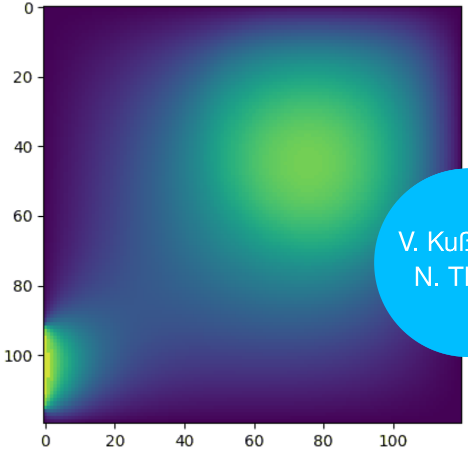
Universität Stuttgart
Department of Mathematics



Project 2: Time dependent heat equation with a source

Scientific Computing

V. Kußmaul,
N. Thorin

Introduction

We solve the **time dependent heat equation**

$$\begin{cases} \partial_t u - \Delta u = f, & \text{in} \quad (0,1)^2, \\ u = g, & \text{on } \partial(0,1)^2. \end{cases}$$

with a source f and boundary condition g.

Discretization

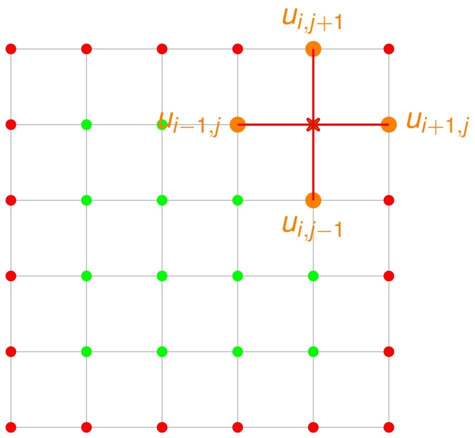
We use finite differences for the Laplacian

$$(\Delta u)(x) \approx \big(u(x_1 - \delta x, x_2) + u(x_1 + \delta x, x_2) + u(x_1, x_2 - \delta x) + u(x_1, x_2 + \delta x) - 4u(x_1, x_2)\big)/\delta x^2$$

and the time derivative

$$\frac{u(x, t + \delta t) - u(x, t)}{\delta t} - (\Delta u)(x, t + \delta t) \approx f(x, t + \delta t)$$

$$\iff u(x, t + \delta t) - \delta t(\Delta u)(x, t + \delta t) \approx \delta t\, f(x, t + \delta t) + u(x, t)$$



Let $u_{ij}^{(\ell)} = u(x_{ij}, t_\ell)$, $0 \leq i, j < N$. Then

$$u_{ij}^{(\ell+1)} - \frac{\delta t}{\delta x^2}(Lu)_{ij}^{(\ell+1)} = \delta t\, f_{ij}^{(\ell+1)} + u_{ij}^{(\ell)}$$

where

$$(Lu)_{ij} = u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{ij}$$

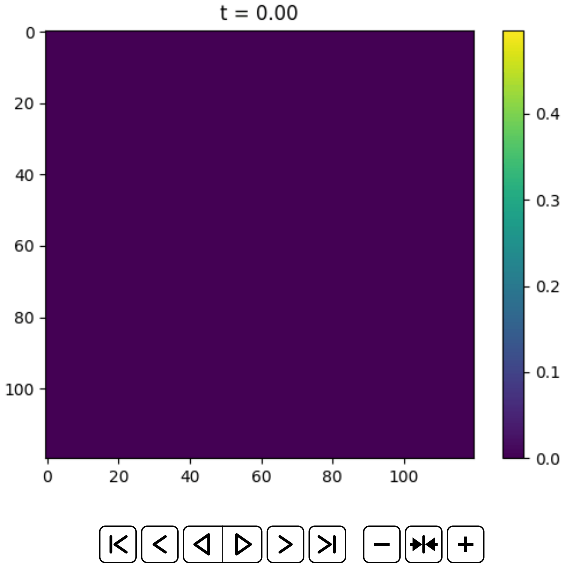
and $u_{ij}^{(\ell)} = g_{ij}^{(\ell)}$ in boundary points.

Implementation details

```
def sparse_laplacian(self) -> csr_matrix:
    def in_bounds(i, j):
        return (0 <= i < self.N) and (0 <= j < self.N)

    L = lil_matrix((self.N**2, self.N**2))
    boundary_points = defaultdict(list)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for i in range(self.N):
        for j in range(self.N):
            L[self.n(i, j), self.n(i, j)] = -4
            for dx, dy in directions:
                i_, j_ = i + dx, j + dy
                if in_bounds(i_, j_):
                    L[self.n(i, j), self.n(i_, j_)] = 1
                else:
                    boundary_points[self.n(i, j)].append(((i_+1)*self.dx, (j_+1)*self.dx))
    self.boundary_points = boundary_points
    return (1/self.dx)**2 * csr_matrix(L)
```

CG Method

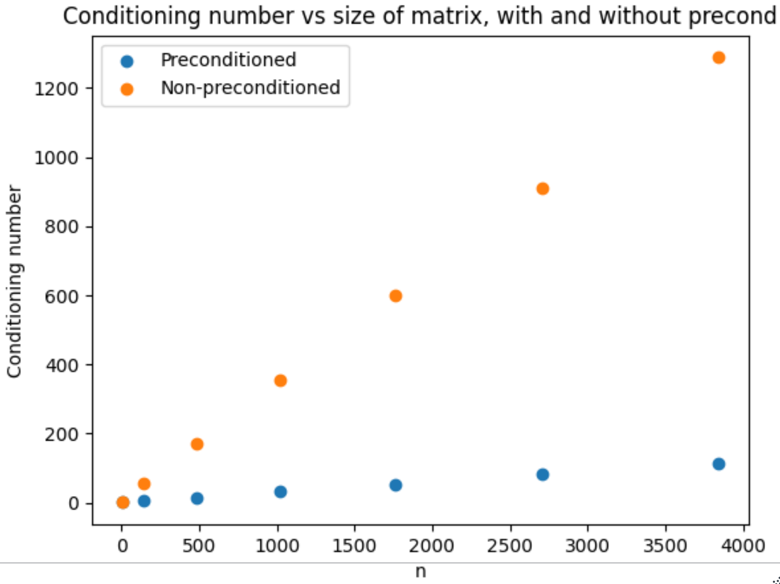
```
def unpreconditioned_solve(self, b: np.ndarray, initial_guess: np.ndarray):
 x = initial_guess
    r = p = b - self.A@x
    alpha = np.dot(r, r)

    for _ in range(self.max_iterations):
        v = self.A@p
        _lambda = alpha / np.dot(v, p)
        x = x + _lambda * p
        r = r - _lambda * v
        alpha_new = np.dot(r, r)
        p = r + (alpha_new / alpha) * p
        alpha = alpha_new

        if alpha < self.tol**2:
            return x

    return None
```


t = 0.00
0
20
40
60
80
100
0
20
40
60
80
100
0.0
0.1
0.2
0.3
0.4

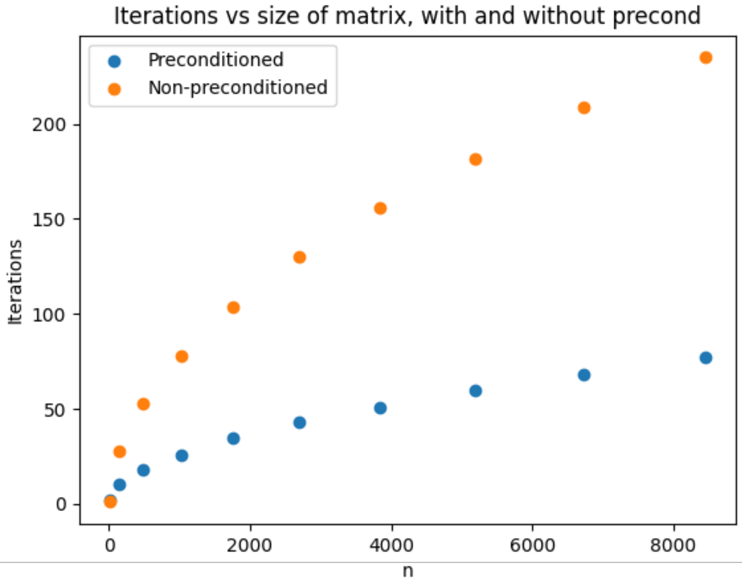
Preconditioning

- A is spd, we are doing CG $\rightsquigarrow$ Incomplete Cholesky as preconditioner.
- We use IC(0) $\rightsquigarrow$ Sparsity pattern is conserved.
- $A = L^T L \rightsquigarrow A \approx \tilde{L}^T \tilde{L}$
- Improvements: Smaller conditioning number $\rightsquigarrow$ Fewer iterations needed
- Downsides: Solve two sparse triangular systems $\rightsquigarrow$ More computing time per iteration

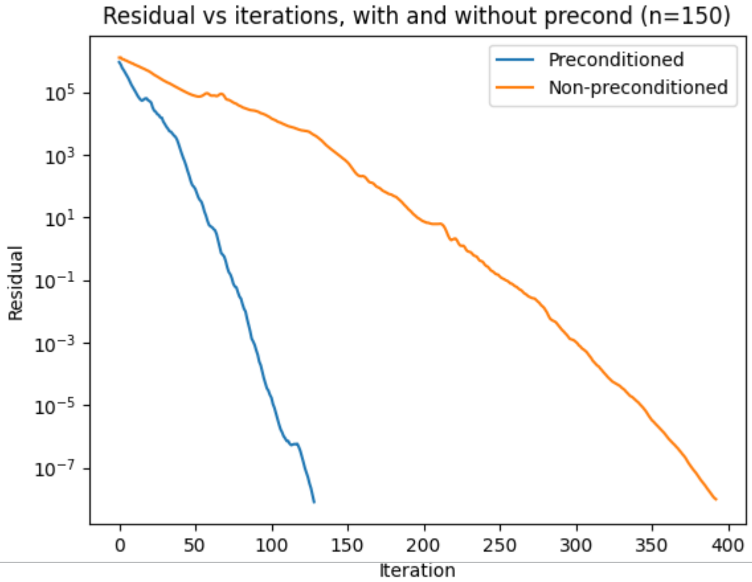
Results (conditioning number)



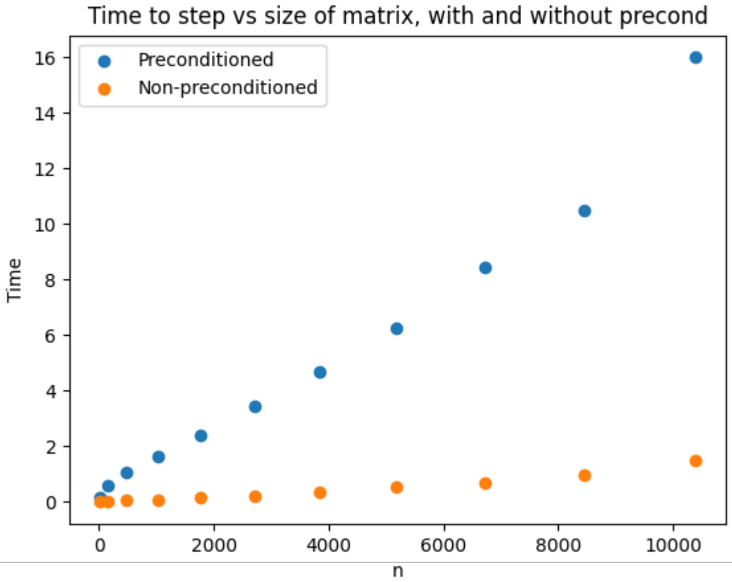
Results (iterations)



Results (residual)



Results (time)



References

[1] Dominik Göddeke.
Scientific computing.
Lecture Notes, June 2024.
Version: 25th June 2024.

ChatGPT was used for implementation details regarding sparse matrices and visualizing our numerically computed solution.