MAT 423: Homework 6 (10/17)

Ex 1.[3pts] [SOR method]
See the code below.

Ex 2. [4pts]

a) We estimate the specturm of the matrix of $J_\omega = (D-\omega L)^{-1}\Big((1-\omega)D+\omega U\Big)$ for ω varying from 0 to 2 on the figure 1. We observe that the minimum around $\omega \approx 1.6$. 2pts

b) Starting from a random vector $\mathbf{b}$, we apply the SOR method with $\omega = \frac{1}{2}$, $\omega = 1$, $\omega = \frac{3}{2}$ and plot the relative error $\|A\mathbf{x}^(k) - \mathbf{b}\|_2$ for $k = 1 \dots 100$ in figure 2. 2pts

Extra) To estimate the decay rate, we perform a linear regression to estimate:

$$\text{error}(k) \approx c_1 k^\alpha \quad \Rightarrow \quad \log \text{error}(k) \approx \log c_1 + \alpha \cdot k.$$

The regression gives the table 1 The decay rate is then compared to spectrum of the matrix J_ω computed previously. The two values are very close, the relative errors between the two are less than 1%. 1pt

ω	α (numeric)	$\rho(J_\omega)$	relative error
0.5	0.97327	0.97330	$3 \cdot 10^{-5}$
1.0	0.92084	0.92063	$2 \cdot 10^{-4}$
1.5	0.72910	0.72801	$1.5 \cdot 10^{-3}$

Table 1: Decay rates for the SOR method.

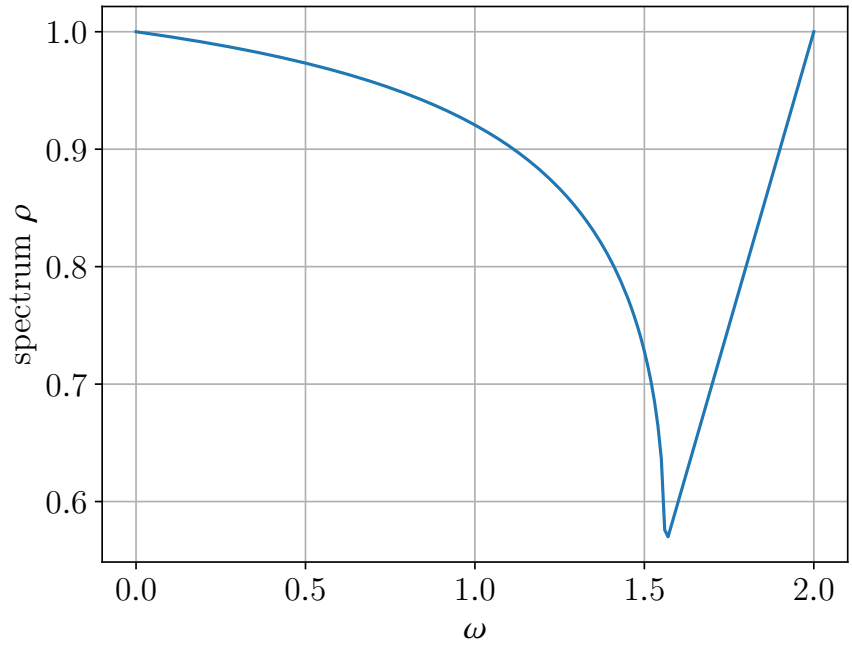

Figure 1: Spectrum of the matrix J_ω for various ω. Notice that the spectrum is always less than 1 and that the minimum is reached around $\omega \approx 1.6$.

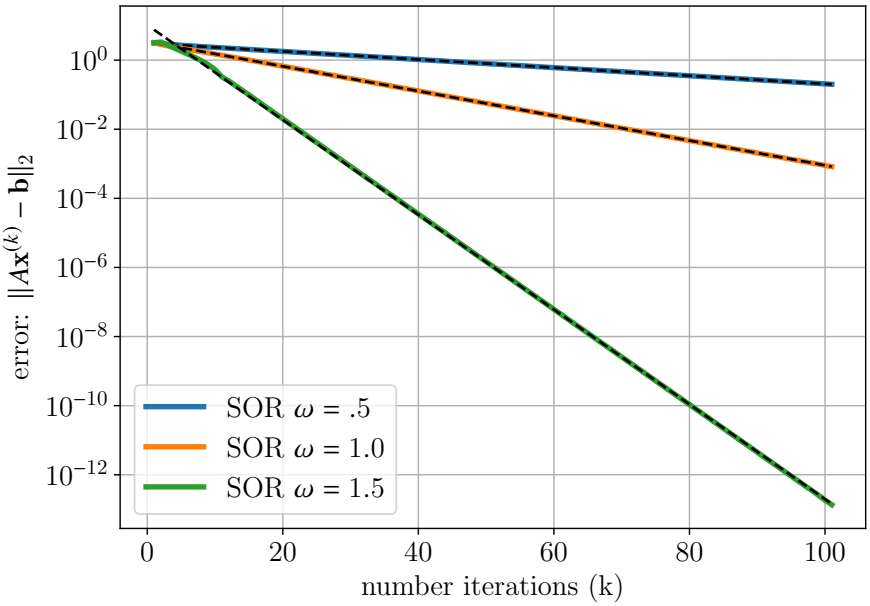

Figure 2: Relative error $\|A\mathbf{x}^{(k)} - \mathbf{b}\|_2$ for the SOR method in semi-log scale using three choices of ω (.5, 1.0, 1.5). The decay rate is faster with $\omega = 1.5$.

Ex 3. [3pts] [Conjugate gradient method]
Consider the matrix A defined in **Ex 2**.

a) See code below. We obtain the following vectors. 2pts

$\mathbf{u}_1$	$\mathbf{u}_2$	$\mathbf{u}_3$	$\mathbf{u}_4$	$\mathbf{u}_5$	$\mathbf{u}_6$	$\mathbf{u}_7$	$\mathbf{u}_8$	$\mathbf{u}_9$	$\mathbf{u}_{10}$
1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\frac{1}{7}$	$\frac{1}{8}$	$\frac{1}{9}$	$\frac{1}{10}$
0	1	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	$\frac{2}{6}$	$\frac{2}{7}$	$\frac{2}{8}$	$\frac{2}{9}$	$\frac{2}{10}$
0	0	1	$\frac{3}{4}$	$\frac{3}{5}$	$\frac{3}{6}$	$\frac{3}{7}$	$\frac{3}{8}$	$\frac{3}{9}$	$\frac{3}{10}$
0	0	0	1	$\frac{4}{5}$	$\frac{4}{6}$	$\frac{4}{7}$	$\frac{4}{8}$	$\frac{4}{9}$	$\frac{4}{10}$
0	0	0	0	1	$\frac{5}{6}$	$\frac{5}{7}$	$\frac{5}{8}$	$\frac{5}{9}$	$\frac{5}{10}$
0	0	0	0	0	1	$\frac{6}{7}$	$\frac{6}{8}$	$\frac{6}{9}$	$\frac{6}{10}$
0	0	0	0	0	0	1	$\frac{7}{8}$	$\frac{7}{9}$	$\frac{7}{10}$
0	0	0	0	0	0	0	1	$\frac{8}{9}$	$\frac{8}{10}$
0	0	0	0	0	0	0	0	1	$\frac{9}{10}$
0	0	0	0	0	0	0	0	0	1

b) Starting from $\mathbf{x}_0 = \mathbf{b} = (1, 0, \ldots, 0)^T$, the conjugate gradient method converges to the exact solution of 10 iterations. The figure 3 shows the decay of the error. 2pts

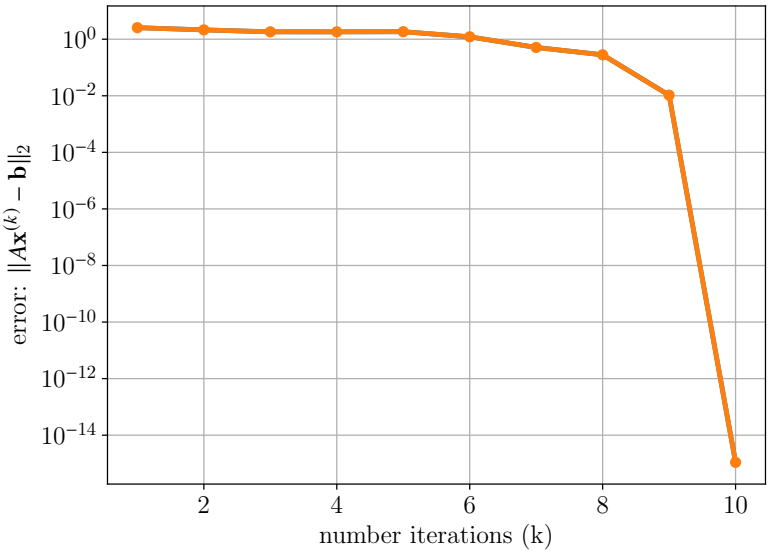

Figure 3: Relative error $\|A\mathbf{x}^{(k)} - \mathbf{b}\|_2$ for the conjugate gradient method in semi-log scale. The method converges after $N = 10$ iterations.

```
function SOR(omega,A,x0,b,nbIter)
    """
      Let A = D-L-U, then
        x_k+1 = (D-omega L)^-1 ((1-omega)D+omega U) x_k + omega(D-omega L)^-1 b
              = (D-omega L)^-1 ( ((1-omega)D+omega U)*x_k + omega b )
    """
    # init
    error = zeros(nbIter+1)
    error[1] = norm(A*x0-b)
    x = copy(x0)
    # matrices
    D = diagm(0=>diag(A))
    L = -(LowerTriangular(A)-D)
    U = -(UpperTriangular(A)-D)
    invMat = inv(D-omega*L)
    directMat = (1-omega)*D + omega*U
    rho = maximum(abs.(eigvals(invMat*directMat)))
    # loop
    for k=1:nbIter
        x .= invMat*( directMat*x + omega*b)
        error[k+1] = norm(A*x-b)
    end
    #println(error)
    return x,error,rho
end
```

```
#=
  Implementation of the conjugrate gradient descent
=#
using LinearAlgebra

# the matrix A
N = 10
D = diagm(0=>2*ones(N))
U = diagm(1=>ones(N-1))
L = diagm(-1=>ones(N-1))
A = D-L-U
# generate the vectors
matrix_vectors = diagm(0=>ones(N)) # canonical basis initially
for i=2:N
    # modify the i-th vector such that
    #      <Au_i,u_j> = for any j=1..(i-1)
    # We use Gram-Schmidt decomposition
    #    u_i = u_i - <Au_i,u_1> u_1/<Au_1,u_1> - ... - <Au_i,u_{i-1}> u_{i-1}/<Au_{i-1}
    u_i = matrix_vectors[:,i]
    for j=1:(i-1)
        u_j = matrix_vectors[:,j]
        u_i .-= (dot(A*u_i,u_j)/(dot(A*u_j,u_j)))*u_j
    end
    matrix_vectors[:,i] .= u_i
end

# implement conjugate gradient method
#     x_k = x_{k-1} - <A x_{k-1} - b, u_k> u}_k/<Au_k,u_k>.
b = zeros(N)
b[1] = 1
x0 = rand(N)
x = copy(x0)
error_seq = zeros(N)
for k=1:N
    u_k = matrix_vectors[:,k]
    x .-= (dot((A*x - b),u_k))*u_k/(dot(A*u_k,u_k))
    error_seq[k] = norm(A*x-b)
end
```