Solving linear systems of equations

Our general strategy for solving a system $Ax=b$ will be to transform it to an equivalent, but easier to solve problem (or problems). An example of an easier sub-problem is a "triangular" system $Ux=b$ where

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 \cdots & 0 & u_{nn} \end{bmatrix}$$

is an upper triangular matrix.

Here is anexample, illustrating how such systems are easy to solve

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & -4 & -6 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ -6 \\ 1 \end{bmatrix}$$

replace x_3 with -1

Solve : $x_3 = -1$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ -12 \\ -1 \end{bmatrix}$$

replace x_2 with 3

Solve : $x_2 = 3$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 3 \\ -1 \end{bmatrix}.$$

We can write this procedure, formally, in pseudocode

Back-substitution for Upper Triangular System

```
for $j = n$ to $1$
    if $u_{jj} = 0$ then stop          [matrix is singular]
    $x_j \leftarrow b_j / u_{jj}$          (1)
    for $i = 1$ to $j-1$
        $b_i \leftarrow b_i - u_{ij} x_j$          (2)
    end
end
```

By counting how many times the loops are executed, we see that $(n-1)$ divisions (line 1) are required, while line 2 is executed $\frac{n(n-1)}{2}$ times. Overall, the cost of back substitution is proportional to $O(n^2)$.

3/29/11

If $L = \begin{bmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 \\ \vdots & & & & \vdots \\ l_{n1} & \cdots & & \cdots & l_{nn} \end{bmatrix}$ is a lower triangular matrix, a similar procedure can be followed to solve $Lx = b$:

<u>Forward substitution for $Lx=b$</u>

```
for $j = 1$ to $n$
    if $l_{jj} = 0$ then stop                [matrix is singular]
    $x_j \leftarrow b_j / l_{jj}$
    for $i = j+1$ to $n$
        $b_i \leftarrow b_i - l_{ij} x_j$
    end
end
```

<u>Cost $= O(n^2)$ again.</u>

The forward- and backward substitution processes can be used to solve a non-triangular system by virtue of the following factorization property:

Thm If A is an $n \times n$ matrix, it can be (generally) written as a product:

$$A = LU.$$

where L = lower triangular & U = upper triangular. Furthermore it is possible to construct L such that all diagonal elements $l_{ii} = 1$

Algorithm: LU factorization by Gaussian Elimination
[Note: This algorithm executes in-place. i.e. the matrix A is replaced by its LU factorization, in compact form]

```
for $k=1$ to $n-1$
    if $a_{kk} = 0$ then stop
        for $i = k+1$ to $n$
            $a_{ik} \leftarrow a_{ik} / a_{kk}$
        end
        for $j = k+1$ to $n$
            for $i = k+1$ to $n$
                $a_{ij} = a_{ij} - a_{ik} a_{kj}$
            end
        end
end.
```

$\approx \frac{n^3}{3}$ times

Cost $= O(n^3)$.

More specifically, this algoritm produces a factorization $A = LU$, where:

$$L = \begin{bmatrix} 1 & & & & & \\ l_{21} & 1 & & & \bigcirc & \\ l_{31} & l_{32} & 1 & & & \\ l_{41} & l_{42} & l_{43} & 1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & u_{33} & \cdots & u_{3n} \\ & & & \ddots & u_{n-1,n} \\ & \bigcirc & & & u_{nn} \end{bmatrix}$$

After the in-place factorization algorithm completes, A is replaced by the following "compacted" encoding of L & U together:

$$A = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & & u_{1n} \\ l_{21} & u_{22} & u_{23} & \cdots & & u_{2n} \\ l_{31} & l_{32} & u_{33} & \cdots & & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & & \vdots \\ & & & & u_{n-1,n-1} & u_{n-1,n} \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{n,n-1} & u_{nn} \end{bmatrix}$$

Existence/Uniqueness and pivoting

When computing the LU factorization, the algorithm will <u>halt</u> if the diagonal element $a_{kk}=0$. This can be avoided by swapping rows of A prior to computing the LU factorization. This is done to always select the largest a_{kk} from the equations that follow

$$A \begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & \textcircled{0} & 3 & 1 \\ 0 & 4 & 1 & -2 \\ 0 & -6 & 0 & 3 \end{bmatrix} \qquad P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

largest

I with swap columns

$$PA = \begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & -6 & 0 & 3 \\ 0 & 4 & 1 & -2 \\ 0 & 0 & 3 & 1 \end{bmatrix}$$

✓

This is pivoting: The pivot a_{kk} is selected to be nonzero. In this process, we can guarrantee uniqueness & existence of LU:

<u>Thm</u> If P is a permutation matrix such that all pivots in the Gaussian Elimination of PA are non-zero, and $l_{ii}=1$, then the LU factorization exists and is unique!

$$PA = LU.$$

<u>Solving $Ax=b$</u>

- Without pivoting:

$$Ax = b$$

$$L\underbrace{Ux}_{=y} = b \quad \Longrightarrow \quad \begin{aligned} &1.\ \text{Solve } Ly=b \quad (\text{F.S}).\\ &2.\ \text{Solve } Ux=y \quad (\text{B.S}).\end{aligned}$$

x is the solution.

In MATLAB 3/29/11

$[L, U] = \text{lu}(A) \Rightarrow \text{Cost } O(n^3)$

$y = b \backslash L \Rightarrow \text{Cost } O(n^2)$

$x = y \backslash L \Rightarrow \text{Cost } O(n^2)$.

If have multiple systems $Ax_i = b_i \Rightarrow$ cost of LU only once.

- With pivoting : For $P^T P = I$

$$Ax = b \Longleftrightarrow PAx = Pb \Leftarrow \cancel{P^T}\cancel{P}Ax = \cancel{P^T}\cancel{P}b$$

$$L\underbrace{Ux}_{y} = Pb$$

$$Ly = Pb$$

$$Ux = y$$

Matlab

```
[L,U,P] = lu(A);     →  Cost O(n^3)
z = Pb;              →  Cost O(n)
y = L\z;             →  Cost O(n^2)
x = U\y;             →  Cost O(n^2)
```

Final variant: <u>Full pivoting</u>

In this case, when we are in the k-th step of the Gauss-Elimination / LU procedure, we pick the pivot element among the <u>entire</u> $(n-k+1)\times(n-k+1)$ lower-rightmost submatrix of A.

pivot position

e.g. $k=2$ and $Ax=b \Rightarrow$

$$\begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & \boxed{0} & 3 & 1 \\ 0 & 4 & 1 & \boxed{-8} \\ 0 & -6 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ 7 \\ 8 \\ 2 \end{bmatrix}$$

submatrix

largest element.

In this case, we can bring (-8) to the pivot position a_{22} by permuting <u>both</u> rows 2-3 <u>AND</u> columns 2-4. Naturally, we will respectively swap rows 2-3 of the RHS, and rows 2-4 of the unknown vector

Thus :
$$\begin{bmatrix} 1 & 2 & 5 & -1 \\ 0 & \boxed{-8} & 1 & 4 \\ 0 & 1 & 3 & 0 \\ 0 & 3 & 0 & -6 \end{bmatrix}\begin{bmatrix} x_1 \\ x_4 \\ x_3 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 7 \\ 2 \end{bmatrix}$$
is the equivalent system !

This process is encoded in the LU factorization using $\underline{\underline{2}}$ permutation matrices P & Q such that $\boxed{PAQ = LU}$

The solution is then computed via:

$$Ax=b \Rightarrow PA\underbrace{QQ^T}_{=I}x = Pb \Rightarrow (LU)(Q^Tx) = Pb$$

$$\Rightarrow L\underbrace{(UQ^Tx)}_{y} = Pb \Rightarrow y \text{ found via lower triangular solve}$$

$$U\underbrace{Q^Tx}_{z} = y \Rightarrow z \text{ found via upper triangular solve}$$

$$Q^Tx = z \Rightarrow QQ^Tx = Qz \Rightarrow \boxed{x = Qz}$$

<u>In Matlab</u> :

```
[L,U,P,Q] = lu(A);   → Cost O(n^3)
  c = Pb;            → Cost O(n)
  y = L\c;           → Cost O(n^2)
  z = U\y;           → Cost O(n^2)
  x = Q*u;           → Cost O(n).
```

Pivoting (§3.6) 3/31/11

We saw that in order to avoid dividing by zero (or, near-zero) values in the LU decomposition, we may need to resort to partial or full pivoting.

- Partial pivoting permutes rows, such that the pivot element in the k-th iteration is the largest number in the $(n-k+1)$ lower entries of the k-th column
  It is written, in the context of LU decomposition, as

$$PA = LU \qquad (P = \text{permutation})$$

- Full pivoting selects the pivot element in the k-th iteration as the largest element of the $(n-k+1)\times(n-k+1)$ lower-rightmost submatrix of A. It operates by permuting rows AND columns and leads to a LU decomposition of.

$$PAQ = LU.$$

However, there are certain categories of matrices for which we can safely use Gauss elimination or LU decomposition without the need for pivoting (i.e. the pivot elements will never be problematically small).

<u>Def</u> A matrix A is called <u>diagonally dominant by columns</u> if the magnitude of every diagonal element is larger than the sum of magnitudes of all other entries in the same column, i.e.

For every $i = 1, 2, \dots, n$ $\qquad |a_{ii}| > \sum_{j \neq i} |a_{ji}|$

If the diagonal element exceeds in magnitude the sum of magnitudes of all other elements in its <u>row</u>, i.e.

for $i = 1, 2, \dots, n$ $\qquad |a_{ii}| > \sum_{j \neq i} |a_{ij}|$

then the matrix is called diagonally <u>dominant by rows</u>.

<u>Def</u>

A symmetric matrix $A \in \mathbb{R}^{n \times n}$ is called positive definite (in short: SPD : for "symmetric positive definite"), if for any $\underline{x} \in \mathbb{R}^n$, $\underline{x} \neq 0$

we have $\quad \underline{x}^T A \underline{x} > 0$

If for any $\underline{x} \in \mathbb{R}$ $\underline{x} \neq 0$ we have $\underline{x}^T A \underline{x} \geq 0$, the matrix is called positive semi-definite.

If the respective properties are $x^T A x < 0$ (or $x^T A x \leq 0$) the [illegible] negative (semi-) definite

Def the k-th leading principal minor of a matrix $A \in \mathbb{R}^{n \times n}$ is the determinant of the top-leftmost $k \times k$ submatrix of A. Thus if we denote this minor by M_k:

$$M_1 = |a_{11}| \qquad M_2 = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \quad \cdots \quad M_k = \begin{vmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & & \vdots \\ a_{k1} & \cdots & a_{kk} \end{vmatrix}$$

Thm If all leading principal minors (i.e. for $k = 1, 2, 3, \dots, n$) of the symmetric matrix A are positive, then A is positive definite

* If $M_k < 0$ for $k =$ odd and $M_k > 0$ for $k =$ even, then A is negative definite.

Thm Pivoting is not necessary when A is diagonally dominant by columns, or symmetric and positive- (or negative-) definite.

These "special" classes of matrices (which appear quite often in engineering and applied sciences) not only make LU decomposition more robust, but also open some additional possibilities for solving $A\underline{x} = \underline{b}$.

Iterative methods for linear systems.

The general idea is similar to the philosophy of iterative methods we saw for nonlinear equations, i.e. we proceed as follows:

→ We write a (matrix) equation

$$\underline{x} = T\underline{x} + \underline{c}$$

in such a way that this equation is equivalent to $Ax=b$.

→ We start with an initial guess $\underline{x}^{(0)}$ for the solution of $Ax=b$

→ We iterate

$$\underset{\sim}{x}^{(k+1)} = T\underset{\sim}{x}^{(k)} + \underset{\sim}{c}$$

→ If properly designed the sequence $x^{(0)}, x^{(1)}, \dots, x^{(k)}, \dots$ converges to $\underline{x}^*$, which satisfies $\underline{x}^* = T\underline{x}^* + c$ and subsequently $Ax^* = b$.

General design approach

We decompose $A = D - L - U$

$$A = \underset{\substack{\uparrow \\ \text{diagonal}}}{D} - \underset{\substack{\uparrow \\ \text{lower triangular}}}{L} - \underset{\substack{\uparrow \\ \text{upper triangular}}}{U}$$

e.g. $$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

$$D = \begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn} \end{bmatrix} \qquad L = \begin{bmatrix} 0 & & & & \\ -a_{21} & 0 & & \text{\Large 0} & \\ -a_{31} & -a_{32} & \ddots & & \\ & & & \ddots & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0 \end{bmatrix}$$

$$U = \begin{bmatrix} 0 & -a_{12} & -a_{13} & & -a_{1n} \\ & \cdot & -a_{23} & & -a_{2n} \\ & & \ddots & & \\ & & & \ddots & -a_{n-1,n} \\ & & & & 0 \end{bmatrix}$$

The Jacobi method

$$A = D - L - U$$

$$Ax = b$$

$$(D-L-U)x = b$$

$$Dx = (L+U)x + b$$

$$x = \underbrace{D^{-1}(L+U)}_{T}x + \underbrace{D^{-1}b}_{c} \qquad (x = Tx + c)$$

Iteration:

$$\underline{x}^{(k+1)} = D^{-1}(L+U)\,\underline{x}^{(k)} + D^{-1}\underline{b} \qquad \underline{\underline{\text{or}}}$$

$$\boxed{D\underline{x}^{(k+1)} = (L+U)\,\underset{\sim}{x}^{(k)} + \underset{\sim}{b}}$$ The Jacobi iteration formula.

→ Solution: easy, since we need to solve a linear system of equations with diagonal coefficient matrix

$$\begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \rightarrow x_i^{(k+1)} = \frac{1}{d_i} c_i$$

→ Convergence: The Jacobi method is guarranteed to converge when A is diagonally dominant by rows

→ Complexity: Each iteration has a cost associated with

→ Solving $Dx^{(k+1)} = c$ ⟹ n divisions

→ Computing $c = (L+U)x^{(k)} + b$ ⟹ as many additions and multiplications as nonzero entries in A worst case $= O(n^2)$, but could be $O(n)$ for sparse matrices

→ Stopping criterion: $\| b - Ax^{(k)} \| < \varepsilon$

or $\| x^{(k+1)} - x^{(k)} \| < \varepsilon$.

There are 3 forms we will see this algorithm, for different purposes:

(i) Matrix form (for proofs) $Dx^{(k+1)} = (L+U)x^{(k)} + b$

(ii) Algorithm form (without in-place update)

Each row of $Dx^{(k+1)} = (L+U)x^{(k)} + b$:

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

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

```
for k = 1, 2, ..., <max>
    for i = 1, 2, ..., n
        $x_i^{(k+1)} \leftarrow \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$
    end
    <check for convergence
end
```

iii) In-place algorithm

(replaces $\underline{x}$ with a better estimate)

```
$\underline{x} \leftarrow$ initial guess
for k = 1, 2, ..., <max>
    for i = 1, 2, ..., n
        $x_i^{new} \leftarrow \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j \right)$
    end
    $\underline{x} \leftarrow \underline{x}^{new}$
    check for convergence
end
```

The Gauss-Seidel method

We again employ the decomposition $A = D - L - U$

$$Ax = b$$

$$(D-L-U)x = b$$

$$(D-L)x = Ux + b$$

At this point, we place $x^{(k+1)}$ on the LHS and $x^{(k)}$ on the RHS

$$\boxed{(D-L)x^{(k+1)} = Ux^{(k)} + b} \qquad (1)$$

The benefit of the Gauss-Seidel method (1) over Jacobi is the improved convergence, which is guaranteed not only for diagonally dominant matrices, but also for <u>symmetric and positive definite</u> matrices

In terms of complexity, each iteration of (1) amounts to solving a lower-triangular system via forward substitution, i.e. incurs a cost $O(k)$, k = # of nonzero entries in A.

Once again, form (1) is useful for proofs, while the pseudocode version is given as:

(Without in-place update)

```
$\underline{x}^{(0)} \leftarrow$ initial guess
for $k = 0, 1, \dots,$ <max>
  for $i = 1, 2, \dots, n$
    $x_i^{(k+1)} \leftarrow \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$
  end
  <check for convergence
end
```

(In-place)

```
$\underline{x} \leftarrow$ initial guess
for $k = 0, 1, \dots,$ <max>
  for $i = 1, 2, \dots, n$
    $x_i \leftarrow \frac{1}{a_{ii}}\left(b_i - \sum_{i \neq j} a_{ij}x_j\right)$ (*)
  end
  <check for convergence>
end.
```

Compare: Jacobi

```
$\underline{x} \leftarrow$ initial guess
for $k = 0, 1, \dots,$ <max>
  for $i = 1, 2, \dots, n$
    $x_i^{new} \leftarrow \frac{1}{a_{ii}}\left(b_i - \sum_{i \neq j} a_{ij}x_j\right)$ (**)
  end
  $\underline{x} \leftarrow x^{new}$ (***)
end
```

The real difference in performance is:

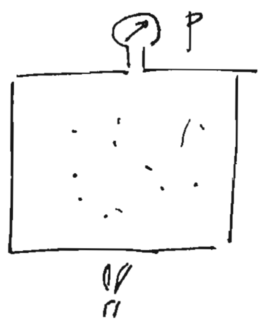
Gauss-Seidel is generally serial in nature (although parallel variants exist), while steps (**) & (***) are highly parallel.

Overdetermined systems

So far, we considered linear systems $Ax=b$ with the <u>same</u> number of equations & unknowns (i.e. $A \in \mathbb{R}^{n\times n}$).

In the case where $A \in \mathbb{R}^{m\times n}$, with $m>n$ (more equations the existence of a true solution is not guaranteed; in this case we look for the "best possible" substitute for a solution. Before analyzing what that means, let's look at how such problems arise:

e.g. In an experiment, we measure the pressure of a gas in a closed container, as a function of the temperature



From physics:

$$pV = nR\,\tfrac{5}{9}\,(T+459.67)$$

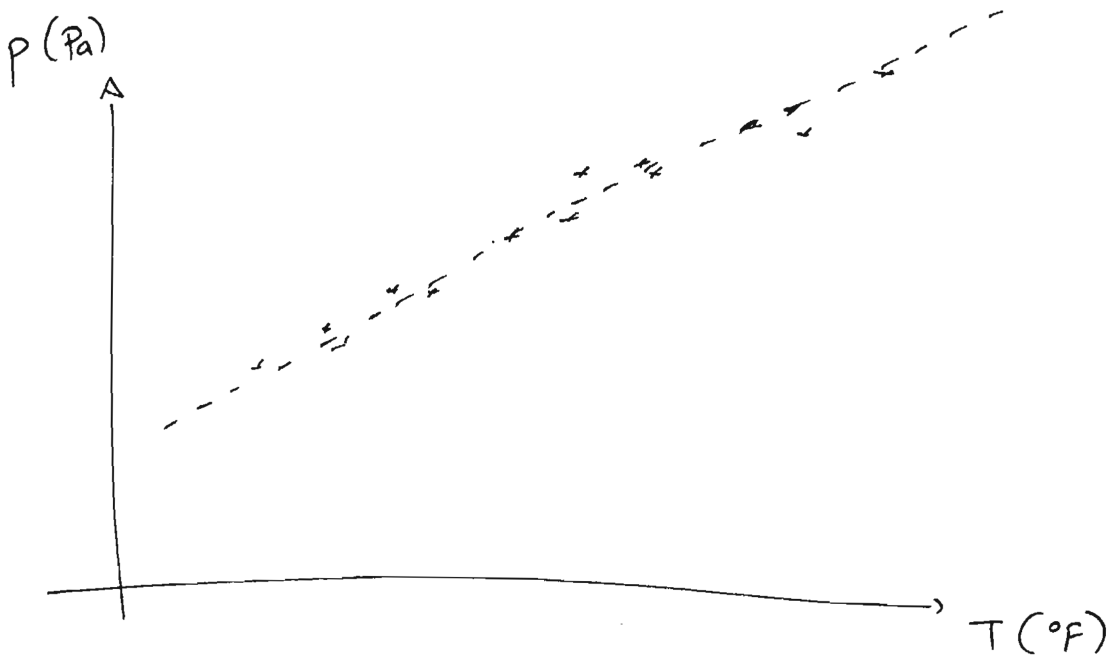
T: Fahrenheit.

i.e $\quad p = \alpha T + \beta$

$$\left(\alpha = \frac{5nR}{9V},\ \beta = \frac{5nR\cdot 459{,}67}{9V}\right)$$

What are α & β?

experimentally:



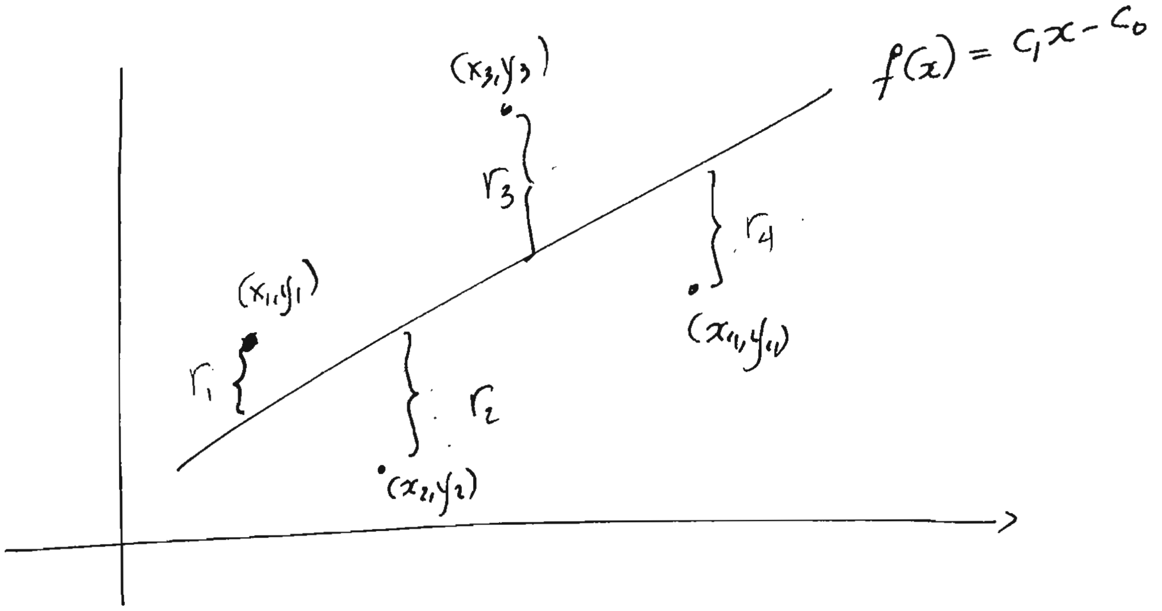
The measurements <u>should ideally</u> lie on a straight line $y = c_1 x + c_0$, but do not, due to measurement error. If we have $\underline{\underline{n}}$ measurement pairs $(x_1, y_1), \dots, (x_n, y_n)$ we would have wanted:

$$\left.\begin{aligned} y_1 &= c_1 x_1 + c_0 \\ y_2 &= c_1 x_2 + c_0 \\ &\vdots \\ y_n &= c_1 x_n + c_0 \end{aligned}\right\} \Rightarrow \underbrace{\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_1 \\ c_0 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{b}$$

Here $\underset{n\times 2}{\underset{\uparrow}{A}}\ \underset{2\times 1}{\underset{\uparrow}{x}} = \underset{n\times 1}{\underset{\uparrow}{b}}$ is a rectangular system.

We cannot hope to find a true solution to this system. Instead let's try to find an "approximate" solution, such that $Ax \approx b$.

Let's look at the residual of this "interpolation"



the residual of the approximation at each data point is

$$r_i = y_i - f(x_i) = y_i - c_1 x_i - x_0$$

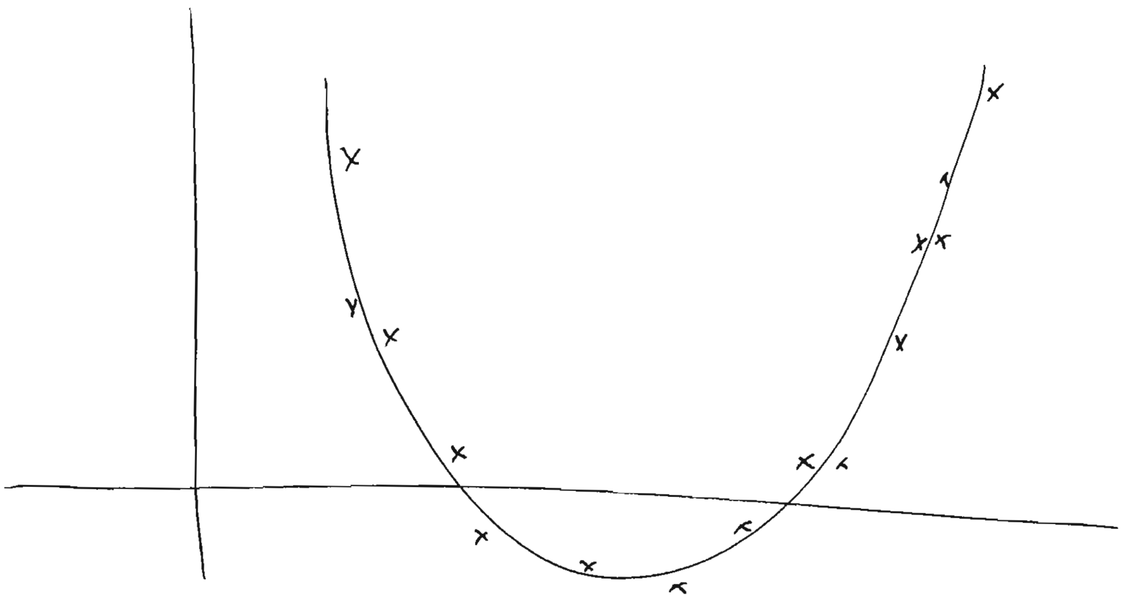
If we write the vector of all residuals:

$$\underline{r} = \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} y_1 - c_1 x_1 - c_0 \\ y_2 - c_1 x_2 - c_0 \\ \vdots \\ y_n - c_1 x_n - c_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \\ x_n & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_0 \end{bmatrix} = \underline{b} - A\underline{x}$$

Although we can't find an $\underline{x}$ s.t. $A\underline{x} = \underline{b}$ (thus $\underline{r} = 0$) we can at least try to make $\underline{r}$ <u>small</u>.

Another example

Find the best parabola $f(x) = c_2x^2 + c_1x + c_0$ that fits measurements $(x_1, y_1), \ldots, (x_n, y_n)$



We would like

$$\left.\begin{array}{l} f(x_1) \approx y_1 \\ f(x_2) \approx y_2 \\ \vdots \\ f(x_n) \approx y_n \end{array}\right\} \quad \left.\begin{array}{c} c_2x_1^2 + c_1x_1 + c_0 \approx y_1 \\ c_2x_2^2 + c_1x_2 + c_0 \approx y_2 \\ \vdots \\ c_2x_n^2 + c_1x_n + c_0 \approx y_n \end{array}\right\} \quad \underbrace{\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ & \vdots & \\ x_n^2 & x_n & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} c_2 \\ c_1 \\ c_0 \end{bmatrix}}_{x} \approx \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{b}$$

Once again we would like to make $\underline{r} = \underline{b} - A\underline{x}$ as small as possible.

How do we quantify $\underline{r}$ being small?

$\Rightarrow$ Using a norm!

We could ask that $\rightarrow$ $\|r\|_1$, $\|r\|_2$, or $\|r\|_\infty$ be as small as possibl

Any of these norms would be intuitive to consider for minimization (esp. 1- and ∞-norms are very intuitive).

However, we typically use the 2-norm for this purpose, because it's the easiest to work with in this problem!

Def The least squares solution of the overdetermined system $A\underline{x} \approx \underline{b}$ is the vector $\underline{x}$ that minimizes

$$\|\underline{r}\|_2 = \|\underline{b} - A\underline{x}\|_2$$

Define $Q(\underline{x}) = Q(x_1, x_2, \ldots, x_n)$

$$= \|\underline{b} - A\underline{x}\|_2^2 \quad \text{where} \quad \underset{\sim}{x} = (x_1, \ldots, x_n)$$

and $A \in \mathbb{R}^{m\times n}$ $b \in \mathbb{R}^m$ $(m>n)$. The least squares solution is the set of values $x_1, \ldots, x_n$ that minimize $Q(x_1, \ldots, x_n)$!

$$Q(x_1, \ldots, x_n) = \|\underline{b} - A\underline{x}\|_2^2 = \|\underline{r}\|_2^2 = \sum_{i=1}^{m} r_i^2$$

$$\underline{r} = \underline{b} - A\underline{x} \Rightarrow r_i = b_i - [Ax]_i \Rightarrow r_i = b_i - \sum a_{ij} x_j$$

$$\Rightarrow Q(x_1, \ldots, x_n) = \sum_{i=1}^{m} \Big(b_i - \sum_{j=1}^{n} a_{ij} x_j\Big)^2.$$

If $x_1, \ldots, x_n$ are those that minimize Q, then:

$$\left.\begin{array}{l} \dfrac{\partial Q}{\partial x_1} = 0 \\ \dfrac{\partial Q}{\partial x_2} = 0 \\ \vdots \\ \dfrac{\partial Q}{\partial x_n} = 0. \end{array}\right\} \text{in order to guarantee a minimum}$$

4/5/11

$$\frac{\partial Q}{\partial x_k} = \frac{\partial}{\partial x_k}\left(\sum_{i=1}^{m}\left(b_i - \sum_{j=1}^{n} a_{ij} x_j\right)^2\right)$$

$$= \sum_{i=1}^{m} \frac{\partial}{\partial x_k}\left(b_i - \sum_{j=1}^{n} a_{ij} x_j\right)^2$$

$$= \sum_{i=1}^{m} 2 \underbrace{\left(b_i - \sum_{j=1}^{n} a_{ij} x_j\right)}_{r_i} \frac{\partial}{\partial x_n}\left(b_i - \sum_{j=1}^{n} a_{ij} x_j\right)$$

$$= \sum_{i=1}^{m} r_i a_{ik} = \sum_{i=1}^{m} [A^T]_{ki} r_i = [A^T r]_k$$

Thus:

$$\left.\begin{array}{l} \frac{\partial Q}{\partial x_1} = 0 \Rightarrow [A^T r]_1 = 0 \\ \vdots \\ \frac{\partial Q}{\partial x_n} = 0 \Rightarrow [A^T r]_n = 0 \end{array}\right\} \quad \boxed{A^T \underline{r} = \underline{0}}$$

Since $\underline{r} = \underline{b} - A\underline{x}$, we have:

$$0 = A^T r = A^T(b - Ax) = A^T b - A^T A x \Rightarrow$$

$$\Rightarrow \boxed{A^T A \underline{x} = A^T b}$$

The system above is called the <u>normal equations system</u>

it is a <u>square</u> system, that has as solution the least-squares approximation of $Ax \approx b$

$$\underbrace{\underset{n\times m}{A^T}\ \underset{m\times n}{A}}_{n\times n}\ \underset{n\times 1}{\underset{n\times 1}{x}} = \underbrace{\underset{n\times m}{A^T}\ \underset{m\times 1}{b}}_{n\times 1}$$

The normal equations <u>always</u> have a solution (with the simple condition that the columns of A have to be linearly independent (usually true)).

<u>Problem</u>: The condition number of $A^T A$ is the <u>square</u> of that of A (if A was square itself!).

Overdetermined systems

We examined the case of systems $Ax = b$ where $A \in \mathbb{R}^{m\times n}$ and $m > n$. In general, a true solution does not exist. We define however the least squares solution as the vector $\underline{x}$ that minimizes

$$\underline{x} = \arg\min \|\underline{b} - A\underline{x}\|_2^2$$

$\underline{x}$ is given by the solution to the system of normal equations

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

System (1) is square ($n\times n$) and invertible (if A has linearly independent columns). However, the condition number of A^TA could be very poor... for example, if A was square, we would have $\text{cond}(A^TA) = [\text{cond}(A)]^2$.

An alternative method that does not suffer from this problematic conditioning is the QR factorization.

Def An $n\times n$ matrix Q is called orthogonal iff

$$Q^TQ = QQ^T = I.$$

Thm Let $A \in \mathbb{R}^{m \times n}$ $(m > n)$ have linearly independent columns. Then a decomposition

$$A = QR$$

exists, such that $Q \in \mathbb{R}^{m \times m}$ is orthogonal and $R \in \mathbb{R}^{m \times n}$ is upper triangular (i.e. $R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ 0 & & \ddots & \\ 0 & 0 & \cdots & r_{nn} \\ & & \vdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}$)

Additionally, given that A has linearly independent columns, we guarrantee that $r_{ii} \neq 0$.

In Matlab, the QR decomposition is obtained via the qr function, i.e

```
[Q, R] = qr (A);
```

Now, let us write

$$Q = \left[\hat{Q} \;\middle|\; Q^* \right]$$

where $\hat{Q} \in \mathbb{R}^{m \times m}$ contains the first m columns of Q and $Q^* \in \mathbb{R}^{(m-n) \times m}$ contains the last (m-n) columns

Respectively, we write:

$$R = \begin{bmatrix} \hat{R} \\ \hdashline O_{(m-n)\times n} \end{bmatrix}$$ where $\hat{R} \in \mathbb{R}^{n\times n}$ (and upper triangula
contains the first n rows of R.
$\hat{R}$ is also <u>nonsingular</u> (for lin. ind. columns of

We can verify the following:

→ $\hat{Q}^T \hat{Q} = I_n$ (although $\hat{Q}\hat{Q}^T \neq I_m$!)

<u>Proof</u>: $$\underset{1\le i,j\le n}{[\hat{Q}^T\hat{Q}]_{ij}} = \sum_{k=1}^{m} [\hat{Q}^T]_{ik} [\hat{Q}]_{kj}$$

$$= \sum_{k=1}^{m} [\hat{Q}]_{ki} [\hat{Q}]_{kj} = \sum_{k=1}^{m} [Q]_{ki} [Q]_{kj}$$

$$= [Q^T Q]_{ij} = [I_m]_{ij}$$

→ $$\underset{(m\times n)}{A} = \underset{(m\times m)}{Q}\underset{(m\times n)}{R} = \underset{(m\times n)}{\hat{Q}} \cdot \underset{(n\times n)}{\hat{R}}.$$

Proof: Similar.

The factorization $A = \hat{Q}\cdot\hat{R}$ is the so-called economy-size QR factorization, and computed in Matlab as:

$$[\hat{Q}, \hat{R}] = qr(A, 0);$$

Once we have $\hat{Q}$ & $\hat{R}$ computed,

we observe that the normal equations can be written as:

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

using $A=\hat{Q}\hat{R}$ $\Longrightarrow$

$$\hat{R}^T \underbrace{\hat{Q}^T \hat{Q}}_{= I_n} \hat{R} \underline{x} = \hat{R}^T \hat{Q}^T b$$

$$\hat{R}^T \hat{R} \underline{x} = \hat{R}^T \hat{Q}^T \underline{b}$$

$\hat{R}$ is invertible $\Longrightarrow$

$$(\hat{R}^{-T})(\hat{R}^T \hat{R} x) = (\hat{R}^{-T})(\hat{R}^T \hat{Q}^T \underline{b})$$

$$\Longrightarrow \boxed{\hat{R}\underline{x} = \hat{Q}^T \underline{b}} \quad (*)$$

Least squares solution using QR decomposition

<u>Matlab</u>:
```
[Q̂,R̂] = qr(A,0)
z = Q̂'*b;
x = R̂\z;
```

<u>Benefit</u>: We can show that $\text{cond}(A^TA) = [\text{cond}(\hat{R})]^2$,

thus, equation (*) is <u>much better</u> conditioned than the normal equations system.