function jacobi(A, b, N)

% Jacobi method for solving Ax = b

% A will be initialized as a 3x3 diagonally dominant coefficient matrix.

% b will be initialized as a 3x1 vector.

% N is the max number of iterations

% The starting vector is the null vector, [0 0 0]

% The output is the solution vector x

n = size(A,1);

% splitting matrix A into the three matrices L, U and D

% diagonal of matrix A returns column vector with pivots

% call diag again and diagonal matrix D is given

% L is the lower triangular matrix of A

% U is upper triangular matrix of U

D = diag(diag(A));

L = tril(-A,-1);

U = triu(-A,1);

% transition matrix and constant vector used for iterations

Tj = inv(D)\*(L+U);

cj = inv(D)\*b;

% x is the starting null vector

% tol is the tolerance we will be running this for

% with 1e-5, this will mean it will be accurate to 4 decimal places

tol = 1e-05;

k = 1;

x = zeros(n,1);

while k <= N

x(:,k+1) = Tj\*x(:,k) + cj;

if norm(x(:,k+1)-x(:,k)) < tol

disp('The procedure was successful')

disp(['The condition was met after ' num2str(k) ' iterations'])

disp('x = ')

disp(x(:,k+1));

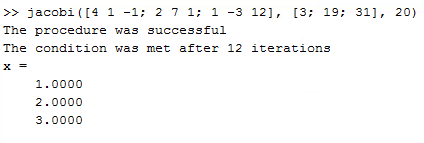
break

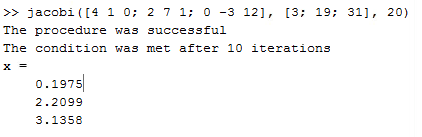
end

k = k+1;

end

Output:





function y = GS(A,b,x,numIters)

clc;

% Gauss-Seidel method for solving Ax = b:

% A will be initialized as a 3x3 diagonally dominant coefficient matrix.

% b will be initialized as a 3x1 vector.

% x is the starting values for our solution vector

% (for our purposes x = [0 0 0]).

% numIters is the number of iterations our Gauss-Seidel

% method will run through.

disp('A =');

disp(A);

disp('x =');

disp(x);

% Isolates the diagonal vector of A

% number of total elements is the length of x (which is 3)

% y is a zero array with size of x elements times our iteration count

D = diag(A);

A = A-diag(D);

numElements = length(x);

x = x(:);

y = zeros(numElements, numIters);

% outer loop iterates Gauss-Seidel

% inner loop goes through every x in x vector

% algorithm is the formula for manual Gauss-Seidel calculation

% y prints the x vector after each iteration

for m = 1:numIters

for n = 1:numElements

x(n) = (b(n)-A(n,:)\*x)/D(n);

end

y(:,m) = x;

end

Output:

