Note) Slides and example codes are available: https://github.com/mint-lab/prog_meets_math

Programming Meets Mathematics: Optimization

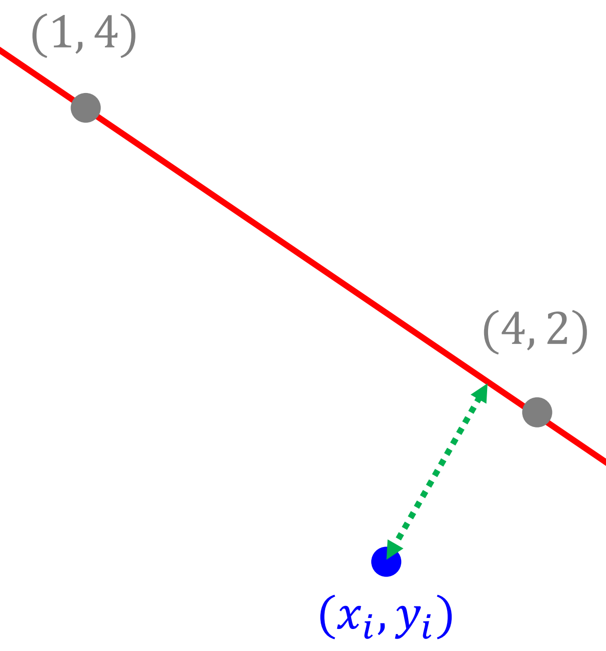
Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

Programming Meets Mathematics

- ~~Calculus~~ **Differentiation**
- ~~Linear Algebra~~ **Vector and Matrix**
  - **NumPy**: `numpy.array` vs. `list/tuple`
  - **Vector**: Why?
    - Note) Norm (~ the distance from the origin, magnitude)
    - Vector multiplication: Dot product, cross product
  - **Matrix**: Why?
    - Matrix multiplication
    - Matrix inverse (square + full rank), pseudo-inverse (full rank)
    - Examples) Line and curve fitting (using matrix operations)
- **Optimization**
- **Probability**
- **Information Theory**

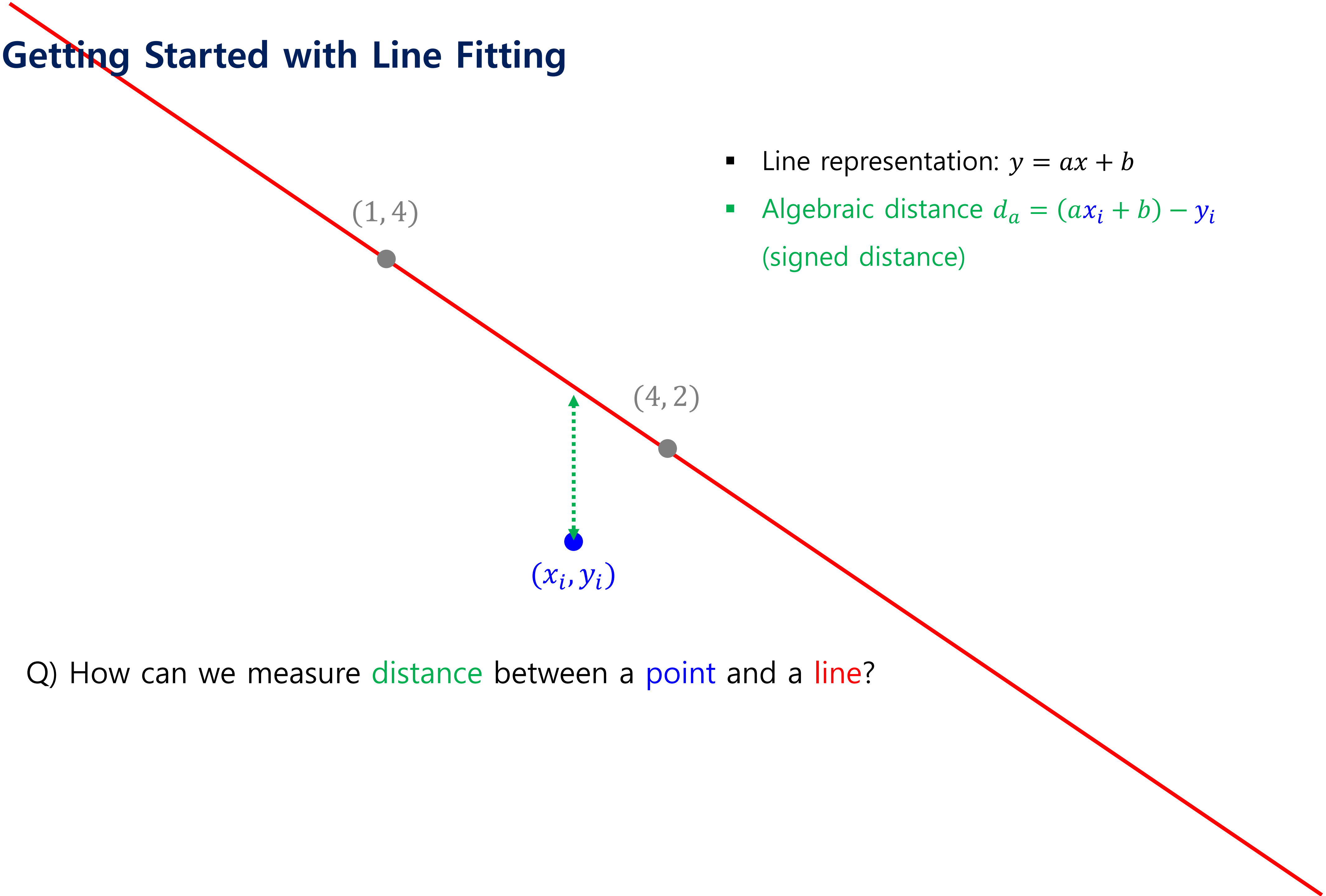
Getting Started with Line Fitting

- Line representation: $ax + by + c = 0$
- Geometric distance $d_g = \frac{ax_i + by_i + c}{\sqrt{a^2 + b^2}}$ (signed distance)



Q) How can we measure distance between a point and a line?

Getting Started with Line Fitting



- Line representation: $y = ax + b$
- Algebraic distance $d_a = (ax_i + b) - y_i$ (signed distance)

Q) How can we measure distance between a point and a line?

Optimization

- **Optimization** is the selection of the best element, with regard to some **criterion**, from a defined domain.
  - Alias: Mathematical programming
    - Linear programming, convex programming, nonlinear programming, …, dynamic programming
  - In the simplest case, optimization is maximizing or minimizing a **objective function**
    - Maximization: Objective functions → profit/utility/fitness/reward/… functions
    - Minimization: Objective functions → loss/cost/error/penalty/… functions
    - Note) Maximization and minimization are dual. → Minimization is usually preferred.
  - Example) Finding x and y for the maximum z with $z = 4 - (x^2 + y^2)$
    - Unknown variable: $\mathbf{x} = [x, y]$ and its domain $\mathbb{R}^2$
    - Objective function: $f(x, y) = 4 - (x^2 + y^2)$ as a maximization problem
    - In short, $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^2$ and $f(\mathbf{x}) = 4 - \|\mathbf{x}\|_2^2$
      - Note) L2-norm (Euclidean norm): $\|\mathbf{x}\|_2 = \sqrt{x^2 + y^2}$ for $\mathbf{x} \in \mathbb{R}^2$

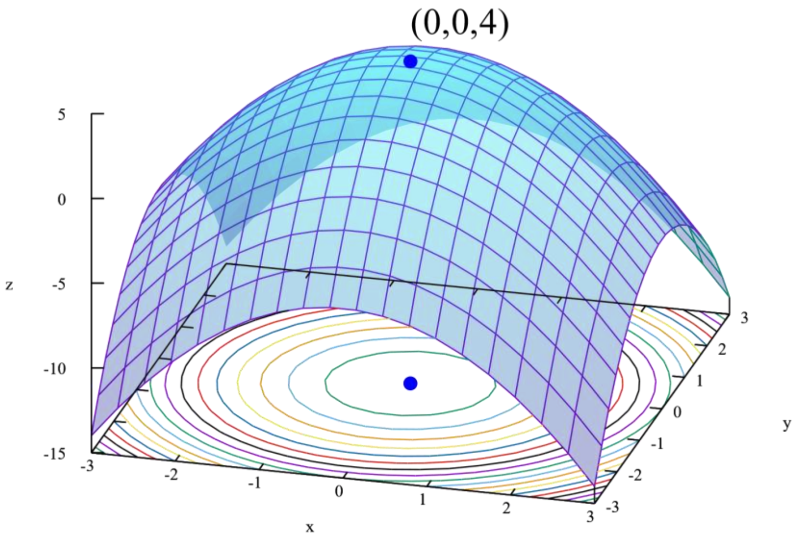


Image: Wikipedia

~~Optimization~~ Nonlinear Optimization

- **Nonlinear optimization** is the process of solving an optimization problem where some of the constraints or the objective function are **nonlinear**.
  - Alias: Nonlinear programming (NLP)
  - Mathematically, $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$ subject to
    $$\begin{aligned} & g_i(\mathbf{x}) \le 0 \ \text{ for each } \ i \in \{1, \dots, m\} \\ & h_j(\mathbf{x}) = 0 \ \text{ for each } \ j \in \{1, \dots, p\} \\ & \mathbf{x} \in \mathrm{X} \quad (\mathrm{X} \text{ is a subset of } \mathbb{R}^n) \end{aligned}$$
    - $f(\mathbf{x})$: The real-valued objective function
    - $g_i(\mathbf{x})$: The i-th real-valued inequality constraint function
    - $h_j(\mathbf{x})$: The j-th real-valued equality constraint function
  - Example) The objective function $f(x, y) = 4 - (x^2 + y^2)$ is nonlinear.

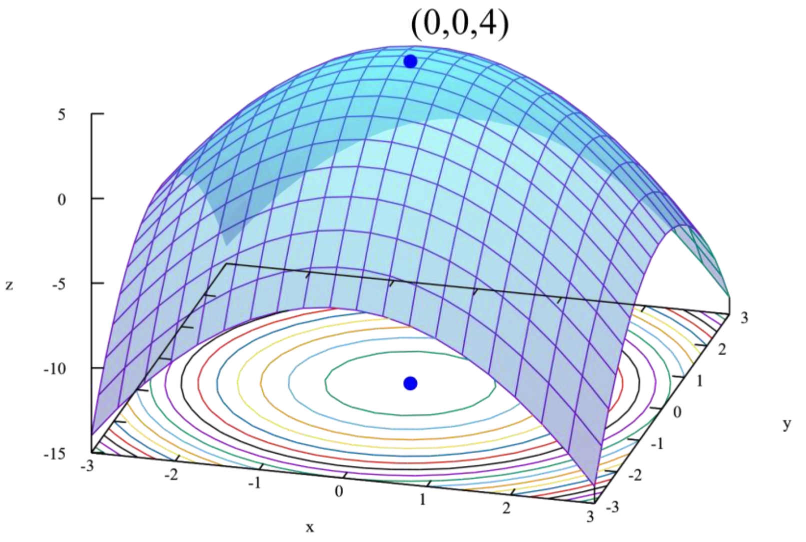
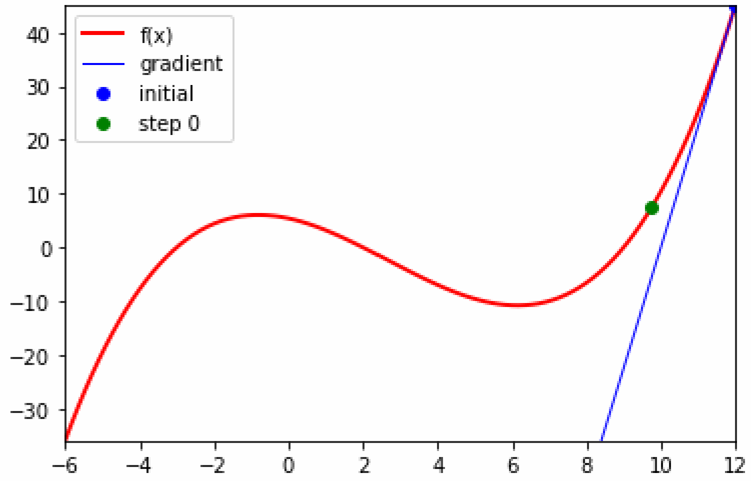


Image: Wikipedia

Nonlinear Optimization

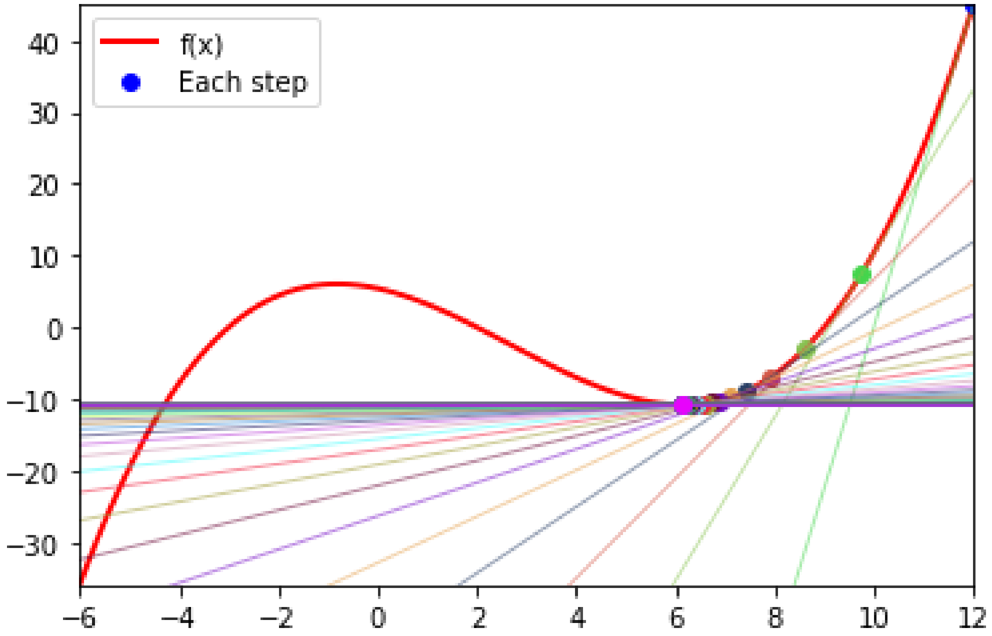
- **Gradient descent**
  - A **first-order iterative algorithm** for finding a local minimum of a differentiable function by pursuiting the opposite direction of the gradient of the function at the current point
  - Mathematically, $x_{t+1} = x_t - \gamma f'(x_t)$
    - γ: The step size (a.k.a. learning rate)



- Note) **Stochastic gradient descent (SGD)**
  - SGD uses an approximated gradient (calculated from a randomly selected subset of the given data) instead of the actual gradient (calculated from the entire data).
  - SGD variants: AdaGrad, RMSProp, Adam, …

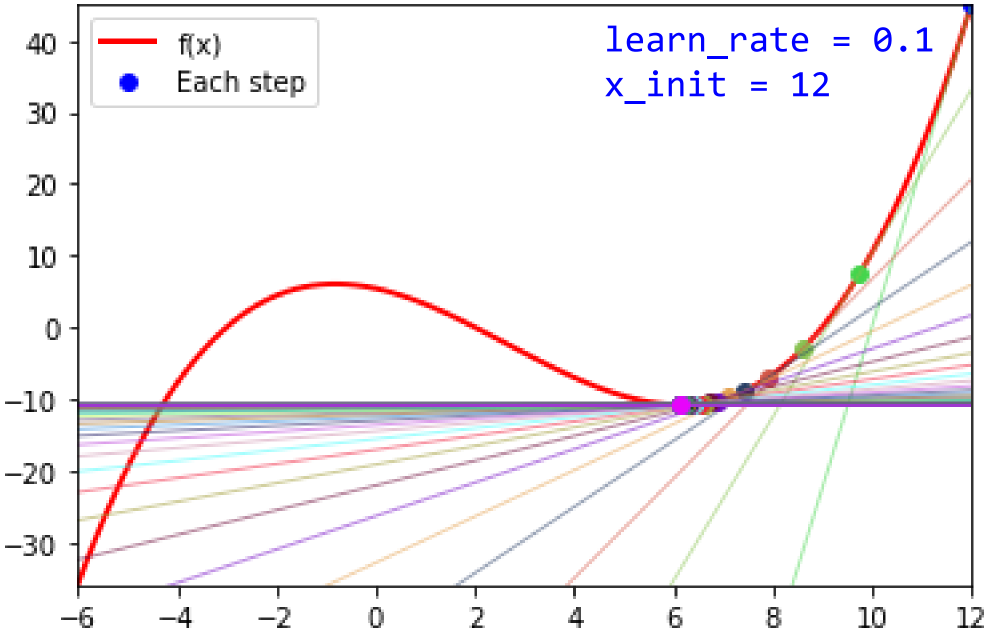
Nonlinear Optimization

- **Gradient descent**
  - Example) Find a local minimum $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$ from $x = 12$

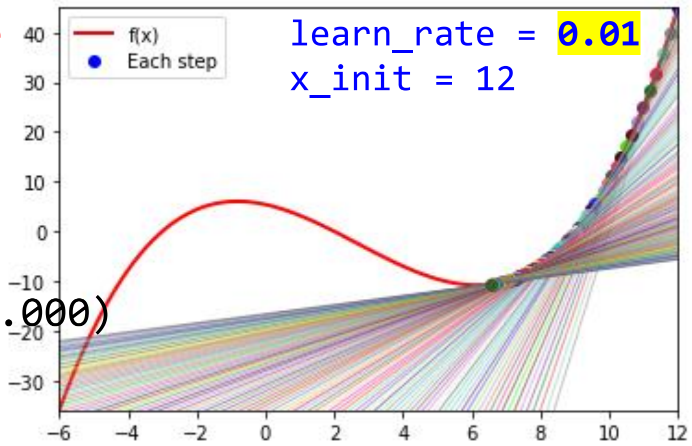


Nonlinear Optimization

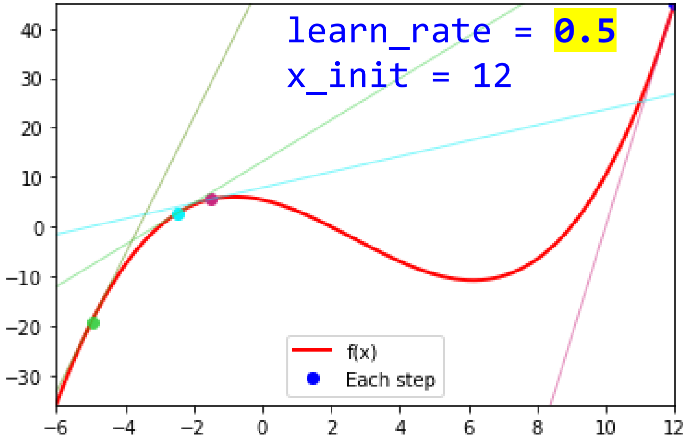
- **Gradient descent**
  - Example) Find a local minimum $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$ from $x = 12$
    - `Iter: 57, x = 6.147 to 6.147, f(x) = -10.822 to -10.822 (f'(x) = 0.000)`



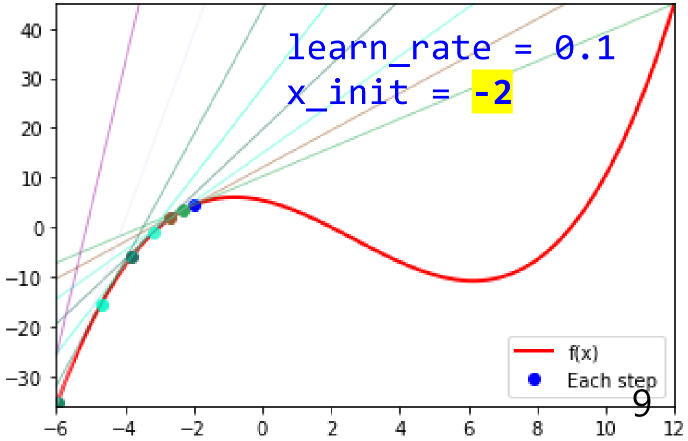
Too small a step size



Too large a step size



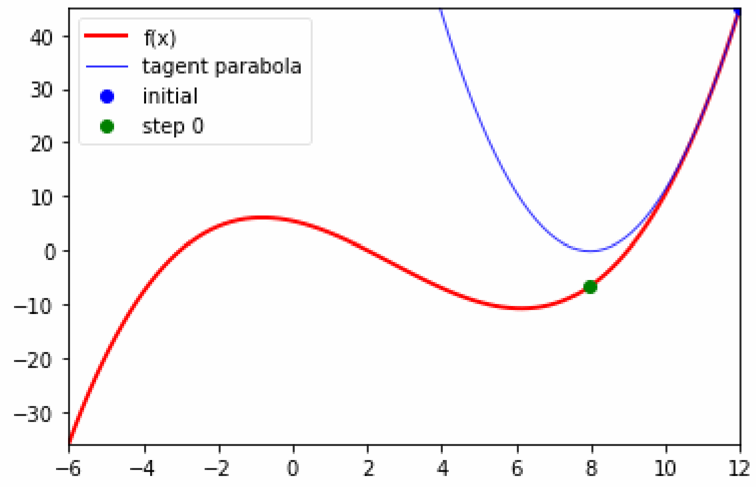
Negative infinity



```python
import numpy as np
import matplotlib.pyplot as plt

f  = lambda x: 0.1*x**3 - 0.8*x**2 - 1.5*x + 5.4
fd = lambda x: 0.3*x**2 - 1.6*x - 1.5
viz_range = np.array([-6, 12])
learn_rate =  0.1 # Try 0.001, 0.01, 0.5, and 0.6
max_iter = 100
min_tol = 1e-6
x_init = 12         # Try -2

# Prepare visualization
xs = np.linspace(*viz_range, 100)
plt.plot(xs, f(xs), 'r-', label='f(x)', linewidth=2)
plt.plot(x_init, f(x_init), 'b.', label='Each step', markersize=12)
plt.axis((*viz_range, *f(viz_range)))
plt.legend()

x = x_init
for i in range(max_iter):
    # Run the gradient descent
    xp = x
    x = x - learn_rate*fd(x)

    # Update visualization for each iteration
    print(f'Iter: {i}, x = {xp:.3f} to {x:.3f}, f(x) = {f(xp):.3f} to {f(x):.3f} (f\'(x) = {fd(xp):.3f})')
    lcolor = np.random.rand(3)
    approx = fd(xp)*(xs-xp) + f(xp)
    plt.plot(xs, approx, '-', linewidth=1, color=lcolor, alpha=0.5)
    plt.plot(x, f(x), '.', color=lcolor, markersize=12)

    # Check the terminal condition
    if abs(x - xp) < min_tol:
        break
plt.show()
```

Nonlinear Optimization

- **Newton's method**
  - A **second-order iterative algorithm** for finding a local minimum of a differentiable function by pursuing the minimum of the locally approximated parabola of the function at the current point
  - Mathematically, $x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$
    - The step size is **not** required.
    - Note) You can regard the step size is adaptively derived as $\gamma = \frac{1}{f''(x_t)}$.



- Note) **Gauss-Newton method**
  - A special case for nonlinear least squares problems
    - When the function has a form of $f(x) = r^2(x)$,
    - Newton's method becomes $x_{t+1} = x_t - \frac{r(x_t)}{r'(x_t)}$ (without the 2nd-order derivative)

Nonlinear Optimization

- **Newton's method**
  - Why does $x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}$ contain the 2nd-order derivative?
    - The tangent parabola: The 2nd-order Tayor series expansion at $(x_t, f(x_t))$

$$p(x) = \frac{1}{2} f''(x_t)(x - x_t)^2 + f'(x_t)(x - x_t) + f(x_t)$$

      - Note) The tangent line: The 1st-order Tayor series expansion at $(x_t, f(x_t))$

$$l(x) = f'(x_t)(x - x_t) + f(x_t)$$

    - Finding the extrema (root) x_{root} of the tangent parabola when $p'(x) = 0$

$$x_{\mathrm{root}} = x_t - \frac{f'(x_t)}{f''(x_t)}$$

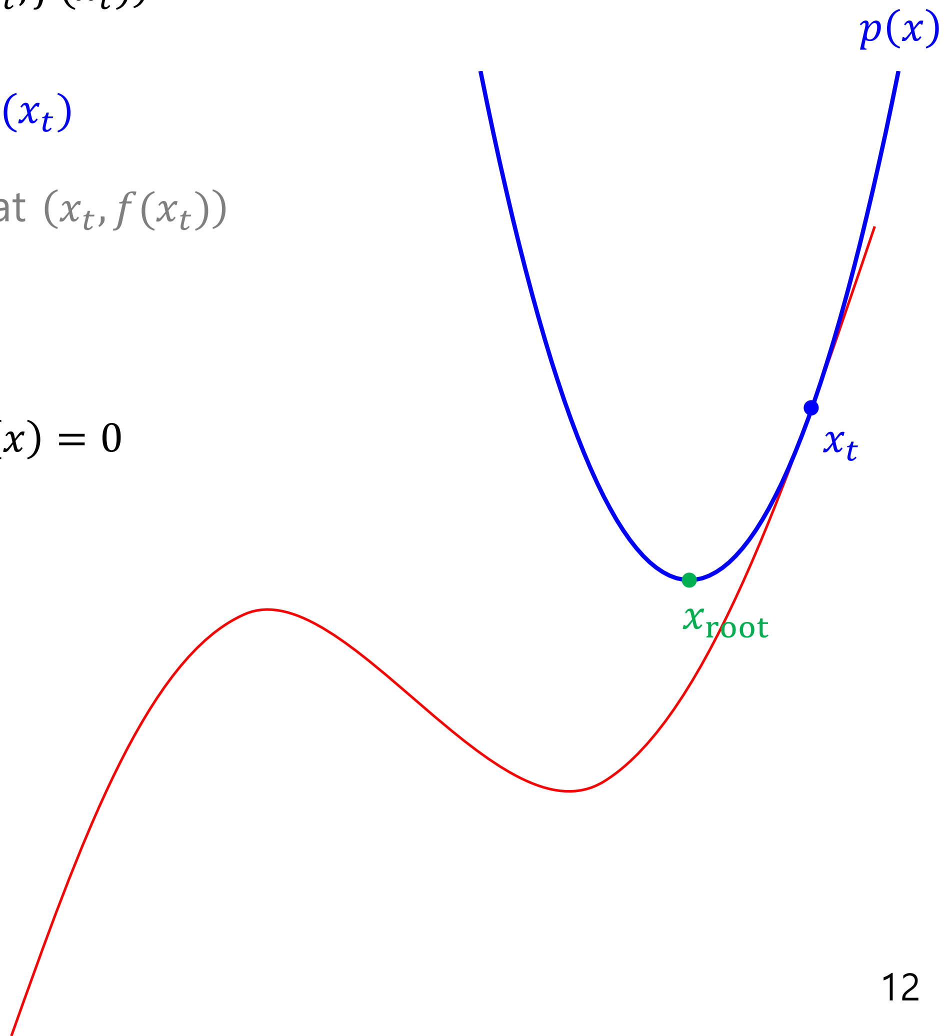
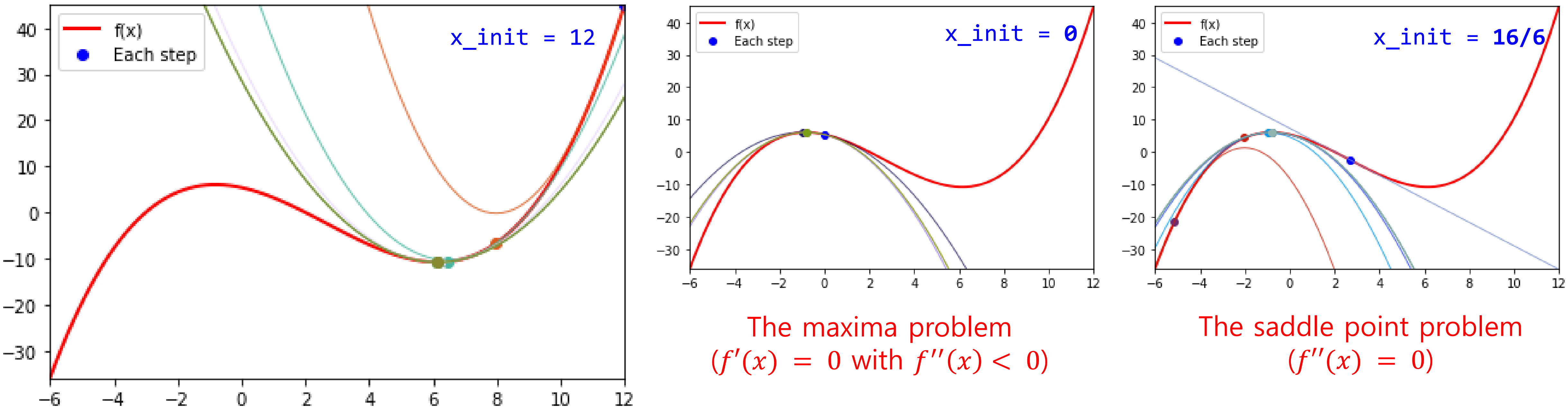


Image: Wikipedia

Nonlinear Optimization

- **Newton's method**
  - Example) Find a local minimum $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$ from $x = 12$
    - `Iter: 57, x = 6.147 to 6.147, f(x) = -10.822 to -10.822 (f'(x) = 0.000)        # GD`
    - `Iter:  5, x = 6.147 to 6.147, f(x) = -10.822 to -10.822 (..., f''(x) = 2.088) # Newton`



The maxima problem
($f'(x) = 0$ with $f''(x) < 0$)

The saddle point problem
($f''(x) = 0$)

```python
import numpy as np
import matplotlib.pyplot as plt

f   = lambda x: 0.1*x**3 - 0.8*x**2 - 1.5*x + 5.4
fd  = lambda x: 0.3*x**2 - 1.6*x - 1.5
fdd = lambda x: 0.6*x - 1.6
viz_range = np.array([-6, 12])
max_iter = 100
min_tol = 1e-6
x_init = 12 # Try -2, 0, and 16/6 (a saddle point)

# Prepare visualization
xs = np.linspace(*viz_range, 100)
plt.plot(xs, f(xs), 'r-', label='f(x)', linewidth=2)
plt.plot(x_init, f(x_init), 'b.', label='Each step', markersize=12)
plt.axis((*viz_range, *f(viz_range)))
plt.legend()

x = x_init
for i in range(max_iter):
    # Run the Newton method
    xp = x
    x = x - fd(x) / fdd(x) # Replace the denominator as abs(fdd(x)) and (abs(fdd(x)) + 1) to resolve the maxima and saddle point problems

    # Update visualization for each iteration
    print(f'Iter: {i}, x = {xp:.3f} to {x:.3f}, f(x) = {f(xp):.3f} to {f(x):.3f} (f\'(x) = {fd(xp):.3f}, f\'\'(x) = {fdd(xp):.3f})')
    lcolor = np.random.rand(3)
    approx = 0.5*fdd(xp)*(xs-xp)**2 + fd(xp)*(xs-xp) + f(xp)
    plt.plot(xs, approx, '-', linewidth=1, color=lcolor, alpha=0.8)
    plt.plot(x, f(x), '.', color=lcolor, markersize=12)

    # Check the terminal condition
    if abs(x - xp) < min_tol:
        break
plt.show()
```

scipy.optimize: Optimization and Root Finding

- scipy.optimize provides functions for various optimization problems (and root finding).
  - Reference: Documentation and Tutorials
  - Example) Find a local minimum $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$ from $x = 12$ using scipy.optimize

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

f = lambda x: 0.1*x**3 - 0.8*x**2 - 1.5*x + 5.4
viz_range = np.array([-6, 12])
max_iter = 100
min_tol = 1e-6
x_init = 12 # Try -2, 0, and 16/6

# Find the minimum by SciPy
result = minimize(f, x_init, tol=min_tol, options={'maxiter': max_iter, 'return_all': True})
print(result)

# Visualize all iterations
xs = np.linspace(*viz_range, 100)
plt.plot(xs, f(xs), 'r-', label='f(x)', linewidth=2)
xr = np.vstack(result.allvecs)
plt.plot(xr, f(xr), 'b.', label='Each step', markersize=12)
plt.legend()
plt.axis((*viz_range, *f(viz_range)))
plt.show()
```

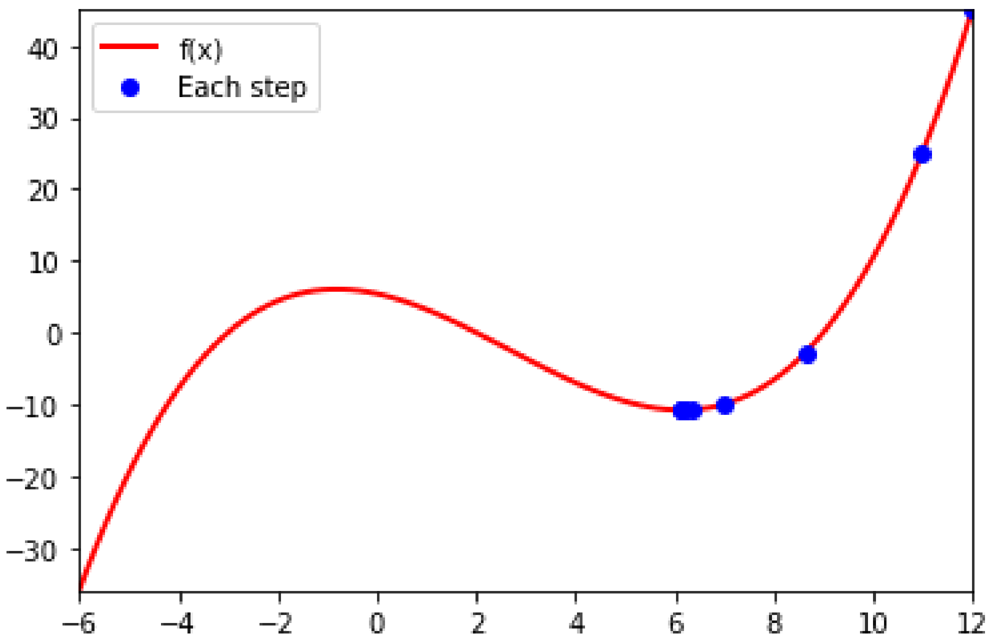
- We don't need to provide derivatives.
- We can control its optimization results using parameters (e.g. `tol` and `options`).

scipy.optimize: Optimization and Root Finding

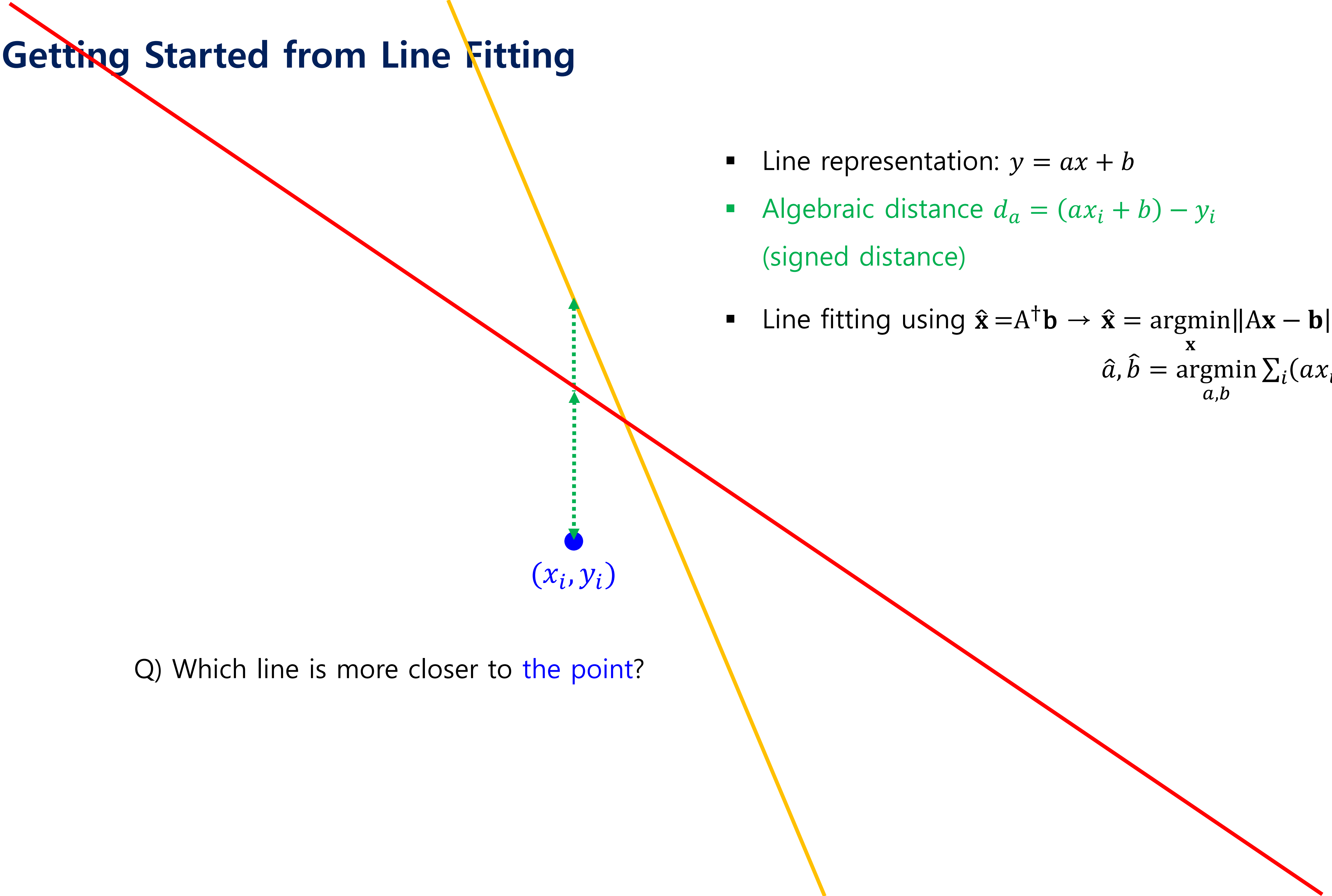
- scipy.optimize provides functions for various optimization problems (and root finding).
  - Reference: Documentation and Tutorials
  - Example) Find a local minimum $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$ from $x = 12$ using scipy.optimize

```
Iter: 57, x = 6.147 to 6.147, f(x) = -10.822 to -10.822 (f'(x) = 0.000)       # GD
Iter:  5, x = 6.147 to 6.147, f(x) = -10.822 to -10.822 (..., f''(x) = 2.088) # Newton
```

```
 allvecs: [array([12.]), array([10.99]), array([8.63764627]), ...]       # SciPy
     fun: -10.822173403490742
hess_inv: array([[0.47882767]])
     jac: array([0.])
 message: 'Optimization terminated successfully.'
    nfev: 18
     nit: 8
    njev: 9
  status: 0
 success: True
       x: array([6.14676882])
```



Getting Started from Line Fitting



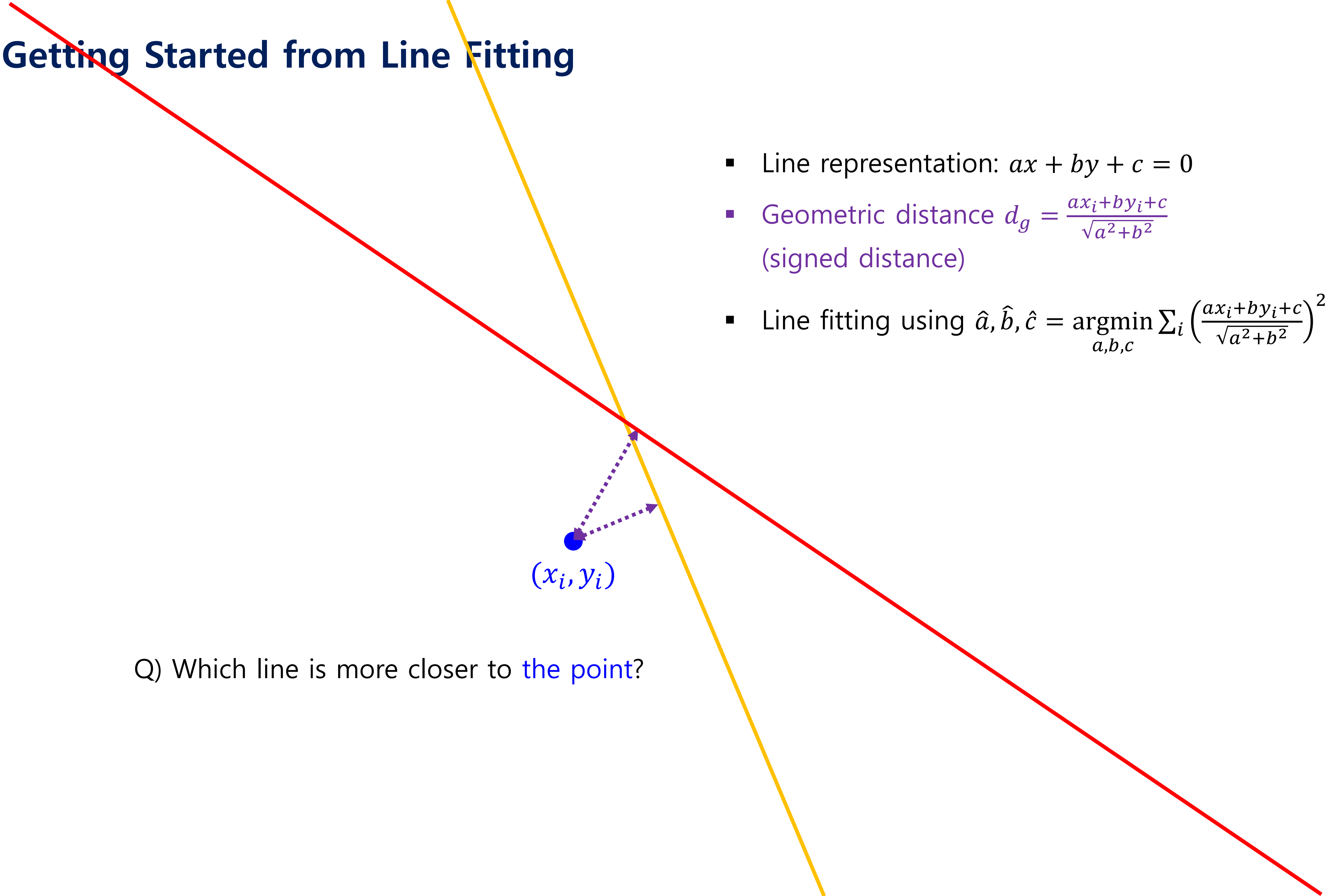
- Line representation: $y = ax + b$
- Algebraic distance $d_a = (ax_i + b) - y_i$
  (signed distance)

- Line fitting using $\hat{\mathbf{x}} = \mathrm{A}^{\dagger}\mathbf{b} \rightarrow \hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \|\mathrm{A}\mathbf{x} - \mathbf{b}\|_2^2$

$$\hat{a}, \hat{b} = \underset{a,b}{\operatorname{argmin}} \sum_i (ax_i + b - y_i)^2$$

Q) Which line is more closer to the point?

Getting Started from Line Fitting

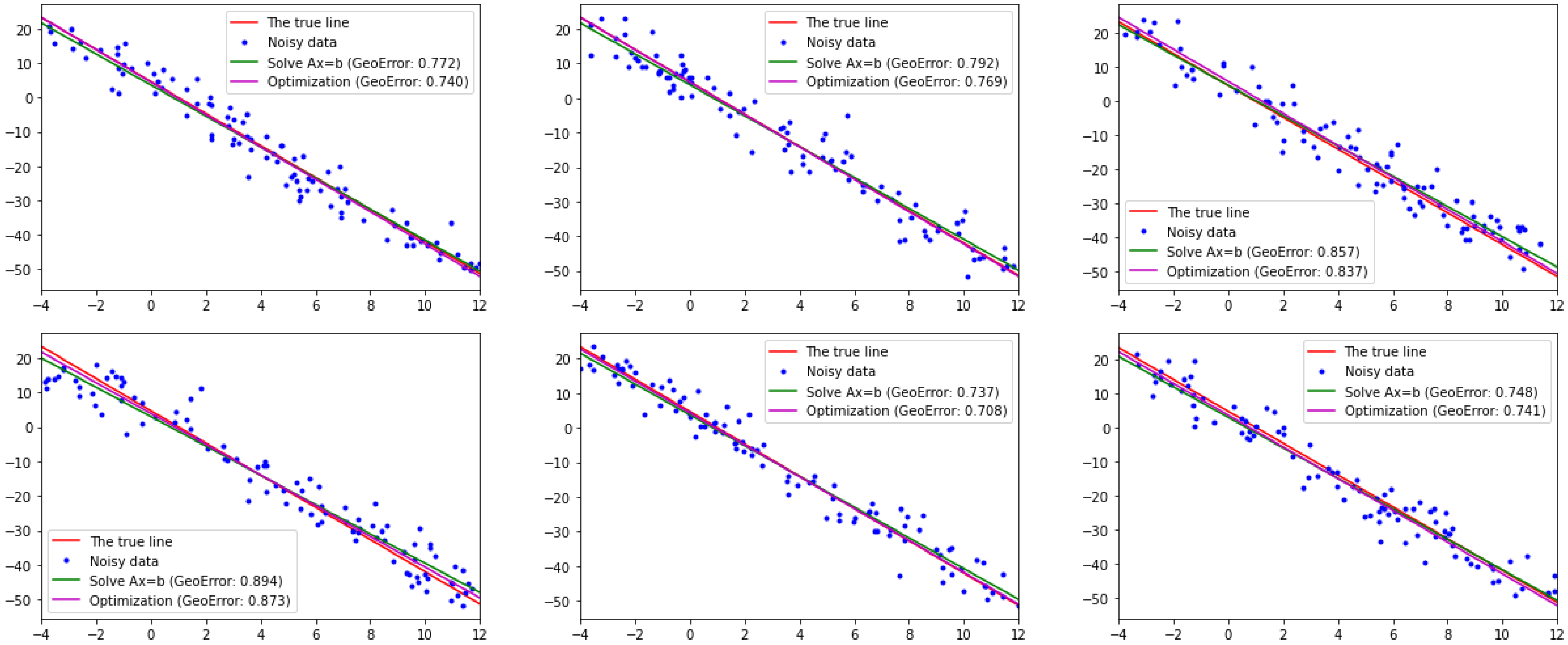


- Line representation: $ax + by + c = 0$
- Geometric distance $d_g = \frac{ax_i + by_i + c}{\sqrt{a^2 + b^2}}$ (signed distance)
- Line fitting using $\hat{a}, \hat{b}, \hat{c} = \underset{a,b,c}{\operatorname{argmin}} \sum_i \left(\frac{ax_i + by_i + c}{\sqrt{a^2 + b^2}} \right)^2$

Q) Which line is more closer to the point?

Objective Functions

- Example) Line fitting with minimizing algebraic distance
  - $\hat{\mathbf{x}} = \mathrm{A}^{\dagger}\mathbf{b} \rightarrow \hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}}\ \|\mathrm{A}\mathbf{x} - \mathbf{b}\|_2^2 = \underset{\mathbf{x}}{\operatorname{argmin}}\ \sum_i (ax_i + b - y_i)^2$ where $\mathbf{x} = [a, b]$
- Example) Line fitting with minimizing geometric distance using [scipy.optimize]()
  - $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}}\ \sum_i \frac{(ax_i - y_i + b)^2}{a^2 + 1}$ where $\mathbf{x} = [a, b]$ for $ax - y + b = 0$
  - Note) Geometric distance will become more helpful when a line has more steeper slope a.



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

true_line = lambda x: -14/3*x + 14/3
data_range = np.array([-4, 12])
data_num = 100
noise_std = 1

# Generate the true data
x = np.random.uniform(data_range[0], data_range[1], size=data_num)
y = true_line(x)

# Add Gaussian noise
xn = x + np.random.normal(scale=noise_std, size=x.shape)
yn = y + np.random.normal(scale=noise_std, size=y.shape)

# Find a line minimizing algebraic distance
A = np.vstack((xn, np.ones(xn.shape))).T
b = yn
l_alg = np.linalg.pinv(A) @ b
e_alg = np.mean(np.abs(l_alg[0]*xn - yn + l_alg[1]) / np.sqrt(l_alg[0]**2 + 1))

# Find a line minimizing geometric distance
geo_dist2 = lambda x: np.sum((x[0]*xn - yn + x[1])**2) / (x[0]**2 + 1)
result = minimize(geo_dist2, [-1, 0]) # The initial value: y = -x
l_geo = result.x
e_geo = np.mean(np.abs(l_geo[0]*xn - yn + l_geo[1]) / np.sqrt(l_geo[0]**2 + 1))

# Plot the data and result
plt.plot(data_range, true_line(data_range), 'r-', label='The true line')
plt.plot(xn, yn, 'b.', label='Noisy data')
plt.plot(data_range, l_alg[0]*data_range + l_alg[1], 'g-', label=f'Solve Ax=b (GeoError: {e_alg:.3f})')
plt.plot(data_range, l_geo[0]*data_range + l_geo[1], 'm-', label=f'Optimization (GeoError: {e_geo:.3f})')
plt.legend()
plt.xlim(data_range)
plt.show()
```

Note) $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i (ax_i + b - y_i)^2$ where $\mathbf{x} = [a, b]$

Note) $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_i \frac{(ax_i - y_i + b)^2}{a^2+1}$ where $\mathbf{x} = [a, b]$

Summary

- ~~Optimization~~ **Unconstrained nonlinear optimization**
  - ~ Finding arguments $\mathbf{x}$ to minimize the *nonlinear* objective function $f(\mathbf{x})$ ~ $\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x})$
  - **Gradient descent** using the **1st-order** approximation and the **given** step size.
    - Possible problems: Too small step size, too large step size
    - Note) Stochastic gradient descent (SGD) uses gradient values derived from randomly selected data.
  - **Newton's method** using the **2nd-order** approximation **without** the step size.
    - Possible problems: The maxima problem, the saddle point problem
    - Note) Gauss-Newton method is a special case for $f(\mathbf{x}) = r^2(\mathbf{x})$.
  - scipy.optimize: A sub-module in SciPy for optimization **without** derivatives
    - You can find minima of any given functions without derivatives.
- **Selecting an objective function is important.**
  - e.g. Algebraic distance vs. geometric distance in line fitting

Our life is full of optimization problems. What is your objective in your life?