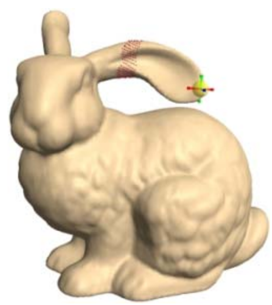
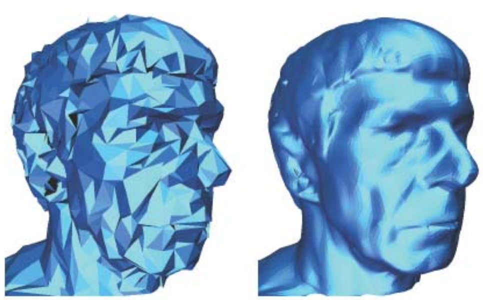
(Sparse) Linear Solvers

$$Ax = B$$

Why?

Many geometry processing applications boil down to:
solve one or more linear systems





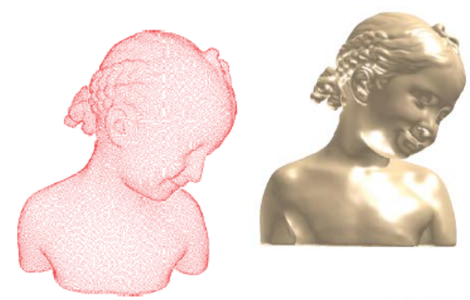
Parameterization

Editing

Reconstruction

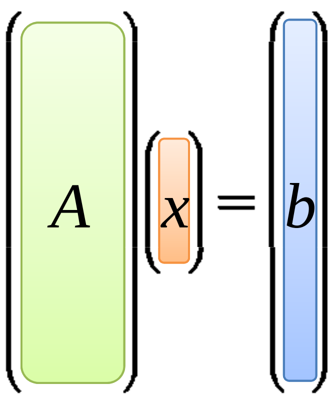
Fairing

Morphing

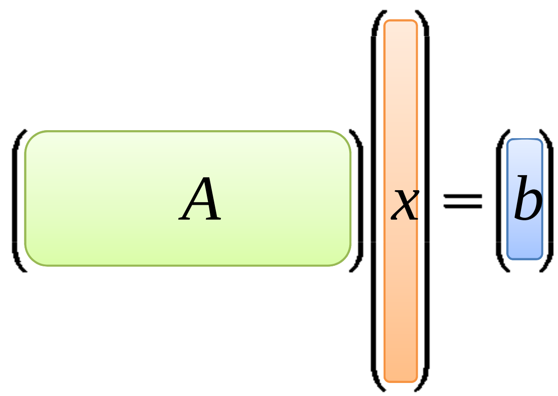


Don't you just invert A?

- Matrix not always invertible
  - Not square



Over determined



Under determined

  - Singular
    - Almost singular

Don't you just invert A?

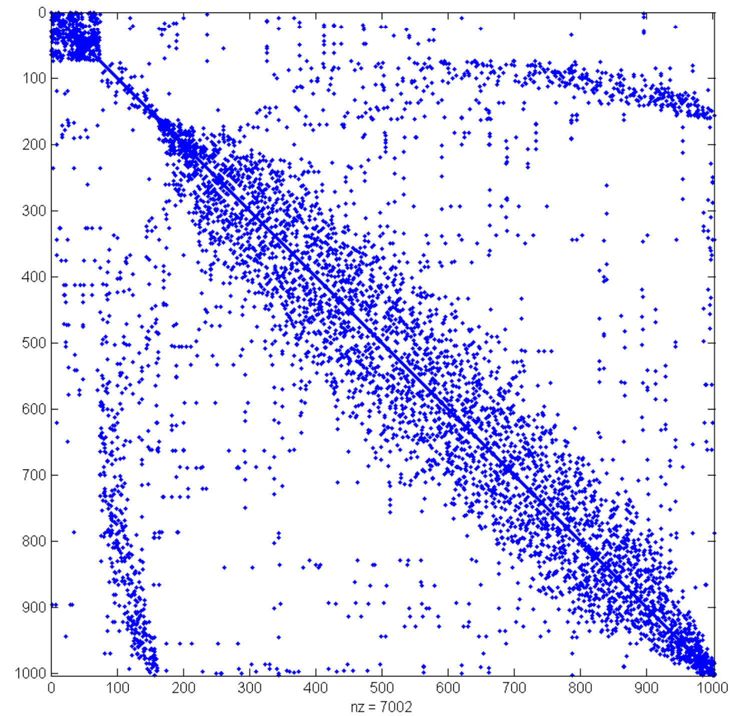
- Even if invertible
  - Very expensive $O(n^3)$
  - Usually n = number of vertices/faces

Problem definition

- Input
  - Matrix A_{mxn}
  - Vector B_{mx1}
- Output
  - Vector x_{nx1}
  - Such that $Ax = B$
- Small time and memory complexity
- Use additional information on the problem

Properties of linear systems for DGP

- Sparse A
  - Equations depend on graph neighborhood
  - Equations on vertices → 7 non-zeros per row on average
  - Number of non zero elements O(n)



n = 1002
Non zeros = 7002

Properties of linear systems for DGP

- Symmetric positive definite (positive eigenvalues)
  - Many times A is the Laplacian matrix
  - Laplacian systems are usually SPD

- A remains, b changes – many right hand sides
  - Interactive applications
  - Mesh geometry – same matrix for X, Y, Z

Linear solvers zoo

- **A is square and regular**

- Indirect solvers – iterative
  - Jacobi
  - Gauss-Seidel
  - Conjugate gradient

- Direct solvers – factorization
  - LU
  - QR
  - Cholesky

- Multigrid solvers

Jacobi iterative solver

- If all variables are known but one, its value is easy to find

$$a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{in}x_n = b_i$$

- Idea:
  - Guess initial values
  - Repeat until convergence
    - Compute the value of one variable assuming all others are known
    - Repeat for all variables

Jacobi iterative solver

- Let x^* be the exact solution of $Ax^* = b$

$$x_i^* = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_{ij}^*\right)$$

- Jacobi method
  - Set $x_i^{(0)} = 0$
  - While not converged
    - For $i = 1$ to n

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_{ij}^{(k)}\right)$$

Values from previous iteration

Jacobi iterative solver

Pros

- Simple to implement
  - No need for sparse data structure
- Low memory consumption $O(n)$
- Takes advantage of sparse structure
- Can be parallelized

Jacobi iterative solver

Cons

- Guaranteed convergence for strictly diagonally dominant matrices

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

  – The Laplacian is *almost* such a matrix

- Converges SLOWLY
  – Smoothens the error
  – Once the error is smooth, slow convergence

- Doesn't take advantage of
  – Same A, different b
  – SPD matrix

Direct solvers - factorization

- If A is diagonal/triangular the system is easy to solve

- Idea:
  – Factor A using "simple" matrices

$$Ax = A_1 \underbrace{A_2 \ldots A_k x}_{x_1} = b$$

  – Solve using k easy systems

$$A_1 x_1 = b \;\rightarrow\; A_2 x_2 = x_1 \;\rightarrow\; \ldots \;\rightarrow\; A_k x = x_{k-1}$$

Direct solvers - factorization

- Factoring harder than solving

- Added benefit – multiple right hand sides
  – Factor only once!

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_{ij}\right)$$

Solving easy matrices

Diagonal $\left(a_{ij} = 0, \quad i \neq j\right)$

$$\begin{pmatrix} \bullet & & & \\ & \bullet & & \\ & & \bullet & \\ & & & \bullet \end{pmatrix} x = b$$

$$x = \begin{pmatrix} \frac{b_1}{a_{11}} & \frac{b_2}{a_{22}} & \dots & \frac{b_n}{a_{nn}} \end{pmatrix}^T$$

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_{ij}\right)$$

Solving easy matrices

Lower triangular $\left(a_{ij} = 0, \quad j > i\right)$

$$\begin{pmatrix} \bullet & & & \\ \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{pmatrix} x = b$$

- Forward substitution
- Start from x_1

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j\right)$$

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij} x_{ij}\right)$$

Solving easy matrices

Upper triangular $\left(a_{ij} = 0, \quad j < i\right)$

$$\begin{pmatrix} \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet \\ & & \bullet & \bullet \\ & & & \bullet \end{pmatrix} x = b$$

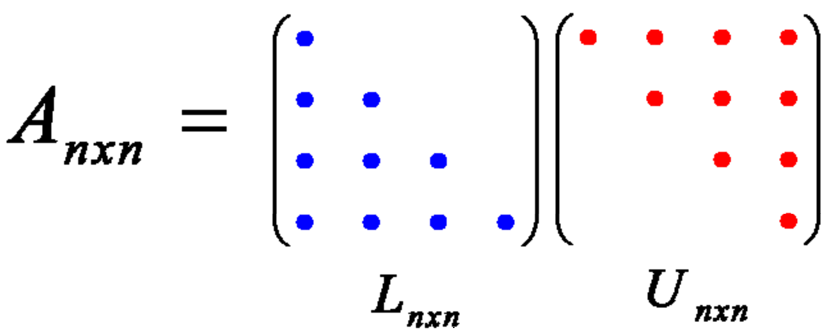
- Backward substitution
- Start from x_n

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j>i} a_{ij} x_j\right)$$

LU factorization

- $A = LU$
  - L lower triangular
  - U upper triangular

- Exists for any non-singular square matrix

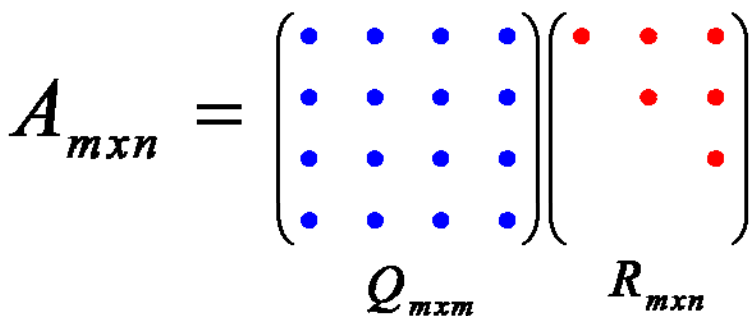


- Solve using $Lx_1 = b$ and $Ux = x_1$

QR factorization

- $A = QR$
  - Q orthogonal → $Q^T = Q^{-1}$
  - R upper triangular

- Exists for any matrix



- Solve using $Rx = Q^Tb$

Cholesky factorization

- $A = LL^T$
  - L lower triangular

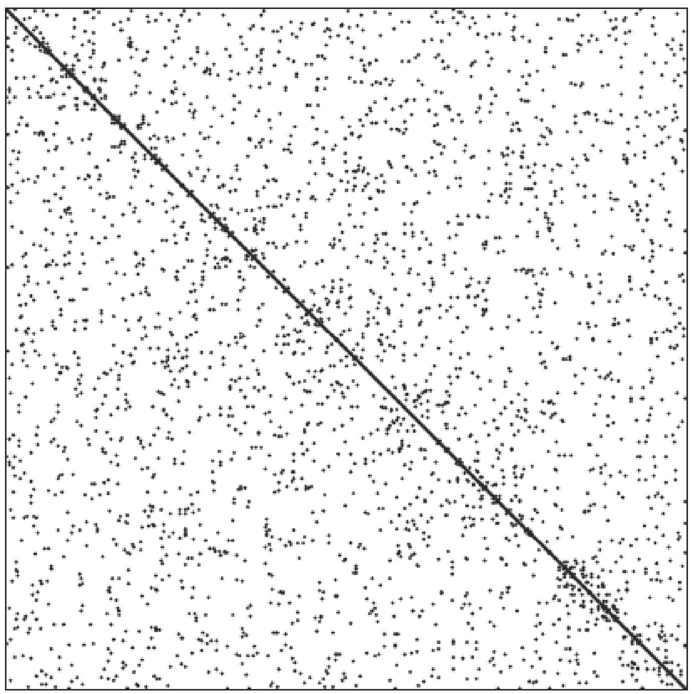
- Exists for square symmetric positive definite matrices

$$A_{nxn} = \underbrace{\begin{pmatrix} \bullet & & & \\ \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \\ \bullet & \bullet & \bullet & \bullet \end{pmatrix}}_{L_{nxn}} \underbrace{\begin{pmatrix} \bullet & \bullet & \bullet & \bullet \\ & \bullet & \bullet & \bullet \\ & & \bullet & \bullet \\ & & & \bullet \end{pmatrix}}_{L^T_{nxn}}$$

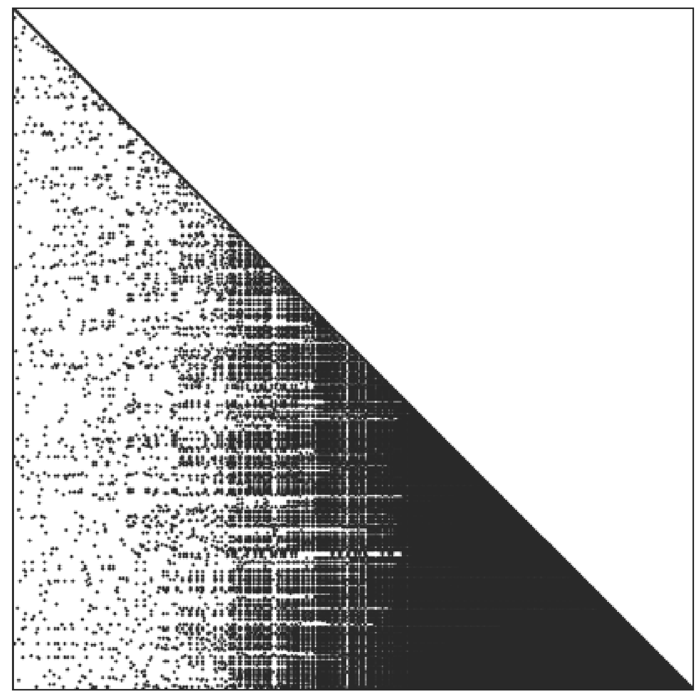
- Solve using $Lx_1 = b$ and $L^T x = x_1$

Direct solvers – sparse factors?

- Usually factors are **not** sparse, even if A is



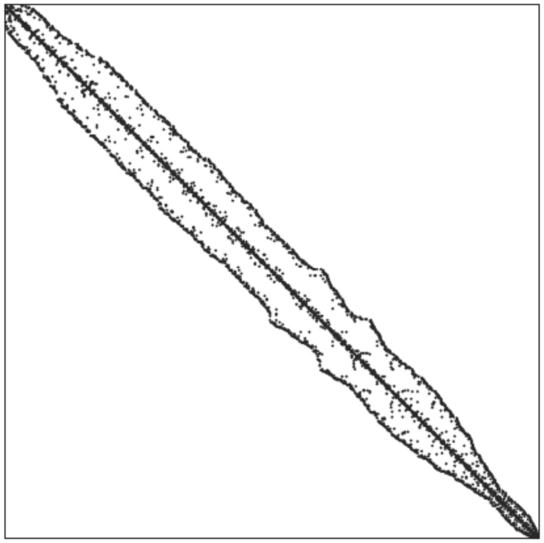
Sparse matrix



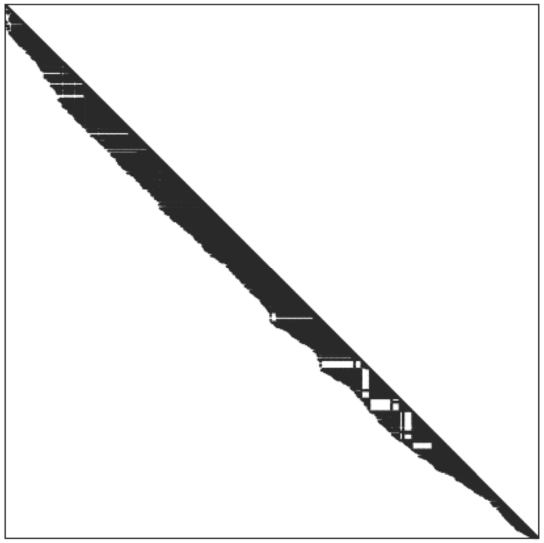
Cholesky factor

Sparse direct solvers

- Reorder variables/equations so factors are sparse
- For example
  - Bandwidth is preserved → Reorder to minimize badnwidth



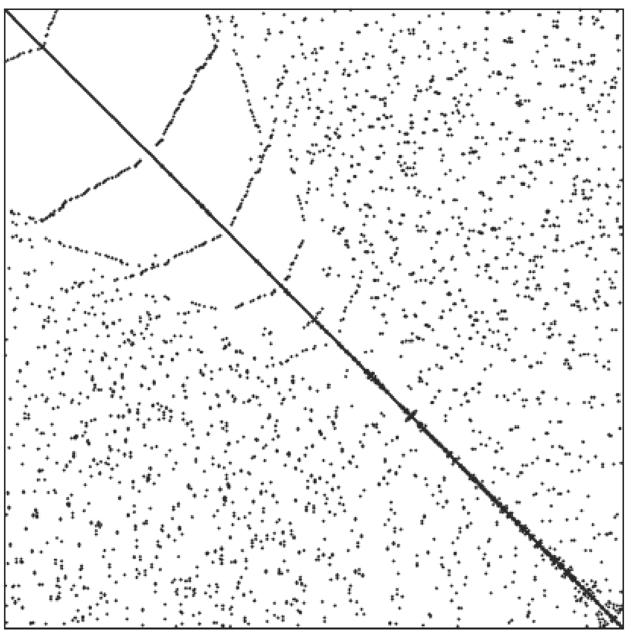
Sparse matrix
bounded bandwith



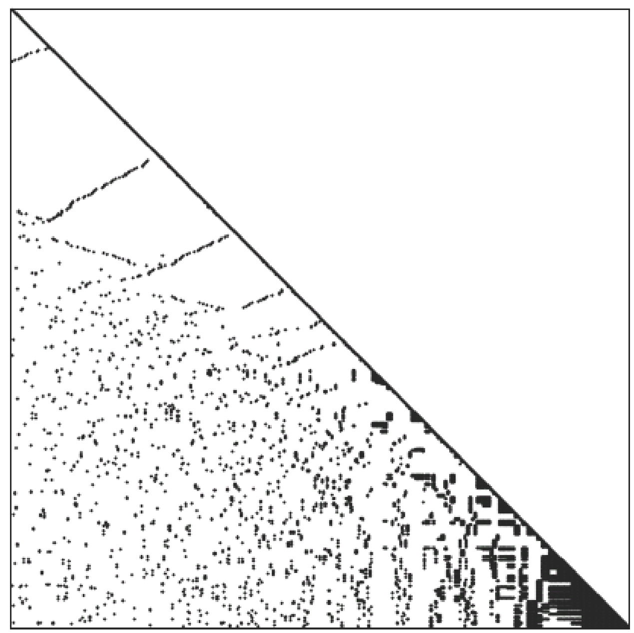
Cholesky factor

Sparse direct solvers

- SPD matrices
  - Cholesky factor sparsity pattern can be derived from matrix' sparsity pattern
    - Reorder to minimize new non zeros (*fill in*) of factor matrix



Sparse matrix - reordered



Cholesky factor

Sparse Cholesky

- Symbolic factorization
  - Can be reused for matrix with same sparsity structure
    - Even if values change!
- Only for SPD matrices
- Reordering is heuristic – not always gives good sparsity
  - High memory complexity if reordering was bad
- BUT
  - Works extremely well for Laplacian systems
  - Can solve systems up to n = 500K on a standard PC

Under-determined systems

- System is under-constrained → too many variables

$$\begin{pmatrix} & & A & & \end{pmatrix}_{mxn} \begin{pmatrix} \\ x \\ \\ \end{pmatrix}_{nx1} = \begin{pmatrix} B \end{pmatrix}_{mx1}$$

- Solution: add constraints by pinning variables
  – Add equations $x_1 = c_1$, $x_2 = c_2$, … , $x_k = c_2$

  Or

  – Replace variables with constants in equations
    - Better – smaller system

Over-determined systems

- System is over-constrained → too many equations

$$\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix}_{mxn} \begin{pmatrix} x \end{pmatrix}_{nx1} = \begin{pmatrix} B \end{pmatrix}_{mx1}$$

- Unless equations are dependent, no solution exists
- Instead, minimize: $\left\| Ax - b \right\|^2$

Over-determined systems

- The minimizer of

$$\left\| Ax - B \right\|^2$$

- Is the solution of

$$A^T Ax = A^T B$$

- A^TA is symmetric positive definite
  - Can use Cholesky

Singular square matrices

- Equivalent to under-determined system
- Add constraints
  - By changing variables to constants
  - NOT by adding equations!
    - Will make system not square
    - Much more expensive

Conclusions

- Many linear solvers exist
- Best choice depends on A
- DGP algorithms generate sparse SPD systems
- Research shows sparse linear solvers are best for DGP
- Sparse algs exist also for non-square systems
- If solving in Matlab – use “\” operator

References

- “Efficient Linear System Solvers for Mesh Processing”, Botsch et al., 2005