triu(iA,1);
    D = diag(diag(iA));
    oA1 = max(max( inv(D) * (L + U)  )); %Jacobi
    oA2 = max(max(  inv(L + D) * U  )); %GS

    if opt >= 1 %Jacobi for 1 or greater
        disp('jacobi');
        oA = oA1;
    else
        disp('gs');
        oA = oA2; %GS for < 1

    end
end
```

- gs is Gauss - Seidel, jacobi2 is Jacobi, luchol is function running LU or Chol based on symmetry

- function lumax runs convergence calculations, based on lecture math provided for such– results less than one should be convergent