RESEARCH UNIVERSITY
EXCELLENCE INITIATIVE



Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

AGH University of Krakow

Numerical Methods in Physics

Project 04

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

Hamzeh Khanpour

hamzeh.khanpour@fis.agh.edu.pl

Faculty of Physics and Applied Computer Science
AGH University of Krakow

Two questions

How do we efficiently store and solve a large sparse system of linear equations in C++?

How does the Successive Overrelaxation (SOR) method improve iterative convergence, and how can we verify the accuracy of the numerical solution?

Scientific workflow

Physical model $\longrightarrow$ discretize $\longrightarrow$ sparse matrix (CSR) $\longrightarrow$ solve using SOR $\longrightarrow$ verify $\longrightarrow$ interpret.

Course sources

S01: Tomasz Chwiej, Numerical Methods in Physics course page.
S02: Laboratory handout: Solving sparse system of linear equations using SOR method.
S03: Lecture: Iterative methods for solving systems of linear equations.

Technical sources

Netlib: Templates for the Solution of Linear Systems – iterative methods and sparse numerical linear algebra.
Sparse matrix storage: Compressed Sparse Row (CSR) format.
Numerical method: Successive Overrelaxation (SOR), based on Gauss–Seidel iteration

A common equation in many branches of physics

A **field** assigns a quantity to every position in space. A **source** produces or influences that field. In many problems, their relationship is described by **Poisson's equation**:

$$\boxed{\nabla^2 u(\mathbf{r}) = s(\mathbf{r})}, \qquad \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}.$$

Here u is the unknown scalar field and s is the prescribed source. The Laplacian $\nabla^2 u$ describes its spatial curvature.

Familiar physical applications

- **Electrostatics:** electric charge produces an electric potential V.
- **Gravitation:** mass produces a gravitational potential Φ.
- **Heat conduction:** internal heating shapes a steady temperature field T.

The question that links these applications to our laboratory

Given a source distribution and suitable boundary conditions, **how can a computer calculate the unknown field?**

Start from Maxwell's equations

In electrostatics, Gauss's law relates the electric field $\mathbf{E}$ to the volume charge density ρ_e:

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

Substitute the potential into Gauss's law:

$$-\nabla \cdot (\nabla V) = \frac{\rho_e}{\varepsilon_0} \quad \Longrightarrow \quad \boxed{\nabla^2 V = -\frac{\rho_e}{\varepsilon_0}}.$$

Reduce to the one-dimensional model of the laboratory

For a configuration varying only along x (for example, an idealized planar geometry),

$$\frac{d^2V(x)}{dx^2} = -\frac{\rho_e(x)}{\varepsilon_0}.$$

With $\rho(x) = \rho_e(x)/\varepsilon_0$, the handout uses $\boxed{V''(x) = -\rho(x)}$. The field follows from $E_x = -dV/dx$ once V and the boundary conditions are known.

Newtonian gravitation

For $\mathbf{g} = -\nabla\Phi$ and $\nabla \cdot \mathbf{g} = -4\pi G\rho_{\mathrm{m}}$,

$$\boxed{\nabla^2\Phi = 4\pi G\rho_{\mathrm{m}}}.$$

Mass density ρ_{m} is the source. Notice the different sign.

Stationary heat conduction with internal heating

For constant conductivity k, Fourier's law is $\mathbf{j}_q = -k\nabla T$. Steady heating Q implies $\nabla \cdot \mathbf{j}_q = Q$, so

$$\boxed{\nabla^2 T = -\frac{Q}{k}}.$$

The unknown is temperature T, not electric potential.

Why this matters for numerical methods

Different physical fields, but the **same Laplacian operator**. We can discretize each problem and solve a sparse linear system.

AGH

Step 1: replace a continuous field by values on a grid

For n positions $x_i = x_{\min} + i\Delta$, the unknowns are $V_i \approx V(x_i)$. In one dimension our physical equation is

$$V''(x) = -\rho(x).$$

Step 2: replace derivatives by a three-point stencil

$$V''(x_i) \;\approx\; \frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta^2} \;=\; -\rho_i.$$

Each row involves only V_{i-1}, V_i, and V_{i+1}: the matrix is **sparse and tridiagonal**. For the physical equation, $A_{-}\mathbf{v} = -\boldsymbol{\rho}$ with diagonal $-2/\Delta^2$ and off-diagonals $+1/\Delta^2$.

Step 3: store, solve, and verify

Physical model $\longrightarrow$ finite differences $\longrightarrow$ $A\mathbf{v} = \mathbf{b}$

$\longrightarrow$ **CSR sparse storage** $\longrightarrow$ **SOR iterations** $\longrightarrow$ residual $\|\mathbf{b} - A\mathbf{v}\|_2$.

AGH

Complete reference examples

- E01–E05: derivative, stencil, CSR and residuals.
- E06–E08: one Jacobi, Gauss–Seidel, and SOR sweep.
- E09: an independent 3×3 convergence CSV and plot.

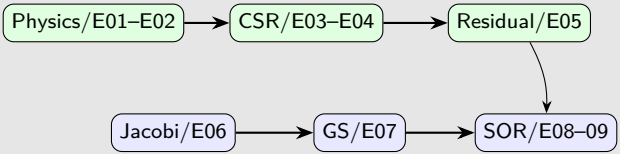
Your assessed project

- Construct the assigned $n = 500$ full-CSR system.
- Integrate and debug your own SOR solver.
- Prepare the four required figures and the homework comparisons.

Student ownership

The examples teach the building blocks. No distributed example solves the assigned full system or produces its completed report.

Learn in the same order as the program is built



Optional later

E10 explores a different two-dimensional stencil. It is not needed for the assigned one-dimensional SOR implementation.

Illustrative 135-minute learning route

0–10 min	Orientation, files and project contract
10–25	Physical model and derivative
25–40	Sparse CSR, row products and residuals
40–55	Jacobi $\rightarrow$ GS $\rightarrow$ SOR
55–120	**Students independently implement and test**
120–135	Debugging, plots and homework guidance

Build only the examples you need

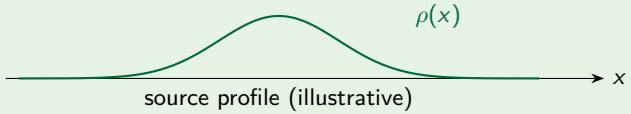
`make help` `make standard` `make check-standard`

The 56-slide file is a detailed reference, not 56 slides that must all be lectured live.

Think first about the physics

In electrostatics a potential describes how a source distribution influences space. Here $V(x)$ is a scalar function and $\rho(x)$ prescribes the source. We use the handout's scaled mathematical form; physical prefactors and units are not specified.

A small sketch



source profile (illustrative)

The numerical question

Given a source and boundary conditions, compute the potential at a finite set of locations.

AGH

Continuous field equation

$$\nabla^2 V(\mathbf{r}) = -\rho(\mathbf{r}), \qquad \nabla^2 = \partial_x^2 + \partial_y^2 + \partial_z^2.$$

If the problem depends only on x, the Laplacian reduces to a second derivative:

The model in this laboratory

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

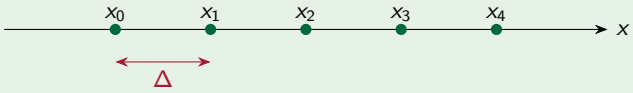
Curvature of the unknown potential is determined by the prescribed source. We must supply two boundary conditions to obtain a unique discrete solution.

Indices and spacing

$$\Delta = \frac{x_{\max} - x_{\min}}{n-1}, \qquad x_i = x_{\min} + i\Delta, \quad i = 0, \ldots, n-1.$$

We store n unknown values $V_i \approx V(x_i)$. The n stored nodes cover $n-1$ intervals.

Five-node example



AGH

Expand at the center x_i

$$f(x_i + \Delta) = f_i + \Delta f_i\prime + \tfrac{\Delta^2}{2} f_i\prime\prime + \tfrac{\Delta^3}{6} f_i^{(3)} + \cdots ,$$

$$f(x_i - \Delta) = f_i - \Delta f_i\prime + \tfrac{\Delta^2}{2} f_i\prime\prime - \tfrac{\Delta^3}{6} f_i^{(3)} + \cdots .$$

Add both rows: the odd powers cancel, leaving the curvature.

Central second difference (E01)

$$f\prime\prime(x_i) = \frac{f_{i-1} - 2f_i + f_{i+1}}{\Delta^2} + O(\Delta^2).$$

On a smooth function, halving the spacing reduces leading truncation error by approximately four.

Calculate by hand before running

$$f(x) = x^4, \quad f''(1) = 12, \quad D_2 f(1; h) = 12 + 2h^2.$$

h	finite difference	absolute error
0.5	12.5	0.5
0.25	12.125	0.125

Run E01 and compare

```
make bin/01_second_difference_error
./bin/01_second_difference_error
```

Expected: `ratio=4`, then `PASS E01`.

Connection to your project

The same centered difference gives the coefficients of the sparse Poisson matrix.

AGH

Substitute finite differences into Poisson

$$\frac{V_{i-1} - 2V_i + V_{i+1}}{\Delta^2} = -\rho_i.$$

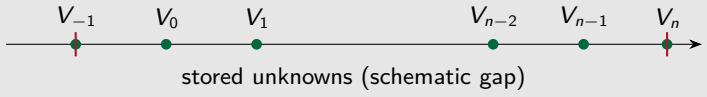
So the algebraic row has a left neighbor coefficient $+1/\Delta^2$, a center coefficient $-2/\Delta^2$, and a right neighbor coefficient $+1/\Delta^2$.

Important difference in the printed assignment

Its differential equation implies $A_{-}\mathbf{v} = -\boldsymbol{\rho}$, but the printed matrix task uses $A_{-}\mathbf{v} = \boldsymbol{\rho}$. The following project instructions retain the **printed algebraic track**; the physical sign-consistent alternative must be identified separately.

Exactly the assigned ghost-node convention

$V_{-1} = 0, \qquad [V_0, V_1, \ldots, V_{n-1}] \text{ unknown}, \qquad V_n = 0.$



Do not change the endpoints by accident

V_0 and V_{n-1} remain unknown. Setting them to zero instead would create a different matrix problem.

AGH

The stencil determines every nonzero entry

$$A_4 = \begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}, \qquad q = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}.$$

First row: $-2q_0 + q_1 = 0$; middle rows: $q_{i-1} - 2q_i + q_{i+1} = 0$; final row: $q_2 - 2q_3 = -5$.

Predict / compile / verify

$$A_4 q = (0, 0, 0, -5)^T.$$

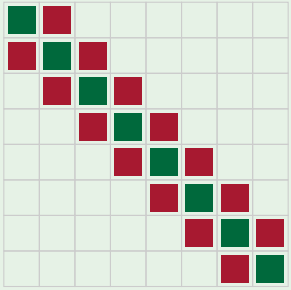
```
make bin/02_tridiagonal_toy_system
./bin/02_tridiagonal_toy_system
```

Expected: four printed rows and `PASS E02`.

Most coefficients are zeros

Dense storage allocates all n^2 elements, even though a three-point stencil couples each node only to itself and its immediate neighbors.

Nonzero pattern of a tridiagonal operator



Only the central band is stored.

AGH

Three kinds of entries

$$\underbrace{n}_{\text{diagonal}} + \underbrace{(n-1)}_{\text{below}} + \underbrace{(n-1)}_{\text{above}} = 3n - 2.$$

The first and last rows have two entries each; all $n-2$ interior rows have three.

Check the arithmetic

$$n = 4: \quad \mathrm{nnz} = 10, \qquad n = 500: \quad \mathrm{nnz} = 1498.$$

The official pseudocode uses $3n$ as a convenient *capacity*, not as the actual number of stored nonzeros.

AGH

Dense double-precision matrix

$$n^2 \cdot 8 = 500^2 \cdot 8 = 2{,}000{,}000 \text{ bytes}.$$

A general dense double matrix stores 250,000 elements.

Full CSR base arrays

$$1498 \cdot 8 + 1498 \cdot 4 + 501 \cdot 4 = 19{,}980 \text{ bytes}.$$

This assumes 64-bit doubles, 32-bit indices and 0-based row pointers; excludes vector storage, the separate diagonal array (4000 more bytes), allocator overhead and padding.

Why this matters

Sparse storage reduces memory and makes an iteration proportional to the number of nonzero entries.

The compact data model

- `values[p]` = nonzero coefficient in row-major order.
- `col[p]` = 0-based column index of that coefficient.
- `row[i]` = first stored index for row i.
- `row[i+1]` = one past the last index for that row.

Independent invariants

$$\texttt{row[0]} = 0, \quad \texttt{row[n]} = \mathrm{nnz}, \quad \texttt{row.size()} = n + 1.$$

An optional separate diagonal array d_i is used for the official SOR update.

AGH

For the four-by-four stencil matrix

$$A_4 = \begin{pmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

$$\texttt{values} = [-2, 1, 1, -2, 1, 1, -2, 1, 1, -2],$$
$$\texttt{col} = [0, 1, 0, 1, 2, 1, 2, 3, 2, 3],$$
$$\texttt{row} = [0, 2, 5, 8, 10].$$

E03

Compile `03_csr_toy_encoding.cpp`; its self-check reconstructs every matrix row before printing `PASS E03`.

AGH

Row zero: positions in [0,2)

`p=0: col=0, value=-2;  p=1: col=1, value=1.`

Row one: positions in [2,5)

p	`col[p]`	`values[p]`	meaning
2	0	1	left neighbor
3	1	-2	diagonal
4	2	1	right neighbor

End-exclusive indexing means the loop uses $p <$ `row[i+1]`.

Predict before coding

Which three columns must row $i = 2$ contain? Check against E03.

AGH

Check construction before the solver

- Pointers are nondecreasing: $0 = \texttt{row[0]} \leq \cdots \leq \texttt{row[n]} = \mathrm{nnz}$.
- Each column index satisfies $0 \leq \texttt{col[p]} < n$.
- Every row has exactly one finite, nonzero diagonal coefficient for this problem.
- Full symmetric CSR stores both $A_{i,i+1}$ and $A_{i+1,i}$.

Debugging test

For $n = 4$, `row.back()==10`; if it is 9 or 11, the final row or capacity accounting is wrong.



E02: explicit dense A_4

Four rows of four entries: easy to draw and multiply by hand; stores six zeros unnecessarily.

E03: equivalent full CSR

Only ten coefficients plus ten column indices and five pointers. Each row spans `[row[i],row[i+1])`.

Your task is the generalization

In the assessed project, construct and validate the $n = 500$ matrix yourself. The examples do not contain the full official assembly driver.

AGH

What the program must print

```
values:  -2 1 1 -2 1 1 -2 1 1 -2
col:  0 1 0 1 2 1 2 3 2 3
row:  0 2 5 8 10
row 1 interval=[2,5)
PASS E03
```

How the verification works

The test does not check only the text `PASS`: it compares all three arrays, their lengths and the reconstructed dense rows with independently specified coefficients.

What to fix if it fails

Inspect row pointer offsets and whether the last nonzero was added before assigning `row[n]`.

AGH

A row is a sum over stored entries

$$y_i = \sum_{p=\texttt{row}[i]}^{\texttt{row}[i+1]-1} \texttt{values}[p]\, x_{\texttt{col}[p]}.$$

If the row has only two nonzero coefficients, the loop performs just two products. There is no need to visit zero entries.

The crucial indexing rule

The index p identifies a stored coefficient; $j = \texttt{col[p]}$ identifies the vector component being multiplied. The work scales as $O(\mathrm{nnz})$.

AGH

Take $q = (1, 2, 3, 4)^T$

$$
\begin{aligned}
y_0 &= -2(1) + 1(2) = 0, \\
y_1 &= 1(1) - 2(2) + 1(3) = 0, \\
y_2 &= 1(2) - 2(3) + 1(4) = 0, \\
y_3 &= 1(3) - 2(4) = -5.
\end{aligned}
$$

Reference vector

$$A_4 q = (0, 0, 0, -5)^T.$$

E04 computes the same four values from CSR arrays and separately checks a dense multiplication.

AGH

The smallest useful algorithmic primitive

```
for (std::size_t i=0; i<x.size(); ++i) {
  for (int p=row[i]; p<row[i+1]; ++p) {
    y[i] += a[p] * x[col[p]];
  }
}
```

Compile and verify

`make bin/04_csr_toy_matvec`

Expected: `y0=0,y1=0,y2=0,y3=-5`, all matching independently computed dense products. The example also rejects malformed CSR pointers.

Student project connection

Reuse the idea, but write and integrate your own validated CSR multiplication routine.

AGH

Three-component exact answer fixture

$$B = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}, \quad x_\star = (1, 2, 3)^T, \quad b = (2, 4, 10)^T.$$

$$x_\star \cdot b = 1(2) + 2(4) + 3(10) = 40, \quad \|x_\star\|_2 = \sqrt{1 + 4 + 9} = \sqrt{14}.$$

What a norm measures

$\|v\|_2 = \sqrt{v \cdot v}$. Program E05 verifies both operations and $\|b\|_2 = \sqrt{120}$, with floating-point tolerance and dimension checks.

AGH

Definition

$$r(x) = b - Bx, \qquad \|r\|_2 = \sqrt{\sum_{i=0}^{2} r_i^2}.$$

At $x^{(0)} = (0,0,0)^T$, $r = b$, hence $\|r\|_2 = \sqrt{120}$.

A known solution has zero residual

$$Bx_\star = b \quad \Longrightarrow \quad r(x_\star) = (0,0,0)^T.$$

Run `05_dot_norm_residual.cpp`. Students should verify a solver by an independently calculated residual, not by a plausible-looking plot.

AGH

Remember the earlier LU laboratory

A direct factorization is a powerful solution strategy. For very large sparse problems, storing and processing fill-in during factorization can be expensive; sparse direct methods still exist and can be excellent.

The iterative alternative

Start with a guess $x^{(0)}$ and repeatedly improve it using the matrix rows. A sweep touches only nonzeros, making CSR an effective companion to iterations.

A limitation worth knowing

Convergence and speed depend on the matrix and method; iterative is not automatically better than sparse LU for every system.

AGH

What the superscript means

$$x^{(0)} \longrightarrow x^{(1)} \longrightarrow x^{(2)} \longrightarrow \cdots \longrightarrow x_\star.$$

A superscript (k) labels the iteration number, not a power. In this lab a complete pass through all unknowns is one *sweep*.

An objective convergence measure

$$r^{(k)} = b - Ax^{(k)}, \qquad R_k = \|r^{(k)}\|_2.$$

If R_k becomes small, the computed vector satisfies the algebraic equations closely. Convergence is not guaranteed for arbitrary matrices.

AGH

Rearrange the row

$$a_{ii}x_i + \sum_{j \neq i} a_{ij}x_j = b_i$$

Provided $a_{ii} \neq 0$, isolate x_i using only the previous sweep:

Jacobi: old values everywhere

$$\boxed{x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij}x_j^{(k)}}{a_{ii}}}.$$

Separate old and new vectors prevent a partly updated row from changing the algorithm.

AGH

Use the independent 3*imes*3 matrix

$$B = \begin{pmatrix} 4 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix}, \quad b = (2, 4, 10)^T, \quad x^{(0)} = 0.$$

All neighbor terms are zero initially:

Three simultaneous updates

$$x_0^{(1)} = 2/4 = 0.5,$$
$$x_1^{(1)} = 4/4 = 1,$$
$$x_2^{(1)} = 10/4 = 2.5.$$

`06_jacobi_one_sweep.cpp` must return $(0.5, 1, 2.5)^T$.

AGH

Read old, write new

```
for (int i=0;i<3;++i) {
  double sum=0;
  for (int j=0;j<3;++j)
    if (j!=i) sum += B[i][j]*old[j];
  next[i]=(b[i]-sum)/B[i][i];
}
```

Run and check

`make bin/06_jacobi_one_sweep`
Expected: 0.5, 1, 2.5, then `PASS E06`.

A common error

If you overwrite `old[i]` inside this loop, the result is no longer Jacobi.

Split lower and upper contributions

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)}}{a_{ii}}.$$

The indices below i have already been updated; those above i still contain the previous approximation.

Matrix form

$$(D + L)x^{(k+1)} = b - Ux^{(k)}, \qquad A = D + L + U.$$

This is a sequential sweep: the order of updates matters.

AGH

Same test problem, now updated values are reused

$$x_0^{(1)} = 2/4 = 0.5,$$
$$x_1^{(1)} = (4 + x_0^{(1)})/4 = 4.5/4 = 1.125,$$
$$x_2^{(1)} = (10 + x_1^{(1)})/4 = 11.125/4 = 2.78125.$$

One-sweep reference

$$x_{\mathrm{GS}}^{(1)} = (0.5,\ 1.125,\ 2.78125)^T.$$

Contrast this with Jacobi's $(0.5, 1, 2.5)^T$.

AGH

Same storage, different read timing

```
for (int i=0;i<3;++i) {
  double s=0;
  for (int j=0;j<3;++j)
    if (j!=i) s+=B[i][j]*x[j];
  x[i]=(b[i]-s)/B[i][i];
}
```

The expected check

`07_gauss_seidel_one_sweep.cpp` prints 0.5, 1.125, 2.78125 and `PASS E07`.

Watch the loop order

At $i = 1$ the array already contains the new x_0. At $i = 2$, it also contains the new x_1.

AGH

A controlled step toward the GS candidate

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega x_i^{\mathrm{GS}}.$$

If $\omega < 1$, the correction is smaller; if $\omega > 1$, we step farther than the GS proposal.

Three regimes

$0 < \omega < 1$	underrelaxation	cautious step
$\omega = 1$	Gauss–Seidel	ordinary step
$1 < \omega < 2$	overrelaxation	larger step

Do not overgeneralize

A larger ω is not automatically faster; behavior depends on the matrix and the initial guess.

AGH

Start from GS but scale the correction

$$Dx^{(k+1)} = (1-\omega)Dx^{(k)} + \omega\big(b - Lx^{(k+1)} - Ux^{(k)}\big).$$

Move the lower-triangle term to the left; collect the old-vector terms:

The standard SOR equation

$$\boxed{(D+\omega L)x^{(k+1)} = \big[(1-\omega)D - \omega U\big]x^{(k)} + \omega b.}$$

At $\omega = 1$ this to Gauss–Seidel.

AGH

First calculate the Gauss–Seidel candidate

$$\widehat{x}_i = \frac{b_i - \sum_{j<i} a_{ij} x_j^{\text{new}} - \sum_{j>i} a_{ij} x_j^{\text{old}}}{a_{ii}}.$$

Apply the relaxation blend to the current (old) x_i.

Equivalent in-place row-residual formula

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

The row sum includes the diagonal contribution computed *before* replacing x_i. This is the form used in the assignment.

Only a single row sweep; no official $n = 500$ driver

```
for each row i in ascending order:
  c = 0
  for p from row[i] to row[i+1]-1:
    j = col[p]
    c += values[p] * x[j]
  x[i] += (omega/d[i]) * (b[i]-c)
```

Relate names to the handout

`values` corresponds to `a`; `col`, `row`, and `d` match the official pseudocode on assignment pp. 2–3. This sweep is one piece of the students'independent integration.

Before programming

Check all diagonal entries, indices and boundary rows; do not confuse j (column) with p (CSR storage index).

AGH

Independent SPD toy, $\omega = 1.25$, starting at zero

$$x_0 = 0 + 1.25(2/4) = 0.625,$$
$$x_1 = 0 + \tfrac{1.25}{4}(4 + x_0) = 1.4453125,$$
$$x_2 = 0 + \tfrac{1.25}{4}(10 + x_1) = 3.57666015625.$$

Two checks in one complete toy program

`08_sor_one_sweep.cpp` verifies these values and independently confirms that $\omega = 1$ reproduces E07's GS sweep. No official CSR/SOR project is assembled.

AGH

The stopping variable is not the same as the update

$$r^{(k)} = b - Bx^{(k)}, \qquad R_k = \sqrt{\sum_i (r_i^{(k)})^2}.$$

For the same B toy, one SOR sweep at $\omega = 1.25$ gives the explicitly checkable vector $x = (0.625, 1.4453125, 3.57666015625)^T$.

Why track a sequence rather than one number?

The residual may not fall on every individual sweep for every method/parameter. Scientific conclusions should use the actual history $R_0, R_1, \ldots$ and a stated stopping rule.

You still need to calculate

Do not infer convergence solely from the distance between two successive iterates.

A method is only part of the story

- **Matrix structure and spectrum**: affects whether and how rapidly errors shrink.
- **Relaxation** ω: modifies correction size.
- **Initial vector**: influences early residuals and required sweeps.
- **Ordering and floating-point arithmetic**: influence finite-precision trajectories.

A useful sufficient theorem

For a real symmetric *positive definite* matrix, classical SOR converges for $0 < \omega < 2$ under the standard assumptions. The assignment's negative-definite matrix is equivalent after multiplying *both* A and b by -1, leaving row updates unchanged.

Four values to investigate in the homework

$$\omega \in \{1,\ 1.5,\ 1.9,\ 1.99\}.$$

$\omega = 1$ is GS. Increasing ω into overrelaxation can accelerate some elliptic problems but can also increase oscillations or slow the method if poorly chosen.

Ask of your own convergence curves

For each run, compare R_k at the same k, final stopping status, number of iterations and computational cost. Plot all four histories on one readable graph for the report.

Keep assessed results independent

We deliberately do not provide the completed $n = 500$ comparison curves.

The assignment gives two termination mechanisms

$$\boxed{R_k \leq \varepsilon \quad \text{or} \quad k = \texttt{itmax}}.$$

The specified tolerance is the *absolute* residual $\varepsilon = 10^{-15}$ and maximum count is 10^4.

Floating-point reality

Cancellation and finite precision can create a residual floor above 10^{-15}. At `itmax`, report whether tolerance was actually reached; do not label every iteration-limit stop as a numerical failure.

Plotting

A logarithmic vertical axis requires strictly positive points. Preserve raw zero residuals in the CSV; omit or clearly annotate them on a semilog figure.

Different tiny system: the 3*imes*3 dense B toy

Use $x^{(0)} = (0, 0, 0)^T$, $\omega = 1.1$; E09 writes one starting row plus a row after each completed sweep:

CSV format

```
iteration,solution_norm,residual_norm
0,0,10.954451150103322
1,...,...
...
```

The exact remaining lines are produced by the compiled program, never invented. The output is deterministic and checked against $x_\star = (1, 2, 3)^T$.

Safe files

E09 rejects overwriting existing outputs unless `-overwrite` is explicitly provided.

AGH

In-laboratory work (handout, Section 2)

- Build the $n = 500$ negative-diagonal tridiagonal matrix in *full* CSR.
- Implement dot product, CSR matvec and the in-place SOR method.
- Record iteration and solution/residual norms; generate four requested figures.

At-home scientific report

Compare specified relaxation factors and the effect of different initial guesses. Answer questions from the PDF using your own computations.

Authoritative specification

Official SOR laboratory assignment. Exact parameters and conventions appear on S48–S53.

Use these inputs as printed

dimension	$n = 500$
grid interval	$\Delta = 1$
iteration cap	$\texttt{itmax} = 10^4$
absolute tolerance	$\varepsilon = 10^{-15}$
initial relaxation	$\omega = 1$
initial iterate	$v_i^{(0)} = 1$

Prescribed source (for each $i = 0, \ldots, n-1$)

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

Project algebra uses $A_v = \rho$ with $a_{ii} = -2/\Delta^2$, $a_{i,i\pm1} = +1/\Delta^2$ *as printed.*

Make sure each primitive is separately correct

- Matrix: both off-diagonals and every diagonal, stored row by row.
- `values` / `col`: exactly $3n - 2$ used entries.
- `row`: length $n + 1$, starts at zero, ends at nnz.
- `d`: accessible finite nonzero diagonal elements.
- Dot products and CSR matvec must pass toy-sized checks.

Student ownership

Write and integrate the project-sized builder yourself; draw on E02, E03, E04 and E05 for patterns and small independently verified results.

AGH

At each completed sweep

$$x \xrightarrow{\text{SOR}} x^{(k)}, \quad r^{(k)} = \rho - A_{-}x^{(k)}.$$

Write three columns to the iteration file: k, $\|x^{(k)}\|_2$, $\|r^{(k)}\|_2$.

Essential run-status information

Record whether $\|r\|_2 \leq 10^{-15}$ or the maximum count 10^4 stopped the iteration. Check NaNs, infinity and unexpected zero diagonal entries.

Do not mix iteration counters

If you additionally record $k = 0$, label it as the *initial* (pre-sweep) state. The handout asks for per-iteration output.

Two convergence diagnostics

1. $\|x^{(k)}\|_2$ versus completed iteration k.
2. $\|r^{(k)}\|_2$ versus k (typically log vertical axis).

Two spatial profiles

3. ρ_i versus $x_i = i\Delta$.
4. Final numerical solution x_i versus $x_i = i\Delta$.

Axes must contain quantities, indices/units if known, and a clear legend where needed.

Assessed work

Create these plots from *your own* $n = 500$ output, not from the E09 three-unknown toy.

AGH

Repeat the assigned system

$$\omega \in \{1,\ 1.5,\ 1.9,\ 1.99\}.$$

The initial $\omega = 1$ result is supplemented by three overrelaxation experiments.

One combined convergence plot

Plot $\|r^{(k)}\|_2$ against iteration on a common graph for all four values. Compare curves on common axes and explicitly record each run's status and iteration count.

Write an interpretation, not only a figure

Is SOR effective for this system? Does larger ω always improve efficiency? Use measured data, not an unsupported general rule.

Modify only the stated settings

$$\omega = 1.99, \quad \varepsilon = 10^{-8}.$$

Run once with all $v_i^{(0)} = 0$ and once with randomly generated $v_i^{(0)} \in [0, 100]$. The handout gives $100 \cdot \texttt{rand()}/\texttt{RAND_MAX}$.

Questions to answer

Did each run reach tolerance? How many iterations did each require? Does an initial guess change the final numerical solution or primarily the path to convergence?

Reproducibility

Document the random generator and seed. Counts from different random samples are not necessarily identical.

AGH

Finish one independently testable step at a time

1. Confirm grid, signs, ghost values and endpoint rows.
2. Construct full CSR and assert pointer invariants.
3. Check dot product and CSR matvec on a tiny matrix.
4. Verify one row/sweep with hand calculations.
5. Add the outer loop and absolute residual stop.
6. Write data, then independently plot and interpret.

Questions at each checkpoint

Can you predict one exact row product? Does `row[n]` equal $3n - 2$? Does $\omega = 1$ agree with a GS sweep? How will your program report `itmax` termination?

Before trusting a final result

- Wrong sign in $b - Ax$ or silently changing the printed physics/matrix convention.
- Endpoint unknowns set to zero instead of using external ghost values.
- Symmetric half-CSR used where the assignment requires both triangles.
- Off-by-one in row pointers, including missing final `row[n]`.
- Row sum formed after updating its own x_i; wrong diagonal or ω factor.
- Logging $\log 0$, ignoring NaNs, or reporting `itmax` as successful convergence.

Where to revisit

E02–E04: rows and CSR. E05: residuals. E06–E08: update timing. E09: safe CSV and plotting.

AGH

The two course references

Official SOR laboratory assignment

Iterative-methods lecture

The official handout defines the assessed tasks; these notes provide a guided mathematical introduction.

Optional E10: a different 2D stencil

A 2×2 interior grid with a five-point operator extends the sparse-stencil idea. It uses a separate matrix and boundary setup and *does not* solve the SOR assignment.

Now build your own solver

Use E01–E09 as verified independent examples, then implement and interpret the $n = 500$ CSR–SOR laboratory yourself.


THANK
YOU!