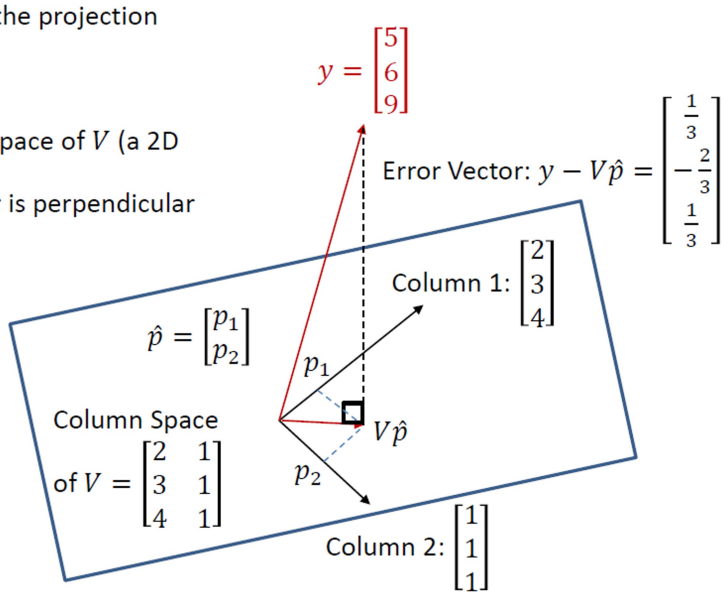
Lecture 20

Projection onto the Column Space

$V\hat{p} = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 4\frac{2}{3} \\ 6\frac{2}{3} \\ 8\frac{2}{3} \end{bmatrix}$ is the projection of $y = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix}$ onto the column space of V (a 2D plane), such that error vector is perpendicular to the column space.

$V^T(y - Vp) = 0$



Regarding the least square solution $\hat{p} = [(V^TV)^{-1}V^T]\, y$, to the problem $Vp = y$:

- $V\hat{p}$ is the projected point of y on the column space of V, by constructing a perpendicular line from y to the column space.
- $E = ||V\hat{p} - y|| = ||y - V\hat{p}||$, is the distance from y to the point $V\hat{p}$ in the column space.
- Searching for the least-square solution, which minimizes E, or equivalently, E^2, is the same as locating the point $V\hat{p}$, that is closer to y than any other points in the column space of V.
- The error vector $(y - Vp)$ or $(Vp - y)$ must be perpendicular to the column space of V.

The projected point $V\hat{p} = V[(V^TV)^{-1}V^T]\, y = Sy$, where the $m \times m$ square matrix $S = V(V^TV)^{-1}V^T$ is called a **Projection Matrix**. It can be shown that in general:

- $S = S^2 = S^3 = \cdots$
- $S^{\mathrm{T}} = S$

Numerical Stability

- The least square solution to a generally inconsistent system $Vp = y$ of m equations in n unknowns satisfies the *normal equation*: $V^T Vp = V^T y$.
- If the columns of V are linearly independent, then $V^T V$ is invertible, we can find $\hat{p} = (V^T V)^{-1} V^T y$, by using the pseudoinverse method.
- How sensitive is the solution $\hat{p}$ to a small change of V?
  - **Condition Number** of the matrix V

$$\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix}$$

V p ? y

SVD

- Solving the *normal equation*: $V^T Vp = V^T y$ might lead to even worse numerical instability due to the squaring of the conditional number $\kappa(V)$.
- Another method is Singular Value Decomposition (SVD), used by sklearn.
- SVD factorize a matrix V into the product of three matrices: $V = ASB^T$, where the middle matrix S contains the singular values.

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

A =

  -0.3913   0.8247
  -0.5606   0.1382
  -0.7298  -0.5484

S =

   5.6402        0
        0   0.4343

B =

  -0.9545  -0.2982
  -0.2982   0.9545
```

```
>> V = [2, 1; 3, 1; 4, 1]

V =

   2   1
   3   1
   4   1
```

```
>> y =[5; 7; 9]

y =

   5
   7
   9

>> p = B*S^(-1)*A'*y

p =

   2.0000
   1.0000
```

$V \times p = y$, where $V = A \times S \times B^T$

$(A \times S \times B^T) \times p = y$, both sides multiplied by $(B \times S^{-1} \times A^T)$, we have
$(B \times S^{-1} \times A^T) \times (A \times S \times B^T) \times p = (B \times S^{-1} \times A^T) \times y$, where

$B \times S^{-1} \times A^T \times A \times S \times B^T \times p = p$, since
$A^T \times A = I$, $S^{-1} \times S = I$, and $B \times B^T = I$

Thus $p = (B \times S^{-1} \times A^T) \times y$

```
>> [A,S,B] = svd (V)

A =

   -0.3913    0.8247    0.4082
   -0.5606    0.1382   -0.8165
   -0.7298   -0.5484    0.4082

S =

    5.6402         0
         0    0.4343
         0         0

B =

   -0.9545   -0.2982
   -0.2982    0.9545
```

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

A =

   -0.3913    0.8247
   -0.5606    0.1382
   -0.7298   -0.5484

S =

    5.6402         0
         0    0.4343

B =

   -0.9545   -0.2982
   -0.2982    0.9545
```

- Solving the *normal equation*: $V^T V p = V^T y$ might lead to even worse numerical instability due to the squaring of the conditional number $\kappa(V)$.
- There is a need to use other methods, e.g., QR decomposition, where R is a upper triangular matrix (square matrix with all the entries below the main diagonal being zero), and Q is a norm-preserving orthogonal matrix (whose columns are orthonormal vectors).

$$Q \times R \times p = y \rightarrow Q^{\mathrm{T}} \times Q \times R \times p = Q^{\mathrm{T}} \times y \rightarrow R \times p = (Q^{\mathrm{T}} \times y),$$

$Vp = y$

$Q^{\mathrm{T}} \times Q = I$

```
>> [Q,R] = qr(V, 0)

Q =

   -0.3714    0.8339
   -0.5571    0.1516
   -0.7428   -0.5307

R =

   -5.3852   -1.6713
         0    0.4549
```

$$\times p = (Q^{\mathrm{T}} \times y),$$

$$\begin{bmatrix} -5.3852 & -1.6713 \\ 0 & 0.4549 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -12.4416 \\ 0.4549 \end{bmatrix}$$

```
>> Q'*y

ans =
```

```
R =

   -5.3852   -1.6713
         0    0.4549

>> Q'*Q

ans =

    1.0000    0.0000
    0.0000    1.0000

>> Q'*y

ans =

   -12.4416
     0.4549

>> mldivide (R, Q'*y ) % Avoid inversion of large matrix

ans =

    2.0000
    1.0000
```

http://www.ece.uah.edu/~dwpan/course/ee610/code/Regression/curvefit_demo.m

Example: $y(x, w) = w_0 + w_1 x + w_2 x^2$

- Training data with four samples: (x_1, t_1), (x_2, t_2), (x_3, t_3), (x_4, t_4).
- Predicted values:

$$y_1 = w_0 + w_1 x_1 + w_2 x_1^2$$
$$y_2 = w_0 + w_1 x_2 + w_2 x_2^2$$
$$y_3 = w_0 + w_1 x_3 + w_2 x_3^2$$
$$y_4 = w_0 + w_1 x_4 + w_2 x_4^2$$

- Using row-vector and matrix operations:

$Y = [y_1\ y_2\ y_3\ y_4],\ W = [w_0\ w_1\ w_2],\ Z = [t_1\ t_2\ t_3\ t_4]$

$$Y = WF_X = [w_0 \quad w_1 \quad w_2] \begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 \end{bmatrix}$$

- We want the best approximation in the least-square sense: $Z \approx WF_X$
- The system of linear equations are overdetermined since there are more equations than unknowns. In Matlab, $W = Z \ / \ F_X$

```
N = 4
% Generate 4 data points for training
rng(1);
x = 10*rand(1, N);

Z = 1 + 2*x + 3*x.^2;  % Target values

% Formulate the input data matrix
Fx = zeros(3,N);
for i = 1:N
   Fx(:,i) = [1, x(i), x(i)^2];
end
>> Fx

Fx =

   1.0000   1.0000   1.0000   1.0000
   4.1702   7.2032   0.0011   3.0233
  17.3907  51.8867   0.0000   9.1405
```

```
>>  W = Z / Fx

W =

   1.0000   2.0000   3.0000

>> W2 = Z * pinv(Fx)

W2 =

   1.0000   2.0000   3.0000
```

Matlab: polyfit ( ) function

```
>> V = fliplr(Fx')

V =

   17.3907    4.1702    1.0000
   51.8867    7.2032    1.0000
    0.0000    0.0011    1.0000
    9.1405    3.0233    1.0000
```

- p = polyfit (x,y,n) returns the coefficients for a polynomial p(x) of degree n that is a best fit (in a least-squares sense) for the data in y. The coefficients in p are in descending powers, and the length of p is n+1.

$$p(x) = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}.$$

- polyfit uses x to form a Vandermonde matrix V with m = length(x) rows and (n+1) columns, resulting in the linear system below, which polyfit solves with p = V\y = pinv(V) * y.

$$\begin{pmatrix} x_1^n & x_1^{n-1} & \cdots & 1 \\ x_2^n & x_2^{n-1} & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_m^n & x_m^{n-1} & \cdots & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix},$$

m x (n+1) (n+1) x 1 m x 1

```
>> [Q,R] = qr(V,0);   % Economy-size QR decomposition
>> p2 = mldivide(R, Q'*Z');
>> p2

p2 =

    3.0000
    2.0000
    1.0000
```

$3x^2 + 2x + 1$