Numerical Methods I

MATH-GA 2010.001/CSCI-GA 2420.001

Benjamin Peherstorfer
Courant Institute, NYU

Based on slides by G. Stadler and A. Donev

Today

Last time

- Solving systems of linear equations
- LU decomposition

Today

- Recap: linear systems. Sparse matrices
- Linear least square problems
- Geometric perspective on the normal equations
- Orthogonalization with Gram-Schmidt

Announcements

- Homework 2 has been posted, due Tue (!), Oct 11 before class
- Final exam on Mon, Dec 19, 5.10 - 7.00pm
- Office hour: Tue, 1pm

Gauss elimination and LU factorization

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & a_{23}^{(1)} \\ a_{31}^{(1)} & a_{32}^{(1)} & a_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}$$

index of step

multiply first row by ℓ_{i1} and subtract from 2nd row $\longrightarrow$

$$\ell_{21} = \frac{a_{21}^{(1)}}{a_{11}^{(1)}}$$

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix}$$

eliminate using

$$\ell_{32} = \frac{a_{32}^{(2)}}{a_{22}^{(2)}}$$

$$\longrightarrow \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{12}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \end{bmatrix}$$

u_{11}	u_{12}	u_{13}
ℓ_{21}	u_{22}	u_{23}
ℓ_{31}	ℓ_{32}	u_{33}

Idea: Store multipliers $\ell_{21}, \ell_{31}, \ell_{32}$ in matrix:

$$\longrightarrow \begin{bmatrix} L \backslash U \end{bmatrix}$$

$$A = LU, \quad L = \begin{bmatrix} 1 & & \\ \ell_{21} & 1 & \\ \ell_{31} & \ell_{31} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & \dots & u_{13} \\ 0 & & \vdots \\ 0 & 0 & \end{bmatrix}$$

LU with pivoting

Zero diagonal entries (pivots) pose a problem ⇝ pivoting by swapping rows

⇝ board
$$\begin{bmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 6 & 4 \end{bmatrix} \xrightarrow{\text{Gauss elimination}} \begin{bmatrix} 1 & 1 & 3 \\ l_{21}=2 & 0 & -4 \\ l_{31}=3 & 3 & -5 \end{bmatrix}$$

Problem! Cannot divide by zero, pivot element is zero.

$$\text{Swap rows} \xrightarrow{} 2 \,\&\, 3 \quad \begin{bmatrix} 1 & 1 & 3 \\ 3 & 3 & -5 \\ 2 & 0 & -4 \end{bmatrix}$$

done – in general, use 3 as pivot.

$$\boxed{LU = PA}$$

P... permutation matrix.

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Theorem: This always exists with appropriate perm. matrix P

Recap

- For any square (regular or singular) matrix $\boldsymbol{A}$, partial (row) pivoting ensures exists of
$$\underline{\boldsymbol{PA} = \boldsymbol{LU}}$$
where $\boldsymbol{P}$ is a permutation matrix
- Pivoting (w.r.t. $\max |a_{ij}|$) leads to a backward stable algorithm.
- There also is full pivoting (rows + columns)
$$\boldsymbol{PAQ} = \boldsymbol{LU}$$
to further increases stability but it usually is not worth it in practice (higher costs to search for pivoting element over rows and columns but little improvement in terms of stability)

Recap: Solving linear systems

- Once an LU factorization is available, solving a linear system is cheap:

$$\underline{\boldsymbol{A}\boldsymbol{x}} = \underline{\boldsymbol{L}\boldsymbol{U}}\boldsymbol{x} = \underline{\boldsymbol{L}}(\underline{\boldsymbol{U}\boldsymbol{x}}) = \boldsymbol{L}\underline{\boldsymbol{y}} = \underline{\boldsymbol{b}}$$

- Solve for $\boldsymbol{y}$ using *forward substitution*
- Solve for $\boldsymbol{x}$ by using *backward substitution* $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$

Forward/backward substitution requires

$$\frac{n(n-1)}{2} \text{ multiplications/additions,}$$
$$n \text{ divisons.}$$

Overall: $\sim n^2$ floating point operations (flops) $\rightsquigarrow$ costs scale as $\mathcal{O}(n^2)$

Recap: Costs

For forward [backward] substitution at step k there are $\approx k$ $[(n-k)]$ multiplications and subtractions plus a few divisions. The total over all n steps is

$$\sum_{k=1}^{n} k \in \mathcal{O}(n^2)$$

$\rightsquigarrow$ the number of floating-point operations (FLOPs) scales as $\mathcal{O}(n^2)$

For Gaussian elimination, at step k, there are $\approx (n-k)^2$ operations. Thus, the total scales as

$$\sum_{k=1}^{n} (n-k)^2 \in \mathcal{O}(n^3)$$

Summary:

- Directly applying Gaussian elimination (=LU + fwd/bwd) scales as $\mathcal{O}(n^3)$
- Computing LU decomposition scales as $\mathcal{O}(n^3)$
- Forward/backward substitution scales as $\mathcal{O}(n^2)$
- LU + forward/backward scales as $\mathcal{O}(n^3)$ $\rightsquigarrow$ can *reuse LU* for other b

Recap: Summary

- The condition of solving a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is determined by the condition number of the matrix $\boldsymbol{A}$

$$\underline{\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\|\|\boldsymbol{A}^{-1}\|} \geq 1$$

- Gaussian elimination can be used to solve general square linear systems and produces a factorization, if it exists

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

- Partial pivoting is sufficient for existence and stability of the LU decomposition

$$\boldsymbol{P}\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}\,, \qquad \boldsymbol{A} = \tilde{\boldsymbol{L}}\boldsymbol{U}$$

- The Cholesky factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ exists if $\boldsymbol{A}$ is spd and then it is the better choice (stabler, cheaper) than LU

- Rely on the highly optimized routines in Matlab (LAPACK) and other software packages than implementing these algorithms yourself $\rightsquigarrow$ take the course on HPC next spring to learn more about the efficient *implementation* of these algorithms

Sparse matrix

- A matrix where a substantial fraction of the entries are zero is called a sparse matrix. Typically, only $\underline{\mathcal{O}(N)}$ non-zero entries are allowed.
- If we have only $\mathcal{O}(N)$ non-zero entries, then store only those; in contrast to dense matrices. Exploiting sparsity is important (life saving) for large matrices
- The structure of a sparse matrix refers to the set of indices i, j such that $|a_{ij}| > 0$ and is visualized in Matlab with `spy`
- The sparsity structure is the most important property that determines whether an efficient method exists
- The structure of sparse matrices comes from the nature of the problem

Banded matrices

Banded matrices are a very special but common type of sparse matrices, e.g., tridiagonal matrices

$$\begin{bmatrix} a_1 & c_1 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ 0 & & b_n & a_n \end{bmatrix}$$

For example, think of the Laplace problem $u''(x) = f(x)$ on the unit interval and a finite-difference discretization

$$u''(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2}$$

on an equidistant grid. This leads to a system of equations with a tridiagonal matrix
$\rightsquigarrow$ Numerical Methods II (Spring semester)

There exist special techniques for banded matrices that are much faster than the general case, e.g., only $8n$ FLOPs

Decomposing sparse matrices

There also are general methods for dealing with sparse matrices, such as the sparse LU factorization.

How well they work depends on the structure of the matrix. What could go wrong?

Decomposing sparse matrices

There also are general methods for dealing with sparse matrices, such as the sparse LU factorization.

How well they work depends on the structure of the matrix. What could go wrong?

When factorizing sparse matrices, the factors, e.g., L and U, can be much less sparse than $A \rightsquigarrow$ fill-in

For many sparse matrices, there is a large fill-in

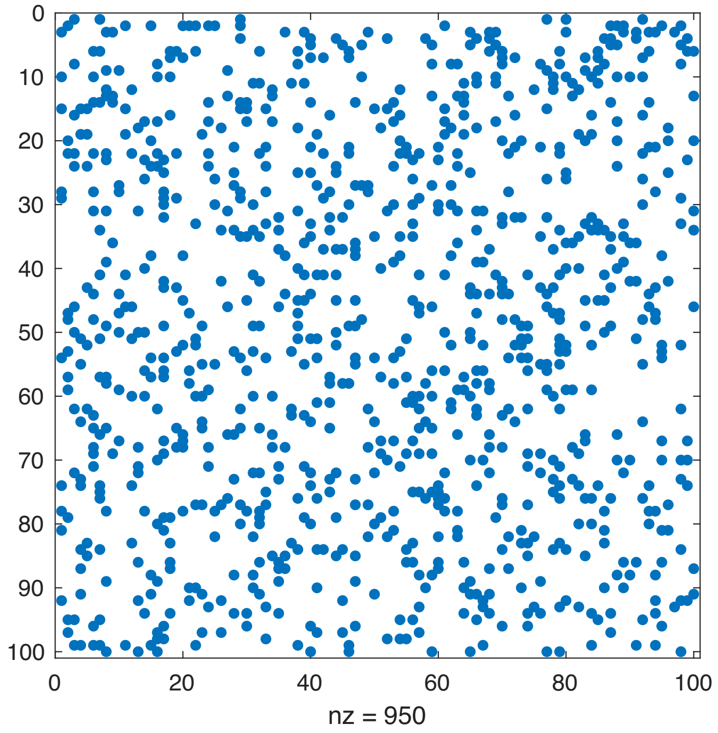
- Pivoting can help to reduce fill-in
- However, often "good" pivoting for sparsity leads to less stable behavior and vice versa

Sparse matrices in Matlab

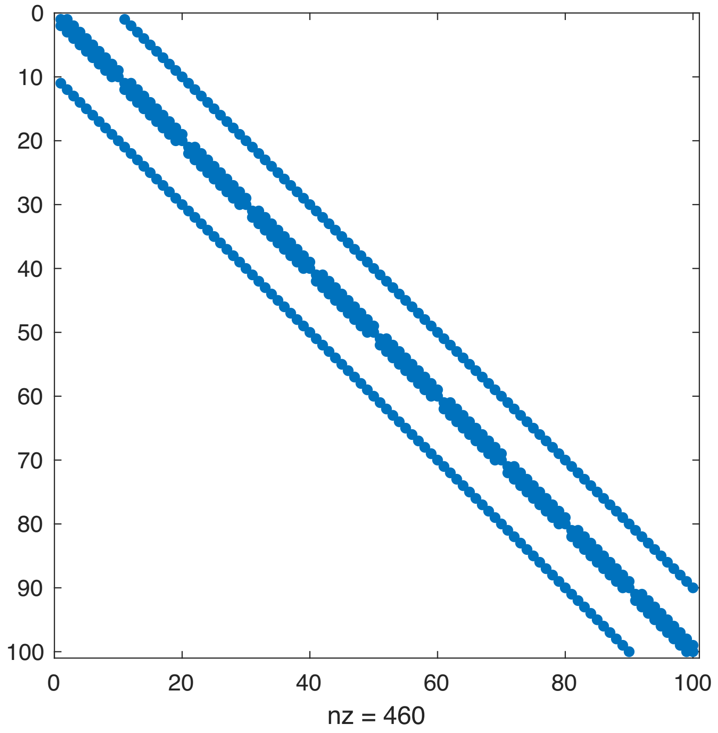
```
% S = sparse(i,j,v) generates a sparse matrix S
% from the triplets i, j, v with S(i(k),j(k)) = v(k).
>> A = sparse([1 2 2 4 4], [3 1 4 2 3], 1:5)
A =
     (2,1)          2
     (4,2)          4
     (1,3)          1
     (4,3)          5
     (2,4)          3
>> whos A
  Name        Size                Bytes  Class
     Attributes
  A           4x4                   120  double     sparse
>> nnz(A)
ans =
     5
```

```
% S = sparse(i,j,v,m,n,nz) allocates space for nz nonzero elements.
% Use this syntax to allocate extra space for nonzero values to be filled in after construction.
>> A = sparse([], [], [], 4, 4, 5);
>> A(2, 1) = 2; A(4, 2) = 4; A(1, 3) = 1; A(4, 3) = 5; A(2, 4) = 3;
>> full(A)

ans =

     0     0     1     0
     2     0     0     3
     0     0     0     0
     0     4     5     0
```

```
% generate a random sparse matrix with density 10% and
    size 100x100
>> B = sprand(100, 100, 0.1);
% the sparse block tridiagonal matrix of order n^2
    resulting from discretizing Poisson's equation with
    the 5-point operator on an n-by-n mesh.
>> X = gallery('poisson', 10);
>> spy(B);
>> spy(X);
```

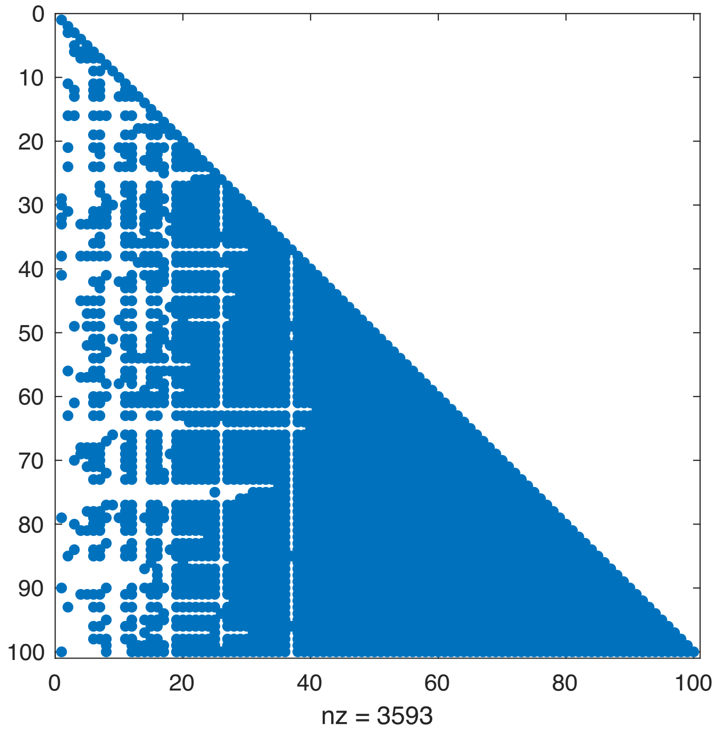


(a) matrix B

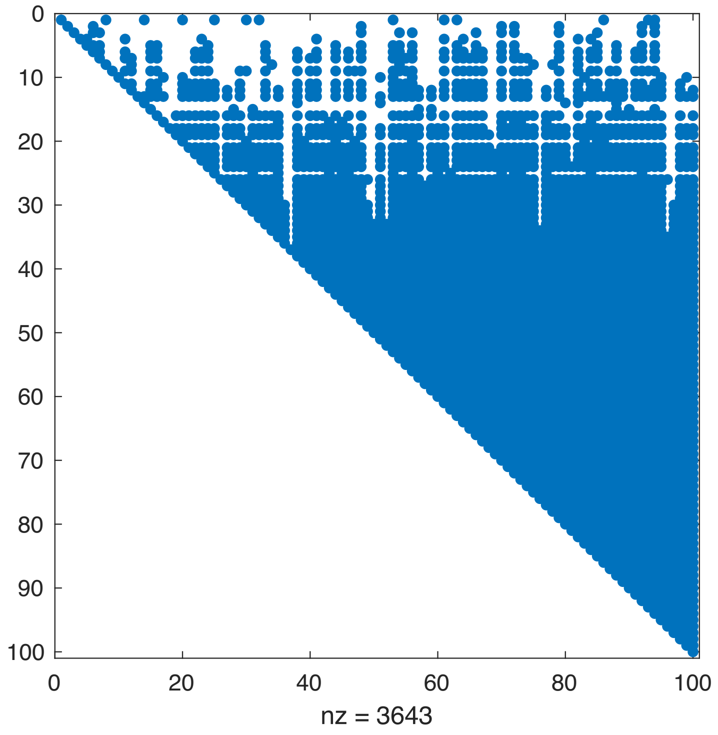


(b) matrix X

```
>> [L, U, P] = lu(B);
>> spy(L);
>> spy(U);

>> [L, U, P] = lu(X);
>> spy(L);
>> spy(U);
```

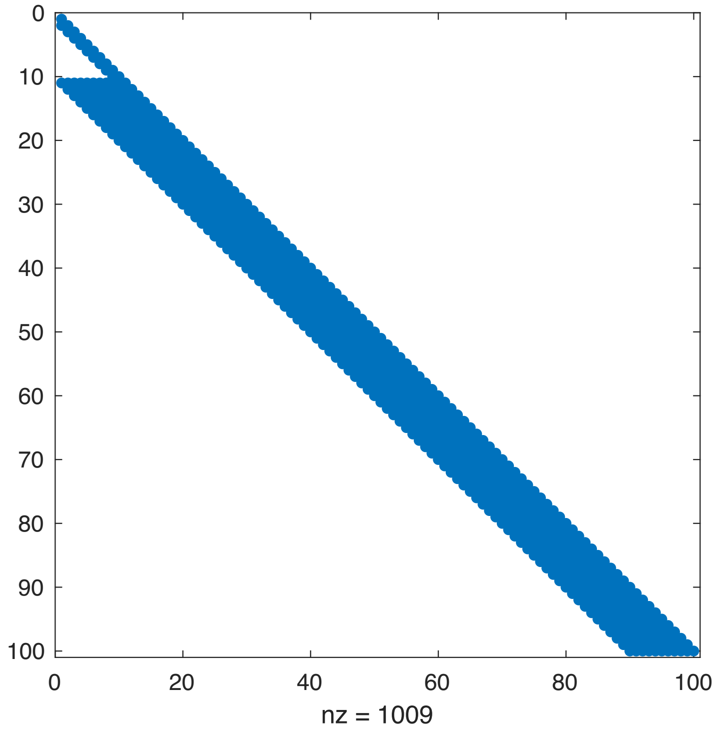


(a) factor L of B

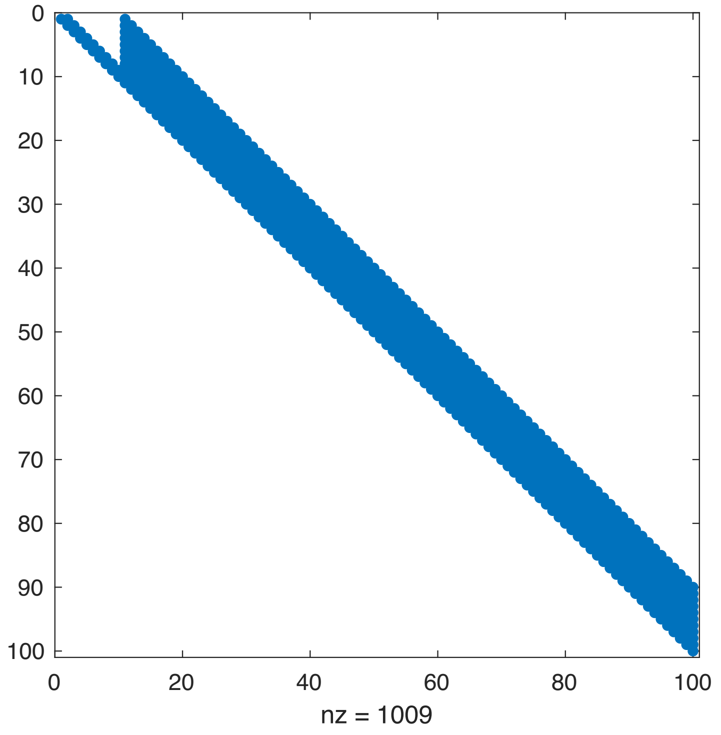


(b) factor U of B

A lot of fill-in! Factors L and U are not sparse, even though matrix B is sparse



(a) factor L of X



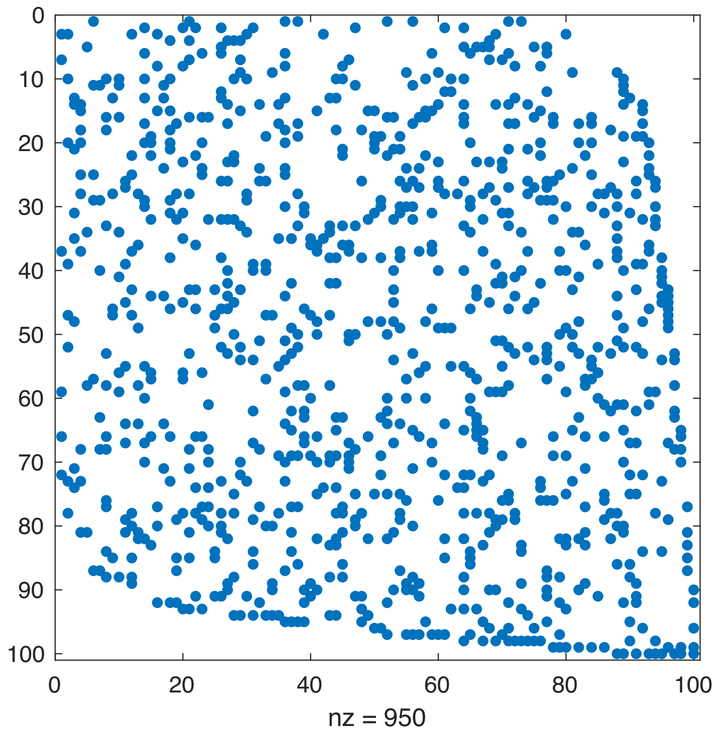
(b) factor U of X

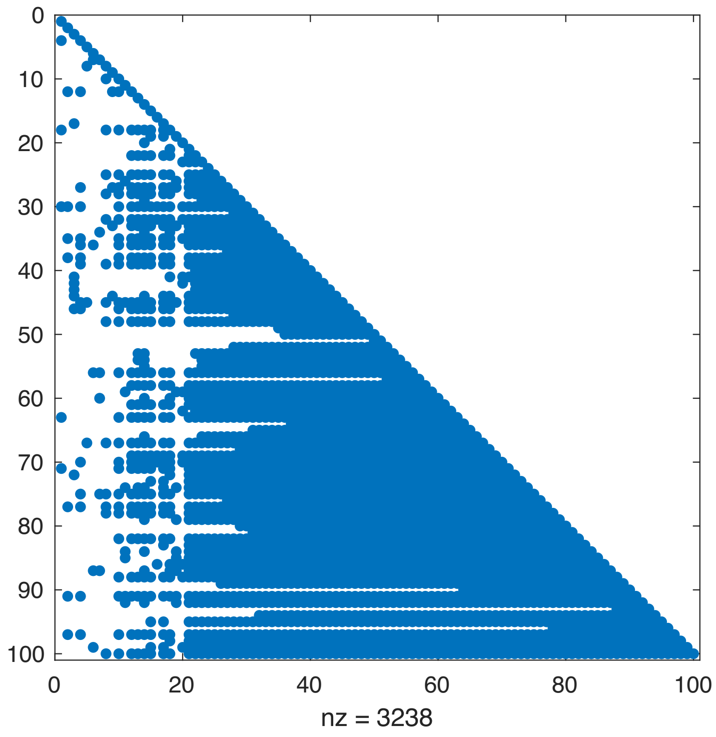
Though better than for matrix B with random sparsity structure, there still are many more non-zero entries in the factors of X than in X itself.

Changing the sparsity structure via re-ordering the matrix can help to reduce fill-in. For example, in Matlab the sparse reverse Cuthill-McKee ordering is implemented.

The re-ordered matrix tends to have its nonzero elements closer to the diagonal. This is a good preordering for LU or Cholesky factorization of matrices.

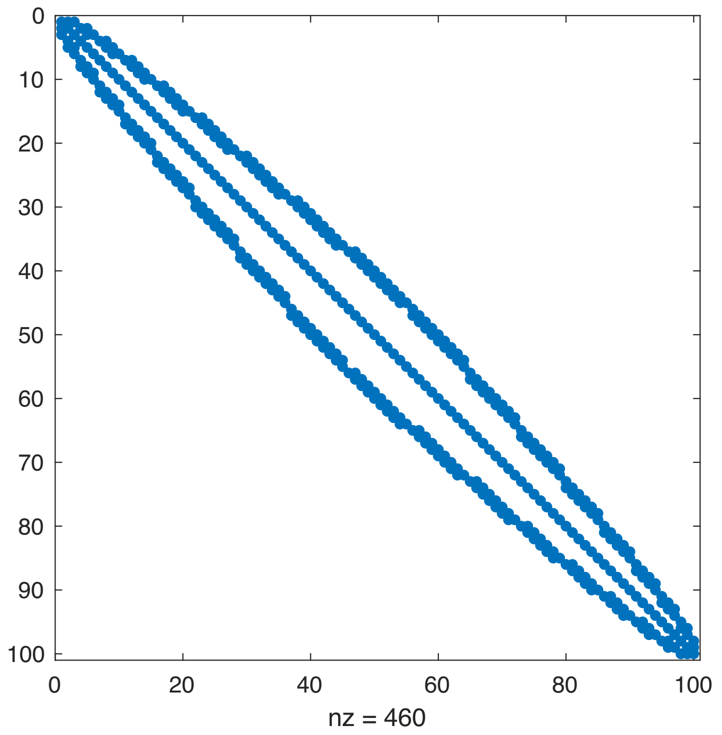
```
>> p = symrcm(B);
>> spy(B(p, p));
>> [L, U, P] = lu(B(p, p));
>> spy(L);
>> spy(U);

>> p = symrcm(X)
>> spy(X(p, p));
>> [L, U, P] = lu(X(p, p));
>> spy(L);
>> spy(U);
```



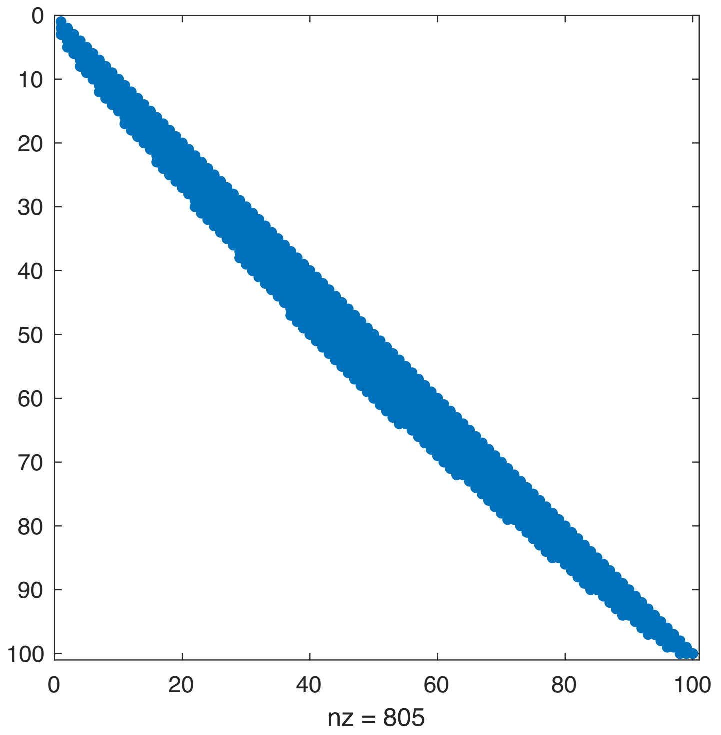


(a) re-ordered B matrix (b) factor L of re-ordered B

- Notice how the non-zero elements tend to be closer to the diagonal of the re-ordered B compared to the original B
- The fill-in is reduced from ~ 3500 to ~ 3200 non-zero entries; reduction of $< 10\%$
- It is hard to find a good ordering for matrix with a random sparsity structure



(a) re-ordered X matrix



(b) factor L of re-ordered X

- Non-zero entries of re-ordered X are closer to the diagonal than for the original X
- Fill-in is reduced by a roughly 20% from ~ 1000 to ~ 800 non-zero entries

Conclusions/summary

While there are general techniques for dealing with sparse matrices that help greatly, it all depends on the structure of the matrix

Pivoting has a dual, sometimes conflicting goal:

1. Reduce fill-in, i.e., improve memory use: Still active subject of research!
2. Reduce roundoff errors, i.e., improve stability. Typically some threshold pivoting is used only when needed

For many sparse matrices *iterative methods* (later) are required when large fill-in.

Least-squares problems

Least-squares problems

Given data points/measurements

$$\boxed{(t_i, b_i),} \quad i = 1, \dots, m$$

and a model function ϕ that relates t and b:

$$b = \phi(t; x_1, \dots, x_n),$$

where $x_1, \dots, x_n$ are model function parameters. If the model is supposed to describe the data, the deviations/errors

$$\Delta_i = b_i - \boxed{\phi(t_i, x_1, \dots, x_n)}$$

should be small. Thus, to fit the model to the measurements, one must choose $x_1, \dots, x_n$ appropriately.

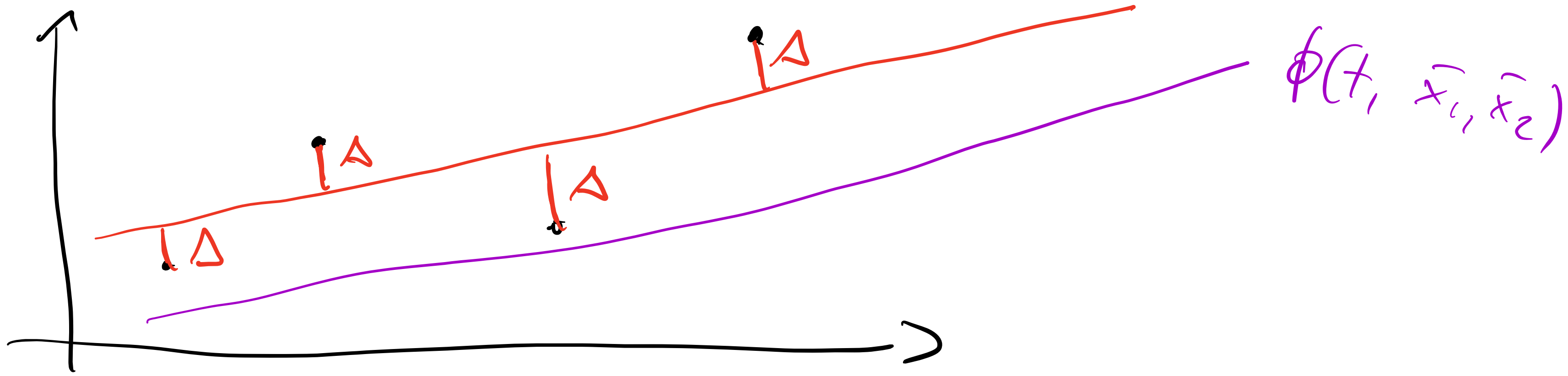
Least-squares problems

data $\{(t_i, b_i)\}_{i=1}^{m}$

parametrization

$$b = \phi(t; x_1, \ldots, x_n)$$

$$\underline{\Delta_i} = b_i - \phi(t_i, x_1, \ldots, x_n)$$



Least-squares problems

Least squares: Find $x_1, \ldots, x_n$ such that

$$\boxed{\frac{1}{2}\sum_{i=1}^{m} \Delta_i^2 \to \min}$$

Weighted least squares: Find $x_1, \ldots, x_n$ such that

$$\underline{\frac{1}{2}\sum_{i=1}^{m} \left(\frac{\Delta_i}{\boxed{\delta b_i}}\right)^2} \to \min,$$

where $\delta b_i > 0$ contain information about how much we trust the ith data point.

Least-squares problems (cont'd)

Alternatives to using squares:

L^1 error: Find $x_1, \ldots, x_n$ such that

$$\underline{\sum_{i=1}^{m} |\Delta_i| \to \min}$$

Result can be very different, other statistical interpretation, more stable with respect to outliers.

$\underline{L^\infty \text{ error:}}$ Find $x_1, \ldots, x_n$ such that

$$\underline{\max_{1 \leq i \leq m} |\Delta_i| \to \min}$$

Keeps the worst-case error small (risk averse)

Linear least-squares

We assume (for now) that the model depends linearly on $x_1, \ldots, x_n$, e.g.:

$$\phi(t; x_1, \ldots x_n) = \boxed{a_1(t)}x_1 + \ldots + a_n(t)x_n$$

$\rightsquigarrow$ board

$$\phi(t; x_1, x_2) = x_1 t^2 + x_2 \exp(t)$$

$$\Delta_1 = b_1 - (x_1 t_1^2 + x_2 \exp(t_1))$$

$$\Delta_2 = b_2 - (x_1 t_2^2 + x_2 \exp(t_2))$$

$$\Delta_3 = b_3 - (x_1 t_3^2 + x_2 \exp(t_3))$$

$$\min_{x_1, x_2} \sum_{i=1}^{3} \Delta_i^2 \rightsquigarrow \text{linear lst sq problem}$$

Linear least-squares

Choosing the least square error, this results in

$$\min_{\boldsymbol{x}} \|\underbrace{A\boldsymbol{x} - \boldsymbol{b}}\|^2,$$

where $\boldsymbol{x} = (x_1, \dots, x_n)^T$, $\boldsymbol{b} = (b_1, \dots, b_m)^T$, and $a_{ij} = a_j(t_i)$.

In the following, we study the overdetermined case, i.e., $m \geq n \rightsquigarrow$ board

$$A = \underbrace{\begin{bmatrix} t_1^2 & \exp(t_1) \\ t_2^2 & \exp(t_2) \\ t_3^2 & \exp(t_3) \end{bmatrix}}_{\in \mathbb{R}^{3\times 2}}, \quad b = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}}_{\in \mathbb{R}^3}, \quad x = \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\in \mathbb{R}^2}$$

Linear least-squares

Different perspective:

$m > n$

Consider non-square matrices $A \in \mathbb{R}^{\underline{m \times n}}$ with $\underline{m \geq n}$ and $\underline{\text{rank(A)} = n}$. Then the system

$$A\boldsymbol{x} = \boldsymbol{b}$$

does, in general, not have a solution (more equations than unknowns). We thus instead solve a minimization problem

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 = \min_{\boldsymbol{x}} \Phi(\boldsymbol{x})$$

How can we solve this optimization problem?

Because we consider the Euclidean norm $\|.\|_2$, we obtain

$$\underline{\Phi(\boldsymbol{x})} = \underline{(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})} = \underline{\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \ldots}$$

which is quadratic in $\boldsymbol{x}$ if $\underline{\boldsymbol{A} \text{ has full rank}}$ $\rightsquigarrow$ $\underline{\text{convex in } \boldsymbol{x}}$

Therefore, the critical point is the global optimum

$$\underline{\nabla\Phi(\boldsymbol{x})} = \underline{\boldsymbol{A}^T(2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}))} = \underline{\mathbf{0}}$$

which satisfies the *normal equations*

$$\boxed{\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}.}$$

If $\boldsymbol{A}$ is full rank, $\underline{\text{rank}(\boldsymbol{A}) = n}$, then $\underline{\boldsymbol{A}^T\boldsymbol{A}}$ is positive definite and the normal equations can be solved with the Cholesky factorization (warning: we shouldn't do this, can you already see why?)

A geometry perspective on the normal equations

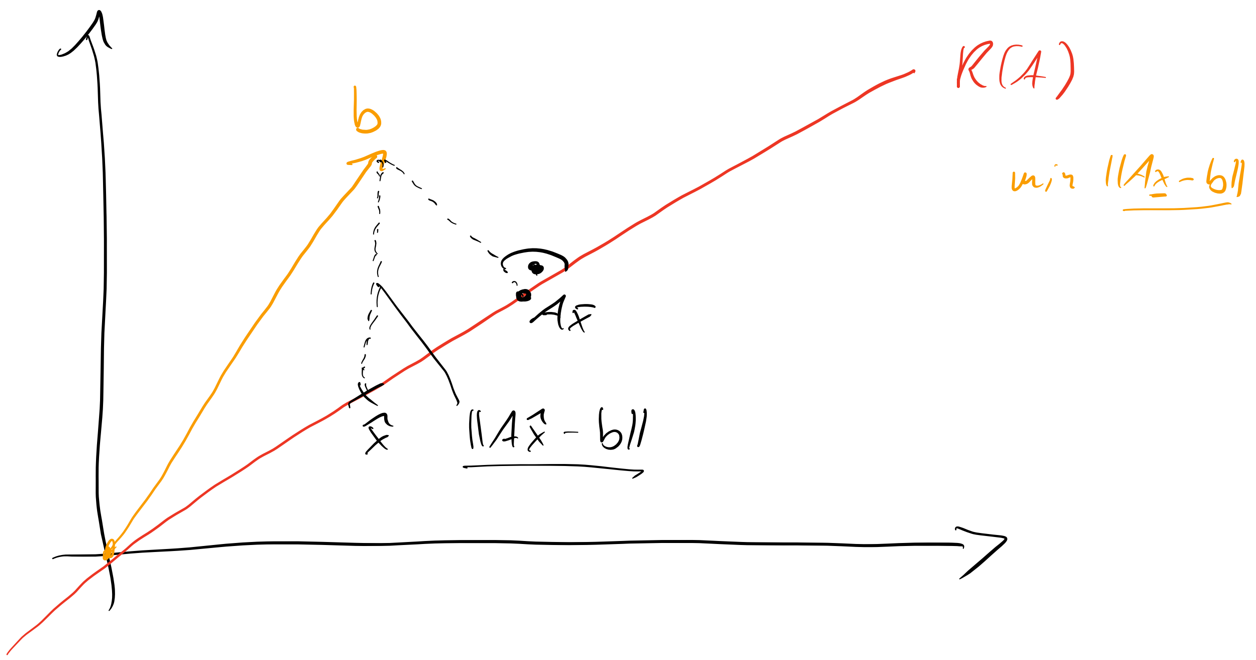
$$\min_{x \in \mathbb{R}^m} \|Ax - b\|_2^2$$

geometric sketch

$$\bar{x} = \arg\min_{x \in \mathbb{R}^m} \|Ax - b\|_2^2$$

$$\Downarrow$$

$$A\bar{x} - b \perp R(A) \Leftarrow \{Ax \mid x \in \mathbb{R}^n\}$$



Orthogonal projection

Finite-dim space $\underline{V}$ with $\langle \cdot, \cdot \rangle$ and subspace $\underline{U} \subset V$. Let $U^\perp$ be the orthogonal complement w.r.t. V

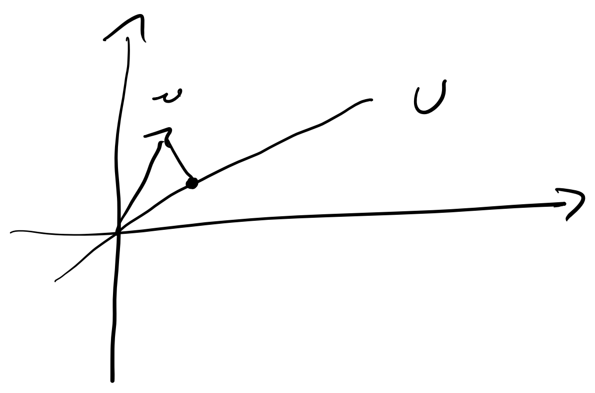
$$U^\perp = \{ \underline{v} \in V \mid \langle v, u \rangle = 0 \quad \forall u \in U \}$$

Then

$$\underset{u' \in U}{\arg\min} \| u' - \underline{v} \| = u \iff v - u \in U^\perp$$

Proof: let $u \in U$ such that $\underline{v - u} \in U^\perp$ for all $u' \in U$

$$\| v - u' \|^2 = \| v - u + u - u' \|^2$$
$$= \| v - u \|^2 + \| u - u' \|^2 \underset{\text{iff } u \neq u'}{>} \| v - u \|^2$$



$u = P(v)$

P is linear in v

thus

$u = Pv$

$$\bar{x} = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \|Ax - b\|_2^2$$

$V = \mathbb{R}^m, \quad U = R(A) \subset V$

$$\|b - Ax\|^2 = \min \iff \langle b - Ax, Ax' \rangle = 0 \quad \forall x' \in \mathbb{R}^n$$

$$\iff \langle A^T(b - Ax), \underline{x}' \rangle = 0$$

$$\iff \boxed{A^T b = A^T A x}$$

$$\iff \underline{x} = (A^T A)^{-1} A^T b$$

$$Ax = Pb = A(A^T A)^{-1} A^T b \qquad x = (A^T A)^{-1} A^T b$$

Linear least-squares problems

Solving the normal equations

$$A^T A\bar{\boldsymbol{x}} = A^T \boldsymbol{b}$$

requires:

- computing $A^T A$ (which is $O(mn^2)$)
- condition number of $A^T A$ is square of condition number of A; (problematic for the Choleski factorization)

$\rightsquigarrow$ derive condition of $A^T A$ on board

$$k_2(A^TA) = k_2(A)^2$$

$$k_2(A)^2 = \|A\|_2^2 \, \|A^{-1}\|_2^2 = \frac{\max\limits_{\|x\|=1} \|Ax\|_2^2}{\min\limits_{\|x\|=1} \|Ax\|_2^2}$$

$$= \frac{\max\limits_{\|x\|=1} \langle Ax, Ax \rangle}{\min\limits_{\|x\|=1} \langle Ax, Ax \rangle}$$

$$= \frac{\max\limits_{\|x\|=1} \boxed{\langle A^TAx, x \rangle}}{\min\limits_{\|x\|=1} \langle A^TAx, x \rangle}$$

$$= \frac{\lambda_{\max}(A^TA)}{\lambda_{\min}(A^TA)}$$

A^TA is spd

$$\lambda_{\max}(A^TA) = \sqrt{\lambda_{\max}(A^TA)^2} \overset{\text{spd}}{=} \sqrt{\lambda_{\max}((A^TA)^2)}$$

$$= \sqrt{\lambda_{\max}((A^TA)^T(A^TA))}$$

$$= \|A^TA\|_2$$

$$\frac{1}{\lambda_{\min}(A^TA)} = \|(A^TA)^{-1}\|_2$$

$$\Rightarrow k_2^2(A) = \|A^TA\|_2 \, \|(A^TA)^{-1}\|_2 = k_2(A^TA)$$

Linear least-squares problems

Conditioning

Solving the normal equation is equivalent to computing $P\boldsymbol{b}$, the orthogonal projection of $\boldsymbol{b}$ onto the subspace V spanned by columns of A.

Let $P : \mathbb{R}^m \to V$ be an orthogonal projection onto $V \subseteq \mathbb{R}^n$. For $b \in \mathbb{R}^m$, denote by θ the angle between b and V defined by

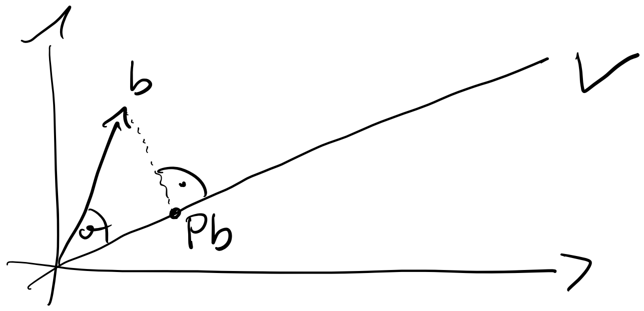
$$\sin(\theta) = \frac{\|b - Pb\|_2}{\|b\|_2} \,.$$

The the relative condition number of projecting b onto V with P with respect to the 2-norm is

$$\kappa_{\text{rel}}(b) = \frac{1}{\cos(\theta(b))} \|P\|_2 \,.$$

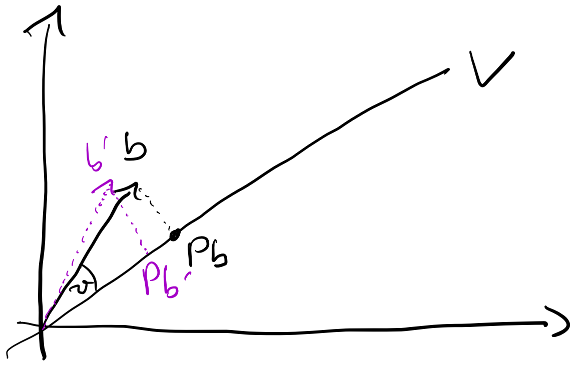
$P : \mathbb{R}^m \to V$

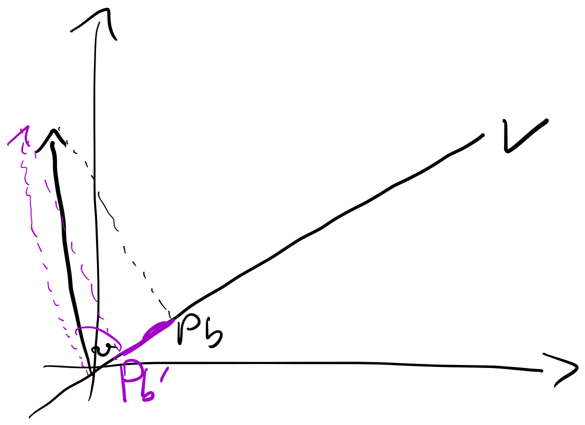
$$\underline{\sin(\vartheta)} = \frac{\|b - Pb\|_2}{\|b\|_2}$$



Relative condition number of (P, b) w.r.t. 2-norm

$$\boxed{\kappa_{rel} = \boxed{\frac{1}{\cos(\vartheta)}} \|P\|_2}$$





$$\kappa_{rel} = \frac{\|x\|}{\|f(x)\|} \|f'(x)\|$$

$$\kappa_{rel} = \frac{\|b\|}{\|Pb\|} \|P'(b)\|$$

(1) $P'(b) = P$

(2) $\frac{\|Pb\|^2}{\|b\|^2} =$

$Pb \perp b - Pb$

$$\|b\|^2 = \|Pb + b - Pb\|^2$$
$$= \|Pb\|^2 + \|b - Pb\|^2$$
$$\|Pb\|^2 = \|b\|^2 - \|b - Pb\|^2$$

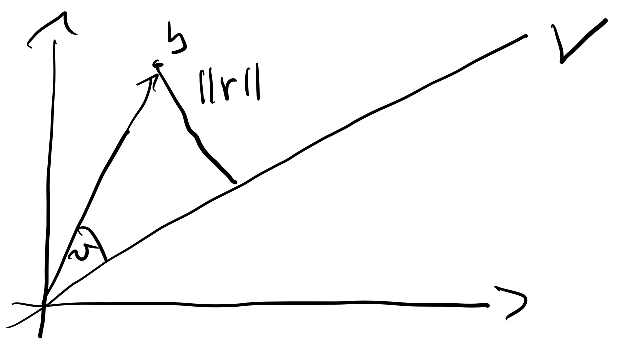
$$= \frac{\|b\|^2 - \|b - Pb\|^2}{\|b\|^2}$$

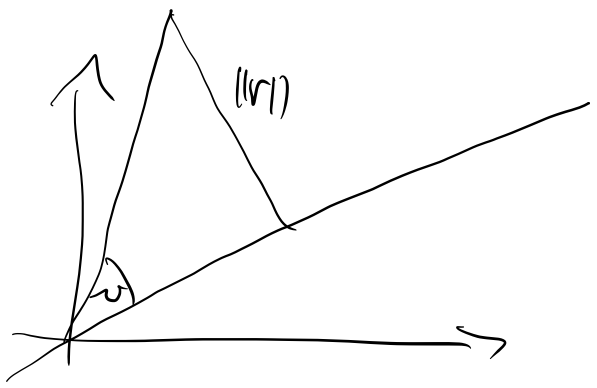
$$= 1 - \sin^2(\vartheta) = \cos^2(\vartheta)$$

$$\kappa_{rel}(b) = \frac{1}{\cos(\vartheta)} \|P\|_2$$

$r = b - Ax$

$$\sin(\theta) = \frac{\|r\|_2}{\|b\|_2}$$





Linear least-squares problems

Conditioning

Now for the least-squares problem $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2$. The relative condition number κ in the Euclidean norm is bounded by

- With respect to perturbations in $\boldsymbol{b}$:

$$\kappa \leq \frac{\kappa_2(A)}{\cos(\theta)}$$

- With respect to perturbations in $\boldsymbol{A}$:

$$\kappa \leq \kappa_2(A) + \kappa_2(A)^2 \tan(\theta)$$

Small residual problems, small angle θ $\cos(\theta) \approx 1$, $\tan(\theta) \approx 0$: behavior similar to linear system.
Large residual problems, large angle θ $\cos(\theta) \ll 1$, $\tan(\theta) \approx 1$: behavior very different from linear system because $\kappa_2(A)^2$ shows up

How should we solve least-squares problems numerically?

We know from the previous slide that if the residual is large, then the condition κ is much larger than $\kappa_2(A)$ (closer to $\kappa_2(A)^2$)

- This is a poorly condition problem; however, do we care?

How should we solve least-squares problems numerically?

We know from the previous slide that if the residual is large, then the condition κ is much larger than $\kappa_2(A)$ (closer to $\kappa_2(A)^2$)

- This is a poorly condition problem; however, do we care?
- If the residual is large, then our $\boldsymbol{Ax}$ won't explain well the right-hand side $\boldsymbol{b}$
- This means that "our curve doesn't fit well the data" and we probably should try to find another space in which to search for a solution (another $\boldsymbol{A}$ with a range that better approximates the projected right-hand side $\boldsymbol{b}$)

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?
- If we choose a numerical method that solves the normal equations

  $$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

  then our problem becomes the problem of solving the linear system with matrix $\boldsymbol{A}^T\boldsymbol{A}$, which has condition number

  $$\kappa_2(\boldsymbol{A}^T\boldsymbol{A}) = \kappa_2(\boldsymbol{A})^2$$

More relevant is the situation with a small residual and then $\kappa \approx \kappa_2(A)$

- Here we have a well condition problem; so what could go wrong?
- If we choose a numerical method that solves the normal equations

  $$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

  then our problem becomes the problem of solving the linear system with matrix $\boldsymbol{A}^T\boldsymbol{A}$, which has condition number

  $$\kappa_2(\boldsymbol{A}^T\boldsymbol{A}) = \kappa_2(\boldsymbol{A})^2$$

  $\leadsto$ we are back in the situation of a poorly condition problem ("solving a linear system with $\boldsymbol{A}^T\boldsymbol{A}$") even though our original problem (least-squares problem) is well condition
- Can we do better and solve the least-squares problem (the problem we are originally interested in) without having to solve a problem with condition that grows with $\kappa_2(\boldsymbol{A})^2$ on the way?

Recall that projecting $\boldsymbol{b}$ onto the column span (range) of $\boldsymbol{A}$ was the key step $\rightsquigarrow$ let's try to find a numerical method that computes an orthonormal basis $\boldsymbol{q}_1, \dots, \boldsymbol{q}_n$ of the rank-n column span of $\boldsymbol{A}$

$$\boldsymbol{A} = \begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \dots & \boldsymbol{a}_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{m \times n}, \qquad m \geq n$$

$$\Downarrow$$

$$\underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \dots & \boldsymbol{a}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & & \vdots \\ & & \ddots & \\ & & & r_{nn} \end{bmatrix}}_{\boldsymbol{R}}$$

with an invertible matrix $\boldsymbol{R}$ so that

$$\operatorname{span}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_k) = \operatorname{span}(\boldsymbol{q}_1, \dots, \boldsymbol{q}_k), \qquad k = 1, \dots, n$$

$$\underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{a}_1 & \dots & \boldsymbol{a}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{q}_1 & \dots & \boldsymbol{q}_n \\ | & & | \end{bmatrix}}_{\boldsymbol{Q}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & & \dots \\ & & \ddots & \\ & & & r_{nn} \end{bmatrix}}_{\boldsymbol{R}}$$

$\Downarrow$ leads to system of equations $\Downarrow$

$$\begin{aligned} \boldsymbol{a}_1 &= r_{11}\boldsymbol{q}_1 \\ \boldsymbol{a}_2 &= r_{12}\boldsymbol{q}_1 + r_{22}\boldsymbol{q}_2 \\ \boldsymbol{a}_3 &= r_{13}\boldsymbol{q}_1 + r_{23}\boldsymbol{q}_2 + r_{33}\boldsymbol{q}_3 \\ &\vdots \\ \boldsymbol{a}_n &= r_{1n}\boldsymbol{q}_1 + r_{2n}\boldsymbol{q}_2 + \cdots + r_{nn}\boldsymbol{q}_n \end{aligned}$$

This motivates a process for computing the basis $\boldsymbol{q}_1, \dots, \boldsymbol{q}_n$

- At step j, we have $\underline{\boldsymbol{q}_1, \dots, \boldsymbol{q}_{j-1}}$ that span $\underline{\text{span}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_{j-1})}$
- We want to find $\underline{\boldsymbol{q}_j}$ orthonormal to $\underline{\boldsymbol{q}_1, \dots, \boldsymbol{q}_{j-1}}$ so that $\underline{\boldsymbol{q}_1, \dots, \boldsymbol{q}_j}$ spans $\underline{\text{span}(\boldsymbol{a}_1, \dots, \boldsymbol{a}_j)}$
- Thus, set

$$\boldsymbol{v}_j = \underline{\boldsymbol{a}_j} - \underline{(\boldsymbol{q}_1^T \boldsymbol{a}_j)}\boldsymbol{q}_1 - \underline{(\boldsymbol{q}_2^T \boldsymbol{a}_j)}\boldsymbol{q}_2 - \cdots - \underline{(\boldsymbol{q}_{j-1}^T \boldsymbol{a}_j)}\boldsymbol{q}_{j-1}$$

and normalize

$$\boxed{\boldsymbol{q}_j = \frac{\boldsymbol{v}_j}{\|\boldsymbol{v}_j\|_2}}$$

Notice that at step j, the quantities $\boldsymbol{q}_1^T \boldsymbol{a}_j, \boldsymbol{q}_2^T \boldsymbol{a}_j, \dots, \boldsymbol{q}_{j-1}^T \boldsymbol{a}_j$ are the values $r_{j,1}, \dots, r_{j,j-1}$ and r_{jj} is responsible for the normalization and set to

$$r_{jj} = \|a_j - \sum_{i=1}^{j-1} r_{ij} q_i\|_2$$

This process is the *classical Gram-Schmidt* procedure to compute the *QR* factorization; however, this process is numerically unstable!