EE 610, ML Fundamentals

Linear Models for Regression

Dr. W. David Pan
Dept. of ECE
UAH

- Linear models for regression
- Polynomial curve fitting as an illustrative example
- Solution to a least-square problem
- Math review
  - Vector Calculus
  - Linear Algebra
    - Vector Space, QR Decomposition, SVD, Condition Numbers, etc.
- Numerical Stability
- Implementations

- The goal of regression is to predict the value of one or more continuous target variables t given the value of a D-dimensional vector $\boldsymbol{x}$ of input variables.
- The polynomial is a specific example of a broad class of functions called linear regression models, which share the property of being linear functions of the adjustable parameters.
- we can also obtain a class of functions by taking linear combinations of a fixed set of nonlinear functions of the input variables, known as *basis functions*.
- Such models are linear functions of the parameters, which gives them simple analytical properties, and yet can be nonlinear with respect to the input variables.

- The simplest linear model for regression is one that involves a linear combination of the input variables. This is known as linear regression.

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D$$

- The key property of this model is that it is a linear function of the parameters $w_0, \ldots, w_D$
- We can extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

Example: Polynomial Curve Fitting

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- The polynomial coefficients $w_0, \ldots, w_M$ are collectively denoted by the vector $\boldsymbol{w}$.
- Although the polynomial function $y(x, \boldsymbol{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\boldsymbol{w}$.
- The values of the coefficients will be determined by fitting the polynomial to the training data.
- This can be done by minimizing an *error function* that measures the misfit between the function $y(x, \boldsymbol{w})$, for any given value of $\boldsymbol{w}$, and the training set data points.
- One common choice of error function is the sum of the squares of the errors between the predictions $y(x_n, \boldsymbol{w})$ for each data point x_n and the corresponding target values t_n, in order to minimize the error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

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:

$$
\begin{aligned}
y_1 &= w_0 + w_1 x_1 + w_2 x_1^2 \\
y_2 &= w_0 + w_1 x_2 + w_2 x_2^2 \\
y_3 &= w_0 + w_1 x_3 + w_2 x_3^2 \\
y_4 &= w_0 + w_1 x_4 + w_2 x_4^2
\end{aligned}
$$

- 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, $W = Z \,/\, F_X$

Matlab: polyfit ( ) function

- 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.

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

- 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

Example: Fit with a straight line

$p_1 x + p_2$, using notations of Matlab (weights are now p_i in reversed order, and the target values are now y_i).

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$, or $V_{(3\times 2)}\, p_{(2\times 1)} = y_{(3\times 1)}$

Given training data samples (x, y): (2, 5), (3, 7), (4, 9), the system of equations (with 2 unknowns and 3 equations):

$$\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix}$$

- Goal: Find a solution vector p such that the approximation error (squared) below is minimized: $E^2(p) = \|Vp - y\|^2$.
- We can use calculus, or geometry and linear algebra to solve the problem.

Gradient of Quadratic Function

$$E^2(p) = \|Vp - y\|^2 = (Vp - y)^T(Vp - y) = (p^T V^T - y^T)(Vp - y)$$

$$= p^T V^T V p - p^T V^T \mathrm{y} - \mathrm{y}^\mathrm{T} V p + \mathrm{y}^\mathrm{T}\mathrm{y}$$

$\nabla E^2(p) = \begin{bmatrix} \frac{\partial E^2}{\partial p_1} \\ \frac{\partial E^2}{\partial p_2} \end{bmatrix} = \nabla_p\left(p^T V^T V \mathrm{p} - p^T V^T \mathrm{y} - \mathrm{y}^\mathrm{T} V p + \mathrm{y}^\mathrm{T}\mathrm{y}\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, in order to determine the *critical point* that can potentially minimize $E^2(p)$, where

$$\nabla_p(p^T V^T V p) = 2(V^T V)p, \qquad \nabla_p(p^T V^T \mathrm{y}) = \nabla_p(y^T V p) = V^T \mathrm{y}, \quad \nabla_p(y^T y) = 0$$

Thus $(V^T V)p - V^T \mathrm{y} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, or $V^T V p = V^T \mathrm{y}$ (Normal Equation in Statistics)

$V^T V$ is invertible when the columns of *V* are linearly independent.
Best estimate (in least square sense): $\hat{p} = [(V^T V)^{-1} V^T]\, y = \mathrm{pinv}(V)\, y$

```
>> V = [2, 1; 3, 1; 4, 1];
>> y = [5 7 9]‘;
```

```
>> inv(V'*V)*V'*y
>> pinv(V)*y
```

```
ans =
   2.0000
   1.0000
```

Hessian Matrix (Derivative of Gradient)

$$\nabla_p^2 E^2(p) = \begin{bmatrix} \frac{\partial^2 E^2}{\partial p_1^2} & \frac{\partial^2 E^2}{\partial p_1 \partial p_2} \\ \frac{\partial^2 E^2}{\partial p_2 \partial p_1} & \frac{\partial^2 E^2}{\partial p_2^2} \end{bmatrix} = \nabla_p \begin{bmatrix} \frac{\partial E^2}{\partial p_1} \\ \frac{\partial E^2}{\partial p_2} \end{bmatrix}^T = \nabla_p (2V^T V p - 2V^T \mathrm{y})^T$$

$$= \nabla_p (2p^T V^T V - 2y^T \mathrm{V}) = 2V^T V$$

- $V^T V$ is always symmetric and positive definite (with all eigenvalues being positive, all pivots being positive), thus $\hat{p} = [(V^T V)^{-1} V^T]\, y$ is not only a critical point, but also a *local* minima.
- In addition, due to the Hessian being a (everywhere in general) positive definite matrix, $E^2(p)$ is a convex function, and $\hat{p}$ is also a **global minima**.

```
>> V'*V
ans =
   29    9
    9    3
```

```
>> det(V'*V)
ans =
    6.0000
```

```
>> EIG = eig(V'*V)
EIG =
    0.1886
   31.8114
```

```
>> EIG(1)*EIG(2)
ans =
    6.0000
```

Symbolic Matrix Operations

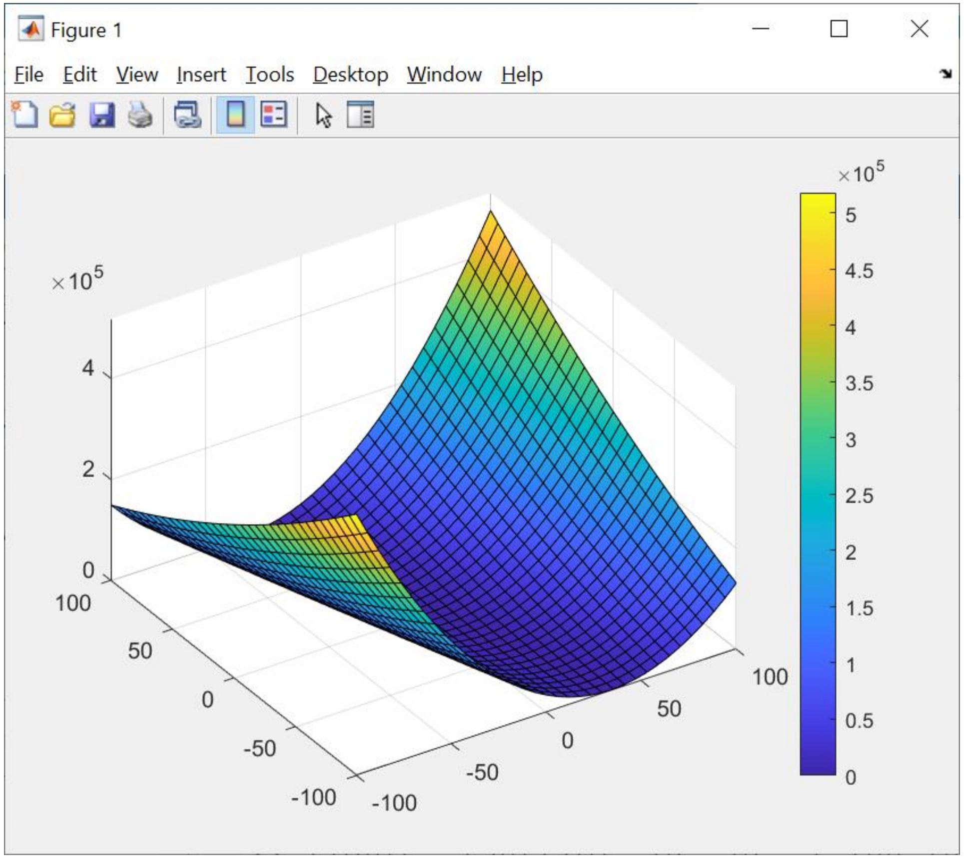
$$E^2(p) = \|Vp - y\|^2 = (Vp - y)^T(Vp - y) = (p^T V^T - y^T)(Vp - y)$$

$$= p^T V^T V p - p^T V^T \mathrm{y} - \mathrm{y}^{\mathrm{T}} V p + \mathrm{y}^{\mathrm{T}}\mathrm{y}$$

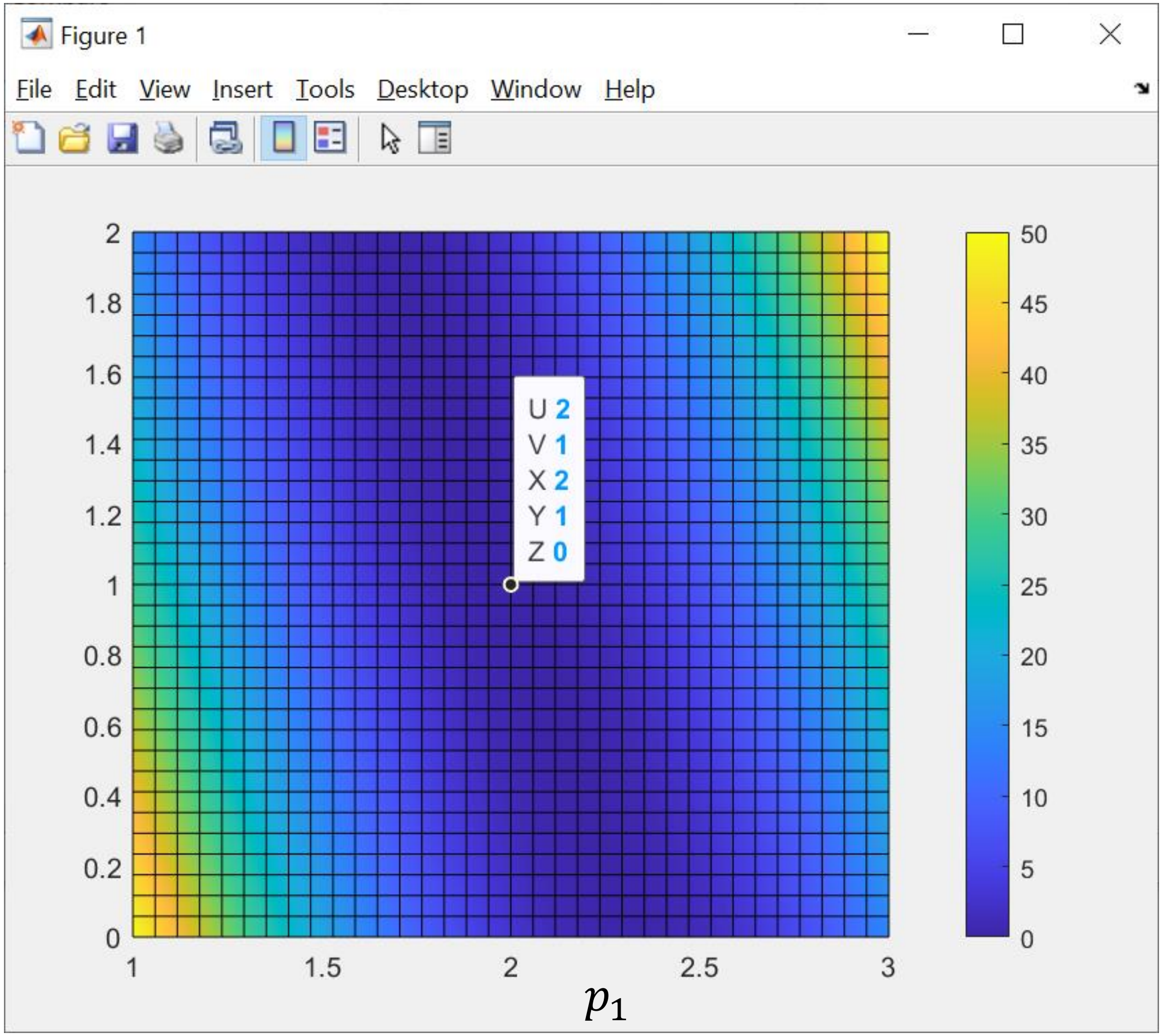
```
>> syms p1 p2 p E2(p1,p2)
p =[p1;p2];
V = [2, 1; 3, 1; 4, 1];
y = [5 7 9]';
E2(p1,p2) = (p.')*(V')*V*p -(p.')*(V')*y -
y'*V*p +y'*y;

>> simplify(E2)
ans = 29*p1^2 + 18*p1*p2 - 134*p1 +
3*p2^2 - 42*p2 + 155

>> fsurf(p1, p2, E2, [-100 100 -100
100]); colorbar;
```



Least Square



p_2

p_1

Geometric Interpretation

- The least square solution to a generally inconsistent system $Vp = y$ of m equations in n unknowns satisfies $V^T V p = V^T y$.
- If the columns of V are linearly independent, then $V^T V$ is invertible, and $\hat{p} = (V^T V)^{-1} V^T y$.
- In this specific example (with zero estimation error), the 3×1 vector y happens to be in the **column space** of the matrix V, with the solution 2×1 vector $\hat{p}$ containing the components (linear combination coefficients).

$$\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} = y, \text{ Solution: } \hat{p} = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$y = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} = p_1 \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + p_2 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Column Space of a Matrix

Given a $m \times n$ matrix V, its **column space** is the vector space formed by the columns of V. The column space contains all linear combinations of the columns of V. It is a subspace of $\boldsymbol{R}^m$.

- The column space consists of all vectors Vp for some $n \times 1$ vector p.
  - For example, $V = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}$ has a column space which is a 2D plane (a subspace in $\boldsymbol{R}^3$).
- Consider the following (slightly changed) least square problem:

$$\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} = y, \text{ then } V^{\mathrm{T}}V = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} = \begin{bmatrix} 29 & 9 \\ 9 & 3 \end{bmatrix}$$

$$\hat{p} = (V^T V)^{-1} V^T y = \begin{bmatrix} 29 & 9 \\ 9 & 3 \end{bmatrix}^{-1} \begin{bmatrix} 2 & 3 & 4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{11}{6} & \frac{1}{3} & -\frac{7}{6} \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 2 \\ \frac{2}{3} \end{bmatrix}$$

$$y = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} \approx \begin{bmatrix} 4\frac{2}{3} \\ 6\frac{2}{3} \\ 8\frac{2}{3} \end{bmatrix} = p_1 \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + p_2 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 2 \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + \frac{2}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Left Nullspace of a Matrix

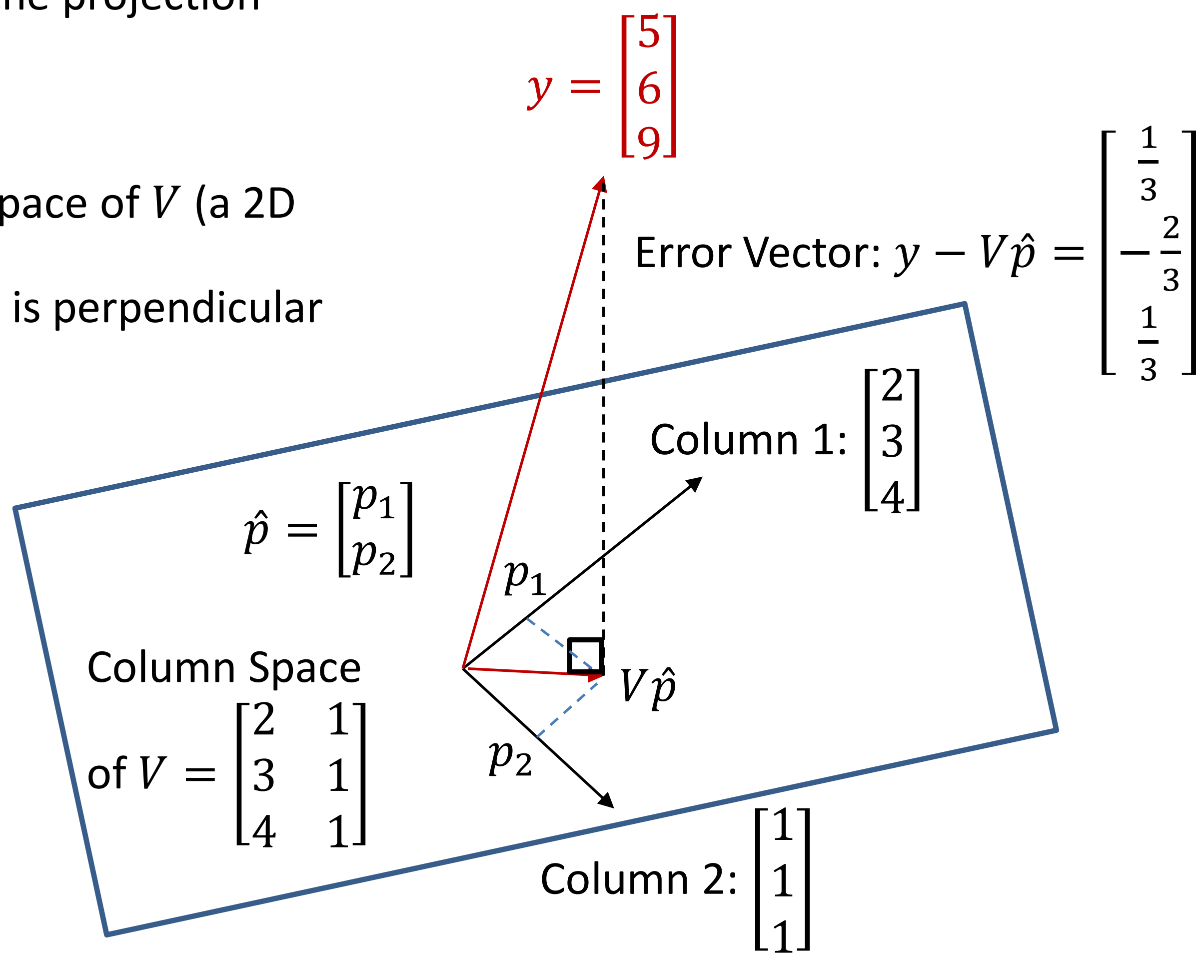
- The **nullspace** of a $m \times n$ matrix V consists of all vectors p such that $Vp = 0$. The nullspace is a subspace of $\boldsymbol{R}^m$, just as the column space.
- The **left nullspace** of a $m \times n$ matrix V is the **nullspace** of V^{T}. The left nullspace contains all vectors p such that $V^T p = 0$.
- $V^T V p = V^T \mathrm{y}$ (Normal Equation), or $V^T (y - Vp) = 0$, indicating the error vector $(y - Vp)$ **must be perpendicular to the column space of** $\boldsymbol{V}$. In other words,
- **The error vector is in the left nullspace of** $\boldsymbol{V}$.

Error Vector: $y - V\hat{p} = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} - \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ \frac{2}{3} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} \\ -\frac{2}{3} \\ \frac{1}{3} \end{bmatrix}$, which is orthogonal to all the column vectors of V, since $V^T (y - V\hat{p}) = \begin{bmatrix} 2 & 3 & 4 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} \\ -\frac{2}{3} \\ \frac{1}{3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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.

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



Equivalence of Algebraic and Geometric Interpretations

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

Projection Matrix

Projection Matrix: $S = V(V^TV)^{-1}V^T$

(1) Given $\begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = Vp = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} = y,$

then $S = \begin{bmatrix} \frac{5}{6} & \frac{1}{3} & -\frac{1}{6} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{6} & \frac{1}{3} & \frac{5}{6} \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 5 & 2 & -1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{bmatrix}$, and $Sy = \frac{1}{6}\begin{bmatrix} 5 & 2 & -1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{bmatrix}\begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} = \begin{bmatrix} 4\frac{2}{3} \\ 6\frac{2}{3} \\ 8\frac{2}{3} \end{bmatrix} \approx y$

(2) Given the same V, but $Vp = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} = y,$

then S is the same as: $\frac{1}{6}\begin{bmatrix} 5 & 2 & -1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{bmatrix}$, and $Sy = \frac{1}{6}\begin{bmatrix} 5 & 2 & -1 \\ 2 & 2 & 2 \\ -1 & 2 & 5 \end{bmatrix}\begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} = y$

Structure Returned by polyfit ( )

```
[p,S] = polyfit(x,y,n) also returns a structure S
that can be used to obtain error estimates.

S is a structure containing three elements:
(1) The triangular factor from a QR decomposition
    of the Vandermonde matrix,
(2) The degrees of freedom and,
(3) The norm of the residuals.

    S.R = R;
    S.df = max(0,length(y) - (n+1));
    r = y - V*p;
    S.normr = norm(r);
```

QR Decomposition

```
% Construct the Vandermonde matrix V = [x.^n ... x.^2 x ones(size(x))]
V(:,n+1) = ones(length(x),1,class(x));
for j = n:-1:1
   V(:,j) = x.*V(:,j+1);
end

% Solve least squares problem p = V\y to get polynomial coefficients p.
[Q,R] = qr(V, 0);       % Economy-size QR Decomposition
% Same as p = V\y
p = matlab.internal.math.nowarn.mldivide(R, Q'*y);
```

$V \times p = y$, where V is the Vandermonde matrix: m by (n+1), p is the output weight vector: (n+1) by 1, and y is the target vector: m by 1.

QR decomposition (Economy-size instead of full-size): $V = Q \times R$, where Q: m by (n+1) with orthonormal columns, i.e., $Q^{\mathrm{T}} \times Q = I$, and R: (n+1) by (n+1) upper triangular matrix.

$Q \times R \times p = y \rightarrow Q^{\mathrm{T}} \times Q \times R \times p = Q^{\mathrm{T}} \times y \rightarrow R \times p = (Q^{\mathrm{T}} \times y)$, which represents a system of linear equations of unknown *p.* The equations can be solved by using mldivide(R, Q'*y).

Numerical Stability

- The least square solution to a generally inconsistent system $Vp = y$ of m equations in n unknowns satisfies the *normal equation*: $V^T V p = V^T y$.
- 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

Condition Number

- The condition number of matrix $V_{m\times n}$ is given by $\kappa(V) = \|V\|\|V^+\|$, where $\|V\|$ is the 2-norm of the matrix V, and V^+ is the pseudo inverse of V.
- The 2-norm of a matrix is V is the largest singular value of V (i.e., the square root of the largest eigenvalue of the matrix V^TV), as given by $\|V\| = \sqrt{\lambda\_\max(V^TV)} = \sigma\_\max(V)$.
- The relative sensitivity of the solution p of $Vp = y$ to the perturbation of the input $\|\Delta p\|$ satisfies $\frac{\|\Delta p\|}{\|p\|} \leq \kappa \frac{\|\Delta y\|}{\|y\|}$.
- $\kappa \geq 1$. The larger the condition number, the worse.

Example

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

```
>> Eig = eig(V'*V)
ans =
   0.1886
  31.8114
>> sqrt(Eig(2))
ans =
   5.6402
```

```
>> Svd = svd(V)
ans =
   5.6402
   0.4343
```

```
>> norm(V)
ans =
   5.6402
```

```
>> Svd(1)*Svd(2)
ans =
   2.4495
```

```
>> sqrt(Eig(1)*Eig(2))
ans =
   2.4495
```

$$\|V\| = \sqrt{\lambda\_\max(V^T V)} = \sigma\_\max(V)$$

```
>> norm(pinv(V))
ans =
   2.3026
```

```
>> norm(V)*norm(pinv(V))
ans =
  12.9869
```

>> **cond(V)**
ans =
  12.9869

- 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).

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

>> cond(V)
ans =
   12.9869

>> cond(V)^2
ans =
  168.6607
```

```
>> VTV = V'*V
VTV =
    29     9
     9     3

>> cond(VTV)
ans =
  168.6607
```

QR Decomposition

```
>> [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
```

```
>> Q'*Q
ans =
    1.0000   -0.0000
   -0.0000    1.0000
```

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

```
>> Q*R
ans =
    2.0000    1.0000
    3.0000    1.0000
    4.0000    1.0000
```

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

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

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

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

```
>> Q'*y
ans =
  -12.4416
   0.4549
```

```
>> mldivide(R, Q'*y)      % Avoid inversion of large matrix
ans = 2.0000
      1.0000
```

SVD

- 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 $\kappa(V)$.
- 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.

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

>> S
S =
   5.6402         0
        0    0.4343
>> S^(-1)
ans =
   0.1773         0
        0    2.3026
```

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

>> A
A =
  -0.3913    0.8247
  -0.5606    0.1382
  -0.7298   -0.5484

>> B
B =
  -0.9545   -0.2982
  -0.2982    0.9545
```

```
>> A'*A
ans =
    1.0000   -0.0000
   -0.0000    1.0000

>> B*B'
ans =
    1.0000   -0.0000
   -0.0000    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 $\quad p = (B \times S^{-1} \times A^T) \times y$

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

```
>> p = B*S^(-1)*A'*y
p =
    2.0000
    1.0000
```

```
>> y =[5; 6; 9]
y =
    5
    6
    9
```

```
>> p = B*S^(-1)*A'*y
p =
    2.0000
    0.6667
```

‘curve_fit_demo.m’

```
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

W = Z / Fx
W2 =  Z * pinv(Fx)


[p,s] = polyfit(x, Z, 2);   % notice the reversed order
wrev(p)   % Show weights from low to high orders
```

```
% Now with noise added
rng(1);
Z = 1 + 2*x + 3*x.^2 + randn(1, N);

% The Vondermonde matrix
V = fliplr(Fx');
[Q,R] = qr(V,0);
Q'*Q
p2 = mldivide(R, Q'*Z');
% Compared wit the structure
returned by polyfit( )
[p,s] = polyfit(x, Z, 2);
p
s.R
s.normr
normr2 = norm(W*Fx - Z)
normr3 = norm(V*p2 - Z')
```

Fitting Noisier Data

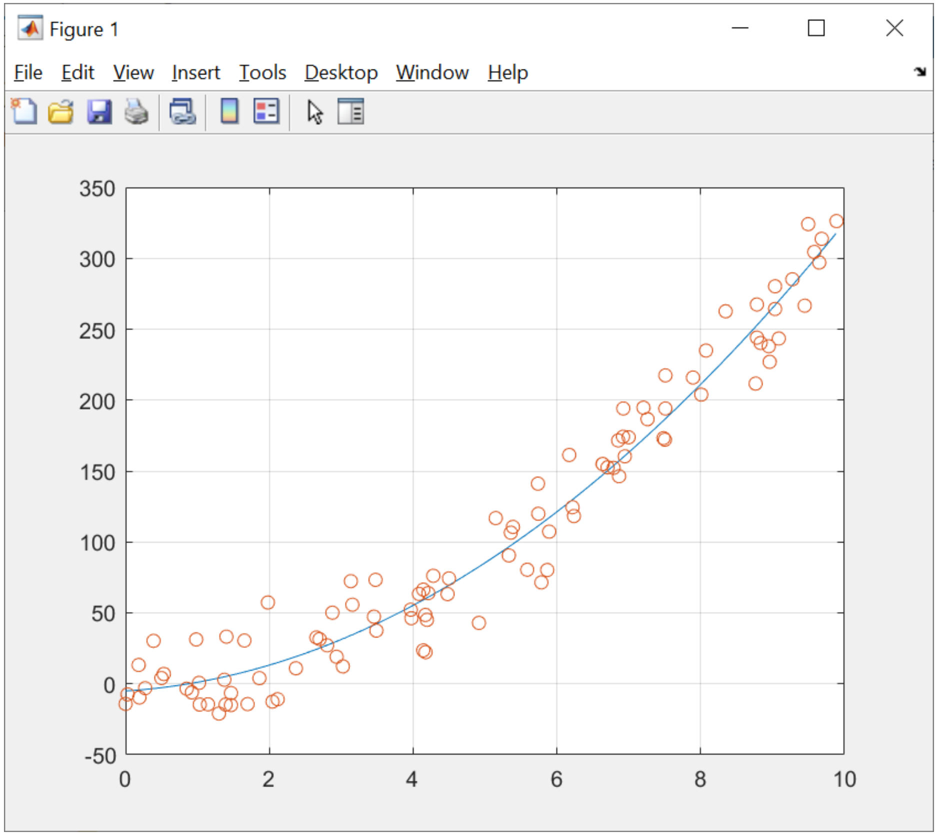
```
% With more training data with much worse noise added
N = 100;
rng(1);
x = 10*rand(1,N);
rng(1);
noise = 20*randn(1, N);
Z = 1 + 2*x + 3*x.^2 + noise;

Fx = zeros(3,N);
for i = 1:N
   Fx(:,i) = [1, x(i), x(i)^2];
end

scatter(x,Z); grid
W = Z / Fx

hold on;
xx = min(x):0.01:max(x);
plot(xx, W(1) + W(2)*xx + W(3)*xx.^2);

[p,s] = polyfit(x, Z, 2);
p
s.R
s.Normr
```




```
W =
  -5.0299   3.1358   2.9858
```

```
% Compared with the added noise norm
norm(noise)
```

Condition Numbers

```
>> V = fliplr(Fx');
>> whos V
  Name        Size            Bytes  Class     Attributes
  V         100x3              2400  double
>> cond(V)
ans =
  133.3990
>> cond(V'*V)
ans =
   1.7795e+04

>> s.R
ans =
 -438.8045  -55.0248   -7.3552
         0  -14.1341   -5.7421
         0         0   -3.5957
```

sklearn

```
ataset = np.loadtxt(infile, delimiter=',')
xdata = dataset[:, 0]
ydata = dataset[:, 1]

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(degree=2)
xdata = xdata[:, np.newaxis]

xdata_poly = poly.fit_transform(xdata)
reg = LinearRegression(fit_intercept=False).fit(xdata_poly, ydata)

reg.coef_
```

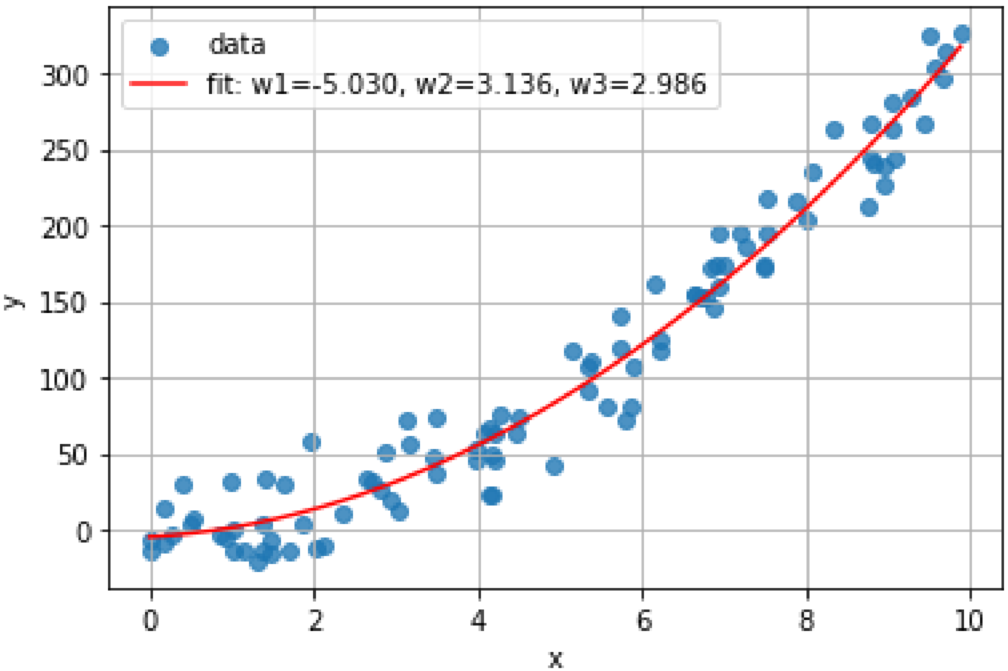
matplotlib

```
import matplotlib.pyplot as plt
plt.scatter(xdata, ydata, label='data', alpha = 0.8)

def func(x, w1, w2, w3):
   return w1 + w2*x + w3*x**2
xdata_clean = np.arange(np.min(xdata),
np.max(xdata), 0.01)

plt.plot(xdata_clean, func(xdata_clean,
*reg.coef_),
        'r', label='fit: w1=%5.3f, w2=%5.3f,
w3=%5.3f' % tuple(reg.coef_))

plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```



SVD

- Matlab

```
[A,S,B]= svd(Fx','econ');
>> p_svd = B*S^(-1)*A'*Z'
P_svd =
  -5.0299
   3.1358
   2.9858
```

```
>> S
S =
  442.3060         0         0
         0   15.2064         0
         0         0    3.3157
```

- Sklearn

```
reg.coef_
array([-5.02994715,  3.13580213,  2.98578577])

reg.singular_
array([442.30603581,  15.20639803,   3.31566156])
```