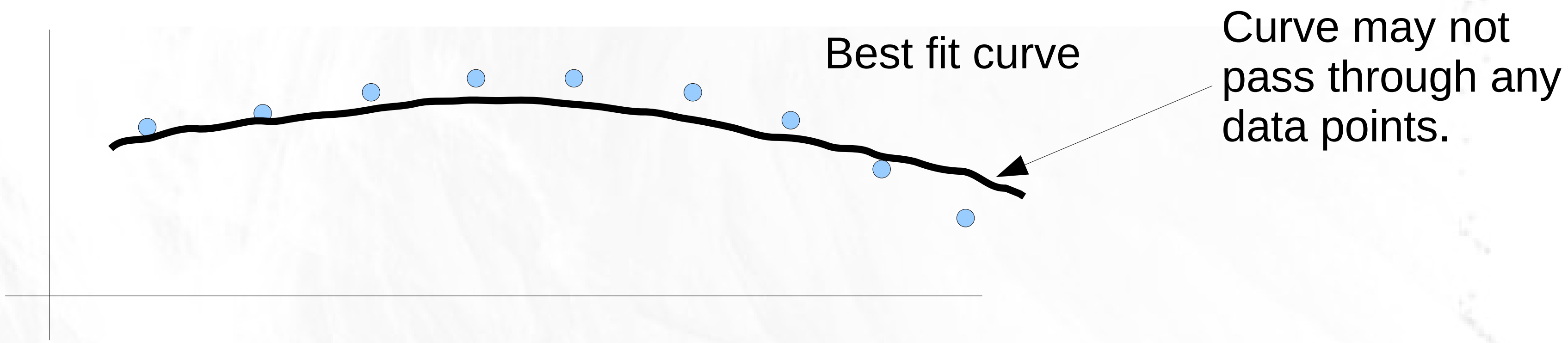
Today's topic:
Linear regression in one and many variables

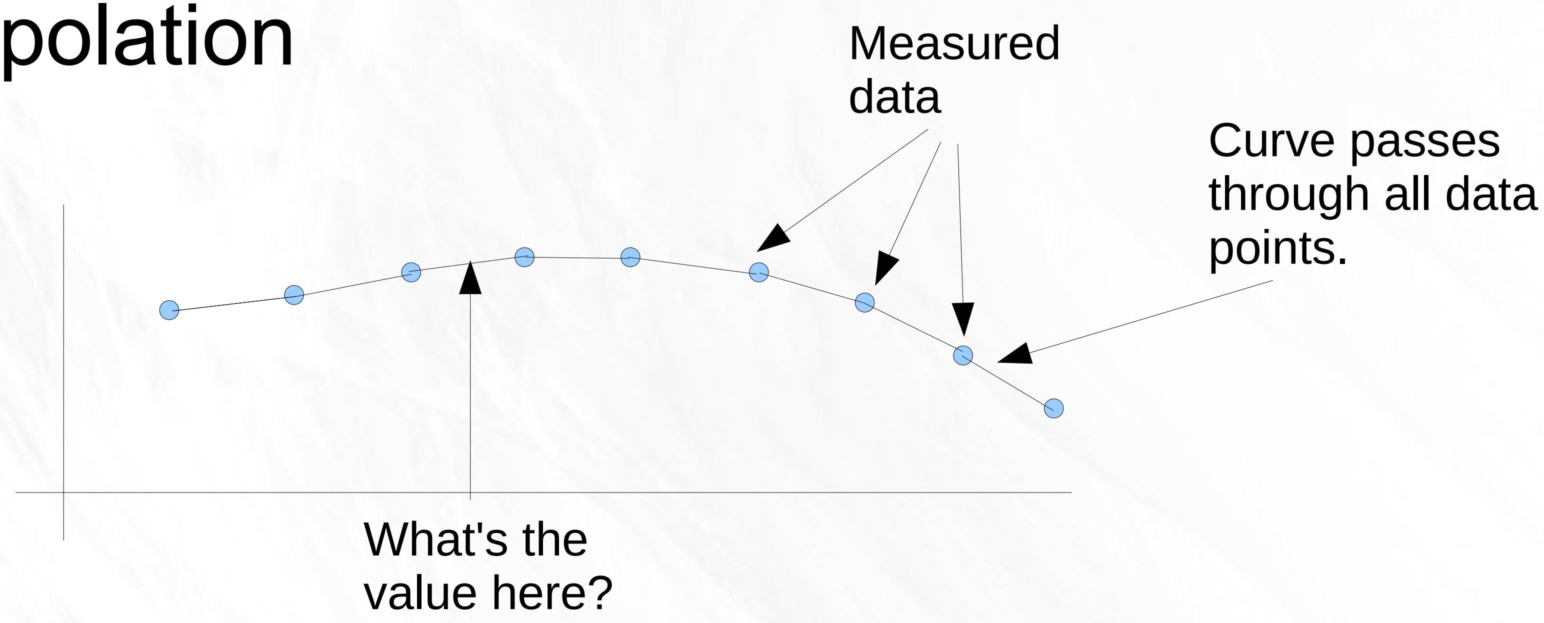
Regression



- Regression:
  - Fit some type of (simple) curve to the data
  - The curve might actually miss the data points
- Variants:
  - One independent variable vs. many independent variables (multiple regression).
  - Linear vs. Non-linear. Today treats only linear.

Regression is different from Interpolation & Extrapolation

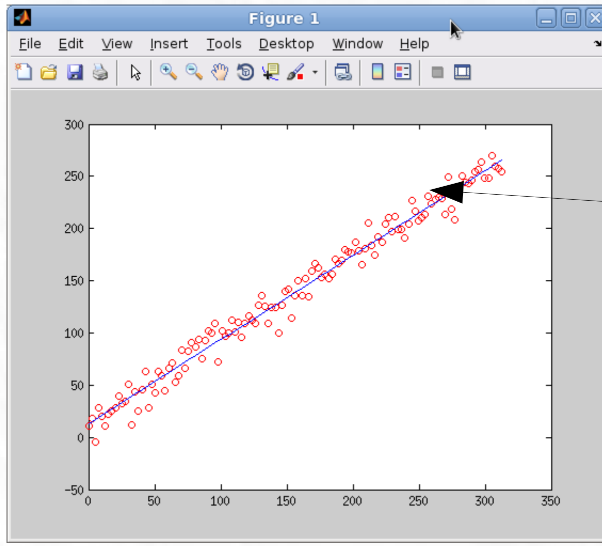
- Interpolation



- Extrapolation

Simplest regression problem: 1D linear regression

- We want to find best-fit line to data



Data is assumed to be *N* (x_i, y_i) pairs

- Find line which minimizes total error (best fit)
  - Line: $y = \beta x + y_0$
  - Error: $e = \sum_i (y_i - y(x_i))^2$

$$= \sum_{i=1}^{N} (y_i - [\beta x_i + y_0])^2$$

Fitting parameters:
Slope β
Intercept y_0

Straight line guess

Actual data pt.

Our goal: find slope and intercept

- Assume fit line: $y = \beta x + y_0$

- Find β, y_0 to minimize the total error:

Consider error as fcn of fitting parameters

$$e(\beta, y_0) = \sum_{i=1}^{N} (y_i - [\beta x_i + y_0])^2$$

Least squares error is always non-negative.

- When error is minimized, the derivatives are zero:

$$\frac{\partial e}{\partial \beta} = 0$$

$$\frac{\partial e}{\partial y_0} = 0$$

Later we will see that this generalizes to an expression involving gradients

Derivatives

- First: $$\frac{\partial e}{\partial \beta} = \sum_i \cancel{2}(y_i - \beta x_i - y_0)(\cancel{-}x_i) = 0$$

$$= \sum_i (y_i x_i - \beta x_i^2 - y_0 x_i) = 0$$

Items in brackets are constants computed from the input data

- Rearrange to get: $$\beta\left(\sum_i x_i^2\right) + y_0\left(\sum_i x_i\right) = \left(\sum_i y_i x_i\right)$$

- Next: $$\frac{\partial e}{\partial y_0} = \sum_i 2(y_i - \beta x_i - y_0)(-1) = 0$$

$$= \sum_i (y_i - \beta x_i - y_0) = 0$$

- Rearrange to get: $$\beta\left(\sum_i x_i\right) + y_0\left(\sum_i 1\right) = \left(\sum_i y_i\right)$$

We have a 2x2 linear system

$$\beta\left(\sum_i x_i^2\right)+y_0\left(\sum_i x_i\right)=\left(\sum_i y_i x_i\right)$$

$$\beta\left(\sum_i x_i\right)+y_0\left(\sum_i 1\right)=\left(\sum_i y_i\right)$$

- Rewrite as matrix expression:

$$\begin{pmatrix}\sum_i 1 & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2\end{pmatrix}\begin{pmatrix}y_0 \\ \beta\end{pmatrix}=\begin{pmatrix}\sum_i y_i \\ \sum_i y_i x_i\end{pmatrix}$$

$$Au=f$$

Solution is easy

$$Au=f \qquad \begin{pmatrix} \sum_i 1 & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix} \begin{pmatrix} y_0 \\ \beta \end{pmatrix} = \begin{pmatrix} \sum_i y_i \\ \sum_i y_i x_i \end{pmatrix}$$

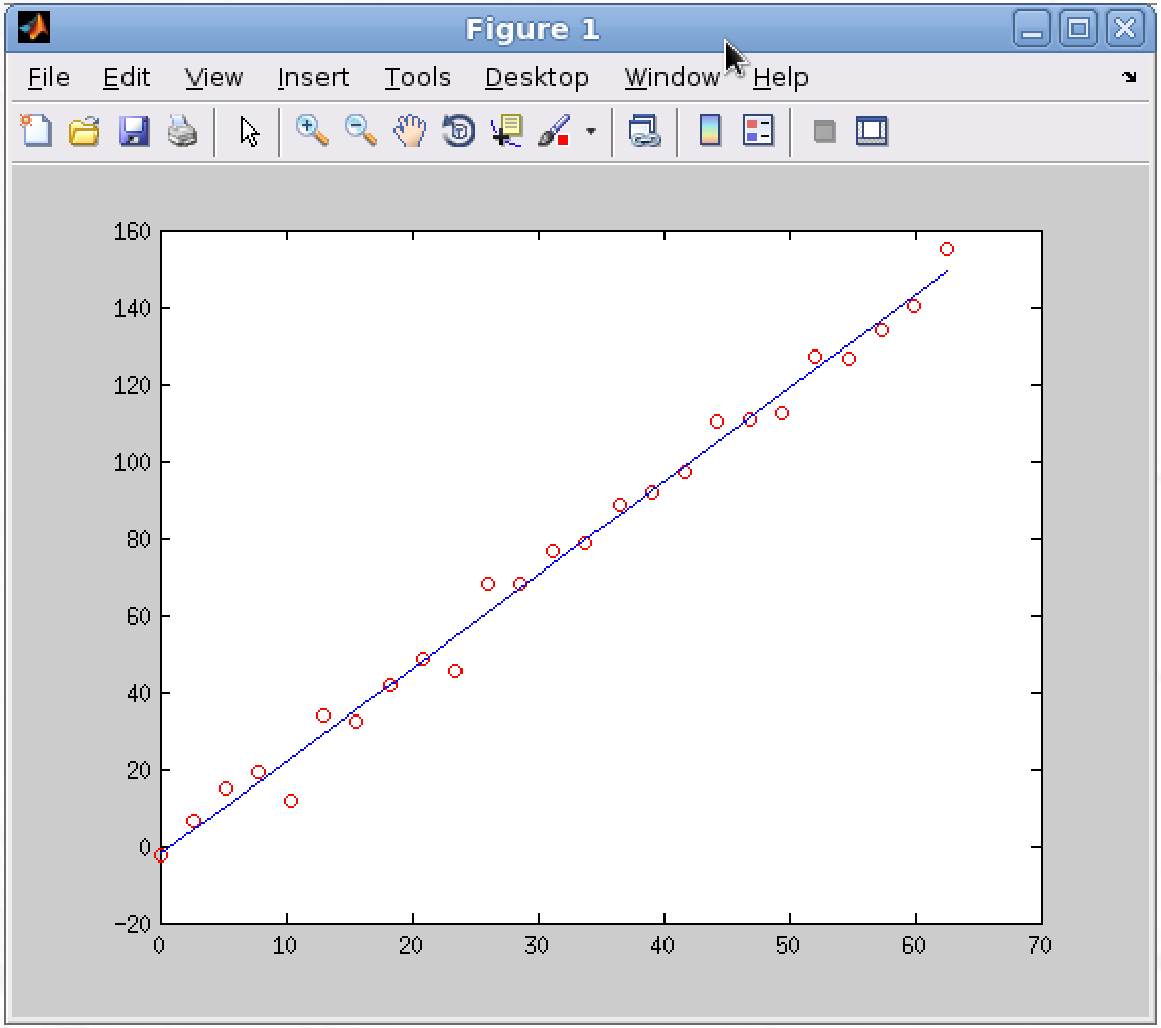
$$u=A\backslash f \qquad \begin{pmatrix} y_0 \\ \beta \end{pmatrix} = \begin{pmatrix} \sum_i 1 & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix} \backslash \begin{pmatrix} \sum_i y_i \\ \sum_i y_i x_i \end{pmatrix}$$

Matlab implementation

```
function [y0, b] = lstsq(x, y)
  % this fcn uses the least squares error approach to
  % compute a linear fit to the data [x, y].  The approach
  % forms the linear system Au = f and then solves it.

  N = length(x);

  % Create A matrix
  A = zeros(2,2);
  A(1,1) = N;
  A(2,1) = sum(x);
  A(1,2) = A(2,1);
  A(2,2) = sum(x.*x);

  % Create f vector
  f = zeros(2, 1);
  f(1,1) = sum(y);
  f(2,1) = sum(y.*x);

  u = A\f;
  y0 = u(1);
  b = u(2);
end
```

A few comments

- Good fit due in part to "averaging" performed by fit.
- Least squares is sensitive to outliers



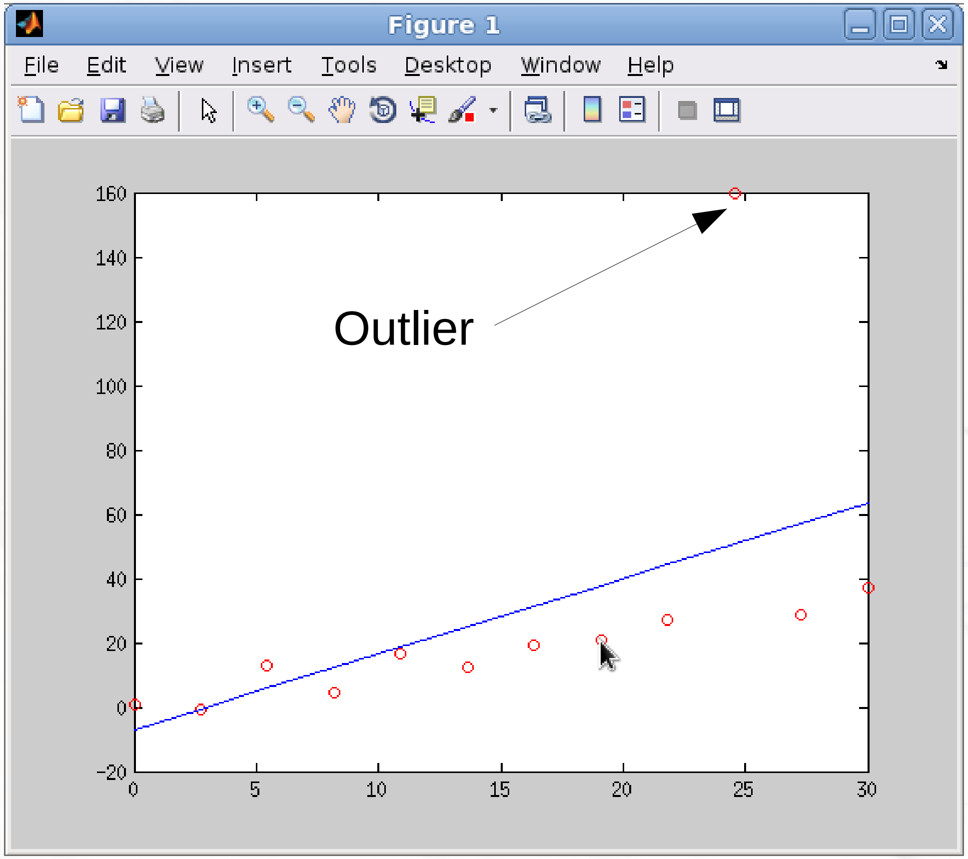
Effect of outlier

- Effect is large because errors are penalized by

$$(y_i - y(x_i))^2$$

- A less sensitive error expression would involve absolute value instead of least-squares:

$$e = \sum_i |y_i - y(x_i)|$$



Next: Linear least squares from a Linear Algebra viewpoint

- Recall goal: find line $y = \beta x + y_0$ which minimizes error.
- Let's define a few objects....

$$A = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \qquad s = \begin{pmatrix} \beta \\ y_0 \end{pmatrix} \qquad t = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

With these definitions....

- Write expression for β, y_0 as matrix equation:

$$As=t \longleftrightarrow \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

- Equivalent linear system:

$$\begin{aligned} y_1 &= \beta x_1 + y_0 \\ y_2 &= \beta x_2 + y_0 \\ y_3 &= \beta x_3 + y_0 \\ \vdots \quad & \quad \vdots \quad \quad \vdots \end{aligned}$$

We have N equations, 2 unknowns. System is overdetermined. No solution exists in general.

Residual

- From last slide – equivalent linear system

$$
\begin{aligned}
y_1 &= \beta\, x_1 + y_0 \\
y_2 &= \beta\, x_2 + y_0 \\
y_3 &= \beta\, x_3 + y_0 \\
\vdots \quad & \quad \vdots \qquad \vdots
\end{aligned}
$$

$$\vec{t} = A\,\vec{s}$$

A not square, can’t solve this system.

- Definition of residual (error):

$$
\begin{aligned}
r_1 &= y_1 - \beta x_1 + y_0 \\
r_2 &= y_2 - \beta x_2 + y_0 \\
r_3 &= y_3 - \beta x_3 + y_0 \\
\vdots \quad & \quad \vdots \qquad \vdots
\end{aligned}
$$

$$\vec{r} = \vec{t} - A\,\vec{s}$$

Consider the residual

- Residual (vector): $r = t - A\,s$

- Suppose we try to minimize the Euclidian norm of the residual?

Dot product of r with itself.

- Euclidian norm squared : $\|r\|_2^2 = \sum_i r_i^2 = r^T r$

- Minimizing the (squared) Euclidian norm of the residual is the same as minimizing the least squares error:

$$e = \sum_{i=1}^{N} (\overbrace{y_i}^{t} - \overbrace{[\beta\,x_i + y_0]}^{A\,s})^2$$

Least squares and linear algebra

- We have related the original problem statement (minimize least squares error) to a linear algebra statement about the residual.

$$\text{minimize} \sum_{i=1}^{N} (y_i - [\beta x_i + y_0])^2 \quad \longleftrightarrow \quad \text{minimize} \|r\|_2^2 = r^T r$$

- Compute norm (squared) of residual:

$$\|r\|_2^2 = (A s - t)^T (A s - t)$$

Find s where this is minimized.

- Goal: Norm is minimized when gradient is zero. Find gradient w.r.t. *s*. The s value where gradient is zero holds the desired fit coefficients.

Compute gradient

$$\nabla_s (As-t)^T (As-t) = A^T (As-t) + (As-t)^T A$$

$$= A^T As - A^T t + s^T A^T A - t^T A$$

$$s = \begin{pmatrix} \beta \\ y_0 \end{pmatrix}$$

Derivative w.r.t. s.

$$A^T As \qquad A^T t$$

- Therefore, $\text{gradient} = (2A^T As - 2A^T t) = 0$
- The “normal equations”: $A^T As = A^T t$
- Solve to get the parameters of the line:

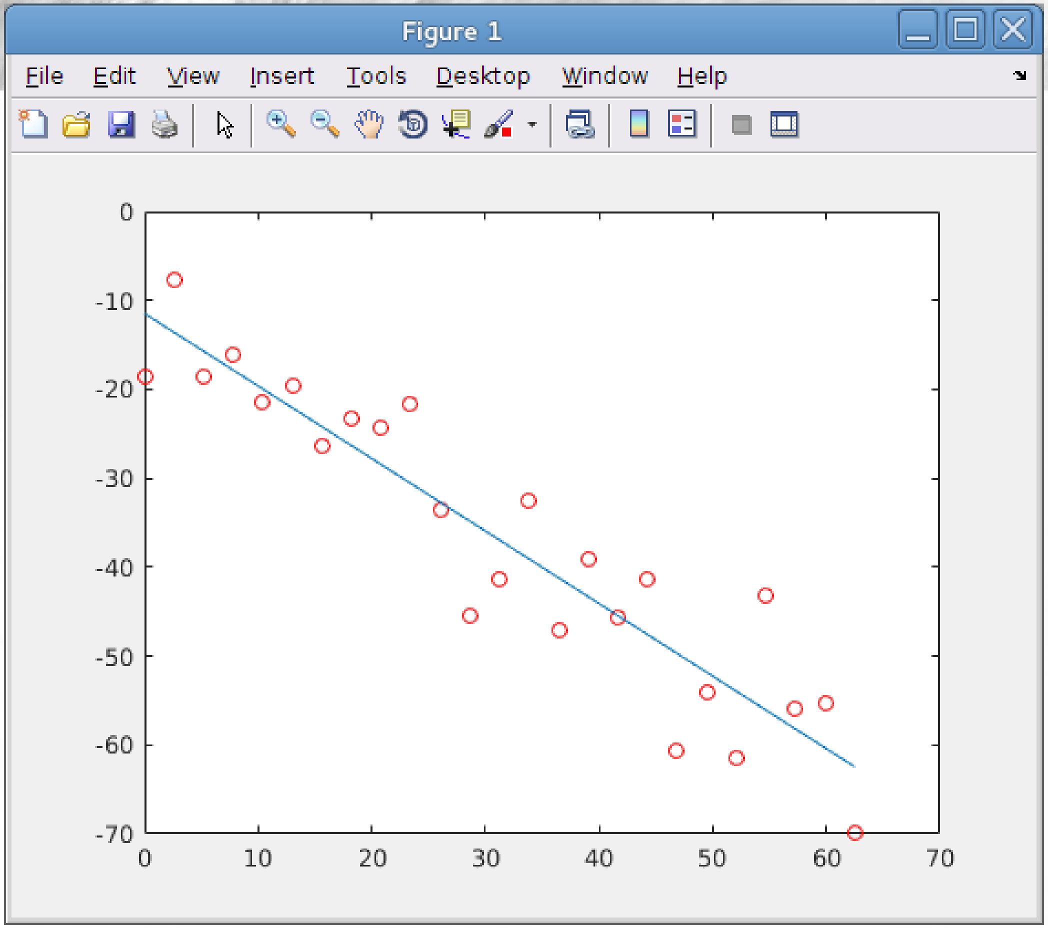
$$\boxed{s = \left(A^T A\right)^{-1} A^T t}$$

Linear least squares fit to data.

Matlab implementation

```
function [y0, b] = normaleqs(x, y)
  % This fcn uses the normal equations to
  % compute a linear fit to the data [x, y].  The approach
  % solves the normal equations using s = (A'*A)\(A'*t)

  N = length(x);

  % Create A matrix
  e = ones(N, 1);
  A = [x, e];

  % Create t vector
  t = y;

  % Solve normal equations
  s = (A'*A)\(A'*t);
  b = s(1);
  y0 = s(2);
end
```

$$s = \left(A^T A\right)^{-1} A^T t$$




```
>> test_normaleqs
-------------------------------------------------
Original yzero = -5.739764, fitted y0 = -2.719575
Original beta = 0.209749, fitted b = -0.077015
-------------------------------------------------
Original yzero = -9.665115, fitted y0 = -11.465547
Original beta = -0.877932, fitted b = -0.815984
-------------------------------------------------
Original yzero = -2.343078, fitted y0 = -4.658100
Original beta = -0.544939, fitted b = -0.539390
```

Another way to get the normal eqs

- Recall we want to “solve” a linear system

$$As=t \longleftrightarrow \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

A, *t* known quantities, solve for *s*.

- “Solve” is in quotes since the best we can do is minimize the residual.
- Problem: A is not square.

A trick.....

- Start with the rectangular system:

$$A\,s=t$$

- Make it square by multiplying through by A^T:

$$A^T A\,s=A^T t$$

A^TA is always square

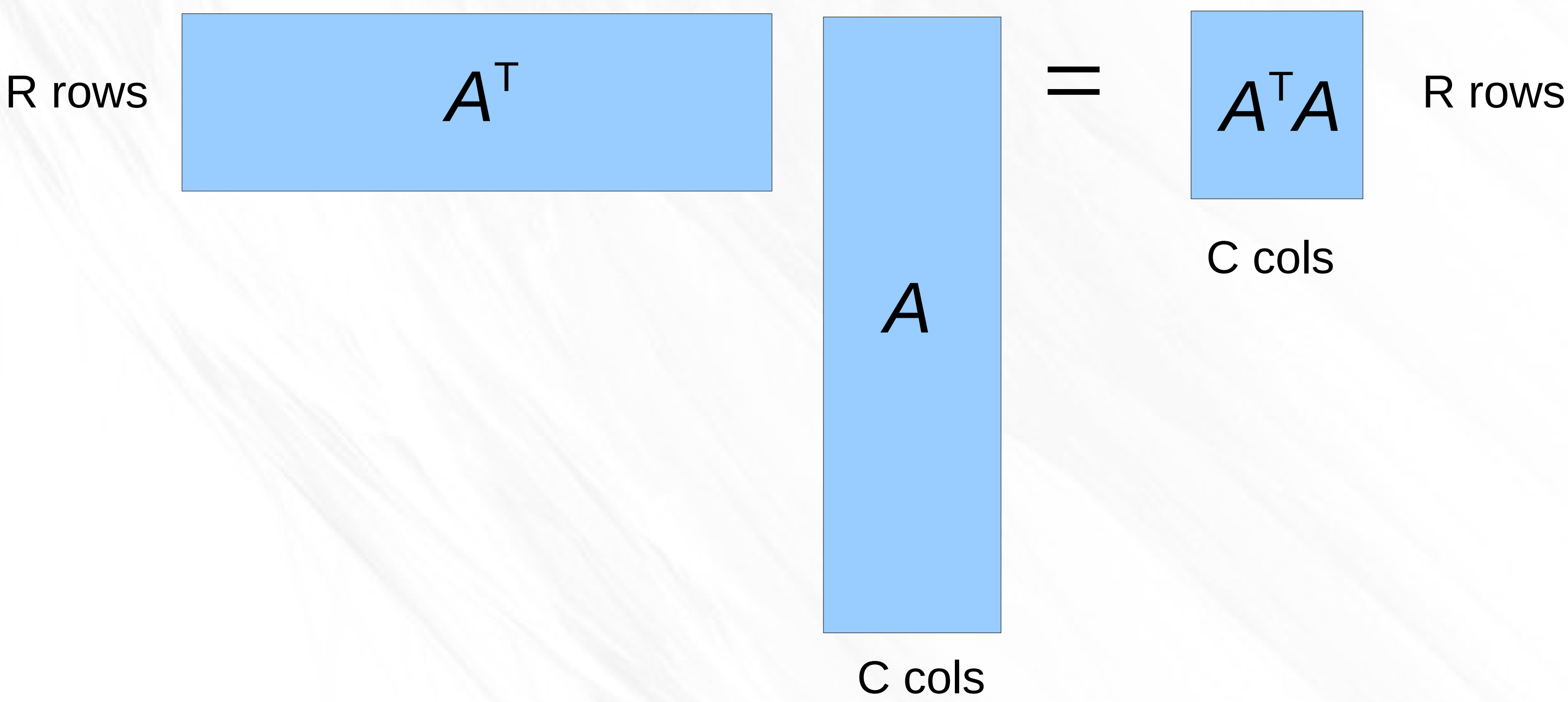
- The LHS matrix is now square. Therefore, we can solve it to get coefficient vector *s*.

$$s=(A^T A)^{-1} A^T t$$

- Voila!

A^TA is always square

- Just draw a picture….



A trick.....

- Start with the rectangular system:

$$A\,s = t$$

A rectangular – can't solve

- Make it square by multiplying through by A^T:

$$A^T A\,s = A^T t$$

$A^T A$ is always square

- The LHS matrix is now square. Therefore, we can solve it to get coefficient vector *s*.

$$s = \left(A^T A\right)^{-1} A^T t$$

- Voila!

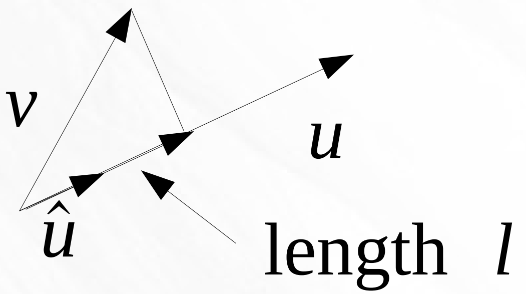
This feels rather suspicious. To make sense of it let's talk about linear algebra again…..

What just happened?

- Consider a vector *v* and vector *u*.
- What is the amount of *v* which points in the same direction as *u*?

Scalar:

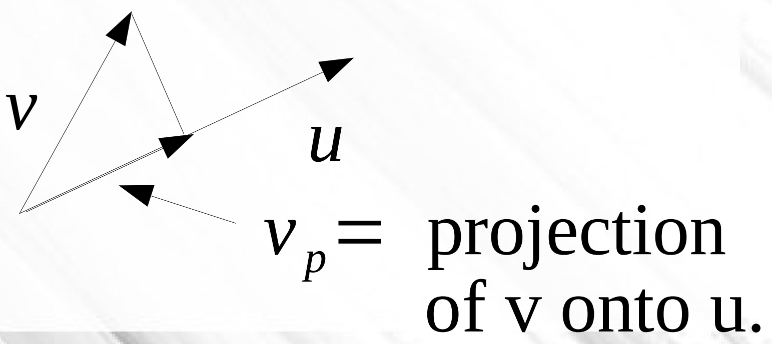
$$l = \hat{u} \cdot \vec{v} = \left(\frac{u^T}{\|u\|} \right) v$$



- What is vector of length *l* pointing in same direction as *u*?

Vector:

$$v_p = \hat{u}\, l = \left(\frac{u}{\|u\|} \right) l = \frac{u\, u^T}{\|u\|^2} v$$



Projection operator

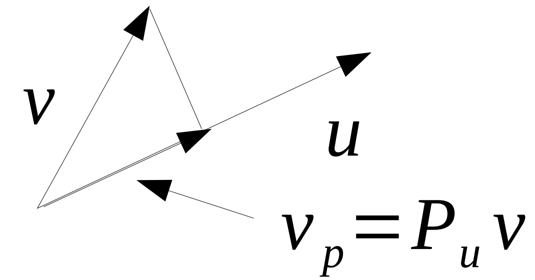
- If $\hat{u}$ is unit vector, the projection of *v* on $\hat{u}$ is

$$v_p = \hat{u}(\hat{u}^T v) = (\hat{u}\hat{u}^T) v$$
$$= P_u v$$

Note that P_u is a matrix formed by an outer product.

where P_u is a “projection operator”.

- If $\hat{u}$ is unit vector, projection operator P_u is matrix which takes arbitrary *v* to vector pointing in direction of $\hat{u}$.

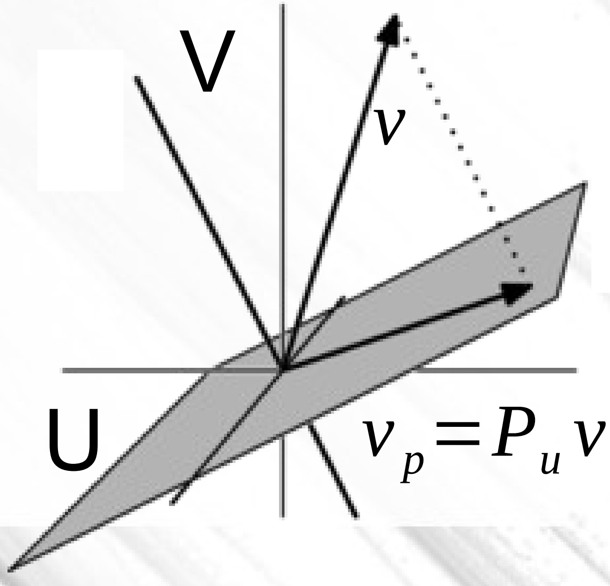


- If *u* is not a unit vector, the projection operator is

$$P_u = \frac{u u^T}{\|u\|^2}$$

What about projecting a vector onto a vector space?

- Consider some vector *v* living in a high-dimensional vector space V.
- Consider a low dimensionality vector space U spanned by orthonormal basis vectors u_i
- We take U as subset of V.
- How can I project *v* onto U?
- Think of projecting vector v in 3D onto plane.

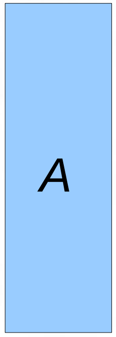


In matrix notation

- Construct matrix from orthonormal basis vectors u_i

$$A = \begin{pmatrix} \vdots & \vdots & \vdots & \\ u_1 & u_2 & u_3 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}$$

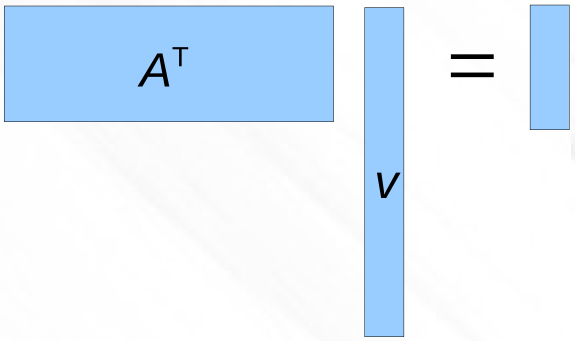
Note A is low rank



- It's easy to show

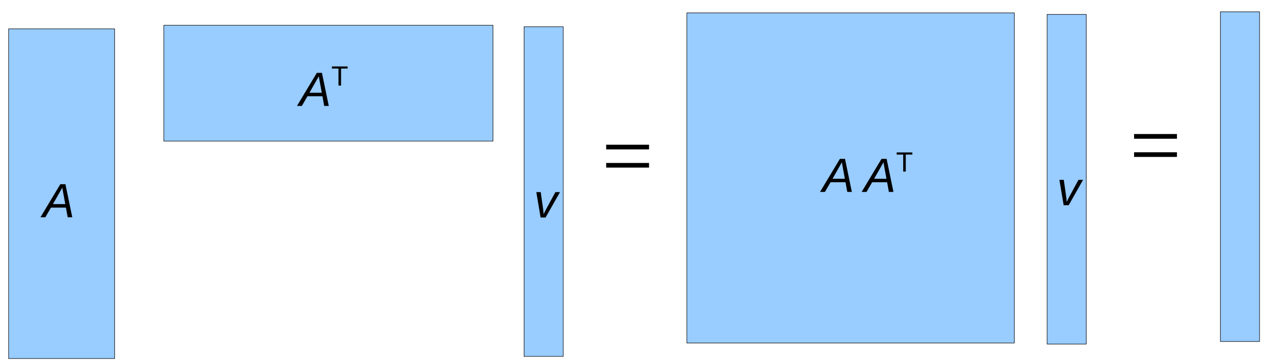
$$A^T v = \begin{pmatrix} \cdots & u_1^T & \cdots \\ \cdots & u_2^T & \cdots \\ \cdots & u_3^T & \cdots \\ & \vdots & \end{pmatrix} v = \begin{pmatrix} u_1^T v \\ u_2^T v \\ u_3^T v \\ \vdots \end{pmatrix}$$

length of *v* in direction u_i



- And,

$$A A^T v = \begin{pmatrix} \vdots & \vdots & \vdots & \\ u_1 & u_2 & u_3 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix} \begin{pmatrix} u_1^T v \\ u_2^T v \\ u_3^T v \\ \vdots \end{pmatrix} = \begin{pmatrix} u_1 u_1^T v \\ u_2 u_2^T v \\ u_3 u_3^T v \\ \vdots \end{pmatrix}$$



$$= P v$$

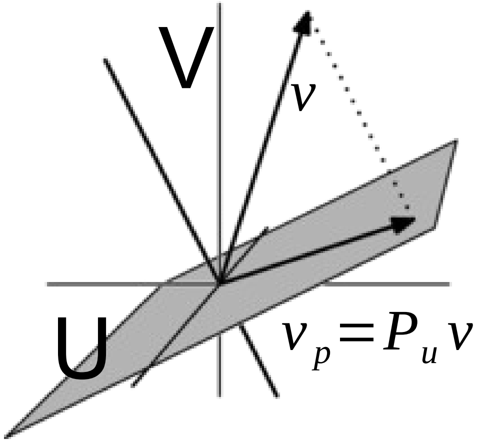
Note P = $A A^T$ is a square, low-rank matrix

- $A A^T$ is projection matrix taking v onto space spanned by u_i.

Projection operator

- Projection operator taking vector *v* and projecting it onto a subspace U:

$$P_u = (A A^T) \qquad A = \begin{pmatrix} \vdots & \vdots & \vdots & \\ u_1 & u_2 & u_3 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix}$$



- Recall this expression was derived assuming orthonormal u_i.
- I claim, if u_i are *not* orthonormal, the projection operator is

Middle piece acts something like a normalization factor.

$$P_u = A(A^T A)^{-1} A^T$$

Now consider what we are doing with 1D linear regression

- Recall we want to “solve” a linear system

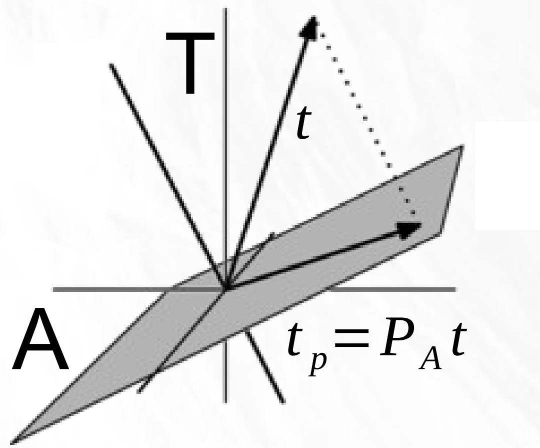
$$As = t \quad \longleftrightarrow \quad \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

- This has no exact solution since it is overdetermined.

- But what I can do is find the vector *As* which is closest to vector *t*.

Find closest vector *As* to vector *t*

- To get the closest vector we project *t* onto the subspace spanned by *A*.



$$t_p = P_A t$$

t_p is closest vector to *t* lying in subspace.

- Substitute the projection operator

$$P_A = A\left(A^T A\right)^{-1} A^T$$

into *As* = *t* equation and get

$$A s = t_p = A\left(A^T A\right)^{-1} A^T t$$

Getting the Normal Equation

- On last slide we used the projection operator to get vector $As = t_p$ closest to t. This gives equation:

$$As = t_p = A\left(A^T A\right)^{-1} A^T t$$

- Assuming A is non-singular, this simplifies to

$$s = \left(A^T A\right)^{-1} A^T t$$

which is what we wanted to show.

- This means normal equation is consequence of projecting t onto subspace spanned by A.

1D Linear regression -- Visualization

1. Recall *t* is vector living in *N* dimensional space

$$A\,s=t \quad \longleftrightarrow \quad \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

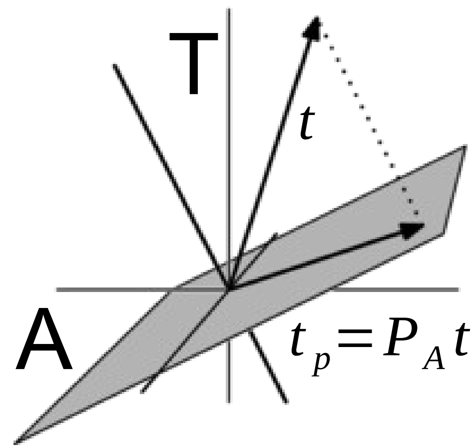
N values

2. *A* is 2 dimensional subspace with basis [u_1, u_2]

Rank 2

$$A=\begin{pmatrix} \vdots & \vdots \\ u_1 & u_2 \\ \vdots & \vdots \end{pmatrix} \qquad u_1=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{pmatrix} \qquad u_2=\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \end{pmatrix}$$

3. Project t onto 2D subspace spanned by A



$$t_p = P_A t$$
$$= A\left(A^T A\right)^{-1} A^T t$$

4. Find s which satisfies the equation

$$A s = t_p$$

Since t_p lives in 2D space spanned by A, this equation can be solved exactly.

5. The solution $s = \begin{pmatrix} \beta \\ y_0 \end{pmatrix}$ are the coefficients corresponding to the desired vector t_p

Another detour: Pseudoinverse

- We started with overdetermined system:

$$A\,s=t \quad \longleftrightarrow \quad \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

- We found (min residual) solution $s=\left[\left(A^T A\right)^{-1} A^T\right]t$

- This suggests the expression $\left(A^T A\right)^{-1} A^T$ is something like an inverse:

$$s=\left[\left(A^T A\right)^{-1} A^T\right]t$$

Pseudoinverse can be computed for any shape matrix.

Moore-Penrose Pseudoinverse

- Pseudoinverse: $A^{+}=\left(A^{T}A\right)^{-1}A^{T}$

- Can be computed for any shape matrix (square, rectangular....)

- For square matrix, it reduces to the usual matrix inverse:

$$A^{+}=\left(A^{T}A\right)^{-1}A^{T}$$

This only makes sense for square A

$$=\left(A^{-1}(A^{T})^{-1}\right)A^{T}$$

$$=A^{-1}$$

Pseudoinverse and the SVD

- SVD $A = U \Sigma V^T$

$$= U \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix} V^T$$

- Pseudoinverse

$$A^+ = V \begin{pmatrix} 1/\sigma_1 & 0 & 0 \\ 0 & 1/\sigma_2 & 0 \\ 0 & 0 & 1/\sigma_3 \end{pmatrix} U^T$$

Demo that pseudoinverse using SVD and pinv are the same

```
>> A = randn(5, 2)

A =

    0.1853   -2.4878
    0.3493    0.1980
   -0.6291    0.9454
    1.0889   -0.7958
    2.3531    0.0897

>> [U, S, V] = svd(A, 'econ')

U =

    0.6618   -0.6132
    0.0258    0.1627
   -0.3719    0.0582
    0.4336    0.1260
    0.4847    0.7605


S =

    3.0297         0
         0    2.4206


V =

    0.6530    0.7574
   -0.7574    0.6530
```

Note I use econ to get SVD with square S matrix.

```
>> Spinv = diag(1./diag(S))

Spinv =

    0.3301         0
         0    0.4131

>> Apinv = V*Spinv*U'

Apinv =

   -0.0492    0.0565   -0.0619    0.1329    0.3424
   -0.3309    0.0374    0.1087   -0.0744    0.0840

>> Apinv*A

ans =

    1.0000   -0.0000
   -0.0000    1.0000

>> pinv(A)

ans =

   -0.0492    0.0565   -0.0619    0.1329    0.3424
   -0.3309    0.0374    0.1087   -0.0744    0.0840
```

$$A^{+} = V\,\Sigma^{-1}\,U^{T}$$

Apinv acts like an inverse.

Same

Pseudoinverse and Matlab

```
>> B = rand(5,3)

B =

    0.9361    0.9295    0.8786
    0.1862    0.1368    0.3874
    0.5074    0.8716    0.2464
    0.1476    0.0124    0.1117
    0.9207    0.7220    0.8468

>> IB = pinv(B)

IB =

   -0.8060   -4.6997   -0.8065    2.7881    2.8535
    0.4643    1.4841    1.6455   -1.3894   -1.4564
    0.9076    3.7390   -0.6319   -1.5142   -1.0878

>> IB*B

ans =

    1.0000    0.0000    0.0000
         0    1.0000    0.0000
    0.0000    0.0000    1.0000
```

pinv()

Next: Multiple linear regression

- Linear algebra approach (normal equations) easily generalizes to multivariate data fitting.
- Multiple linear regression:

Many independent variables

Scalar dependent variable

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots$$

- Choose concrete example: house prices.
- Assume simple house price model, where price is sum of:
    - Base price
    - Cost per bedroom
    - Cost per square foot

House price model

- Model: Price = base + Pbr*Nbr + Psq*A

Base price

Bedroom coefficient

# of bedrooms

Area coefficient

Area of house

- Goal: Fit a dataset to extract coefficients base, Pbr, Psq.
- This is a classic example of multiple linear regression

$$y=\beta_0+\beta_1 x_1+\beta_2 x_2+\beta_3 x_3+\cdots$$

Example dataset

Nbr	Area	Sale price
3	2379	698000
1	1263	370000
4	1069	750000
3	2616	722000
2	2514	610000
3	2777	775000
1	1871	552000
1	2516	603000
3	2141	748000
4	2580	802000

I assume no correlation between Nbr and Area. This makes everything simple...

- Model: Price = base + Pbr*Nbr + Psq*A

House price model

- Recall 1D formula:

$$X\beta = y \quad \longleftrightarrow \quad \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

Note change of
nomenclature

- Generalize to 2D:

# of bedrooms

Area

Price

$$X\beta = y \quad \longleftrightarrow \quad \begin{pmatrix} 1 & Nbr_1 & A_1 \\ 1 & Nbr_2 & A_2 \\ 1 & Nbr_3 & A_3 \\ 1 & Nbr_4 & A_4 \\ \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ \vdots \end{pmatrix}$$

Model coefficients to find

Solving multiple linear regression

- We have the overdetermined system:

Matrix of known x values arranged in columns.

$$X\beta = y$$

Known y values (column vector)

- Solve to get the desired parameters:

$$\boxed{\beta = \left(X^T X\right)^{-1} X^T y}$$

$$\begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} \mathrm{base} \\ \mathrm{Pbr} \\ \mathrm{Psq} \end{pmatrix}$$

- Coefficients used in model:

Price = base + Pbr*Nbr + Psq*A

```
function make_prices()
  % This fcn creates house prices for use in later
  % multi-variate regression

  FID = fopen('HousePrices.csv', 'w');

  % Define model parameters
  base = 250000;  % Base price
  Pbr = 100000;   % Price increase per bedroom
  Psqft = 70;     % Price increase per square foot

  % Nbr = Number of bedrooms
  % Sqft = Area of house in square feet
  % Create our price model
  p = @(N, S) base + Pbr*N + Psqft*S + 50000*randn();


  N = 50;  % Number of houses to create
  for idx = 1:N
    Nbr = randi(4, 1);
    Sqft = round(1000 + 2000*rand());
    price = 1000*round(p(Nbr, Sqft)/1000);

    fprintf('Nbr = %d, Sqft = %d, price = %d\n', Nbr, Sqft, price);
    fprintf(FID, '%d, %d, %d\n', Nbr, Sqft, price);
  end
  fclose(FID)

end
```

```
3, 2379, 698000
1, 1263, 370000
4, 1069, 750000
3, 2616, 722000
2, 2514, 610000
3, 2777, 775000
1, 1871, 552000
1, 2516, 603000
3, 2141, 748000
4, 2580, 802000
3, 1510, 656000
1, 1112, 456000
3, 1163, 657000
3, 2712, 750000
2, 1214, 473000
3, 2398, 697000
2, 1732, 461000
1, 1242, 453000
```

This fcn reads in file containing house price data. Then does linear regression, and reports the computed regression coefficients.

```
function fit_prices()

  filename = 'HousePrices.csv';

  % Read in data and then split it up.  Data lines are of form
  % Nbr, Sqft, price
  M = csvread(filename);

  Nbr = M(:, 1);
  Sqft = M(:, 2);
  price = M(:, 3);

  % Now create matrix of independent variables
  e = ones(length(Nbr), 1);
  A = horzcat(e, Nbr, Sqft);

  % Now do regression
  s = (A'*A)\(A'*price);

  % print out results
  base = s(1);
  Pbr = s(2);
  Psqft = s(3);

  fprintf('Base = %f, Pbr = %f, Psqft = %f\n', base, Pbr, Psqft)

end
```

$$\beta = \left(X^T X \right)^{-1} X^T y$$

House price regression

- make_prices.m:

```
% Define model parameters
base = 250000;  % Base price
Pbr = 100000;   % Price increase per bedroom
Psqft = 70;     % Price increase per square foot
```

- fit_prices.m:

```
>> fit_prices
Base = 264256.016019, Pbr = 110657.206348, Psqft = 48.959504
```

- predict_prices.m:

```
>> predict_prices(3, 1500)
Base = 264256.016019, Pbr = 110657.206348, Psqft = 48.959504
Predicted house price = $ 669666.89
```

Another example: fit car prices

- Imagine a model for automobile prices. What are the independent variables?
  - Age of car (years)
  - Number of miles driven
  - Length of car
  - Weight of car

Idea: Older cars are more worn out, need more repair.

Idea: Bigger cars cost more.

- Other possible independent variables:
  - Car manufacturer
  - Car model

“Catagorical” variables, can't treat with linear regression.

Real data

- Collect car price data from web
- Do multiple least squares fit to extract:
  - Base price
  - Price/age
  - Price/mileage

Price decreases with age

Price decreases with mileage

```
>> pd = fitprices_normal('DodgeDurango.csv');
cond(X) = 1.113634e+02
base = 38897.370864, age = -2298.457053, mileage = -100.668865
>> pd = fitprices_normal('FordExplorer.csv');
cond(X) = 1.499720e+02
base = 33414.212943, age = -2417.365056, mileage = 119.790824
```

????

Problem: statistical correlation

$$\beta = \left(X^T X\right)^{-1} X^T y$$

- Predictors:
  - Age of car (years)
  - Number of miles driven
- These quantities are highly correlated.
- Effect:
  - high condition number
  - If quantities are exactly correlated, the data matrix is singular.
  - Fit coefficients can vary with noisy *X*

$k(X)$ large $\Rightarrow$

$k(X^T X)$ large^2

Look at condition number....

```
>> pd = fitprices_normal('DodgeDurango.csv');
cond(X) = 1.113634e+02
base = 38897.370864, age = -2298.457053, mileage = -100.668865
>> pd = fitprices_normal('FordExplorer.csv');
cond(X) = 1.499720e+02
base = 33414.212943, age = -2417.365056, mileage = 119.790824
```

$$k(X) \text{ large} \Rightarrow$$
$$k(X^T X) \text{ large}^2$$

- Normal equations:

$$\beta = \left(X^T X\right)^{-1} X^T y$$

- Effect: Fit very sensitive to errors in input data.

Consider the limit: Pseudoinverse chokes on singular matrix

```
>> A = [1 2 3 4 5; 1 2 3 4 5; 1 2 3 4 5]'

A =

     1     1     1
     2     2     2
     3     3     3
     4     4     4
     5     5     5

>> (A'*A)\A'
Warning: Matrix is singular to working precision.

ans =

   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN

>>
```

Data in different columns is perfectly correlated. Perfectly correlated data corresponds to singular matrix.

Next: Multiple linear regression using QR

- Problem: Pseudoinverse method computes

$$\left(X^T X\right)^{-1} X^T$$

Pseudoinverse of *X*

  - Product $X^T X$ can be badly conditioned.
- Are there methods which don't compute $X^T X$?
  - QR
  - SVD

Won't do this one here, but
the concept is similar to QR.

Recall QR decomposition

- For overdetermined system, QR exists

$$X\beta = y \quad \longleftrightarrow \quad \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \end{pmatrix} \begin{pmatrix} \beta \\ y_0 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \end{pmatrix}$$

- QR decomposition: $QR = X$
  - Q orthogonal: $Q^T = Q$
  - R upper triangular, zero padded:

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ 0 & r_{22} & r_{23} & r_{24} \\ 0 & 0 & r_{33} & r_{34} \\ 0 & 0 & 0 & r_{44} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Multiple linear regression using QR

- Start with: $X\beta = y$
- QR decomposition: $QR\beta = y$

Condition number of Q = 1

- Q is orthogonal: $R\beta = Q^{-1}y = Q^T y$

R is upper triangular, solve operation is well conditioned.

- Solve for beta: $\beta = R^{-1}Q^T y$
- This method is better conditioned than normal equations.

Matlab: qr()

```
>> A = [1 2 3 4 5 6 7; 7 6 5 4 3 2 1; 1 2 3 4 3 2 1]'

A =

     1     7     1
     2     6     2
     3     5     3
     4     4     4
     5     3     3
     6     2     2
     7     1     1

>> [Q, R] = qr(A)

Q =

   -0.0845   -0.6761    0.4717    0.1221   -0.0746   -0.2713   -0.4681
   -0.1690   -0.5071    0.1048    0.1798    0.3194    0.4590    0.5986
   -0.2535   -0.3381   -0.2621   -0.7259   -0.4149   -0.1040    0.2070
   -0.3381   -0.1690   -0.6290    0.5968   -0.2809   -0.1586   -0.0363
   -0.4226    0.0000   -0.2621   -0.2304    0.7681   -0.2335   -0.2350
   -0.5071    0.1690    0.1048   -0.0576   -0.1830    0.6916   -0.4338
   -0.5916    0.3381    0.4717    0.1152   -0.1340   -0.3833    0.3675


R =

  -11.8322   -7.0993   -5.4090
         0   -9.4657   -2.7045
         0         0   -2.7255
         0         0         0
         0         0         0
         0         0         0
         0         0         0
```

Matlab implementation

```
function c = fitprices_qr(filename)
  FID = fopen(filename, 'r');
  M = csvread(filename, 1, 0);

  e = ones(size(M, 1), 1);
  X = [e, M(:, 1:2)];
  p = M(:, 3);
  fprintf('cond(X) = %e\n', cond(X))

  [Q, R] = qr(X);

  % Assumes model of form p = Xc
  c = R\Q'*p;

  fprintf('base = %f, age = %f, mileage = %f\n', c(1), c(2), c(3))
end
```

$$\beta = R^{-1} Q^T y$$

Session summary

Works for models which are linear in the coefficients.

- Linear regression -- 1D
- Two approaches:
  - Minimize least squares error
  - Normal equations. $s=\left(A^T A\right)^{-1} A^T t$
- Multiple linear regression
  - Normal equations $\beta=\left(X^T X\right)^{-1} X^T y$
  - Pseudoinverse
  - QR method $\beta=R^{-1} Q^T y$