Numerical Linear Algebra
with examples in geometry processing

Gaël Guennebaud

Outline

- How to choose the right solver?
  - dense, sparse, direct, iterative, preconditioners, FMM, etc.
- Smoothness?
- Quadratic constraints
- Overview of other classical building-blocks

A zoo of linear solvers

SVD

- Singular Value Decomposition

$$A = V \Sigma W^* \quad \rightarrow \quad x = A^+ b = W \Sigma^+ V^* b$$

  - Welcome default behavior:
    - over-constrained → Least-Square solution
    - rank-deficient → Least-Norm solution
  - Down-side:
    - involve iterative decomposition algorithms
    - overkill for linear solving?

QR

- QR decomposition $AP = QR$
  - Least-square solution: $x = P R^{-1} Q^T b$
  - with column-pivoting → rank revealing
    - rank-deficient:

$$AP = Q \begin{pmatrix} R_1 & R_2 \\ 0 & 0 \end{pmatrix}$$

→ complete orthogonalization (eliminate R_2)

$$AP = Q \begin{pmatrix} T_{11} & 0 \\ 0 & 0 \end{pmatrix} Z$$

→ yields minimal norm solution :)

LU

- LU decomposition

$$A P = L U$$

  - based on Gaussian elimination
  - good for square, non symmetric problems
  - mostly useful for sparse problems

Cholesky

- Cholesky decomposition
  - For SPD matrices

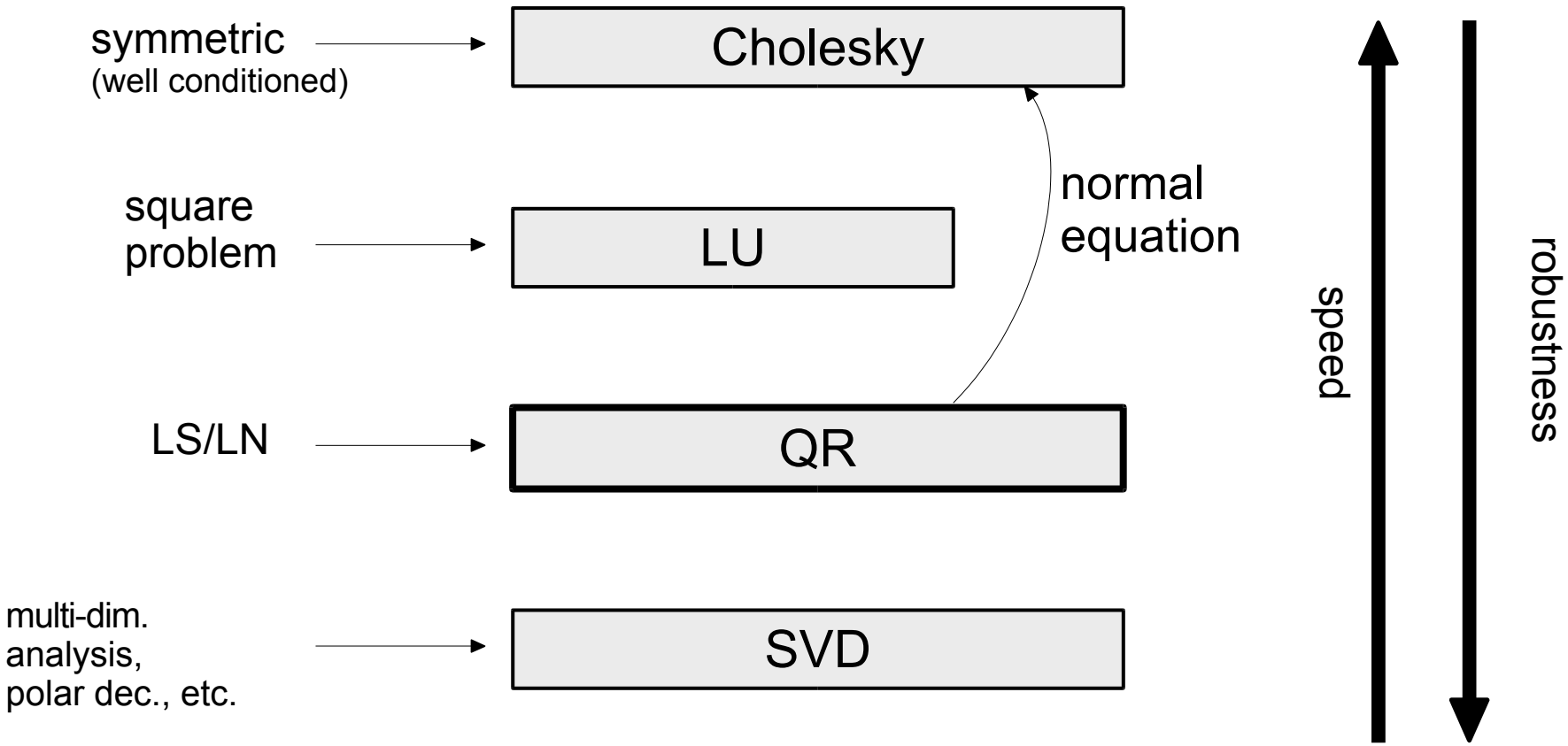
$$A = L L'$$

  - For symmetric indefinite matrices:

$$P^T A P = L D L'$$

    - as fast
    - numerical stability:
      - pivoting
      - or 2x2 diagonal blocks

Dense solvers – Summary



Example

- Scattered data interpolation/approximation
  - problem statement

input:
- sample positions $\mathbf{p}_i$
- with associated values f_i

output:
- a **smooth** scalar field $f : \mathbb{R}^d \rightarrow \mathbb{R}$

  s.t., $f(\mathbf{p}_i) \approx f_i$

Discretization

- Decomposition on a set of basis functions

$$f(\mathbf{x}) = \sum_j \alpha_j \varphi_j(\mathbf{x})$$

unknowns

  - linear LS minimization:

$$\boldsymbol{\alpha} = \operatorname{argmin} \sum_i \left\| \sum_j \alpha_j \varphi_j(\mathbf{p}_i) - f_i \right\|^2$$

  - plus, *f* has to be **smooth**
    - how to mathematically defines “smooth”?

      → seek for a (poly-)harmonic solution:

$$\Delta^k f = 0$$

Smoothness & RBF

- Solution 1: Enforce smoothness *by construction*
  - Choose (poly-)harmonic basis functions:

$$\Delta^k \varphi_i = 0$$

  - Example: Radial Basis Functions
    - centered at *nodes* $\mathbf{q}_j$: $f(\mathbf{x}) = \sum_j \alpha_j \varphi(\|\mathbf{x} - \mathbf{q}_j\|)$
    - polyharmonic splines: $\varphi(t) = t^k \quad , \quad k = 1,3,5,\ldots$
      $\varphi(t) = t^k \ln(t) \quad , \quad k = 2,4,6,\ldots$
    - thin-plate spline : $\varphi(t) = t^2 \ln(t)$

RBF in practice

- Leads to a **dense** LS problem:

$$\begin{bmatrix} \cdots & \vdots & \cdots \\ & \varphi(\|\mathbf{p}_i - \mathbf{q}_j\|) & \\ & \vdots & \end{bmatrix} \cdot \boldsymbol{\alpha} = \begin{bmatrix} \vdots \\ f_i \\ \vdots \end{bmatrix} \quad \Leftrightarrow \quad A\,\boldsymbol{\alpha} = \mathbf{b}$$

  - Choice of the $\mathbf{q}_j$?
    - take $\mathbf{q}_j = \mathbf{p}_j$ → interpolation!
  - Solver choice?
    - square & non-symmetric → LU
  - Conditioning
    - depends on the sampling

RBF in practice

- Globally supported basis
  - storage: $O(n^2)$
  - solving: $O(n^3)$
  - 1 evaluation: $O(n)$

    → very expensive for numerous nodes
    - max: a few thousands
  - For *n* large: Fast Multipole Method (FMM)
    - iterative and hierarchical approach
    - somewhat complicated, rarely used in practice

Global to Local Basis

- Solution 2: enforce smoothness through a PDE
  - the key problem is now to solve for

$$\Delta^k f = 0$$

  - subject to *boundary* constraints, e.g.: $f(\mathbf{p}_i) = f_i$
  - advantage:
    - enable locally supported basis functions (e.g., box-splines)

→ Finite Element Method (FEM)

Laplacian equation

- Example: $\Delta f = 0$

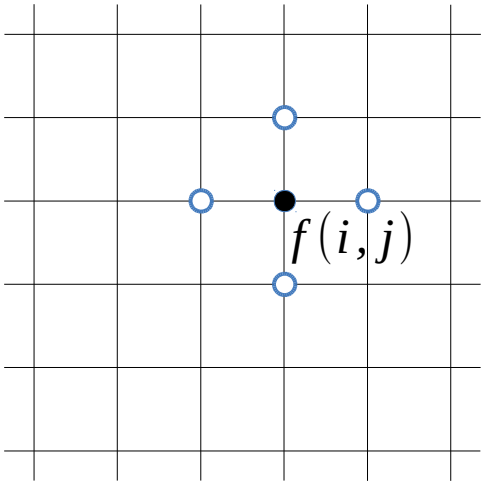
$$\left(\Delta f = \nabla \cdot \nabla f = \frac{\partial^2 f_x}{\partial x^2} + \frac{\partial^2 f_y}{\partial y^2} + \cdots \right)$$

  - fundamental in many applications
    - interpolation
    - smoothing
    - regularization
    - deformations
    - parametrization
    - etc.

FD Discretization

- Example on a 2D grid
  - finite differences

$$\Delta \Leftrightarrow \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



$$\Delta f(i,j) = \frac{\left(f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1)\right)}{4} - f(i,j) = 0$$

  - Matrix form: $\mathbf{L}\,\mathbf{f} = 0$

FEM Discretization

- Leads to a **sparse** linear system of equations

$$\mathrm{L}\,\mathbf{u} = 0 \qquad \text{with} \qquad L_{i,j} = \langle \nabla \varphi_i, \nabla \varphi_j \rangle$$

  - L is called the *stiffness* matrix
  - φ_i are compactly supported → most of the $L_{i,j} = 0$
  - L is usually huge, e.g.
    - ~ number of pixels of an image
    - ~ number of vertices of a mesh

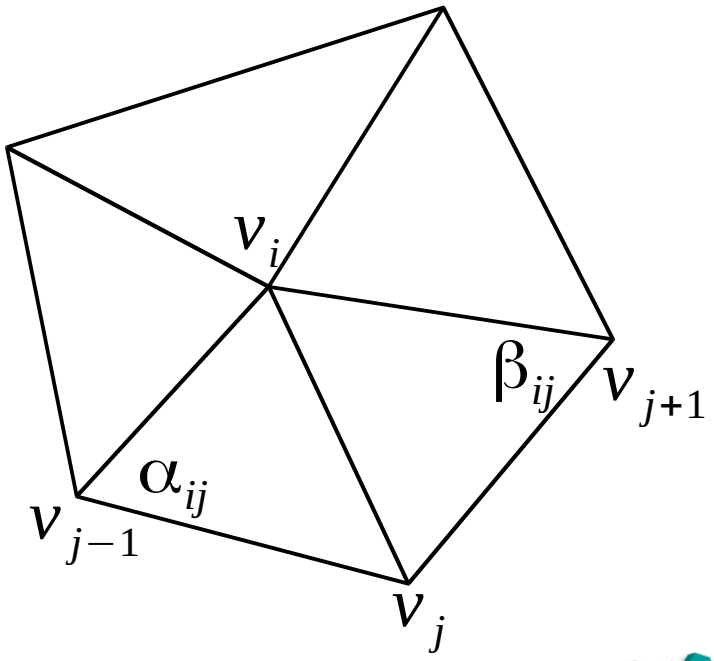
→ How to exploit sparsity in linear solvers?

FEM Discretization

- On a triangular mesh
  - φ_i= linear basis (aka barycentric coordinates)
  - famous “cotangent formula”:

$$L_{i,j} = <\nabla \varphi_i, \nabla \varphi_j> = \cot \alpha_{ij} + \cot \beta_{ij}$$

$$L_{i,i} = - \sum_{v_j \in N_1(v_i)} L_{i,j}$$



Sparse representation?

- Naive way: `std::map<pair<int,int>, double>`
- Compressed {Row,Column} Storage
  - the most commonly used

0	3	0	0	0
22	0	0	0	17
7	5	0	1	0
0	0	0	0	0
0	0	14	0	8

Values:	22	7	3	5	14	1	17	8
InnerIndices:	1	2	0	2	4	2	1	4
OuterStarts:	0	2	4	5	6	*8*		

  - need special care to “assemble” the matrix
    - warning: might be time consuming!
  - variant: store small blocks

Sparse solver classifications

- Direct methods
  - Simplicial versus Super{nodal,frontal}
  - Fill-in ordering
- Iterative methods
  - Preconditioning
- Multi-grid & Hybrid methods

Direct methods

- General principle
  - adapt matrix decompositions to sparse storage
    - Cholesky, LU, QR, etc.
- Main difficulties:
  - matrix-updates introduce new non-zeros

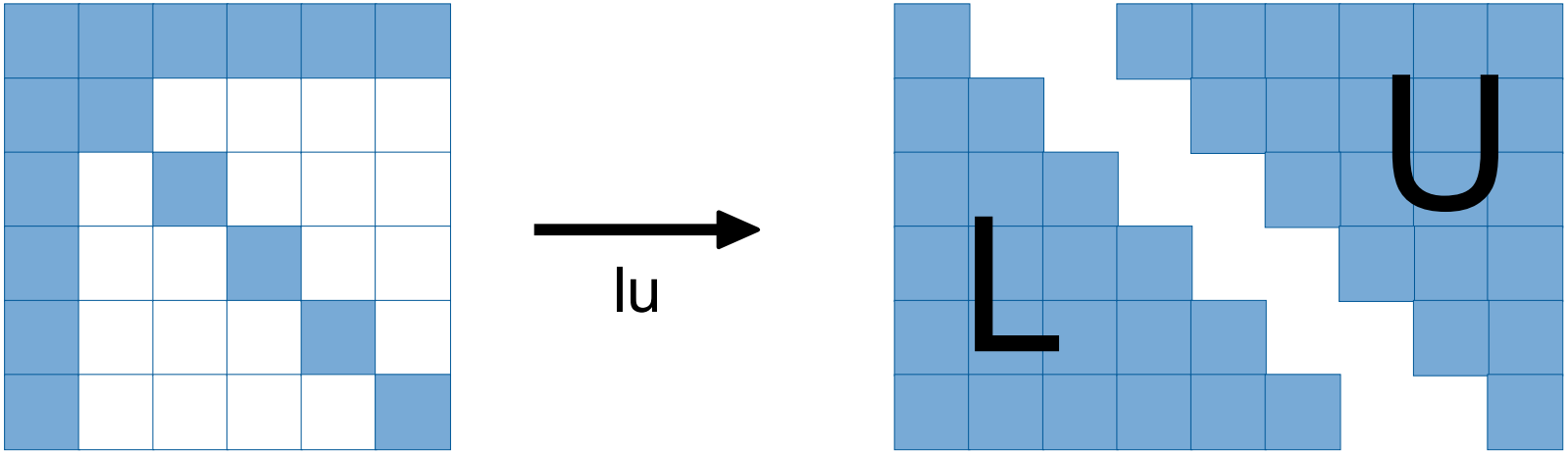
    → need to predict their positions to avoid prohibitive memory reallocation/copies

    → need to reduce the number of new non-zeros (fill-in)
  - scalar-level computation is slow

    → need to leverage dense matrix operations

Fill-in

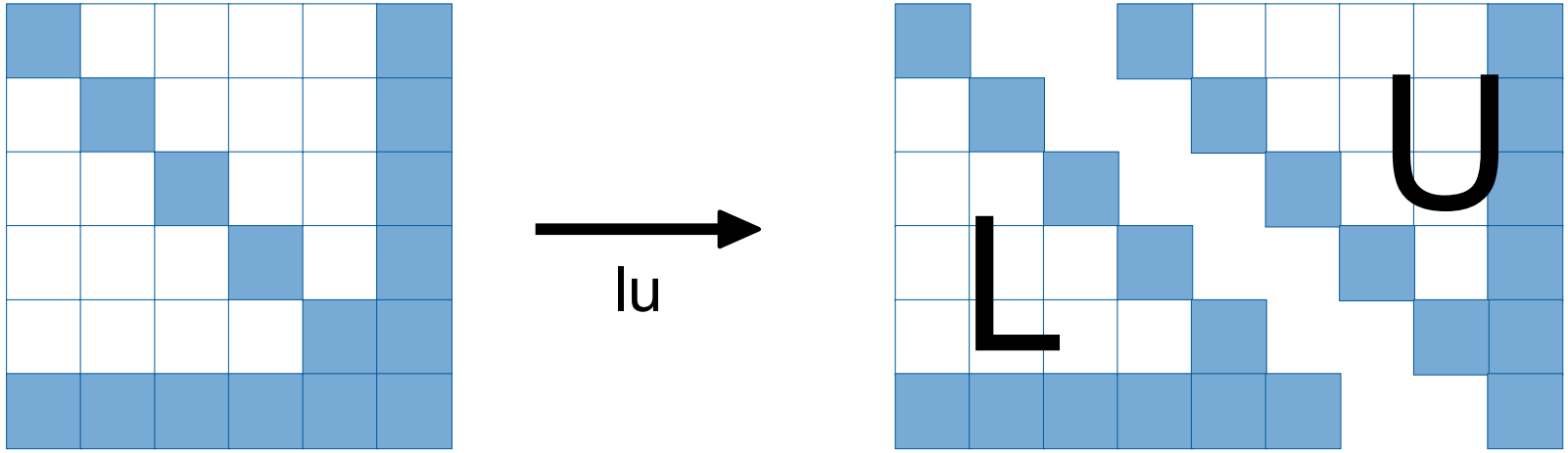
- Fill-in depends on row/column order!
  - i.e., on the arbitrary choice of the numbering of the unknowns & constraints
  - pathological example:



sparse input ***dense*** *factors* :(

Fill-in

- Fill-in depends on row/column order!
  - i.e., on the arbitrary choice of the numbering of the unknowns & constraints
  - pathological example:



sparse input
after re-ordering

sparse *factors* :)

Fill-in

- Fill-in depends on row/column order!
  - i.e., on the arbitrary choice of the numbering of the unknowns & constraints

  → re-ordering step prior to factorization
- tricky:
  - must be faster than the factorization!
  - must trade numerical stability!
  - must preserve symmetry

Fill-in ordering

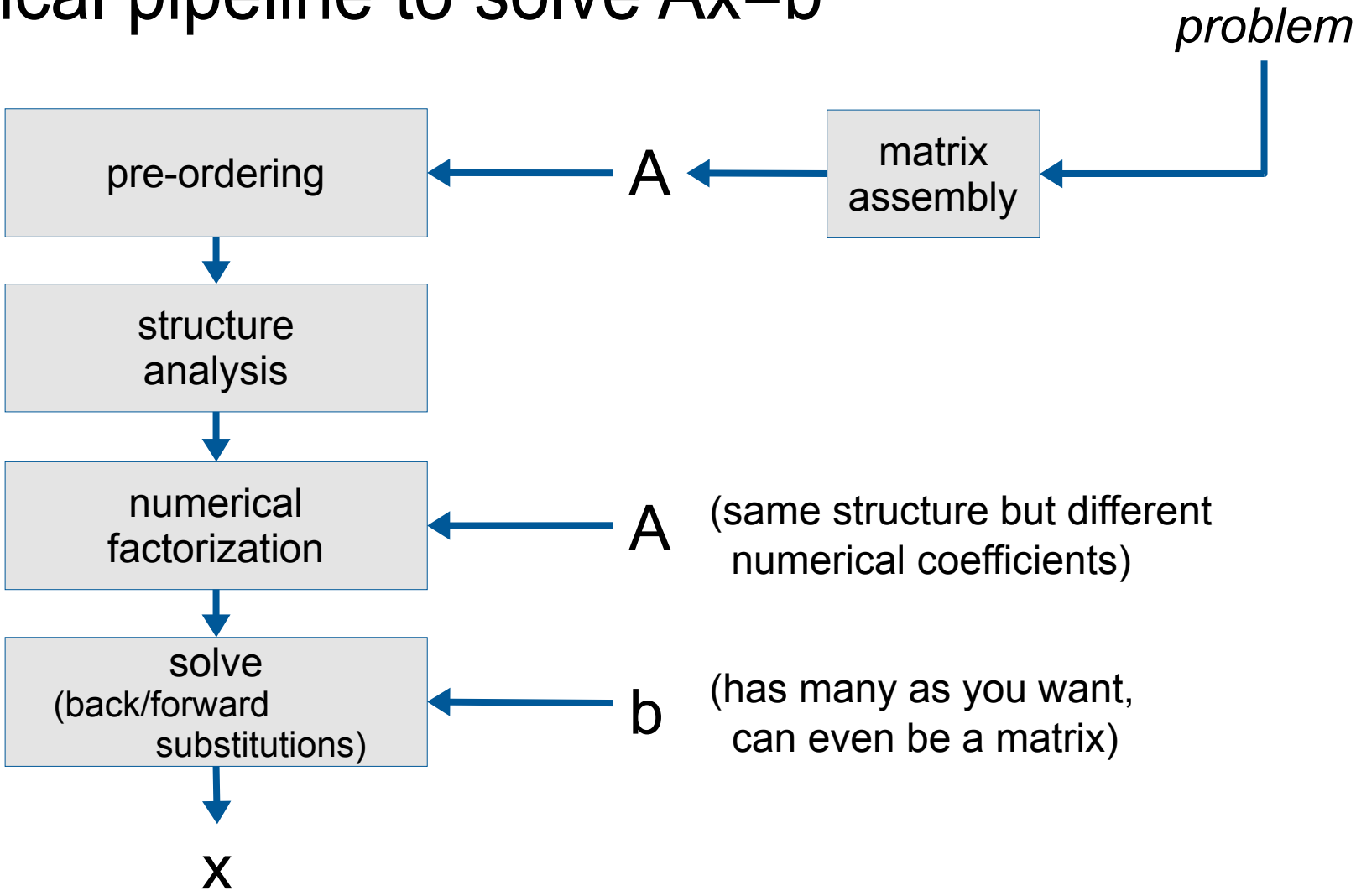
- Many heuristics
  - Band limiting
  - Nested discestion
  - **approximate minimum degree** (AMD)
    - symmetric and symmetric variants

Performance issue

- Sparse structure
  - → indirect memory accesses
    - bad pipelining
    - bad cache usage
- Need to leverage dense matrix computations
  - several variants: multinodal, multifrontal, etc.
  - makes sense for not too sparse problems
    - e.g., Poisson eq. on a 3D domain

Direct solvers – summary

- Typical pipeline to solve Ax=b



Direct solvers – summary

- Pros
  - solve for multiple right-hand sides
  - very fast for very sparse problems (e.g., 2D Poisson)
- Cons
  - high memory consumption
    - ok for 2D domains
    - huge for 3D domains
  - (very) difficult to implement

Iterative methods

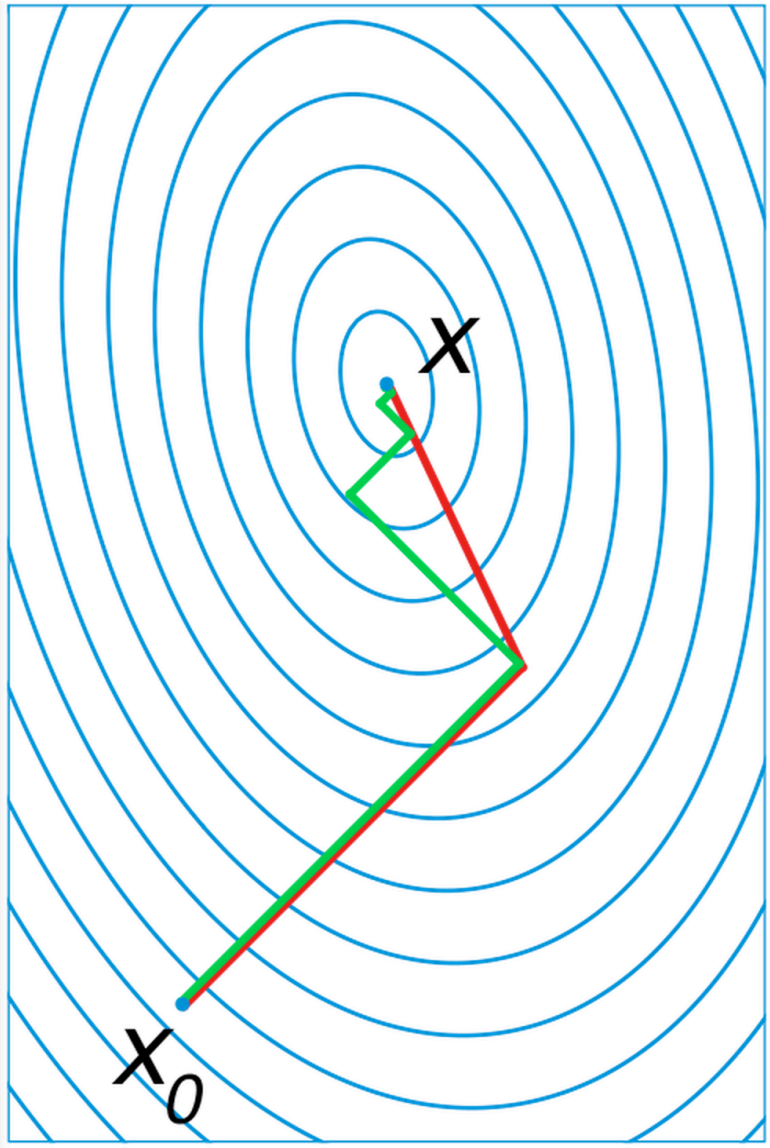
- Jacobi iterations, Gauss-Seidel
  - stationary methods based on matrix splitting:
    - Jacobi : $x^{(i+1)} = D^{-1}\left(b - R\,x^{(i)}\right)$ $\quad A = D + R$
    - Gauss-Seidel : $x^{(i+1)} = L^{-1}\left(b - U\,x^{(i)}\right)$ $\quad A = L + U$
  - easiest to implement but...
  - slow convergence
  - needs to be diagonally dominant (or SPD)

Iterative methods

- Conjugate Gradient (CG)
  - non-stationary method
  - SPD: convergence with decreasing error
  - principle
    - descent along a set of optimal search directions:

$$\{d_1, \ldots, d_i\}$$

with $d_j^T A d_i = 0$



Conjugate Gradient

- In practice
  - dominated by matrix-vector products: $\mathrm{A}\,\mathrm{d}_i$
  - no need to “assemble” the matrix A
    - operator approach
    - easy to implement on the GPU
  - much faster convergence with a pre-conditioner
    - Jacobi, (S)SOR → easy, matrix-free and GPU friendly
    - Incomplete factorization → more involved

Least-Square & CG

- Conjugate Gradient for Least-Square problems
  - The bad approach: form the normal equation

$$A^T A x = A^T b$$

  - LSCG
    - solve for the normal equation without computing $A^T A$
    - numerically more stable
    - matrix-free & GPU friendly

Iterative methods

- Iterative methods for non-symmetric problems
  - Bi-CG(STAB)
    - close to CG but...
    - convergence not guaranteed
    - error may increase!
  - GMRES
    - error monotonically decreases but...
    - may stall until the *n*-th iteration!
    - memory consumption
      - has to store a list of basis vectors (hundreds)

Sparse solvers – Summary

	memory	mat-free	multiple rhs	2D domain	3D domain
Direct (simplicial)	-	-	***	***	*
Direct (with dense blocks)	-	-	***	*	**
Iterative methods	***	***	-	*	***

- **Symmetry** Positive Definite is important
  - simpler implementation
  - up to an order of magnitude faster
  - more robust

Solver Choice

- Questions:
  - Solve multiple times with the same matrix?
    - yes → direct methods
  - Dimension of the support mesh
    - 2D → direct methods
    - 3D → iterative methods
  - Can I trade the performance? Good initial solution?
    - yes → iterative methods
  - Hill conditioned?

- Still lost? → online sparse benchmark

→ *demo*

Let's go back to our Laplacian problem...

Laplacian problem

- Laplacian matrix on a triangular mesh

$$\Delta u = 0 \quad \Leftrightarrow \quad \mathrm{L}\,\mathbf{u} = 0$$

  - with $L_{i,j} = \cot\alpha_{ij} + \cot\beta_{ij}$, $L_{i,i} = -\sum L_{i,j}$
  - symmetric
  - conditioning depends on triangle shapes
  - SPD for well shaped triangles
  - solver choice: *direct simplicial LDL^T*

Laplacian problem

$$\Delta u = 0 \quad \Leftrightarrow \quad \mathrm{L}\,\mathbf{u} = 0$$

- This is an *abstract* problem
  - need to add constraints to make it meaningful
- Fix values at vertices, i.e., $u_i = \bar{u}_i$ for some i
  - remove smoothness constraints at these vertices
  - and reorder:

$$\begin{bmatrix} \mathrm{L}_{00} & \mathrm{L}_{01} \\ \mathrm{L}_{10} & \mathrm{L}_{11} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \bar{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} 0 \\ \theta \end{bmatrix} \quad \Rightarrow \quad \mathrm{L}_{00} \cdot \mathbf{u} = -\mathrm{L}_{01} \cdot \bar{\mathbf{u}}$$

  - problem is still SPD :)

Laplacian problem

- Add linear constraints: $\mathbf{C}\,\mathbf{u}=\mathbf{b}$
  - Solution 1:
    - reduce the solution space through the null-space of C
    - reduce problem size :)
    - problem is not symmetric anymore :(

Laplacian problem

- Add linear constraints: $\mathbf{C}\,\mathbf{u}=\mathbf{b}$
  - Solution 2:
    - Lagrange multipliers yields

$$\begin{bmatrix} \mathbf{L} & \mathbf{C}^T \\ \mathbf{C} & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{b} \end{bmatrix}$$

    - not SPD :(
    - but symmetric indefinite → LDL^T if well conditioned

Regularizing homogeneous equations

with

quadratic constraints

A first example

- How to fit a hyper-plane trough points?
  - Search a plane with center **c** and normal **n** to a set of points $\mathbf{p}_i$
  - Minimize least-square error :

$$E(\mathbf{c},\mathbf{n}) = \sum_i \left| (\mathbf{p}_i - \mathbf{c})^T \mathbf{n} \right|^2$$

  - Subject to $\|\mathbf{n}\| = 1$

  $\rightarrow$ at a first glance, non linear problem...

Plane fitting

- E(c,n) minimum when its derivative wrt. c vanish :

$$\frac{\partial E(\mathbf{c},\mathbf{n})}{\partial \mathbf{c}} = \ldots = -2\,\mathbf{n}\,\mathbf{n}^T \sum_i (\mathbf{p}_i - \mathbf{c}) = 0$$

  - implies that

$$\sum_i (\mathbf{p}_i - \mathbf{c}) = 0 \quad \Rightarrow \quad \mathbf{c} = \frac{1}{n} \sum_i \mathbf{p}_i$$

Plane fitting

- Reformulate E(c,n):

$$E(\mathbf{c},\mathbf{n}) = \mathbf{n}^T \left| \sum_i (\mathbf{q}_i - \mathbf{c})(\mathbf{q}_i - \mathbf{c})^T \right| \mathbf{n} = \mathbf{n}^T \mathbf{C}\, \mathbf{n} \rightarrow min$$

  - subject to $\|\mathbf{n}\| = 1$
  - Lagrange multiplier : $\mathbf{n}^T \mathbf{C}\, \mathbf{n} - \lambda(\mathbf{n}^T \mathbf{n} - 1) \rightarrow min$
  - Differentiate on **n** yields an eigenvalue problem :

$$\mathbf{C}\, \mathbf{n} = \lambda\, \mathbf{n}$$

    - residual: $\mathbf{n}^T \mathbf{C}\, \mathbf{n} = \lambda$

→ **n** is eigenvector of smallest eigenvalue

A second example

- How to fit an hyper-sphere to points?
  - Search a sphere with center $\mathbf{c}$ and radius r to a set of points $\mathbf{p}_i$
  - Minimize least-square error :

$$E(\mathbf{c}, r) = \sum_i \left(\|\mathbf{p}_i - \mathbf{c}\| - r \right)^2$$

    - non-linear energy → see previous session (need an initial guess)
    - numerically unstable for flat area ($\mathbf{c}, r \to \infty$)

Sphere fitting

- Linearized energy:

$$E(\mathbf{c},r) = \sum_i \left(\|\mathbf{p}_i - \mathbf{c}\|^2 - r^2 \right)^2$$
$$= \sum_i \left(\mathbf{c}^2 - r^2 - 2\,\mathbf{p}_i^T\,\mathbf{c} + \mathbf{p}_i^2 \right)^2$$
$$= \sum_i \left(u_c + \mathbf{p}_i^T\,\mathbf{u_l} + \mathbf{p}_i^2 \right)^2$$

  - metric is not Euclidean anymore
  - still unstable for flat area

Sphere fitting

- Linearized energy:

$$
\begin{aligned}
E(\mathbf{c},r) &= \sum_i \left(\|\mathbf{p}_i - \mathbf{c}\|^2 - r^2 \right)^2 \\
&= \sum_i \left(\mathbf{c}^2 - r^2 - 2\,\mathbf{p}_i^T\,\mathbf{c} + \mathbf{p}_i^2 \right)^2 \\
&= \sum_i \left(u_c + \mathbf{p}_i^T\,\mathbf{u_l} + \mathbf{p}_i^2 \right)^2 \\
&= \sum_i \left(u_c + \mathbf{p}_i^T\,\mathbf{u_l} + u_q\,\mathbf{p}_i^2 \right)^2
\end{aligned}
$$

  - metric is not Euclidean anymore
  - again, needs to avoids trivial solution $\mathbf{u} = 0$

Algebraic sphere fitting

- Some bad ideas:
  - fix some values, e.g.: $u_q = 1$
  - linear equality: $\sum_j u_j = 1$
  - unit norm: $\|\mathbf{u}\| = 1$
- What do we want?
  - be invariant to similarity transformations
  - mimic Euclidean norm

Algebraic sphere fitting

- Solution:
  - constraint $\|\nabla f(\mathbf{x})\| = 1 \quad \text{at} \quad f(\mathbf{x}) = 0$
  - algebraic distance close to Euclidean one nearby region of interest
- In practice:

$$\mathbf{u}^{\mathrm{T}} Q \mathbf{u} = 1$$

  - with Q symmetric
  - solve E over the unit ball induced by Q

Quadratic constraints

- The general problem is now:
  - minimize $\|\mathrm{A}\,\mathbf{u}\|^2$
  - subject to $\mathbf{u}^{\mathrm{T}}\,\mathrm{Q}\,\mathbf{u}=1$
  - through Lagrange multipliers, we end up with a *generalized eigenvalue* problem:

$$\mathrm{A}\,\mathbf{u}=\lambda\,\mathrm{Q}\,\mathbf{u}$$

  - residual = λ
  - $\mathbf{u}$ is the eigenvector of the smallest eigenvalue

Quadratic Constraints

- Other examples:
  - Unsigned surface reconstruction
  - Smooth *n*-direction fields

- Taking home message
  - the choice of the regularization norm is crucial!
  - taking $||\mathbf{x}||=1$ is unlikely the right choice!

Eigenvalue problems

- How to solve?
  - closed forms 2x2 and 3x3
  - iterative algorithms otherwise
  - need only the largest → Power iterations
    - fast, easy, GPU-friendly, sparse-friendly
    - be careful with repeated eigenvalues
  - need only the smallest → Inverse Power iterations
    - slightly more tricky: needs a linear solver
- Can-it be considered as a direct method?
  - numerically no, but
  - it provides many of the advantages of simple linear problems such as analytic derivatives

Other classical approaches in geometry processing

- Alternate solution
  - chicken-egg problems
    - fix one part of the equation, solve for the second part
    - fix the second part, solve for the first one
    - repeat
  - ex.: ARAP energy
- Barriers
  - replace inequality constraints with penalty functions
  - much more tricky than it looks like

Other classical approaches in geometry processing

- Smooth functions on meshes
  - linear basis are unnecessarily numerous
  - compute a small set of smooth eigenfunctions
    - typically: a few hundreds
    - many kernels, e.g., heat-kernel, Laplacian
  - your solution becomes “smooth by construction”
  - permits to work with medium-size dense algebra
  - overheads: initialization, conversions, storage