SF2520 — Applied numerical methods

Lecture 15

Numerical linear algebra
Iterative methods, convergence

Anna Nissen
Numerical analysis
Department of Mathematics, KTH

2023-11-14

Today's lecture

- Iterative methods for linear systems of equations
  - Search methods (Krylov subspace methods)
    - Steepest descent
    - Conjugate gradient
    - GMRES
- Convergence theory
- Preconditioning

Iterative methods for linear systems of equations

We consider

$$A\boldsymbol{x} = \boldsymbol{b}, \qquad A \in \mathbb{R}^{N\times N}, \quad \boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^N.$$

- Solve this via iteration $\Rightarrow$

$$\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$$

- Primarily used for sparse linear systems. Methods are mostly insensitive to the precise sparsity structure. No need to worry about "fill-in" as in direct methods.
- Stationary methods:
  Split $A = M - T$, so that $M\boldsymbol{x} = T\boldsymbol{x} + \boldsymbol{b}$ and iterate

$$M\boldsymbol{x}_{k+1} = T\boldsymbol{x}_k + \boldsymbol{b}.$$

- Example 1: $M =$ diagonal of A gives "Jacobi method"
- Example 2: $M =$ lower triangular part of A gives "Gauss–Seidel method"

Search methods

- In search methods, the iteration is of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \qquad \alpha_k \in \mathbb{R}, \quad \boldsymbol{p}_k \in \mathbb{R}^N,$$

  where α_k and $\boldsymbol{p}_k$ (the search direction) are computed based on the residual(s) $\boldsymbol{r}_\ell = \boldsymbol{b} - A\boldsymbol{x}_\ell$, with $\ell \leq k$.
- **Examples:**
  - Steepest descent method (SD)
  - Conjugate gradient method (CG)
  - GMRES ("generalized minimum residual" method)
- SD and CG require symmetric positive definite matrices. GMRES works for any nonsingular matrix.

Steepest descent method

Let A be a symmetric positive definite matrix. We note first that then

$$A\boldsymbol{x} = \boldsymbol{b} \qquad \Leftrightarrow \qquad \boldsymbol{x} \text{ minimizes } \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b} =: \phi(\boldsymbol{x}),$$

since

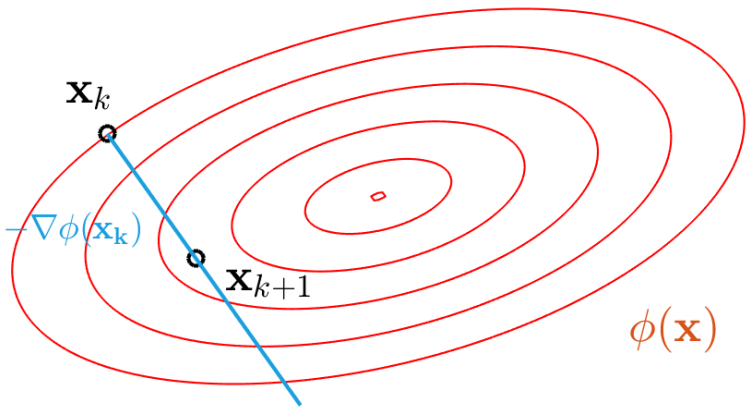
$$\nabla\phi(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}, \qquad D^2\phi(\boldsymbol{x}) = A > 0.$$

- In steepest descent we iterate

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla\phi(\boldsymbol{x}_k), \qquad \boldsymbol{x}_0 = 0,$$

where α_k is chosen to minimize ϕ along the gradient direction,

$$\alpha_k = \text{argmin}_\alpha \ \phi(\boldsymbol{x}_k - \alpha\nabla\phi(\boldsymbol{x}_k)).$$



Intuition: Move in direction of largest change ($\nabla\phi$) to lowest point in that direction.

Steepest descent method

We have

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k,$$

where

- The search direction is the residual,

$$\boldsymbol{p}_k = -\nabla\phi(\boldsymbol{x}_k) = \boldsymbol{b} - A\boldsymbol{x}_k = \boldsymbol{r}_k.$$

- α_k minimizes $\phi(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k)$, i.e.

$$\begin{aligned} 0 = \frac{d}{d\alpha}\phi(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) &= \boldsymbol{p}_k^T \nabla\phi(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) = \boldsymbol{p}_k^T(A(\boldsymbol{x}_k + \alpha\boldsymbol{p}_k) - \boldsymbol{b}) \\ &= -\boldsymbol{p}_k^T\boldsymbol{r}_k + \alpha\boldsymbol{p}_k^T A\boldsymbol{p}_k. \end{aligned}$$

- Therefore,

$$\alpha_k = \frac{\boldsymbol{p}_k^T\boldsymbol{r}_k}{\boldsymbol{p}_k^T A\boldsymbol{p}_k} = \frac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{r}_k^T A\boldsymbol{r}_k}.$$

- Note also:

$$\boldsymbol{r}_{k+1} = \boldsymbol{b} - A\boldsymbol{x}_{k+1} = \boldsymbol{b} - A(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k) = \boldsymbol{r}_k - \alpha_k A\boldsymbol{p}_k.$$

Steepest descent method

In summary, the method can be written

Steepest descent method

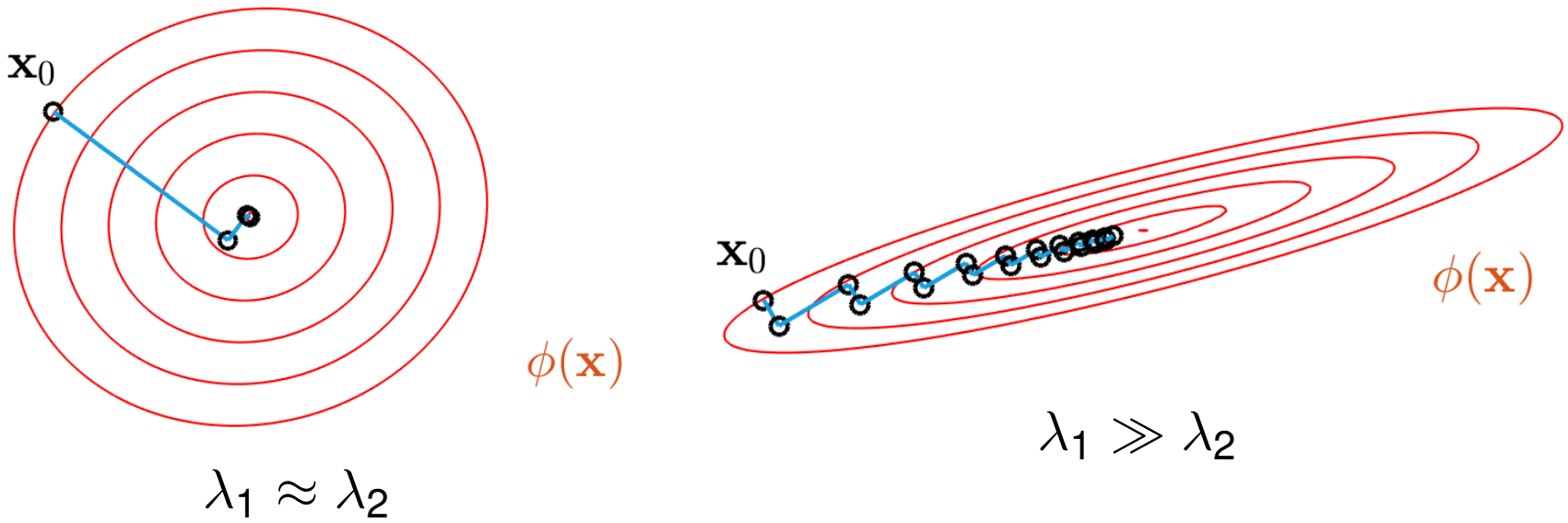
0. $\boldsymbol{x}_0 = 0$, $\boldsymbol{r}_0 = \boldsymbol{b}$.
1. $\boldsymbol{p}_k = \boldsymbol{r}_k$
2. $\alpha_k = \dfrac{\boldsymbol{p}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
3. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
4. $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
5. Goto 1

Note: The main cost of the iteration is the matrix-vector multiply $A\boldsymbol{p}_k$. This should only be computed once per iteration!

(Of course in practice we would simplify and only keep $\boldsymbol{r}_k$, not $\boldsymbol{p}_k$ in the algorithm. I kept $\boldsymbol{p}_k$ here for comparison with the conjugate gradient method later on.)

Steepest descent method

Remark:
When the eigenvalues of A differ a lot, $\lambda_1 \gg \lambda_2$, the contour lines of ϕ become very elongated ellipses. This can lead to slow convergence. Compare:



Note that $\lambda_1 \gg \lambda_2$ means that the condition number $\kappa(A) \gg 1$. In general a large condition number of A leads to slow convergence.

Conjugate gradient method

The conjugate gradient method (CG) uses more optimally chosen search directions $\boldsymbol{p}_k$ in

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \qquad \boldsymbol{x}_0 = 0.$$

In the end, the rest of the algorithm will be the same as SD.

- Observation: Regardless of how $\boldsymbol{p}_k$ are chosen we have

$$\boldsymbol{x}_{k+1} = \alpha_0 \boldsymbol{p}_0 + \alpha_1 \boldsymbol{p}_1 + \cdots + \alpha_k \boldsymbol{p}_k \in \text{span}\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_k\}.$$

- Conjugate gradient idea:
  - Let $\boldsymbol{x}_{k+1}$ be the best value in all of $\text{span}\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_k\}$,

$$\boldsymbol{x}_{k+1} = \operatorname*{argmin}_{\boldsymbol{x} \in \text{span}\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_k\}} \phi(\boldsymbol{x}).$$

  - Choose $\boldsymbol{p}_k$ so that:
    - All $\{\boldsymbol{p}_k\}$ are linearly independent. (This implies full convergence in at most N iterations.)
    - $\boldsymbol{p}_{k+1}$ and $\boldsymbol{x}_{k+1}$ can be computed easily from $\boldsymbol{p}_k$ and $\boldsymbol{x}_k$.

Conjugate gradient method – sketch of derivation

- Let P be the matrix of search directions upto $k-1$

$$P = \begin{pmatrix} | & | & & | \\ \boldsymbol{p}_0 & \boldsymbol{p}_1 & \cdots & \boldsymbol{p}_{k-1} \\ | & | & & | \end{pmatrix}$$

and, since $\boldsymbol{x}_{k+1}$ should be in span$\{\boldsymbol{p}_j\}$, make the ansatz

$$\boldsymbol{x}_{k+1} = P\boldsymbol{y} + \alpha \boldsymbol{p}_k.$$

We need to determine $\boldsymbol{y}$, α and $\boldsymbol{p}_k$.

- They should be selected so that
  1. $\boldsymbol{p}_k$ is not in span$\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{k-1}\}$.
  2. We minimize

$$\phi(\boldsymbol{x}_{k+1}) = \ldots = \phi(P\boldsymbol{y}) + \alpha \boldsymbol{y}^T P^T A \boldsymbol{p}_k + \frac{1}{2}\alpha^2 \boldsymbol{p}_k^T A \boldsymbol{p}_k - \alpha \boldsymbol{p}_k^T \boldsymbol{b}.$$

- We therefore pick $\boldsymbol{p}_k$ such that it is "A conjugate" to $\boldsymbol{p}_j$, $j < k$,

$$\boldsymbol{p}_j^T A \boldsymbol{p}_k = 0, \qquad j = 0, \ldots, k-1$$

$\Rightarrow \{\boldsymbol{p}_k\}$ are linearly independent and $\alpha \boldsymbol{y}^T P^T A \boldsymbol{p}_k = 0$.

Conjugate gradient method – sketch of derivation

- We are left to minimize, over $\boldsymbol{y}$ and α

$$\phi(\boldsymbol{x}_{k+1}) = \underbrace{\phi(P\boldsymbol{y})}_{\text{only depends on } y} + \underbrace{\frac{1}{2}\alpha^2 \boldsymbol{p}_k^T A \boldsymbol{p}_k - \alpha \boldsymbol{p}_k^T \boldsymbol{b}}_{\text{only depends on } \alpha}.$$

- We can minimize the two terms independently:
  - Minimum of $\phi(P\boldsymbol{y}) = \phi(\boldsymbol{x}_k)$, the solution in the previous iteration, by the definition of how $\boldsymbol{x}_k$ are chosen. Hence $P\boldsymbol{y} = \boldsymbol{x}_k$.
  - Optimal α in second term obtained by putting α-derivative to zero,

$$\alpha_k = \frac{\boldsymbol{p}_k^T \boldsymbol{b}}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} = \ldots = \frac{\boldsymbol{p}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}.$$

- We get

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$$

- Finally, one can show that

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}, \qquad \beta_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^T \boldsymbol{r}_{k-1}}$$

  is A-conjugate to $\boldsymbol{p}_j$ for $j < k$ (and easy to compute).

Conjugate gradient method

In summary, the method can be written

Conjugate gradient method

0. $\boldsymbol{x}_0 = \boldsymbol{p}_{-1} = 0$, $\boldsymbol{r}_0 = \boldsymbol{b}$, $\beta_0 = 0$.
1. $\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}$ and $\beta_k = \dfrac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{r}_{k-1}^T \boldsymbol{r}_{k-1}}$ (when $k > 0$)
2. $\alpha_k = \dfrac{\boldsymbol{p}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
3. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
4. $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
5. Goto 1

Note: As in the steepest descent method, the main cost of the iteration is the matrix-vector multiply $A\boldsymbol{p}_k$. It should only be computed once per iteration!

Conjugate gradient method

Remarks:

- The span of $\boldsymbol{p}_k$ satisfies

$$\text{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k\} = \text{span}\{\boldsymbol{b}, A\boldsymbol{b}, A^2\boldsymbol{b}, \ldots, A^k\boldsymbol{b}\},$$

  which is called a Krylov space, denoted by $\mathcal{K}(b, A, k)$.
- $\boldsymbol{x}_k$ in each iteration minimizes the "A-norm" of the error,

$$||\boldsymbol{x} - \boldsymbol{x}^*||_A^2 =: (\boldsymbol{x} - \boldsymbol{x}^*)^T A(\boldsymbol{x} - \boldsymbol{x}^*), \qquad (A\boldsymbol{x}^* = \boldsymbol{b} \text{ here}),$$

  over all vectors in $\mathcal{K}(b, A, k)$. (Note $||\cdot||_A$ is a norm since A is symmetric positive definite.)
- CG converges in at most N iterations. (Although this is seldom of practical use.)
- Convergence for CG is much faster than for SD.
- Like SD, convergence is slower for matrices with large condition numbers $\kappa(A)$.
- $\phi(\boldsymbol{x}_{k+1}) < \phi(\boldsymbol{x}_k)$, for both SD and CG.
- Keep in mind: SD and CG only work for symmetric positive definite matrices.

GMRES

- GMRES is a Krylov method for non-symmetric A.
- $\boldsymbol{x}_k$ minimizes $||A\boldsymbol{x}_k - \boldsymbol{b}||$ over all vectors in $\mathcal{K}(b, A, k)$.
- Implementation and algorithm much more complicated than CG. E.g. $\boldsymbol{p}_k$ depends on all previous $\boldsymbol{p}_j$, $j \leq k$.
- Algorithm builds an orthonormal basis for $\mathcal{K}(b, A, k)$ via modified Gram-Schmidt (Arnoldi) in each iteration.
- Matlab commands:

```
>> pcg(A,b);
>> gmres(A,b);
```

+ many other arguments and options possible

Multigrid

- Multigrid is an iterative method tailored for discretizations of elliptic PDEs (primarily).
- Solves PDE on different grid sizes N, $N/2$, $N/4$, ... and uses differences between numerical solutions to eliminate coarse scale errors.
- Uses simple stationary methods (e.g. Jacobi) to eliminiate fine scale errors.
- Highly efficient.

Computational costs

To achieve a certain given accuracy, the costs can be divided as

$$\text{computational cost} = \text{cost/iteration} \times \#\,\text{iterations needed.}$$

Cost/iteration

We suppose $A \in \mathbb{R}^{N\times N}$ is sparse with $O(N)$ non-zero elements. (Typical for PDE discretizations.) Then

- Stationary methods

  $$M\boldsymbol{x}_{k+1} = T\boldsymbol{x}_k + \boldsymbol{b}, \qquad A = M - T.$$

  Since A sparse, then so is M and T, and the cost of $T\boldsymbol{x}_k$ is $O(N)$. If also $M\boldsymbol{x} = \boldsymbol{f}$ can be solved in $O(N)$ time (as it can for Jacobi and Gauss–Seidel), then cost/iteration is $O(N)$.
- Search methods

  Main cost is the matrix-vector mulitply $A\boldsymbol{p}_k$ which costs $O(N)$.

Conclusion: Cost/iteration is $O(N)$. This is true for any sparsity pattern with $O(N)$ non-zero elements, not just banded matrices.

Convergence rate

- For the methods discucsed one can derive convergence estimates of the type, that if $A\boldsymbol{x}^* = \boldsymbol{b}$, then

$$||\boldsymbol{x}_k - \boldsymbol{x}^*|| \leq C\beta^k ||\boldsymbol{x}_0 - \boldsymbol{x}^*||,$$

  for some norm, constant C and $0 < \beta < 1$ called the convergence rate. It depends on A and on the method.
- Smaller β gives faster convergence. Often $\beta = 1 - \delta$ with $\delta \ll 1$ however.

Convergence rate

Stationary methods

- We can write the stationary methods in concise form as

$$\boldsymbol{x}_{k+1} = R\boldsymbol{x}_k + \boldsymbol{c}, \qquad R = M^{-1}T, \quad \boldsymbol{c} = M^{-1}\boldsymbol{b}.$$

- Since $\boldsymbol{x}^* = R\boldsymbol{x}^* + \boldsymbol{c}$ we get

$$\boldsymbol{x}_k - \boldsymbol{x}^* = R\boldsymbol{x}_{k-1} + \boldsymbol{c} - R\boldsymbol{x}^* - \boldsymbol{c} = R(\boldsymbol{x}_{k-1} - \boldsymbol{x}^*) = R^k(\boldsymbol{x}_0 - \boldsymbol{x}^*)$$

- And therefore,

$$||\boldsymbol{x}_k - \boldsymbol{x}^*|| \leq ||R^k||\,||\boldsymbol{x}_0 - \boldsymbol{x}^*||.$$

- Finally, if R can be diagonalized, then

$$||R^k|| \leq \rho(R)^k$$

  where $\rho(R)$ is the spectral radius of R.
- We conclude that one can take $\beta = \rho(R)$, if R can be diagonalized.
- In stationary methods, the convergence rate thus depends on the structure of M and T in a complicated way.

Convergence rate

Search methods

- In search methods β depends on the condition number κ of the matrix A.
- For steepest descent,

$$\beta = 1 - \frac{1}{\kappa}.$$

- For conjugate gradient,

$$\beta = 1 - \frac{1}{\sqrt{\kappa}} \qquad \left(\text{which is} < 1 - \frac{1}{\kappa} \text{ for } \kappa \text{ large} \right)$$

- Well-conditioned matrices A give faster convergence for search methods.

Convergence rate vs # iterations needed

Suppose we want an error $\leq \varepsilon$ and $\beta = 1 - \delta$ with $0 < \delta \ll 1$. How many iterations are then needed?

- We need to find the number of iterations mï¿½such that

$$||\boldsymbol{x}_m - \boldsymbol{x}^*|| \leq C\beta^m \underbrace{||\boldsymbol{x}_0 - \boldsymbol{x}^*||}_{e_0} \leq \varepsilon.$$

- This will be satisfied when

$$m \ln(\beta) \leq \ln\left(\frac{\varepsilon}{Ce_0}\right) \quad \Rightarrow \quad m|\ln(\beta)| \geq \left|\ln\left(\frac{\varepsilon}{Ce_0}\right)\right| \quad \Rightarrow \quad m \geq \frac{D}{|\ln(\beta)|},$$

  where $D = |\ln(\varepsilon/(Ce_0))|$ only depends on e_0 and ε.
- Since $\delta \ll 1$,

$$\ln(\beta) = \ln(1 - \delta) \approx -\delta.$$

- Consequently, we need, roughly,

$$m \approx \frac{D}{\delta} \quad \text{iterations.}$$

- Small δ means more iterations m needed. Precise number also depends on initial error e_0 and tolerance ε.

Computational costs

In summary, to achieve a certain given accuracy, we have

$$\text{computational cost} = \text{cost/iteration} \times \#\,\text{iterations needed.}$$

- Assuming that the number of non-zero entries in A is $O(N)$, then

  $$\text{Cost}=O(Nm) = O(N/\delta),$$

  where m is the number iterations needed and $\beta = 1 - \delta$.
- For stationary methods $\beta = \rho(R)$, so $\delta = 1 - \rho(R)$. This gives

  $$\text{Cost}=O\left(\frac{N}{1-\rho(R)}\right).$$

- For steepest descent, $\delta = 1/\kappa$. This gives

  $$\text{Cost}=O\left(N\kappa\right).$$

- For conjugate gradient, $\delta = 1/\sqrt{\kappa}$. This gives

  $$\text{Cost}=O\left(N\sqrt{\kappa}\right).$$

  (Also true for GMRES when A close to normal.)

Costs of solving Poisson equation

Consider a finite difference discretization of

$$-\Delta u = f,$$

in d dimensions. This leads to a matrix equation

$$A\boldsymbol{u} = \boldsymbol{f}, \qquad A \in \mathbb{R}^{N\times N}.$$

Suppose discretization has n grid points in each coordinate direction. For direct methods we saw before that

Dimension	Unknowns (N)	Bandwidth (p)	Cost ($\sim Np^2$)
1	n	1	$O(n)$
2	n^2	n	$O(n^4)$
3	n^3	n^2	$O(n^7)$

Remark: Direct methods can sometimes be improved if problem is highly structured (square, constant coefficients, ...). E.g. $d = 2$ can be solved in $O(n^3)$ï¿½or even $O(n^2 \log n)$ time.

Costs of solving Poisson equation

For iterative methods we have when $A\boldsymbol{x} = \boldsymbol{b}$ comes from a finite difference discretization of Poisson:

- For Jacobi, Gauss–Seidel: $\rho(R) \sim 1 - 1/n^2$
  (Can be derived from discretization.)
- Condition number $\kappa \sim h^{-2}$, where $h = O(1/n)$, for all d.
  (We derived this in one dimension when we talked about elliptic equations.)
- Multigrid (MG): $\beta \leq \beta_0 < 1$ independent of N.

	δ	$m \sim 1/\delta$	cost ($Nm = n^d m$)
Jacobi	$1/n^2$	n^2	n^{d+2}
Gauss–Seidel	$1/n^2$	n^2	n^{d+2}
SD	$1/\kappa \sim 1/n^2$	n^2	n^{d+2}
CG	$1/\sqrt{\kappa} \sim 1/n$	n	n^{d+1}
GMRES	$1/\sqrt{\kappa} \sim 1/n$	n	n^{d+1}
Multigrid	$O(1)$	1	n^d

Costs of solving Poisson equation

In summary, we have the following costs for solving the Poisson equation with finite differences in 1D, 2D and 3D.

Dimension	Direct method	Jacobi/GS/SD	CG/GMRES	MG
1	n	n^3	n^2	n
2	n^4	n^4	n^3	n^2
3	n^7	n^5	n^4	n^3

- Little or no gain from iterative methods in 1D, compared to direct methods.
- Large gains in 3D.
- Unclear gains in 2D.
- Multigrid superior to the other methods.
- Jacobi/GS/SD not competetive. Mostly used as components in Multigrid or as "preconditioners".

Preconditioning

- For CG and GMRES the convergence rate is in general

$$\beta = 1 - \frac{1}{\sqrt{\kappa}} \quad \Rightarrow \quad m \sim \sqrt{\kappa}.$$

  Slow convergence therefore for ill-conditioned matrices.
- Preconditioning can be used to lower κ.
- Idea:
  Replace $A\boldsymbol{x} = \boldsymbol{b}$ by equivalent system $M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$, where
  (a) $\kappa(M^{-1}A)$ smaller than $\kappa(A)$
  (b) $M\boldsymbol{x} = \boldsymbol{y}$ is easy to solve

  Then iterate with CG/GMRES on $M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}$ instead of $A\boldsymbol{x} = \boldsymbol{b}$.

  $\Rightarrow$ Faster convergence.
- Note that (a) and (b) are conflicting goals. Best for (a) is to take $M = A$, which gives $\kappa = 1$, but then (b) is not satsified. Best for (b) is to take $M = I$, but then κ does not change. Find good trade-off!

Preconditioning

The preconditioner M can be quite simple and still help. Some examples:

- $M =$ diagonal part of A (similar to one step with Jacobi, with $\boldsymbol{x}_0 = 0$).
- Multiple steps with Jacobi or Gauss–Seidel.
- Incomplete LU- or Cholesky factorizations.
- Good choice highly problem dependent.

Preconditioned Conjugate Gradient (PCG)

- Need extra trick to precondition conjugate gradient since $M^{-1}A$ is in general not symmetric positive definite (SPD) even if both M and A are.
- Let M be SPD and set $C = M^{1/2}$, i.e. an SPD matrix such that $C^2 = M$. (Such C exists.)
- Then $C^{-1}AC^{-1}$ is also SPD and since

$$A\boldsymbol{x} = \boldsymbol{b} \quad \Rightarrow \quad C^{-1}AC^{-1}C\boldsymbol{x} = C^{-1}\boldsymbol{b}$$

we can apply conjugate gradient to the SPD system

$$C^{-1}AC^{-1}\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{b}}, \qquad C\boldsymbol{x} = \tilde{\boldsymbol{x}}, \quad C\tilde{\boldsymbol{b}} = \boldsymbol{b},$$

instead.

- This only changes CG very little in the end. (Main change: In each iteration an additional system $M\boldsymbol{z}_k = \boldsymbol{r}_k$ must be solved.)

Incomplete LU/Cholesky factorization

- Perform standard LU/Cholesky (via gaussian elimination) but let elements of L and U be zero whenever corresponding element of A is zero ($\ell_{j,k} = u_{j,k} = 0$ if $a_{j,k} = 0$).
- Gives approximate factors $\tilde{L}$ and $\tilde{U}$ with $\tilde{L}\tilde{U} \approx A$ for LU and $\tilde{L}\tilde{L}^T \approx A$ for Cholesky.
- In Matlab: `[L,U]=ilu(A)` and `L=ichol(A)`
- Cost = $O(N)$ if there is a fixed number of non-zeros in each row and column.
- Use $M = \tilde{L}\tilde{U}$ and $M = \tilde{L}\tilde{L}^T$ as preconditioners. Then $M\boldsymbol{x} = \boldsymbol{y}$ can easily be solved in $O(N)$ time.
- In Matlab, call `pcg` and `gmres` as

```
>> pcg(A,b,tol,maxiter,M);          (if M is preconditioner)
>> pcg(A,b,tol,maxiter,L,L');       (if LL^T is preconditioner)
>> gmres(A,b,tol,maxiter,M);        (if M is preconditioner)
>> gmres(A,b,tol,maxiter,L,U);      (if LU is preconditioner)
```