Project 04: Solving Sparse Systems of Linear Equations
Using Successive Overrelaxation (SOR)

1. What are we trying to do?

In this project, **we must write a C++ program** to solve a large system of linear equations using the **Successive Overrelaxation (SOR)** method. Rather than finding the solution by a direct matrix factorization, we start with an initial guess and improve it through repeated numerical updates. We also learn to store a matrix efficiently and determine whether our computed answer satisfies the equations.

2. Why does Poisson's equation appear in this project?

In electrostatics, an electric-charge distribution produces an electric potential V. Gauss's law and $\mathbf{E} = -\nabla V$ lead to

$$\nabla \cdot \mathbf{E} = \frac{\rho_e}{\varepsilon_0}, \qquad \nabla^2 V = -\frac{\rho_e}{\varepsilon_0}.$$

For our laboratory we use a simplified, scaled one-dimensional model:

$$\boxed{\frac{d^2V(x)}{dx^2} = -\rho(x).}$$

The same mathematical idea also appears in Newtonian gravitation and steady heat conduction with sources. **Poisson's equation provides the physical example; SOR is the numerical method we study.**

3. How do we obtain a sparse linear system?

We replace the continuous region by grid points x_i separated by Δ. A central finite difference approximates the second derivative:

$$\left.\frac{d^2V}{dx^2}\right|_{x_i} \approx \frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta^2}.$$

This gives one algebraic equation per unknown potential value. Collecting the equations produces $A\mathbf{v} = \mathbf{b}$. For four unknowns, with $\Delta = 1$, the matrix is

$$A = \begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}, \qquad \mathbf{v} = (V_0, V_1, V_2, V_3)^T.$$

In the actual task we have $n = 500$ unknowns, but the matrix has only $3n - 2 = 1498$ nonzero entries out of $500^2 = 250\,000$. Our boundary convention sets the *external* values $V_{-1} = V_n = 0$; the stored endpoint potentials V_0 and V_{n-1} remain unknown.

Important sign note. The stated physical equation implies $A\mathbf{v} = -\boldsymbol{\rho}$ for the matrix above, whereas the printed laboratory matrix equation uses $A\mathbf{v} = \boldsymbol{\rho}$. We must distinguish these conventions; for the assigned algebraic computation we follow the handout's printed equation unless instructed otherwise.

4. Why do we use CSR storage?

Most entries of A are zero, so we need not store all 250 000 values. In **Compressed Sparse Row (CSR)** format, we keep the nonzero coefficients (`values`), their column indices (`col`), and row starting positions (`row`). This makes sparse matrix-vector multiplication much more economical in memory.

From the SOR Algorithm to Our Laboratory Work

5. How does SOR work?

We start with an approximation $\mathbf{v}^{(0)}$ and repeatedly update each component. SOR uses the newest available values, as in Gauss–Seidel, together with a relaxation parameter ω. A convenient in-place update is

$$\boxed{v_i \leftarrow v_i + \frac{\omega}{a_{ii}}\left(b_i - \sum_j a_{ij} v_j\right).}$$

For $\omega = 1$, this is Gauss–Seidel. Values $1 < \omega < 2$ are overrelaxation and can accelerate convergence for suitable systems. We check each approximation with the **residual**:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{v}^{(k)}, \qquad \|\mathbf{r}^{(k)}\|_2 = \sqrt{\sum_i \left(r_i^{(k)}\right)^2}.$$

A small residual means that our approximation nearly satisfies the *discrete algebraic* equations. We stop when the prescribed tolerance is reached or the maximum number of iterations has been completed.

6. What exactly must we do during the laboratory?

We must develop our own C++ program and complete the following tasks:

1. **Build the matrix:** construct the 500×500 tridiagonal matrix with the handout's coefficients and store its nonzero entries in full CSR format.
2. **Implement basic operations:** write scalar-product, vector-norm and CSR matrix-vector multiplication routines.
3. **Implement SOR:** update the potential vector in place, using the correct diagonal entries and relaxation factor.
4. **Monitor convergence:** after each iteration, record the iteration number, the solution norm $\|\mathbf{v}^{(k)}\|_2$, and residual norm $\|\mathbf{r}^{(k)}\|_2$.
5. **Save and plot results:** create the four required plots using results from our own program.

Dimension	$n = 500$	Grid spacing	$\Delta = 1$
Maximum iterations	10^4	Tolerance	$\varepsilon = 10^{-15}$
Relaxation	$\omega = 1$	Starting values	$v_i^{(0)} = 1$

$$\rho_i = 0.1\left[\sin\left(\frac{2\pi i}{n-1}\right)\right]^{10}, \quad i = 0, \ldots, n-1.$$

Our **four plots** are: (1) solution norm versus iteration; (2) residual norm versus iteration; (3) source ρ_i versus position x_i; and (4) final potential versus x_i. The very strict tolerance may be limited by floating-point precision, so we must report honestly whether it was attained.

7. What must we investigate for homework?

We must repeat the calculations with $\omega = 1,\ 1.5,\ 1.9,\ 1.99$, compare the residual histories, and explain how ω changes convergence. Then, using $\omega = 1.99$ and $\varepsilon = 10^{-8}$, we must compare **two initial guesses**: all zeros and random entries between 0 and 100. We report whether convergence occurs and how many iterations each case requires.

8. What do we learn from the teaching examples?

We use complete, small C++ demonstrations as preparation: **E01**–**E02** finite differences and a tridiagonal matrix; **E03**–**E04** CSR storage and multiplication; **E05** norms and residuals; **E06**–**E07** Jacobi and Gauss–Seidel; **E08** one SOR sweep; and **E09** iteration history and plotting. These examples teach us the individual tools; **we must integrate them into the official $n = 500$ project ourselves.**

Official materials: SOR laboratory assignment · Iterative-methods lecture.