Case 1: $m = n$

- For example,

$$\begin{cases} 3x + 2y - z = 1 \\ x - y + 2z = -1 \\ -2x + y - 2z = 0 \end{cases}$$

```
>> A = [3 2 -1; 1 -1 2; -2 1 -2];
>> b = [1; -1; 0];
>> x = A \ b

 1
-2
-2
```

Case 2: $m > n$

- For example,

$$\begin{cases} 2x - y = 2 \\ x - 2y = -2 \\ x + y = 1 \end{cases}$$

```
1       >> A = [2 -1; 1 -2; 1 1];
2       >> b = [2; -2; 1];
3       >> x = A \ b
4
5       1
6       1
```

Case 3: $m < n$

- For example,

$$\begin{cases} x + 2y + 3z = 7 \\ 4x + 5y + 6z = 8 \end{cases}$$

```
>> A = [1 2 3; 4 5 6];
>> b = [7; 8];
>> x = A \ b

-3
0
3.333
```

- Note that this solution is a basic solution, one of infinitely many.
- How to find the directional vector? (Try **cross**.)

Gaussian Elimination Algorithm[1]

- First we consider the linear system is represented as an augmented matrix $[\,A\,|\,y\,]$.
- We then transform A into an upper triangular matrix

$$[\,\bar{A}\,|\,y\,] = \left[\begin{array}{cccc|c} 1 & \bar{a}_{12} & \cdots & \bar{a}_{1n} & \bar{y}_1 \\ 0 & 1 & \cdots & \bar{a}_{2n} & \bar{y}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \bar{y}_n \end{array}\right].$$

  where $\bar{a}_{ij}$'s and $\bar{y}_i$'s are the resulting values after elementary row operations.
- This matrix is said to be in reduced row echelon form.

[1] See `https://en.wikipedia.org/wiki/Gaussian_elimination`.

- The solution can be done by backward substitution:

$$x_i = \bar{y}_i - \sum_{j=i+1}^{n} \bar{a}_{ij} x_j,$$

  where $i = 1, 2, \cdots, n$.
- Time complexity: $O(n^3)$.

Exercise

```
1  clear; clc;
2
3  A = [3 2 -1; 1 -1 2; -2 1 -2];
4  b = [1; -1; 0];
5  A \ b % check the answer
6
7  for i = 1 : 3
8      for j = i : 3
9          b(j) = b(j) / A(j, i); % why first?
10         A(j, :) = A(j, :) / A(j, i);
11     end
12     for j = i + 1 : 3
13         A(j, :) = A(j, :) - A(i, :);
14         b(j) = b(j) - b(i);
15     end
16 end
17 x = zeros(3, 1);
```

```
18  for i = 3 : -1 : 1
19      x(i) = b(i);
20      for j = i + 1 : 1 : 3
21          x(i) = x(i) - A(i, j) * x(j);
22      end
23  end
24  x
```

Selected Functions of Linear Algebra[2]

- Matrix properties: **norm**, **null**, **orth**, **rank**, **rref**, **trace**, **subspace**.
- Matrix factorizations: **lu**, **chol**, **qr**.

[2] See `https://www.mathworks.com/help/matlab/linear-algebra.html`.

Numerical Example: 2D Laplace's Equation

- A partial differential equation (PDE) is a differential equation that contains unknown multivariable functions and their partial derivatives.[3]
- Let $\Phi(x, y)$ be a scalar field on $\mathbb{R}^2$.
- Consider Laplace's equation[4] as follows:

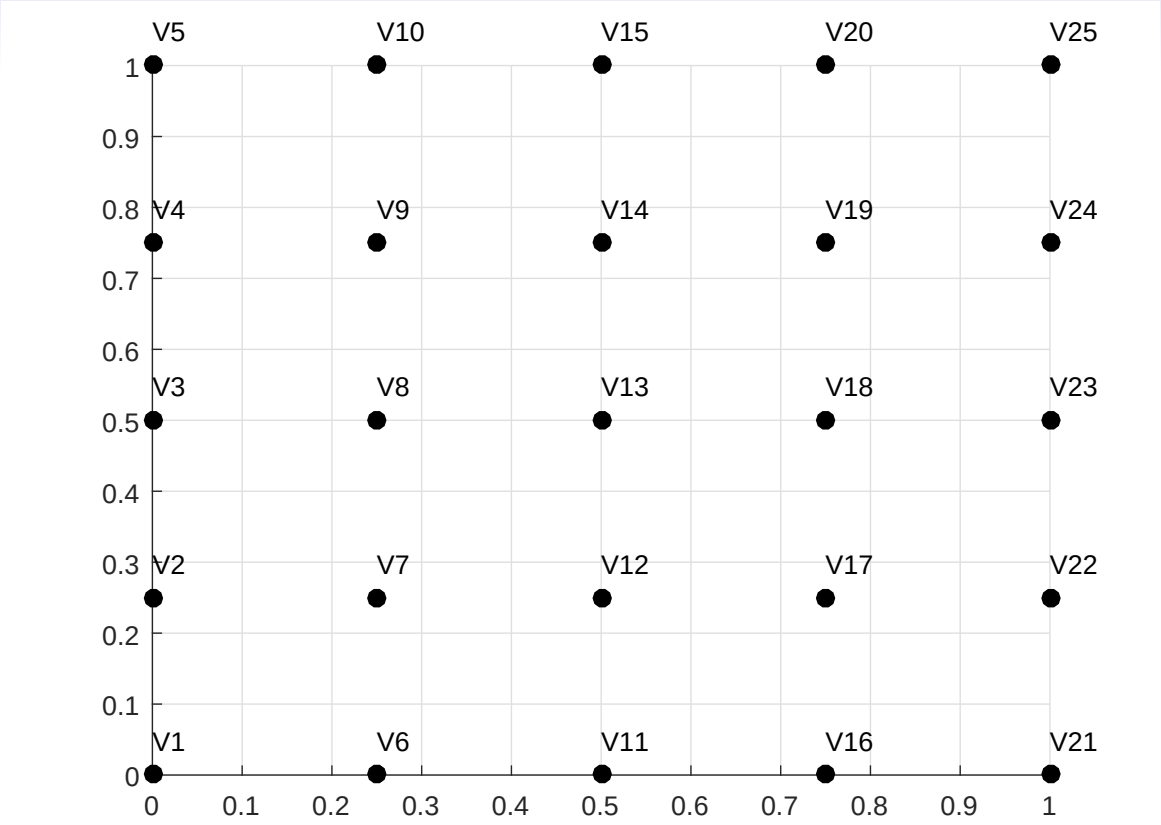
$$\nabla^2 \Phi(x, y) = 0,$$

  where $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ is the Laplace operator.
- Consider the system shown in the next page.

---

[3] See `https://en.wikipedia.org/wiki/Partial_differential_equation`.

[4] Pierre-Simon Laplace (1749–1827).



- Consider the boundary condition:
  - $V_1 = V_2 = \cdots = V_4 = 0.$
  - $V_{21} = V_{22} = \cdots = V_{24} = 0.$
  - $V_1 = V_6 = \cdots = V_{16} = 0.$
  - $V_5 = V_{10} = \cdots = V_{25} = 1.$

An Simple Approximation[5]

- As you can see, we partition the region into many subregions by applying a proper mesh generation.
- Then $\Phi(x, y)$ can be approximated by

$$\Phi(x, y) \approx \frac{\Phi(x+h, y) + \Phi(x-h, y) + \Phi(x, y+h) + \Phi(x, y-h)}{4},$$

where h is small enough.

[5] See https://en.wikipedia.org/wiki/Finite_difference_method#Example:_The_Laplace_operator.

Matrix Formation

- By collecting all constraints, we have $Ax = b$ where

$$A = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{bmatrix}$$

and

$$b = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T.$$

Dimension Reduction by Symmetry

- As you can see, $V_7 = V_{17}, V_8 = V_{18}$ and $V_9 = V_{19}$.
- So we can reduce A to A'

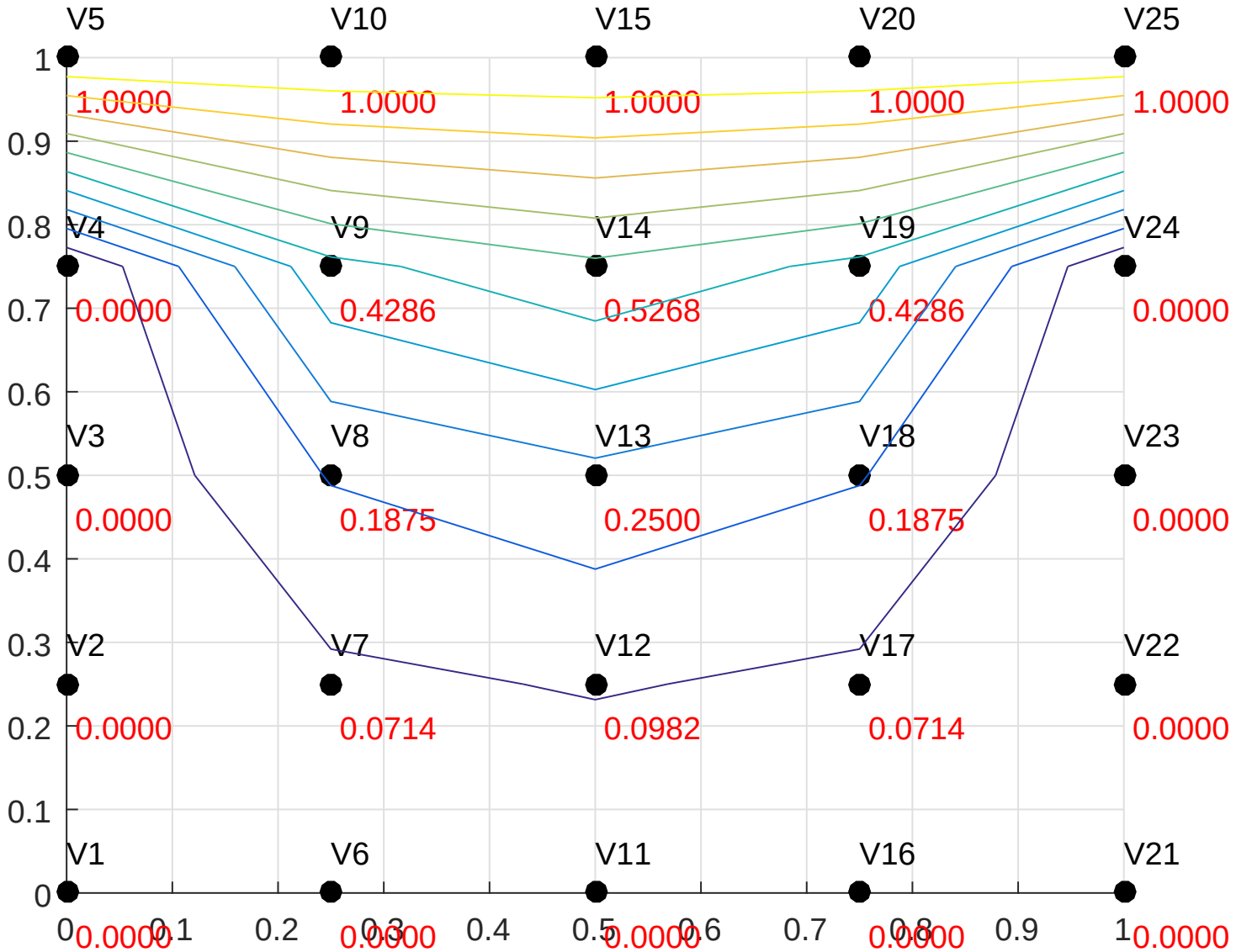
$$A' = \begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 \\ -2 & 0 & 0 & 4 & -1 & 0 \\ 0 & -2 & 0 & -1 & 4 & -1 \\ 0 & 0 & -2 & 0 & -1 & 4 \end{bmatrix}$$

and

$$b' = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}^T.$$

- The dimensions of this problem are cut to 6 from 9.
- This trick helps to alleviate the curse of dimensionality.[6]

[6] See `https://en.wikipedia.org/wiki/Curse_of_dimensionality`.


V5
V10
V15
V20
V25
1.0000
1.0000
1.0000
1.0000
1.0000
V4
V9
V14
V19
V24
0.0000
0.4286
0.5268
0.4286
0.0000
V3
V8
V13
V18
V23
0.0000
0.1875
0.2500
0.1875
0.0000
V2
V7
V12
V17
V22
0.0000
0.0714
0.0982
0.0714
0.0000
V1
V6
V11
V16
V21
0.0000
0.0000
0.0000
0.0000
0.0000
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Remarks

- This is a toy example for numerical methods of PDEs.
- We can use the PDE toolbox for this case. (Try.)
  - You may consider the finite element method (FEM).[7]
  - The mesh generation is also crucial for numerical methods.[8]
  - You can use the Computational Geometry toolbox for triangular mesh.[9]

[7] See https://en.wikipedia.org/wiki/Finite_element_method.

[8] See https://en.wikipedia.org/wiki/Mesh_generation.

[9] See https://www.mathworks.com/help/matlab/computational-geometry.html.

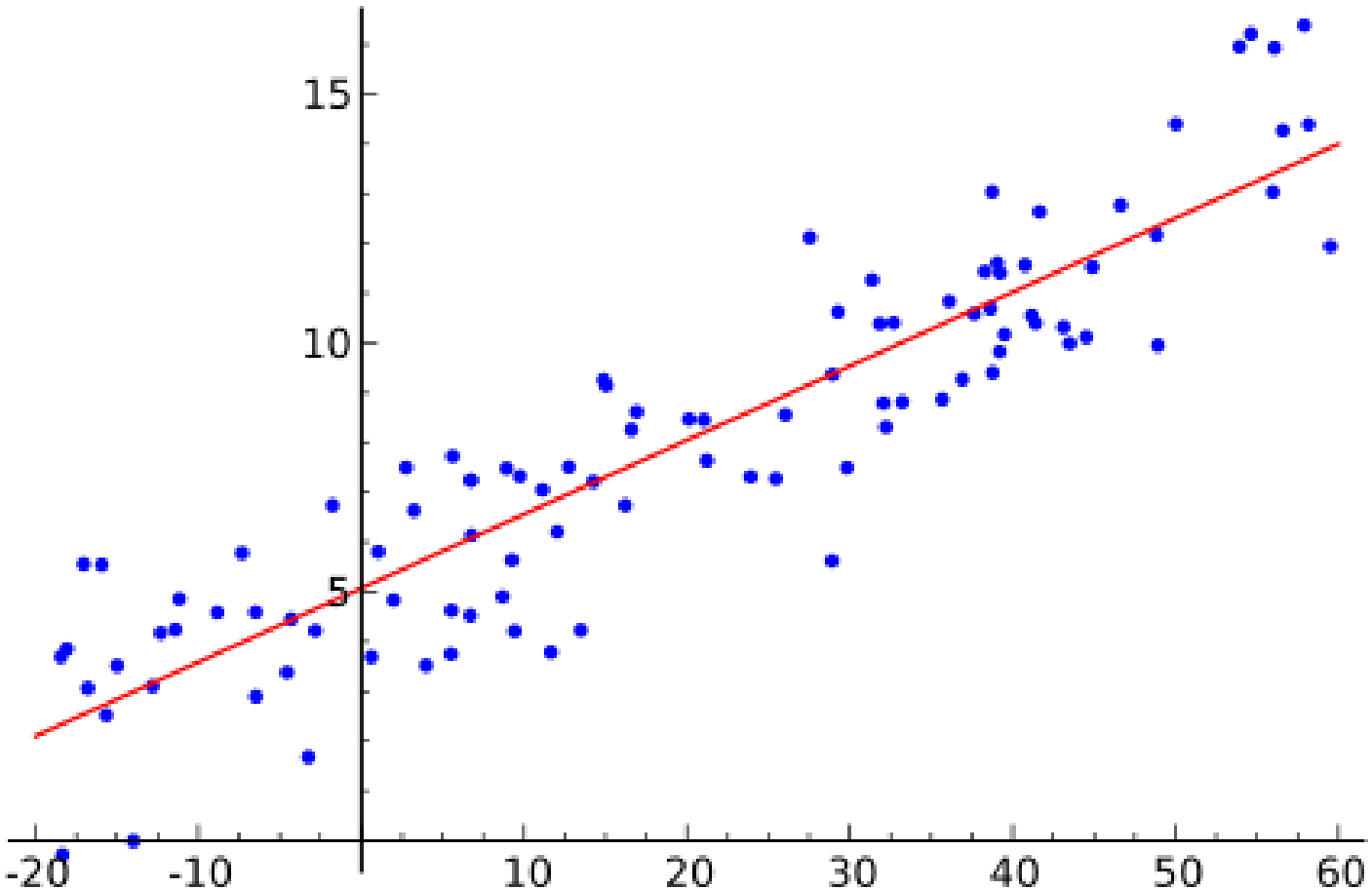
Numerical Example: Method of Least Squares[10]

- The method of least squares is a standard approach to the approximate solution of overdetermined systems ($m > n$).
- Let $\{\hat{y}_i\}_{i=1}^n$ be the observed response values and $\{y_i\}_{i=1}^n$ be the fitted response values.
- Let $\varepsilon_i = \hat{y}_i - y_i$ be the residual for $i = 1, \ldots, n$.
- Then the sum of square residuals estimates associated with the data is given by

$$S = \sum_{i=1}^{n} \epsilon_i^2.$$

- The best fit in the least-squares sense minimizes the sum of squared residuals.

[10] See `https://en.wikipedia.org/wiki/Least_squares`.



https://commons.wikimedia.org/wiki/File:Linear_regression.svg

Linear Least Squares

- The approach is called linear least squares since the assumed function is linear in the parameters to be estimated.
- For example, consider

$$y = ax + b,$$

where a and b are to be determined.

- Then we have $\epsilon_i = (ax_i + b) - \hat{y}_i$ so that

$$S = \sum_{i=1}^{n} ((ax_i + b) - \hat{y}_i)^2 .$$

- Now consider the partial derivatives of S with respective to a and b:

$$\frac{\partial S}{\partial a} = -2\sum_{i=1}^{n} x_i(y_i - (ax_i + b)) = 0,$$

$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{n} (y_i - (ax_i + b)) = 0.$$

- We reorganize the above equations as follows:

$$a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i,$$

$$a\sum_{i=1}^{n} x_i + nb = \sum_{i=1}^{n} y_i.$$

- It could be done by using normal equations.[11]

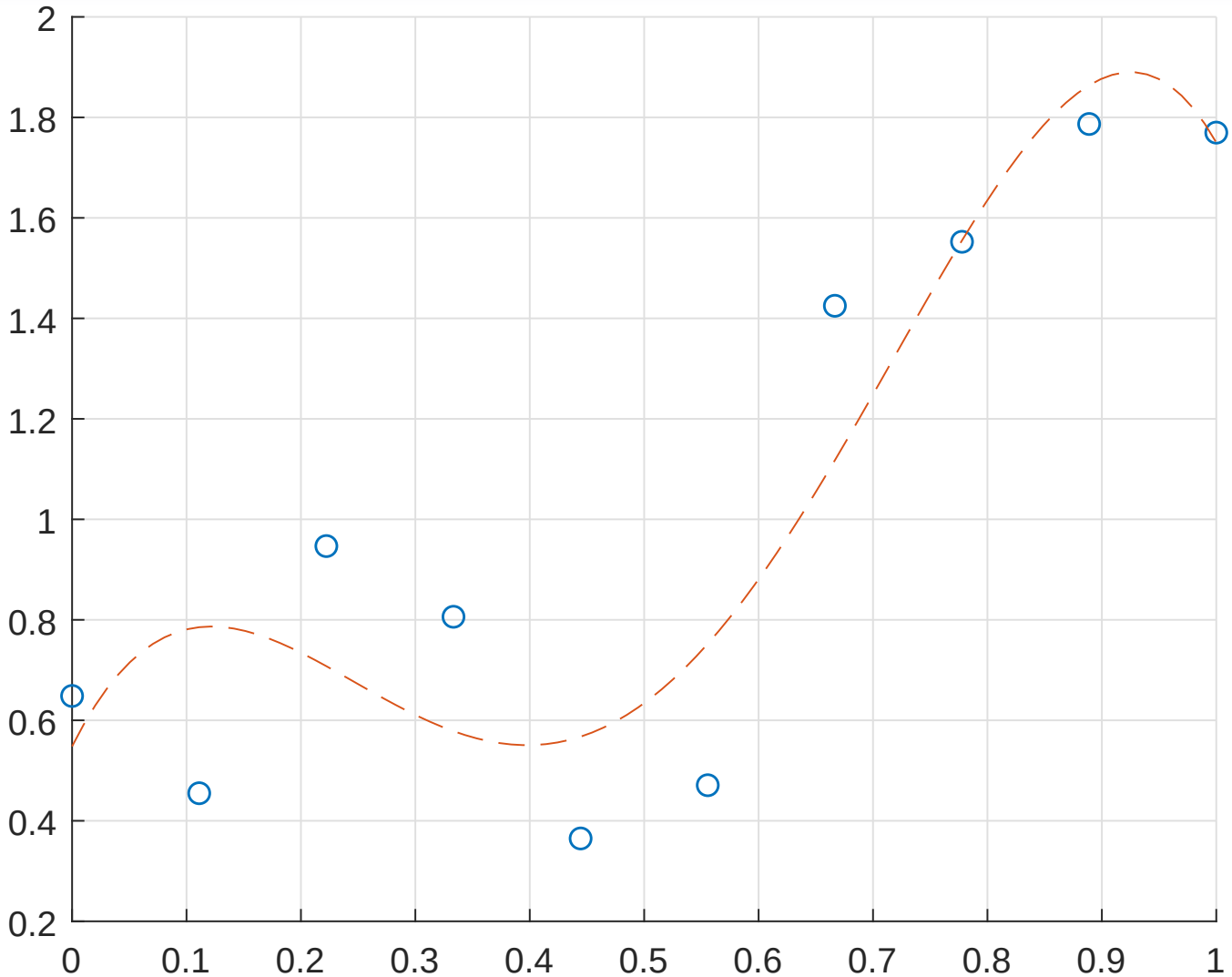
[11] See https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)#Derivation_of_the_normal_equations.

Example

```
clear; clc; close all;

rng(3); % fix the random seed
N = 10;
x = linspace(0, 1, N); x = x(:);
y = cos(rand(size(x)) * pi / 2) + x .^ 2;
figure; hold on; grid on; plot(x, y, 'o');
degree = 4;

M = @(x, degree) repmat(x, 1, degree + 1);
A = @(mat) bsxfun(@(x, i) x .^ i, mat, ...
                  size(mat, 2) - 1 : -1 : 0);
pp = A(M(x, degree)) \ y % show the coefficients
xq = linspace(min(x), max(x), 100); xq = xq(:);
yq = A(M(xq, degree)) * pp;
plot(xq, yq, '--');
```


2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Polynomial Regression

- **polyfit**(x, y, n) returns the coefficients for a polynomial of degree n that is a best fit for the set of sample data (x, y) (in a least-squares sense).

Example

```
clear; clc; close all;

rng(3);
N = 10;
x = linspace(0, 1, N); x = x(:);
y = cos(rand(size(x)) * pi / 2) + x .^ 2;
figure; hold on; grid on; plot(x, y, 'o');
degree = 4;

p = polyfit(x, y, degree)
xq = linspace(0, 1, 50);
yq = polyval(p, xq);
plot(xq, yq);
```

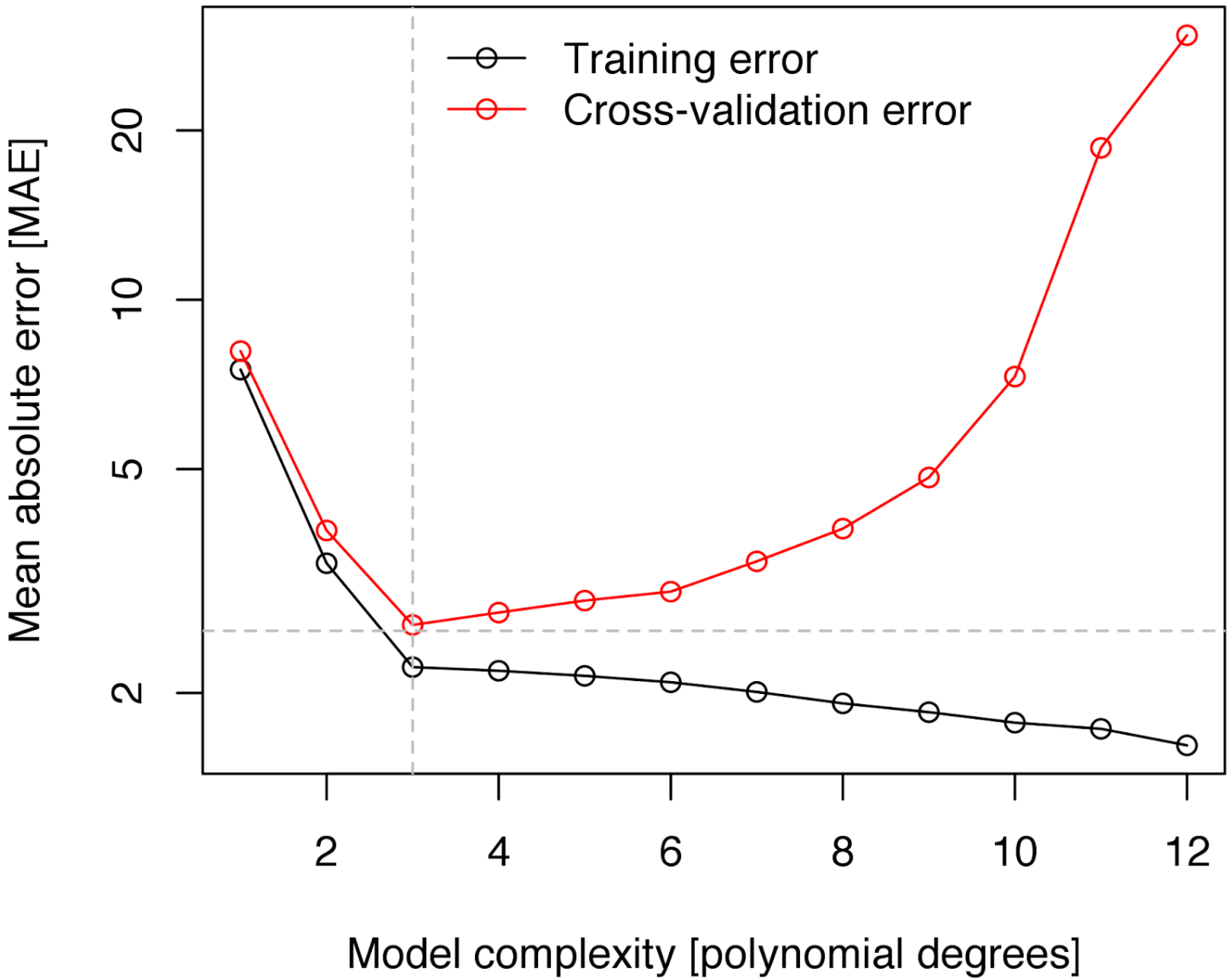
- The result is identical to the figure shown before.

Overfitting

- Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.
  - In other words, the overfitted model is perfect to in-sample data but not robust in out-of-sample data.
  - For example, Runge's phenomenon.[12]
- Law of parsimony[13] states that simpler solutions are more likely to be correct than complex ones.

[12] See `https://en.wikipedia.org/wiki/Runge's_phenomenon`.

[13] Aka Occam's Razor.


Training error
Cross-validation error
Mean absolute error [MAE]
2
5
10
20
2
4
6
8
10
12
Model complexity [polynomial degrees]

Polynomials

- Let $n \in \mathbb{N} \cup \{0\}$, and $x, a_0, \ldots, a_n \in \mathbb{R}$.
- $f(x)$ is said to be a polynomial with degree n provided that

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0,$$

  where $a_n \neq 0$.
- We often express a polynomial by its coefficient vector $[a_n, a_{n-1}, \ldots, a_0]$.
- In fact, the set of polynomials with coefficients in $\mathbb{R}$ is a vector space over $\mathbb{R}$, denoted by $\mathbb{P}_n$.[14]

[14] See `https://en.wikipedia.org/wiki/Examples_of_vector_spaces#Polynomial_vector_spaces`.

Arithmetic Operations of Polynomials

- Let p_1 and p_2 be two coefficient vectors of polynomials of the same degree.
- Then we have the following operations:
  - addition and subtraction: $p_1 \pm p_2$.
  - multiplication: $\mathbf{conv}(p_1, p_2)$.[15]
  - division: $[\, q, r \,] = \mathbf{deconv}(p_1, p_2)$.[16]
- Use $\mathbf{polyval}(p_1, x)$ to calculate the function values of $p1$ on x.

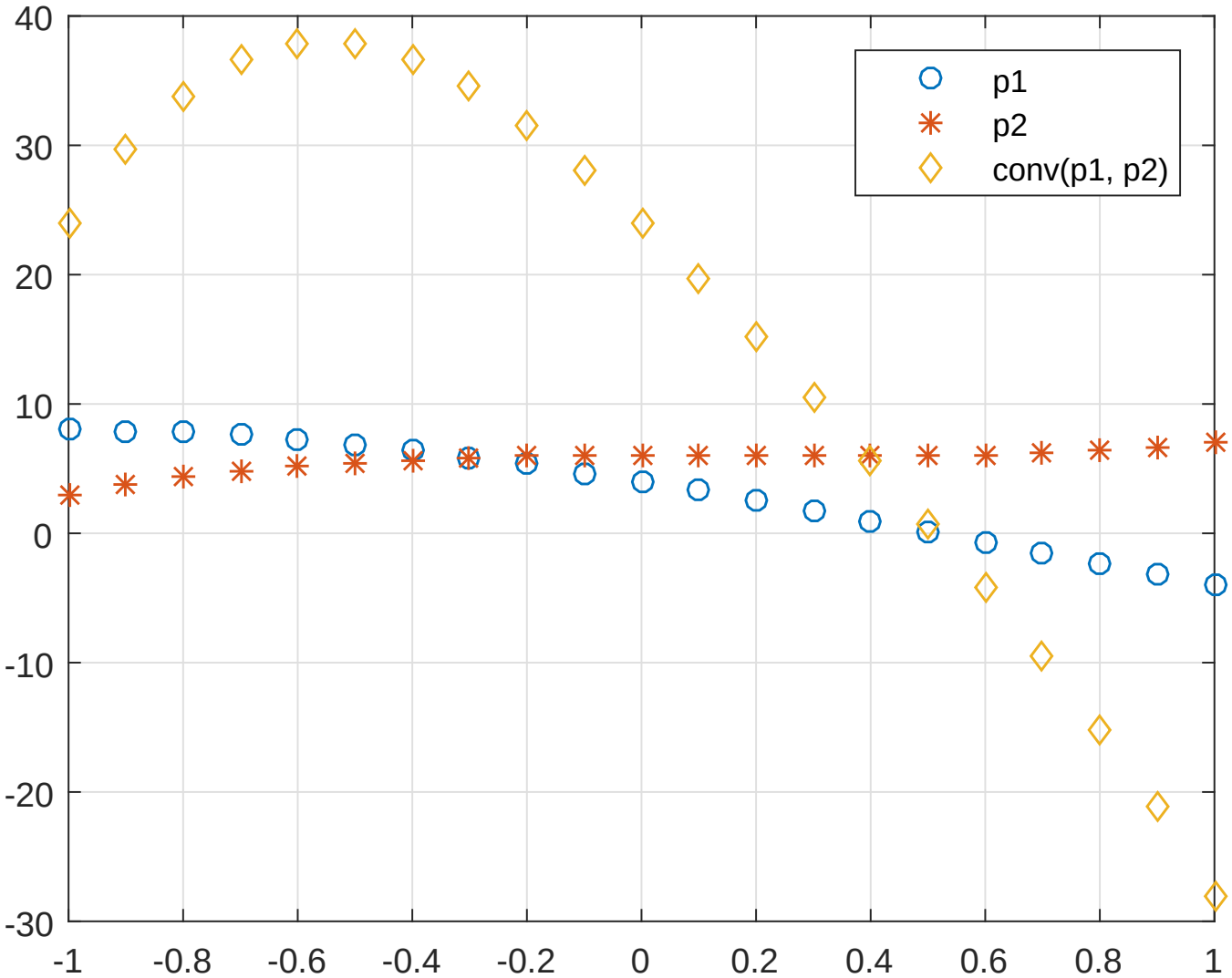
[15] See `http://en.wikipedia.org/wiki/Convolution`.

[16] Equivalently, $v = \text{conv}(u, q) + r$. Also see `http://en.wikipedia.org/wiki/Euclidean_division`.

```
clear; clc;

p1 = [1 -2 -7 4];
p2 = [2 -1 0 6];
p3 = p1 + p2
p4 = p1 - p2
p5 = conv(p1, p2)
[q, r] = deconv(p1, p2)

x = linspace(-1, 1, 20);
plot(x, polyval(p1, x), 'o', x, polyval(p2, x), ...
    '*',  x, polyval(p5, x), 'd');
grid on; legend('p1', 'p2', 'conv(p1, p2)');
```


40
30
20
10
0
-10
-20
-30
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
p1
p2
conv(p1, p2)

Finding Roots of Polynomial

- Use **roots**(p) for all roots of the polynomial p.[17]
- For example,

```
1 clear; clc; close all;
2
3 p = [1, 3, 1, 5, -1];
4 r = roots(p) % find all roots of p
5 polyval(p, r) % why not zeros?
```

[17] See https://en.wikipedia.org/wiki/Jenkins-Traub_algorithm.

Exercise: Internal Rate of Return (IRR)[18]

- Consider two assets.
- For asset A, you are promised to receive the cash flows as follows:

$$C_0 = -100, C_1 = 0, C_2 = 0, C_3 = 120.$$

- For asset B, the cash flows are

$$C_0 = -100, C_1 = 6, C_2 = 6, C_3 = 108.$$

- Which asset is more desirable?

[18] See `https://en.wikipedia.org/wiki/Internal_rate_of_return`.

- Given a collection of pairs (time, cash flow), the IRR is a rate of return when the net present value is zero.
- Explicitly, the IRR can be calculated by solving

$$\sum_{i=0}^{N} \frac{C_i}{(1+r)^i} = 0,$$

  where C_i is the cash flow at time i.
- So the IRR is 6.27% for A and 6.62% for B.

"Time is free, but its priceless.
You can't own it, but you can use it.
You can't keep it, but you can spend it.
Once you've lost it, you can never get it back."
– Harvey MacKay

"No man can achieve success
if he didn't first know the value of time."
– Sunday Adelaja

Integral and Derivative of Polynomials

- **polyder**(p) returns the derivative of the polynomial p.
- **polyint**(p, k) returns a polynomial representing the integral of polynomial p, using a scalar constant of integration k.

```
1 clear; clc;
2
3 p = [4 3 2 1];
4 p_der = polyder(p)
5 p_int = polyint(p, 0) % k = 0
```

Exercise

- Let p be the coefficient vector for any polynomial with degree 3.
- Write a program to calculate the coefficients of its derivative and integration.
- Also, you may write down the matrix representation of differentiation and integration of p.
- Do not use the built-in functions.

```
clear; clc;

p = randi(100, 1, 4)
q1 = [0, p(1 :  end - 1) .* [length(p) - 1 : -1 : 1]]
q2 = [p ./ [length(p)  : -1 : 1], 0]

T1 = [0 0 0 0;
      3 0 0 0;
      0 2 0 0;
      0 0 1 0];
T1 * p'
T2 = [0 1/4   0   0   0;
      0   0 1/3   0   0;
      0   0   0 1/2   0;
      0   0   0   0   1;
      0   0   0   0   0];
T2 * [0 p]'
```

Eigenvalues and Eigenvectors

- Let $A \in \mathrm{M}_{n\times n}(\mathbb{R})$, I be the identity, and $v \in \mathbb{R}^n$ be nontrivial.
- An eigenvalue problem[19] is a system which follows

$$Av = \lambda v.$$

- Then u is an eigenvector associated with the eigenvalue λ by solving $\det(A - \lambda I) = 0$, aka the characteristic polynomial.
  - Use $[\, V, D \,] = \mathbf{eig}(A)$ produces a diagonal matrix D of eigenvalues and a full matrix V whose columns are the corresponding eigenvectors so that $AV = VD$.

[19] See `https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors#Applications`.

Example: Google PageRank Algorithm[20]

- PageRank (PR) is an algorithm used by Google Search to rank web pages in their search engine results.
- PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is.
- The underlying assumption is that more important websites are likely to receive more links from other websites.

[20] Larry Page and Sergey Brin (1998).

Singular Value Decomposition (SVD)

- Let $A \in \mathrm{M}_{m\times n}(\mathbb{R})$, $u \in \mathbb{R}^m$, $v \in \mathbb{R}^n$, and $\sigma \in \mathbb{R}$.
- σ is called a singular value associated with the left singular vector u and the right singular vector v for A provided that

$$A = \sum_{i=1}^{n} \sigma_i u_i v_i^T.$$

- In matrix form,

$$A = U\Sigma V^T,$$

  where U and V consist of the left and right singular vectors, respectively, and Σ is a diagonal matrix whose diagonal entries are the singular values of A.
- Use $[\,U, S, V\,] = \mathbf{svd}(A)$ for SVD.[21]

[21] See https://www.mathworks.com/help/matlab/math/singular-values.html.

Example: Image Compression by Low-Rank Approximation

- We can have an image extremely similar to the original one, but with a smaller image size by keeping the vectors associated with a few number of first large principal components, aka Principal Component Analysis (PCA).[22]
- PCA can be done by **svd**.

[22] See `http://setosa.io/ev/principal-component-analysis/`.