Assignment 1

Question

- User the SOR method to solve the following linear equation set the relaxation factor $\boldsymbol{\omega}$ = **0.5**, **1.0** and **1.7** respectively. When $\|\mathbf{X}^{(k+1)} - \mathbf{X}^{(k)}\|_\infty <= \mathbf{10^{-5}}$ stop iteration. Compare the iteration numbers of the three cases.

$$\begin{pmatrix} 5 & -1 & -1 & & & & & \\ -1 & 5 & -1 & -1 & & & & \\ -1 & -1 & 5 & -1 & -1 & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & -1 & -1 & 5 & -1 & -1 \\ & & & -1 & -1 & 5 & -1 \\ & & & & -1 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{48} \\ x_{49} \\ x_{50} \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 2 \\ \vdots \\ 2 \\ 2 \\ 1 \end{pmatrix}.$$

Algorithm

- SOR stands for (Successive Overrelaxation) which is another iterative method in generalization and improvement of the Gauss-Seidel method. In any iterative method, in finding $\mathbf{X}^{(k+1)}$ from $\mathbf{X}^{(k)}$, we try to move some certain amount in a particular direction from $\mathbf{X}^{(k)}$ to $\mathbf{X}^{(k+1)}$. If we assume that the direction from $\mathbf{X}^{(k)}$ to $\mathbf{X}^{(k+1)}$ is taking us closer, but not to the true solution $\mathbf{X}$, then it would make sense to move in the same direction $\mathbf{X}^{(k+1)} - \mathbf{X}^{(k)}$, but further along that direction.

$$x_i^{(k+1)} = x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} a_{ij} x_j^{(k)} \right)$$

```
Choose an initial guess φ to the solution
repeat until convergence
    for i from 1 until n do
        σ ← 0
        for j from 1 until n do
            if j ≠ i then
                σ ← σ + a_ij φ_j
            end if
        end (j-loop)
        φ_i ← (1 − ω)φ_i + (ω / a_ii)(b_i − σ)
    end (i-loop)
    check if convergence is reached
end (repeat)
```

SOR algorithm

Analysis

- Usually the relaxation factor **ω** depends upon the properties of the coefficient matrix and we are interested in faster convergence rather than just convergence. A poor choice for **ω** leads to slower convergence whereas a good choice for **ω** leads to faster convergence. Since all the given values of **ω** are in the range 0 - 2 it's a good thing. And as **ω** increases from 0.5 to 1.5 the SOR converges much faster by putting higher values on the newly computed values.

Matlab Code

```
tol = input('Tolerance:  ');
m = input('Max iterations:  ');
w = input('omega:  ');
n = input('Enter number of equations, n:  ');
A = zeros(n,n+1);
x1 = zeros(1,n);

A = [input_maxtix]
X = [initial approximation values]
k = 1;
```

```
while  k <= m
  err = 0;
  for i = 1 : n
    s = 0;
    for j = 1 : n
      s = s-A(i,j)*x1(j);
    end
    s = w*(s+A(i,n+1))/A(i,i);
    if abs(s) > err
      err = abs(s);
    end
    x1(i) = x1(i)+s;
  end

  if err <= tol
    break;
  else
    k = k+1;
  end
end

fprintf('Solution: \n', k);
for i = 1 : n
fprintf('   %11.8f \n', x1(i));
end
```

Results

- The results shown below use the following parameters for the SOR algorithm, **number of iterations 20, the tolerance 10-5, ω 0.5 - 1.5.**

ω = 0.5	**ω = 1.0**	**ω = 1.5**
0.03981499	-74584.22269491	181632767.02474564
1.34685434	-4927125.76805898	21606243642.149433
-31.28515247	21839817.52536922	-50247995663.129272
-2517.67388012	2054902010.99628210	-5832281177979.6591

-93.82595091	-2697633394.605933	13665771060683.7460
23559.23077273	-15207057386.696414	20212954058624.9179
-6152.58822508	46123153095.9107589	-169012112570096.46
-175452.95264310	45696467589.962883	282048042943255.125
109306.49208293	-401914262720.92449	486641802866036.50
1041726.65590730	362040408403.974121	-3207219420235515.5

The first 10 results after computing with different ω values with SOR

Conclusion

- The SOR method is sort of an extension to the Gauss-Seidel method with faster convergence for solving linear system equations. Generally the value of **ω** is between 0 and 2. When the relaxation factor **ω** increases the convergence becomes faster which puts more weight on the recently computed values. Finding the optimal value for **ω** is quite difficult but for some special matrices their optimal **ω** is already known and can be used off the shelf. The source code for this computation and the results are attached on github https://github.com/mikias21/SOR-method-matlab/blob/main/sor.m