

香 港 大 學
THE UNIVERSITY OF HONG KONG

# Algorithms for Unconstrained Programs

**Prof. Li Jiayang**

**DASE7113 Optimization Methods for Intelligent Systems**

# Optimality Condition

## Unconstrained Convex Programs

**Given a differentiable convex function $f: \mathbb{R}^n \to \mathbb{R}$, $x^*$ is the solution to the following optimization problem if and only if $\nabla f(x^*) = 0$.**

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

# Logistic Regression

## Question

How would you model the relationship between $y$ and $x_1, x_2$ defined as follows?

$y$: whether or not a college school student is admitted by a master's program

$y = 1$ if admitted

$y = 0$ if not admitted

$x_1$: the student's college GPA (scaled to [0, 1])

$x_2$: whether or not the student has publications

# Logistic Regression

## Question

How would you model the relationship between $y$ and $x_1, x_2$ defined as follows?

$y$: whether or not a college school student is admitted by a master's program

$x_1$: the student's college GPA (scaled to [0, 1])

$x_2$: whether or not the student has publications

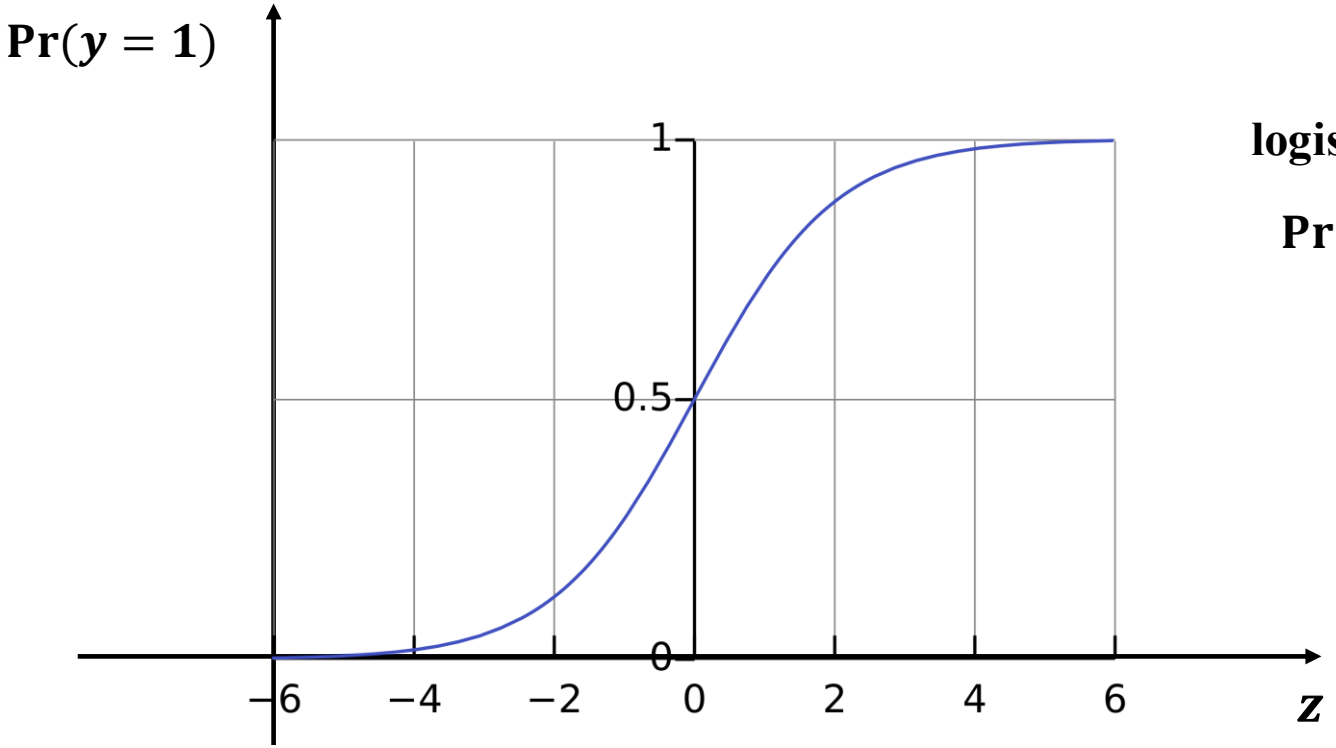
A continuous variable $z = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2$ → The "potential" to be admitted

The higher $z$ is, the more likely the student is to be admitted

# Logistic Regression

A continuous variable $z = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2$ → The "potential" to be admitted

The higher $z$ is, the more likely the student is to be admitted




logistic (sigmoid) function

$$\Pr(y = 1) = \frac{1}{1 + e^{-z}}$$

# Logistic Regression

$$z_1 = \beta_0 + \cdots + \beta_2 \cdot x_{12}$$

$$\vdots$$

$$z_n = \beta_0 + \cdots + \beta_2 \cdot x_{n2}$$

**Predicted probability**

$$\Pr(y_1 = 1) = \frac{1}{1 + e^{-z_1}}$$

$$\vdots$$

$$\Pr(y_n = 1) = \frac{1}{1 + e^{-z_n}}$$

**Real observation**

$$y_1 = 1$$

$$\vdots$$

$$y_n = 0$$

$$\min_{\beta} \sum_{i=1}^{n} -y_i \cdot \log(p_i) - (1 - y_i) \cdot \log(1 - p_i).$$

# Logistic Regression

## Question

Consider the following logistic regression model

$$\underset{\boldsymbol{\beta}}{\text{minimize}}\ l(\boldsymbol{\beta}) = \sum_{i=1}^{n} -y_i \cdot \log(p_i) - (1-y_i) \cdot \log(1-p_i),$$

$$\text{where } p_i = \frac{1}{1+e^{-z_i}},\ z_i = \beta_0 + \beta_1 \cdot x_{i1} + \cdots + \beta_m \cdot x_{im}.$$

Solve the optimal $\boldsymbol{\beta}^*$.

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{nm} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_m \end{bmatrix} \quad \boldsymbol{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix} \quad \boldsymbol{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$z = X\beta \qquad p = \frac{1}{1+\exp(-z)}$$

$$l(\beta) = -y^{\mathrm{T}}\log(p) - (1-y)^{\mathrm{T}}\log(1-p)$$

# Logistic Regression

$$l = l(\beta) = -y^{\mathrm{T}}\log(p) - (1-y)^{\mathrm{T}}\log(1-p), \quad \text{where} \quad p = \frac{1}{1+\exp(-z)} \quad \text{and} \quad z = X\beta$$

$$\Rightarrow \quad \dot{p} = \left(\frac{\partial l}{\partial p}\right)^{\mathrm{T}} = -\frac{y}{p} + \frac{1-y}{1-p} \quad \Rightarrow \quad \dot{z} = \left(\frac{\partial l}{\partial z}\right)^{\mathrm{T}} = \left(\frac{\partial p}{\partial z}\right)^{\mathrm{T}} \dot{p}$$

$$\Rightarrow \quad \nabla l(\beta) = \left(\frac{\partial l}{\partial \beta}\right)^{\mathrm{T}} = X^{\mathrm{T}}\dot{z} \quad \Rightarrow \quad \nabla l(\beta) = \left(\frac{\partial l}{\partial \beta}\right)^{\mathrm{T}} = X^{\mathrm{T}}\mathrm{diag}\big(p \circ (1-p)\big)\left(-\frac{y}{p} + \frac{1-y}{1-p}\right) = X^{\mathrm{T}}(p-y)$$

Setting $\nabla l(\beta^*) = X^{\mathrm{T}}(p-y) = 0$.....

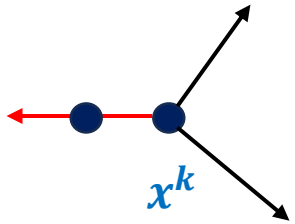
$$X^{\mathrm{T}}\left(\frac{1}{1+\exp(-X\beta^*)} - y\right) = 0$$

⇒ Do you know how to solve this equation?

# Gradient Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

If $x^k$ is your current solution, how can you find a better solution $x^{k+1}$ ?



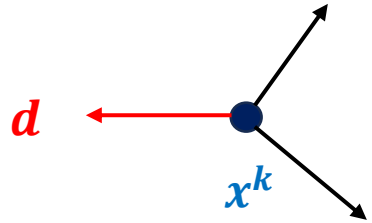

**Find a descent direction and move an appropriate distance in that direction!**

# Gradient Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**What is a descent direction?**

$$g(\alpha) = f(x^k + \alpha \cdot d)$$




$g(\alpha) \leq g(0)$ for all $\alpha \geq 0$……

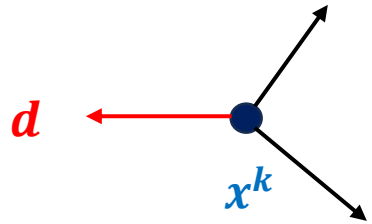
Too strong!

# Gradient Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**What is a descent direction?**

$$g(\alpha) = f(x^k + \alpha \cdot d)$$




$g(\alpha) \leq g(0)$ as long as $\alpha$ is sufficiently small

# Gradient Descent

## Coding

**Gradient:** $\begin{bmatrix} 4x_1 + 4x_2 - 8 \\ 4x_1 + 10x_2 - 6 \end{bmatrix}$

Consider $f: \mathbb{R}^2 \to \mathbb{R}$ defined as $f(x_1, x_2) = 2x_1^2 + 4x_1x_2 + 5x_2^2 - 8x_1 - 6x_2 + 10$.

At the point $(x_1, x_2) = (2, 2)$, set $d_1 = \nabla f(x_1, x_2)$ and $d_2 = -\nabla f(x_1, x_2)$

(1) Plot $g(\alpha) = f(x^k + \alpha \cdot d_1)$




(2) Plot $g(\alpha) = f(x^k + \alpha \cdot d_2)$



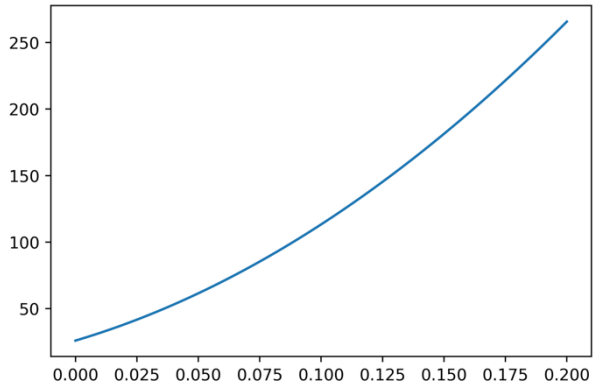
# Gradient Descent

## Coding

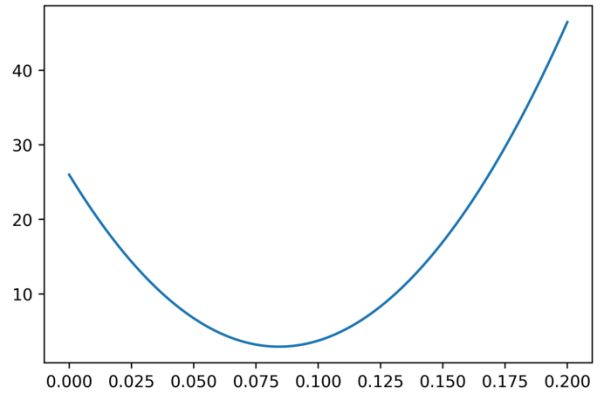
Consider $f: \mathbb{R}^2 \to \mathbb{R}$ defined as $f(x_1, x_2) = 2x_1^2 + 4x_1x_2 + 5x_2^2 - 8x_1 - 6x_2 + 10$.

At the point $(x_1, x_2) = (2, 2)$, set $d_1 = \nabla f(x_1, x_2)$ and $d_2 = -\nabla f(x_1, x_2)$

(1) Plot $g(\alpha) = f(x^k + \alpha \cdot d_1)$




(2) Plot $g(\alpha) = f(x^k + \alpha \cdot d_2)$

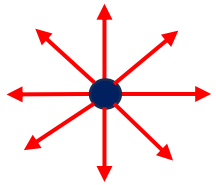


# Gradient Descent

## Coding

**Consider $f: \mathbb{R}^2 \to \mathbb{R}$ defined as $f(x_1, x_2) = 2x_1^2 + 4x_1x_2 + 5x_2^2 - 8x_1 - 6x_2 + 10$.**

**At the point $(x_1, x_2) = (2, 2)$, scan over all unit directions**



```
theta = np.linspace(0, 2 * np.pi, 100)
unit_directions =  np.array([(np.cos(t), np.sin(t)) for t in theta])
for direction in unit_directions:
    print(direction)
```

**Move a distance of $\delta = 0.01$ in every direction:**

**(1) Visualize all descent directions; (2) Find the direction that leads to the most decrease**

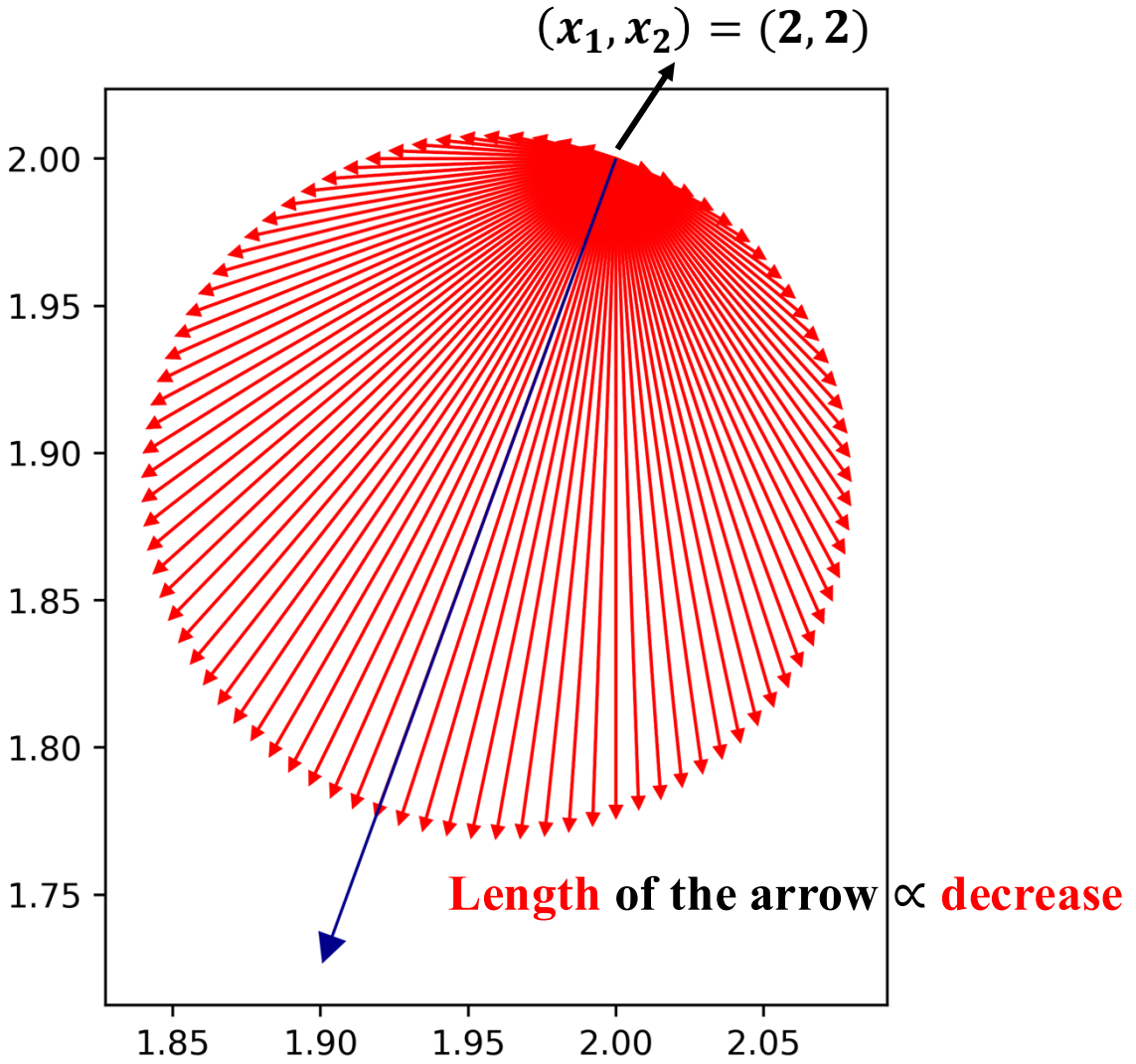
# Gradient Descent

**Conclusion 1:**

**Descent directions:** $\{\boldsymbol{d} \in \mathbb{R}^{\boldsymbol{n}}: \langle \boldsymbol{d}, -\nabla \boldsymbol{f}(\boldsymbol{x}) \rangle \geq \mathbf{0}\}$

**Conclusion 2:**

**Steepest descent direction:** $-\nabla \boldsymbol{f}(\boldsymbol{x})$



# Steepest Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**The steepest descent method:**

For $k = 0, 1, \ldots$

Compute $\nabla f(x^k)$

Set $x^{k+1} = x^k - \alpha^k \cdot \nabla f(x^k)$

If XXXX, break

This is called the step size

This is called the termination condition

# Steepest Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**When can we stop iterating?**

$||\nabla f(x^k)||_2 < \epsilon$ **because** $x^*$ **is the optimal solution if and only if** $\nabla f(x^*) = 0$.

$||x^k - x^{k-1}||_2 < \epsilon$ **i.e. the decision variable is almost not moving**

$|f(x^k) - f(x^{k-1}) < \epsilon$ **i.e. the objective value is almost not changing**

# Steepest Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**How to determine the step size $\alpha^k$ ?**

**Strategy I (fixed step size):**

**Fix $\alpha^k$ to be a sufficiently small constant**

**Let's try this on logistic regression**

# Steepest Descent

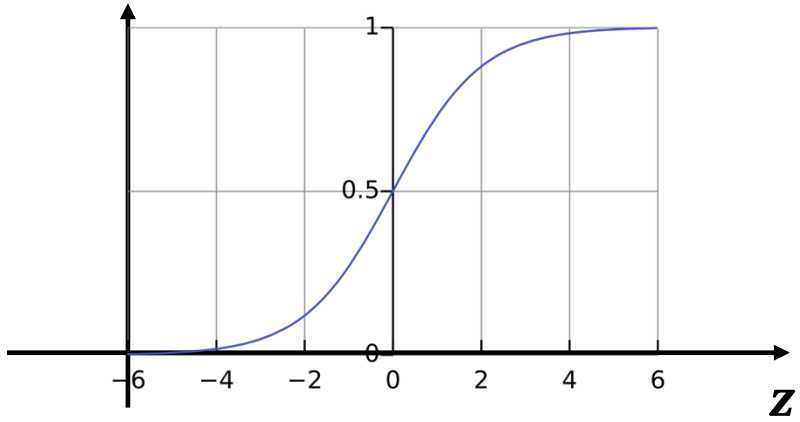
## Setting

$x_1$: college GPA (scaled to [0, 1])

$x_2$: having publication or not

$y$: admitted or not

$$z = 100 \cdot x_1 + 10 \cdot x_2 - 85$$




**Borderline case:**

- 0.85 GPA with no publication
- Or 0.75 GPA with some publications

$$\Pr(y = 1) = \frac{1}{1 + e^{-z}}$$

# Steepest Descent

## Coding

**Train a logistic model to regress $y$ on $x_1$ and $x_2$**

**$x_1$: college GPA (scaled to [0, 1])**

**$x_2$: having publication or not**

**$y$: admitted or not**

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

# Steepest Descent

## Coding

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(penalty="none",
fit_intercept=False)
model.fit(X, y)

print(model.coef_)
```

**Coefficient: 96.88  9.84  -82.47**

# Steepest Descent

## Coding

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

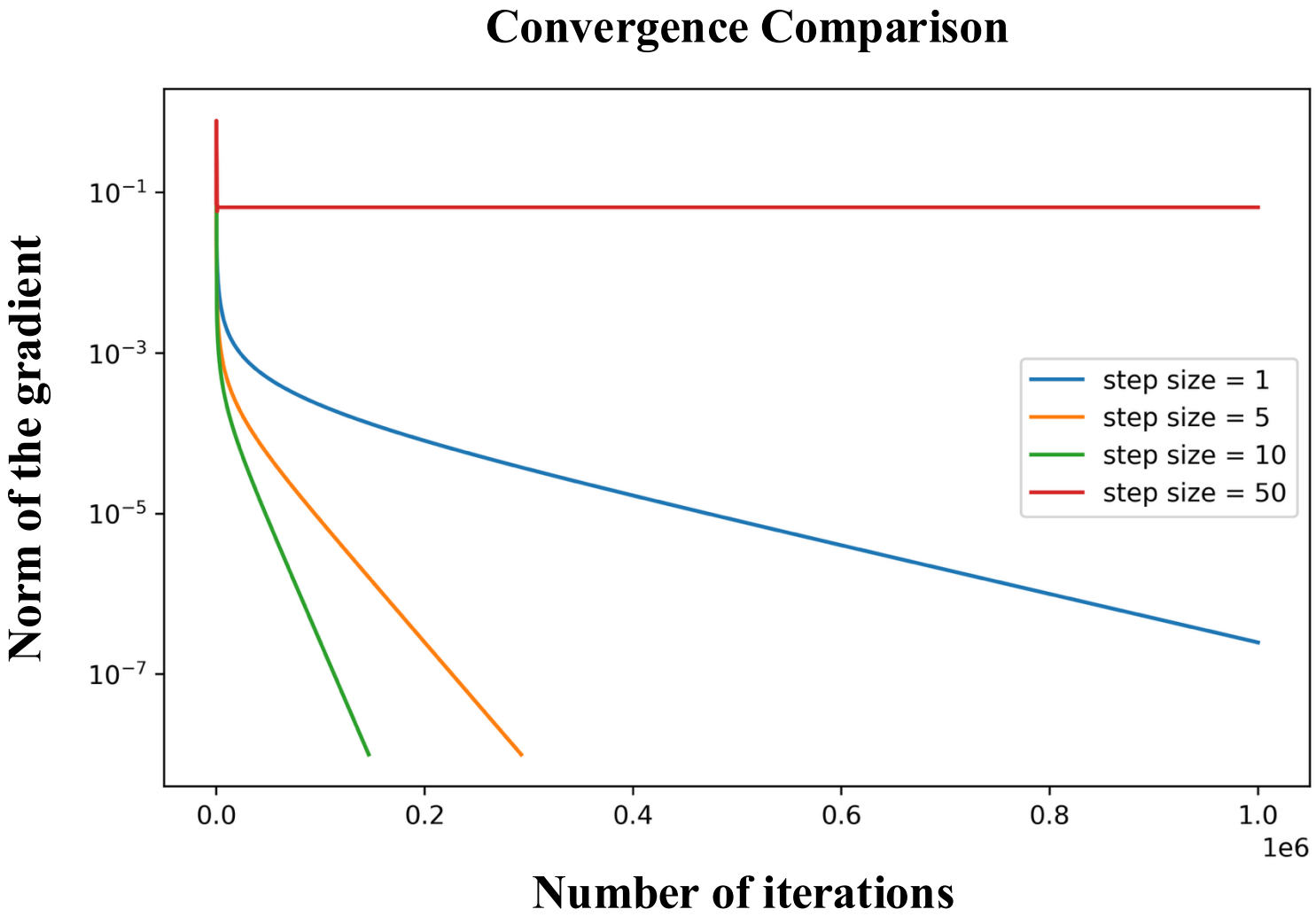
Try some different step sizes!

```
step_size= 1
num_iterations = 1000000
w = np.zeros(X.shape[1])

for i in range(num_iterations):
    z = X @ w
    y_pred = 1 / (1 + np.exp(-z))
    gradient = X.T @ (y_pred - y) / N

    if np.linalg.norm(gradient, 2) < 1e-8:
        break
    w -= step_size * gradient
```

# Steepest Descent



# Steepest Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**How to determine the step size $\alpha^k$ ?**

**Strategy II (exact line search):**

$$\alpha^k = \operatorname{argmin}_{\alpha \geq 0} f(x^k - \alpha \cdot \nabla f(x^k))$$

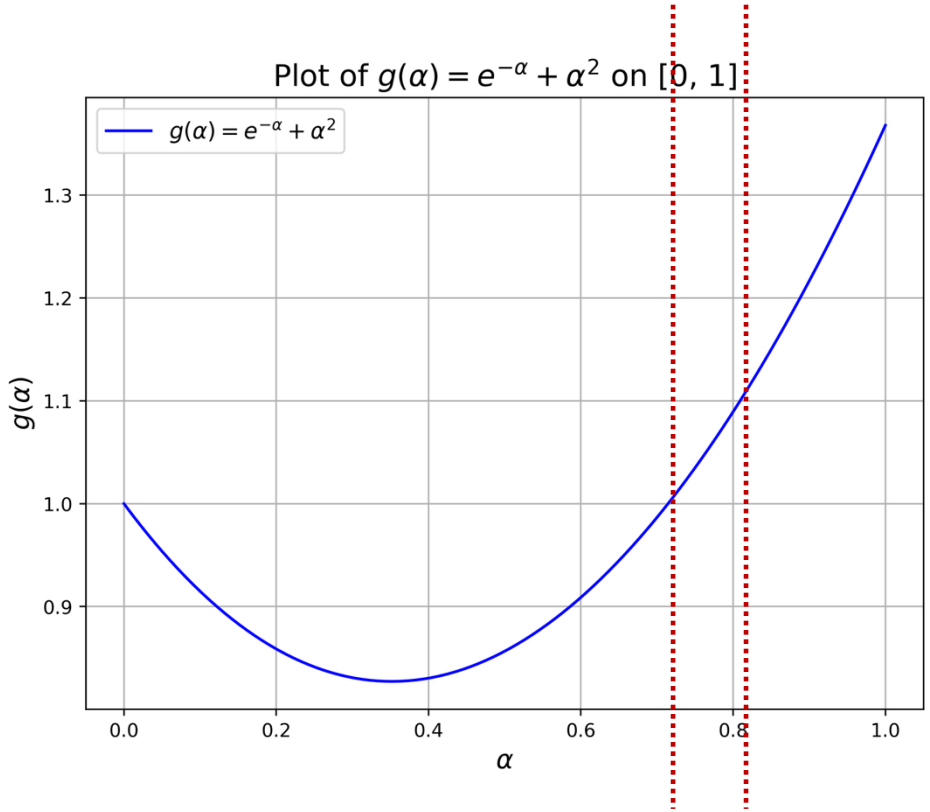
$$g(\alpha) = f(x^k - \alpha \cdot \nabla f(x^k))$$

**How to minimize $g(\alpha)$ over $[0, \infty)$?**

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**




**Step 1: Identify an interval $[0, \alpha_0]$ that contains the minimum point**

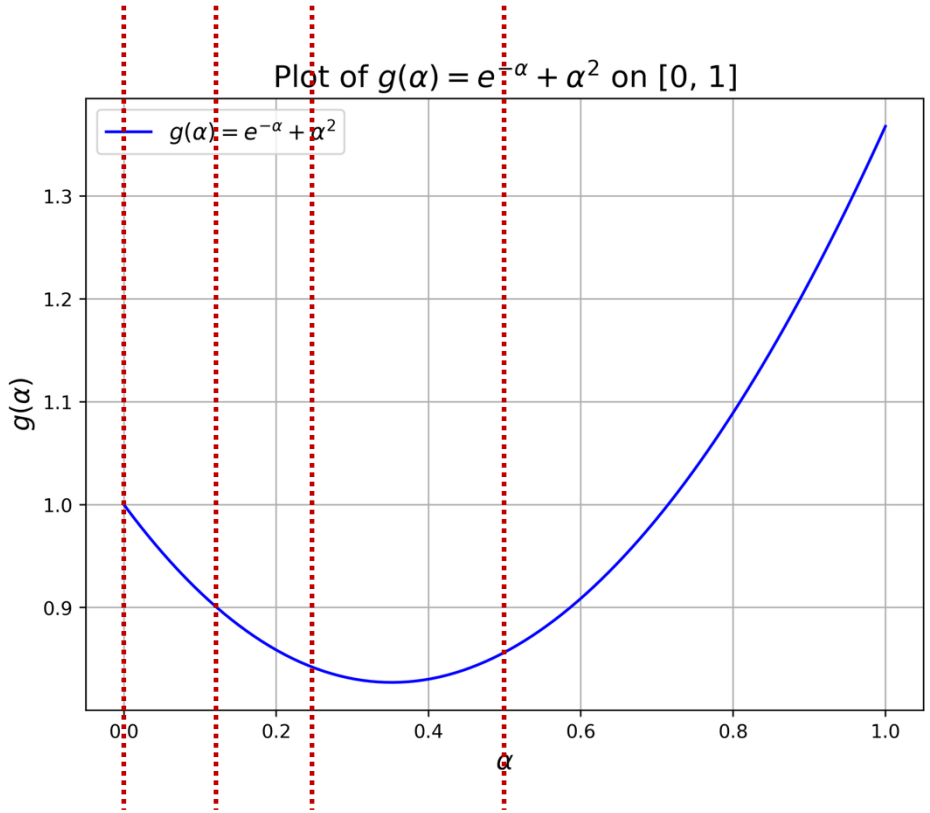
**Basic idea:**

**As long as there exists some $\alpha < \alpha_0$**
**such that $g(\alpha) < g(\alpha_0)$**

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**




**Step 1: Identify an interval $[0, \alpha_0]$ that contains the minimum point**

**Set $\alpha = 0.125$ and $g_0 = g(0)$**

**While True**

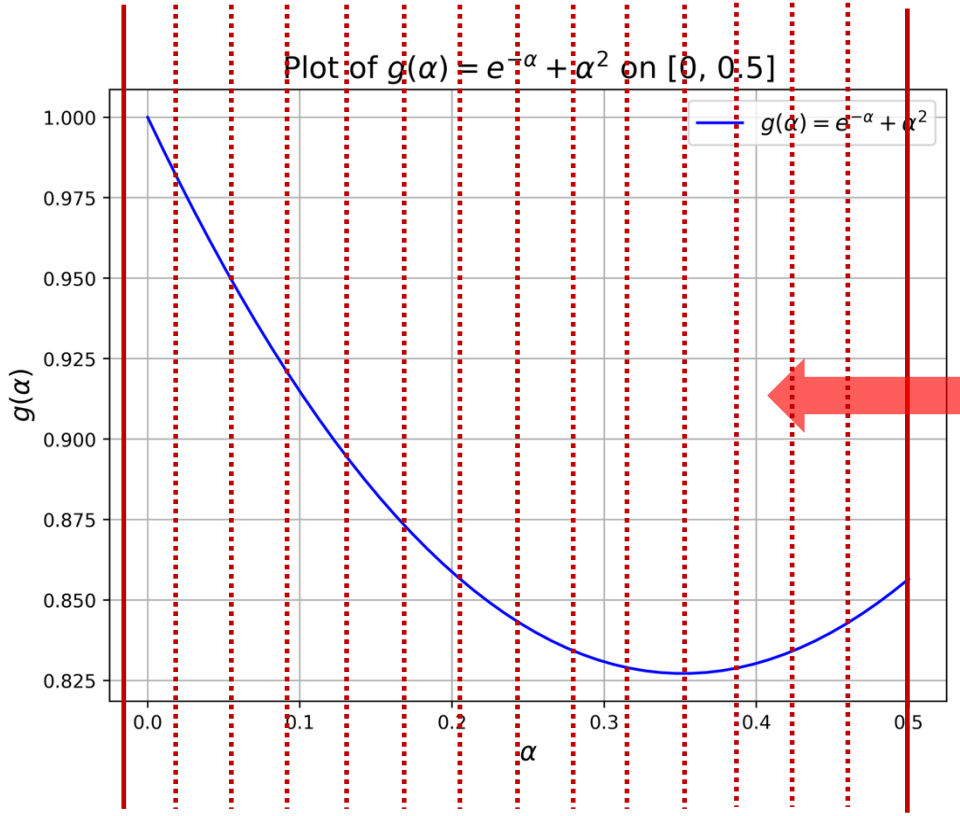
**If $g(\alpha) \geq g_0$, return $\alpha_0 = \alpha$**

**Else, set $g_0 = g(\alpha)$ and $\alpha = 2 \cdot \alpha$**

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**




**Step 2: Search within $[0, \alpha_0]$ to find the minimum point**

scipy.optimize.brute

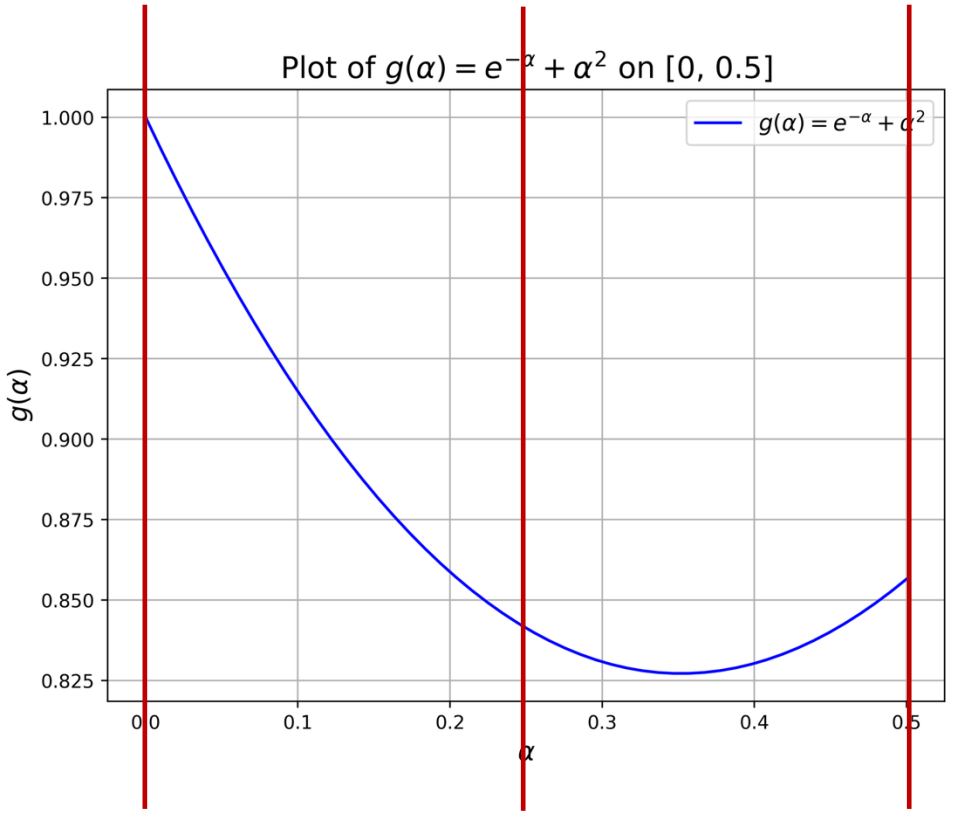
**A brute-force approach: Grid search**

**If you want to roughly locate the minimum point,**
**this is nice**

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**

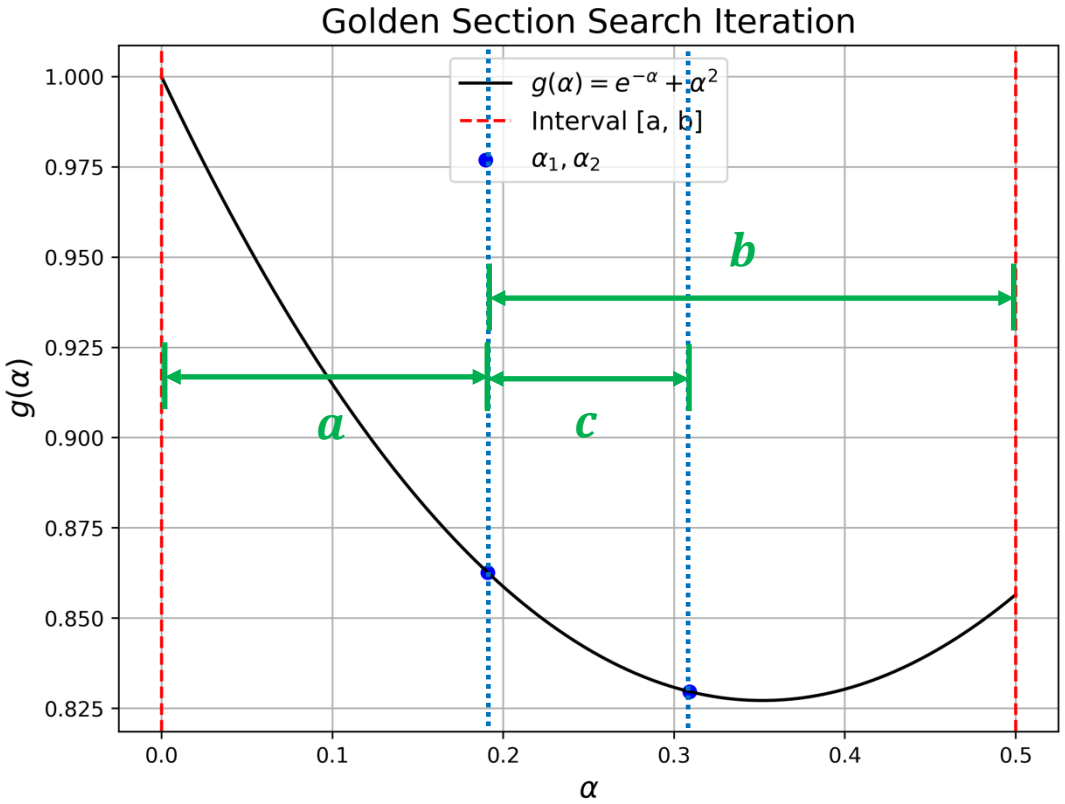



**Step 2: Search within $[0, \alpha_0]$ to find the minimum point**

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**




**Golden-section search**

$$\frac{c}{a} = \frac{a}{b} \Rightarrow \frac{b-a}{a} = \frac{a}{b}$$

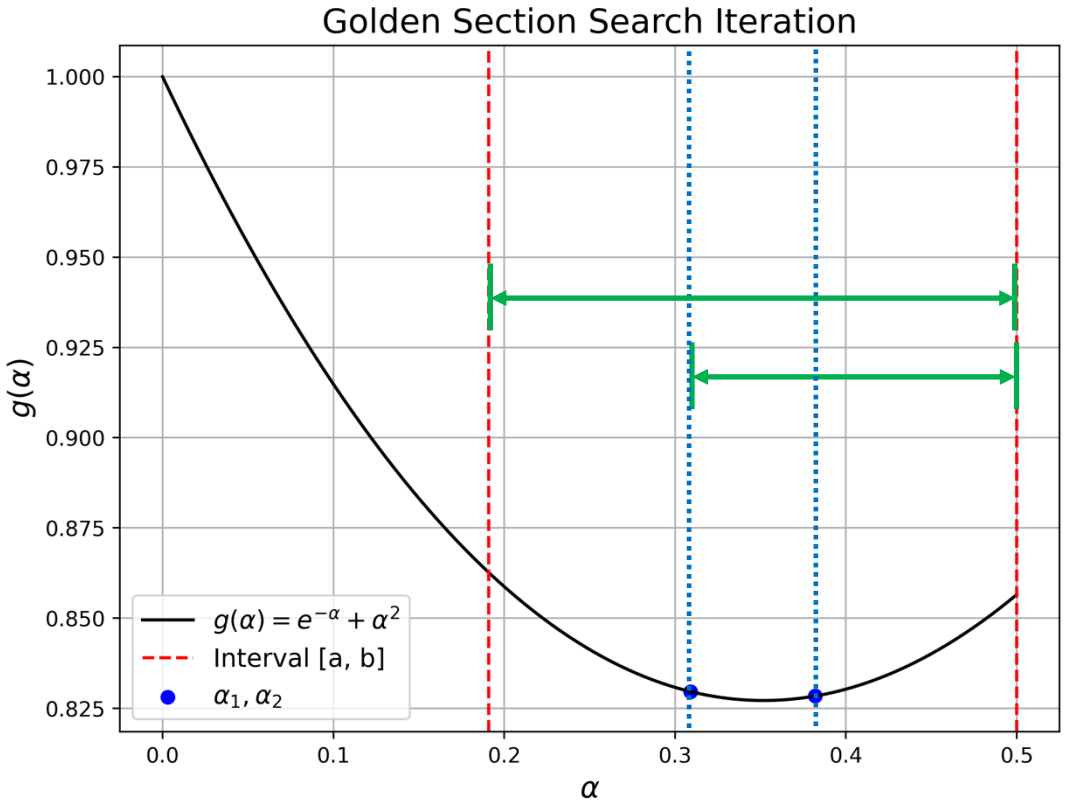
$$\Rightarrow \frac{b}{a} - 1 = \frac{a}{b}$$

$$\Rightarrow 1 - \frac{a}{b} = \left(\frac{a}{b}\right)^2 \Rightarrow \frac{a}{b} = \frac{\sqrt{5}-1}{2} \approx 0.618$$

# Exact Line Search

## Question

**Given a convex function $g(\alpha)$ defined on $[0, \infty)$, how can we solve its minimum point?**



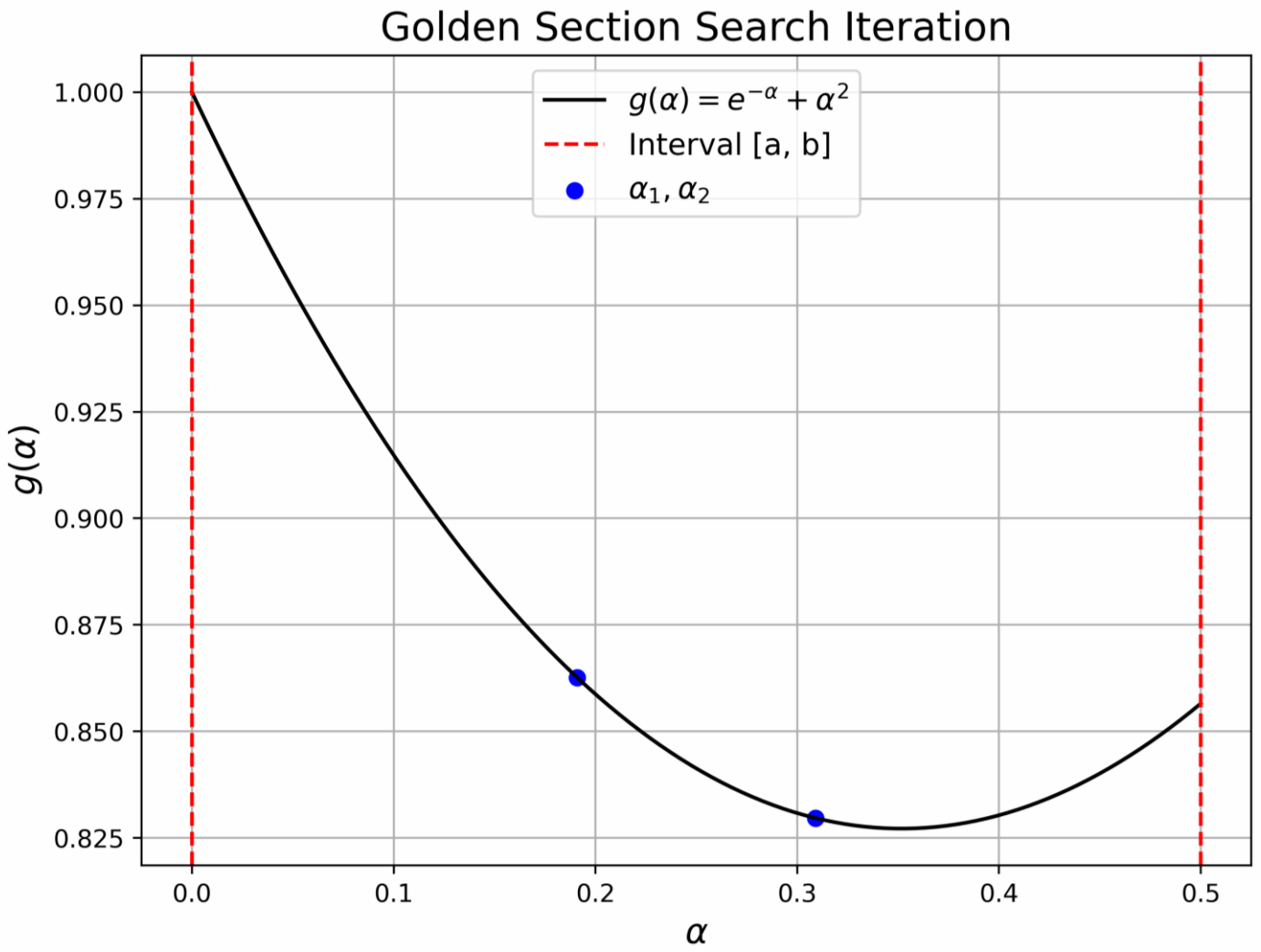

**Golden-section search**

$$\frac{c}{a} = \frac{a}{b} \Rightarrow \frac{b-a}{a} = \frac{a}{b}$$

$$\Rightarrow \frac{b}{a} - 1 = \frac{a}{b}$$

$$\Rightarrow 1 - \frac{a}{b} = \left(\frac{a}{b}\right)^2 \Rightarrow \frac{a}{b} = \frac{\sqrt{5}-1}{2} \approx 0.618$$

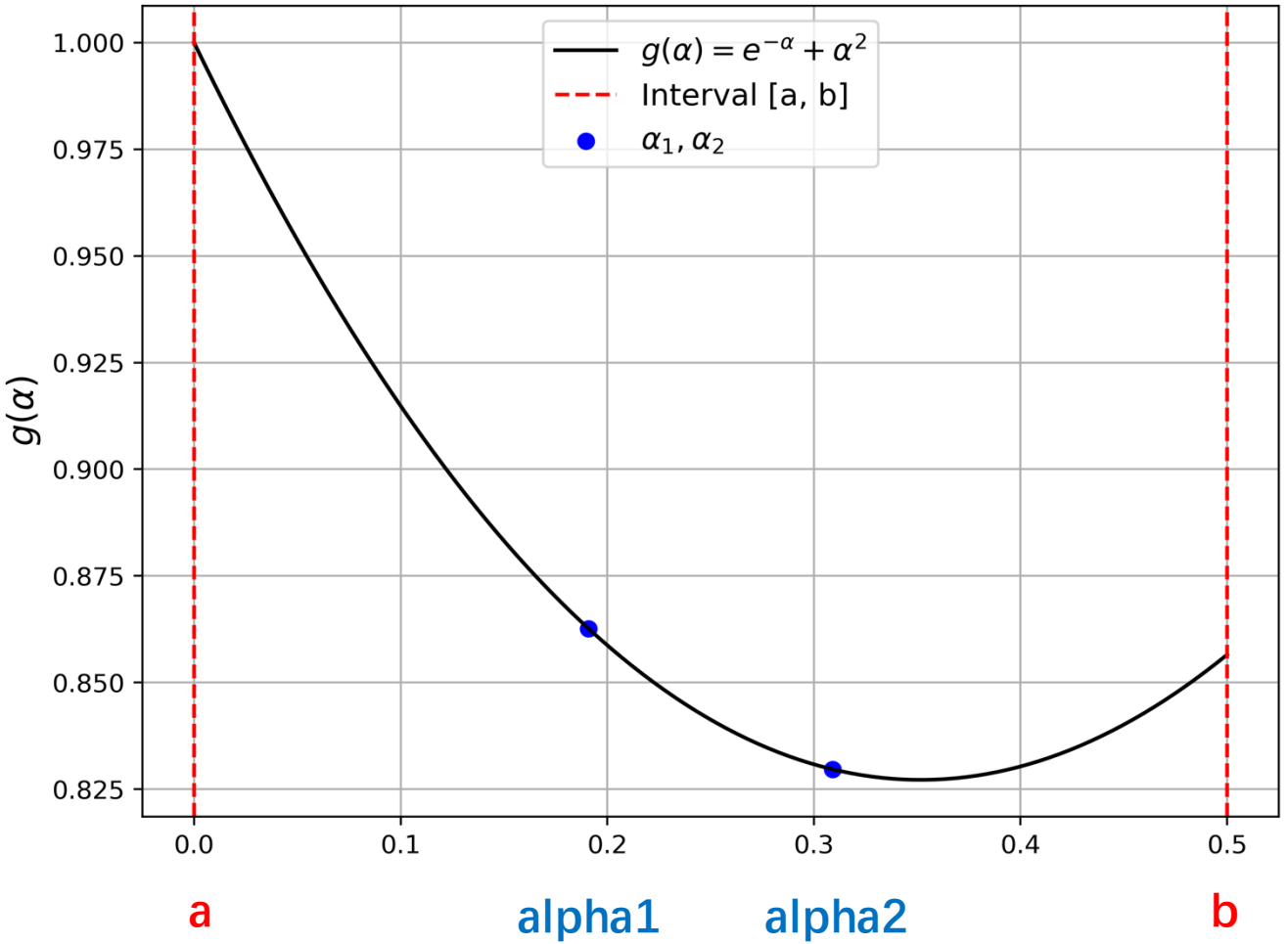
# Exact Line Search



# Exact Line Search

```
phi = (np.sqrt(5) - 1) / 2
alpha1 = a + (1 - phi) * (b - a)
alpha2 = a + phi * (b - a)
g1 = g(alpha1)
g2 = g(alpha2)

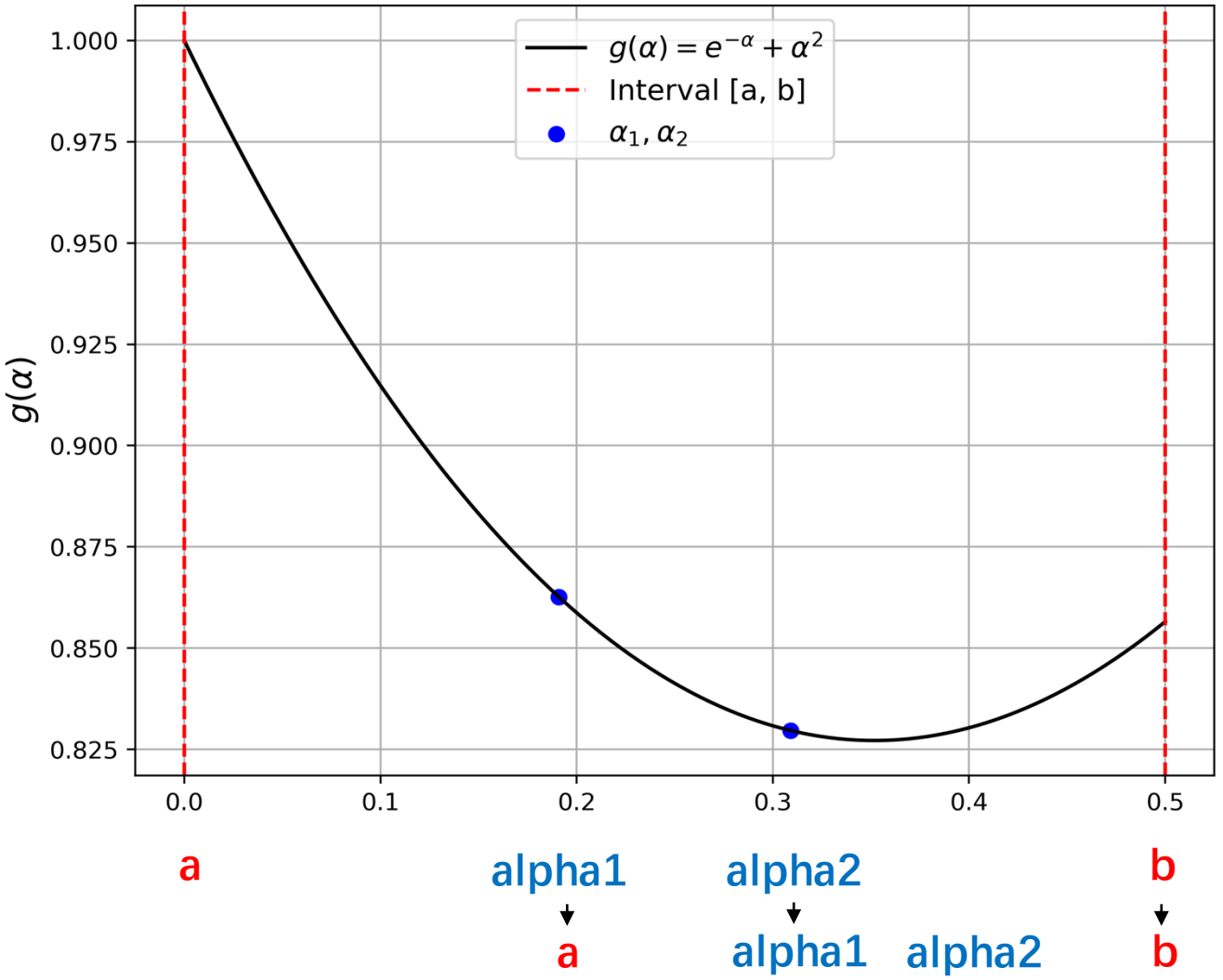
while b - a > tol:
    if g1 < g2:
        # Minimum is in [a, alpha2]
        b = alpha2
        alpha2 = alpha1
        g2 = g1
        alpha1 = a + (1 - phi) * (b - a)
        g1 = g(alpha1)
```



# Exact Line Search

```
phi = (np.sqrt(5) - 1) / 2
alpha1 = a + (1 - phi) * (b - a)
alpha2 = a + phi * (b - a)
g1 = g(alpha1)
g2 = g(alpha2)

while b - a > tol:
    if g1 >= g2:
        # Minimum is in [alpha1, b]
        a = alpha1
        alpha1 = alpha2
        g1 = g2
        alpha2 = a + phi * (b - a)
        g2 = g(alpha2)
```



# Exact Line Search

scipy.optimize.

## minimize_scalar

**minimize_scalar**(*fun*, *bracket=None*, *bounds=None*, *args=()*, *method=None*, *tol=None*, *options=None*) [source]

Local minimization of scalar function of one variable.

```
from scipy.optimize import minimize_scalar

def g(alpha):
    return np.exp(-alpha) + alpha**2

result = minimize_scalar(g, method='golden', bracket=(0, 0.5))
```

# Steepest Descent

## Coding

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

```
step_size= 1
num_iterations = 1000000
w = np.zeros(X.shape[1])

for i in range(num_iterations):
    z = X @ w
    y_pred = 1 / (1 + np.exp(-z))
    gradient = X.T @ (y_pred - y) / N

    if np.linalg.norm(gradient, 2) < 1e-8:
        break
    w -= step_size * gradient
```

**Now, try to determine the step size via exact line search**

# Steepest Descent

## Coding

```
import numpy as np

np.random.seed(0)

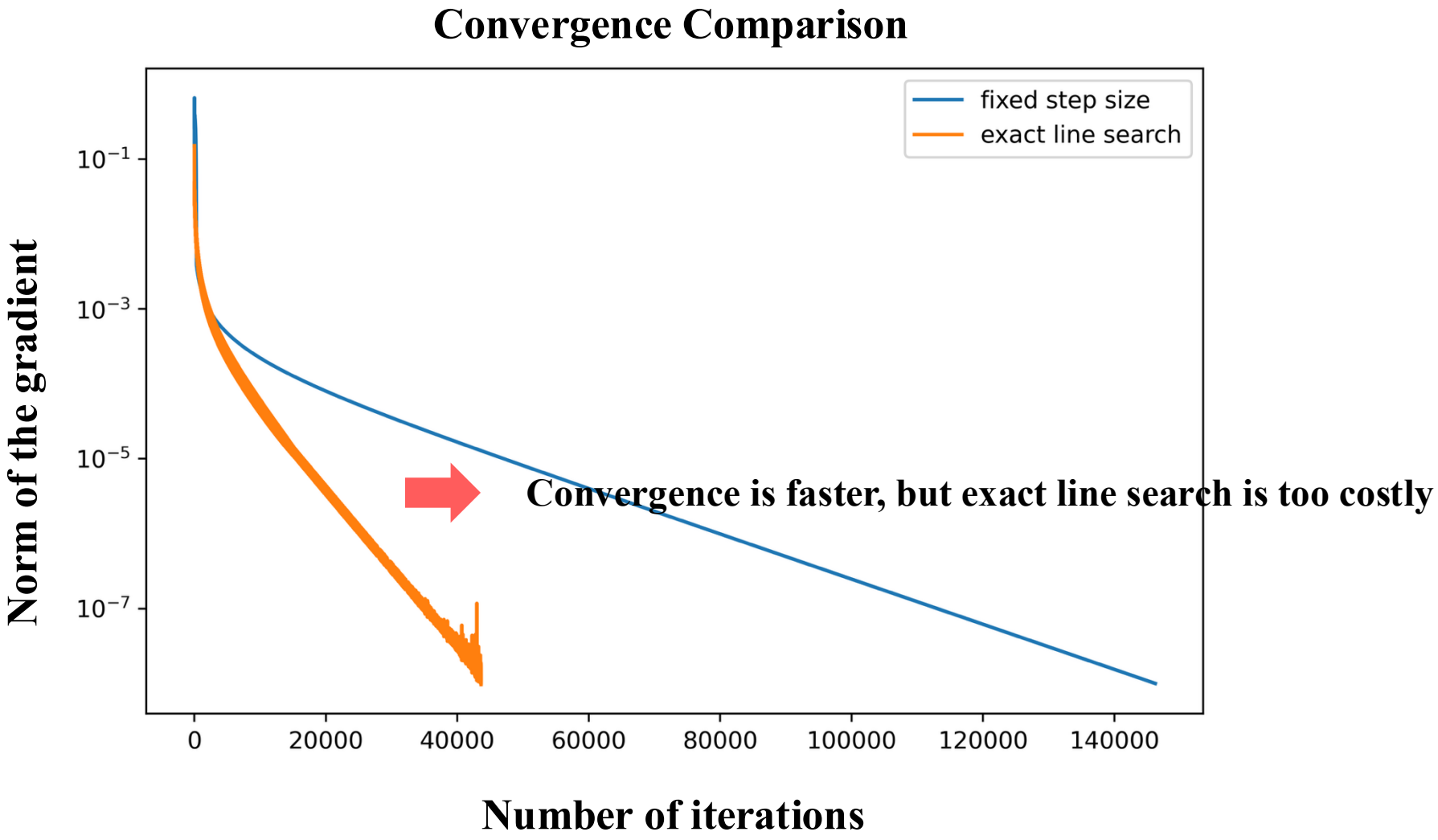
N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

```
def g(alpha):
    w_new = w - alpha * gradient
    z = X @ w_new
    y_pred = 1 / (1 + np.exp(-z))
    return -np.mean(y * np.log(y_pred) + (1 - y) *
np.log(1 - y_pred))

result = minimize_scalar(g, method='golden',
bracket=(0, 50))

learning_rate = result.x
```

# Steepest Descent



# Steepest Descent

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**How to determine the step size $\alpha^k$ ?**

**Strategy III (inexact line search):**

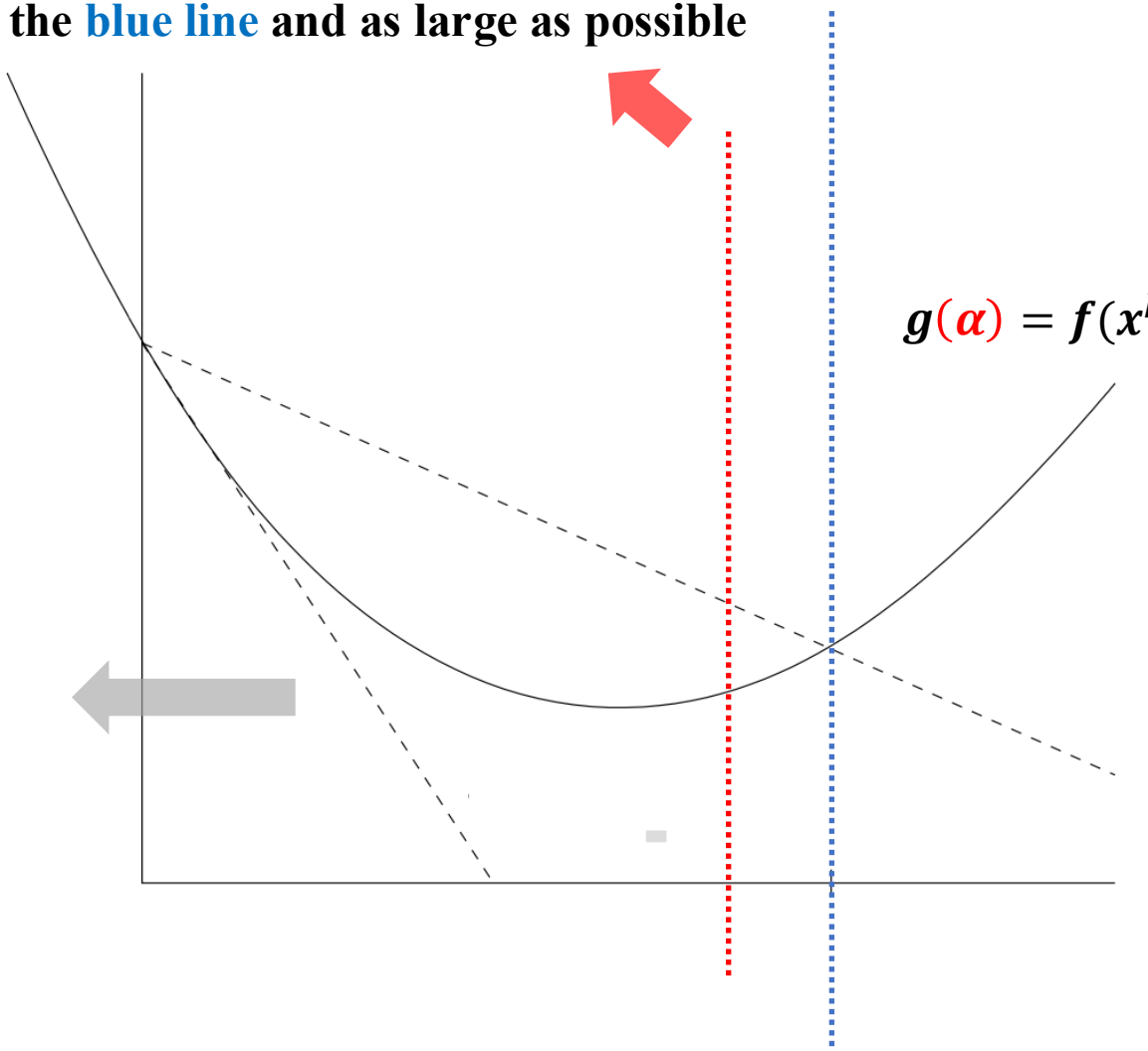
**Perform the line search in a "take-it-easy" manner**

$$g(\alpha) = f(x^k - \alpha \cdot \nabla f(x^k))$$

**Find an approximated minimizer of $g(\alpha)$**

# Steepest Descent

Find some $\alpha$ left to the blue line and as large as possible



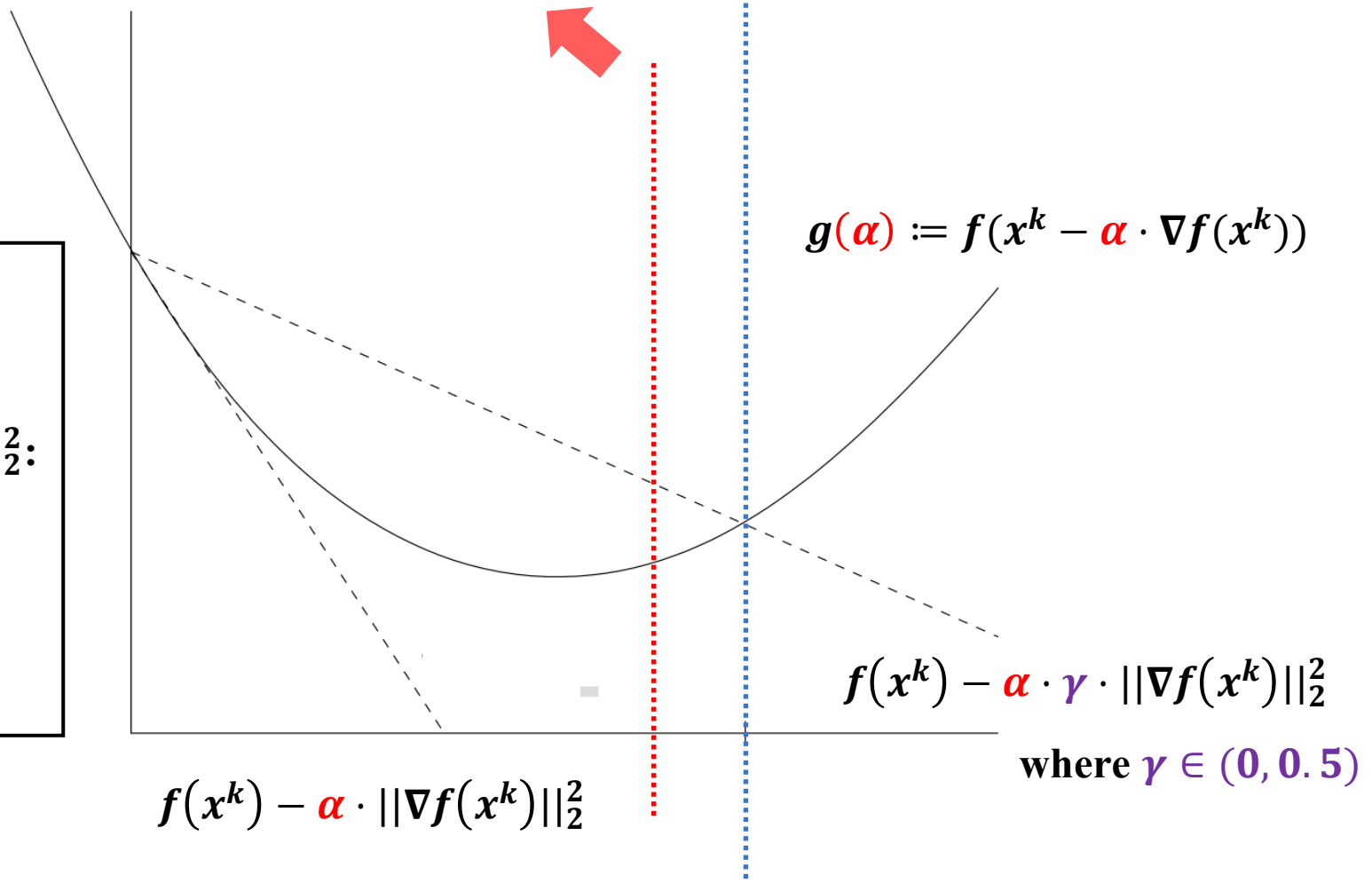
$$g(\alpha) = f(x^k - \alpha \cdot \nabla f(x^k))$$

Compute the slope of this line

$$g'(0) = -||\nabla f(x^k)||_2^2$$

# Steepest Descent

Find some $\alpha$ left to the blue line and as large as possible



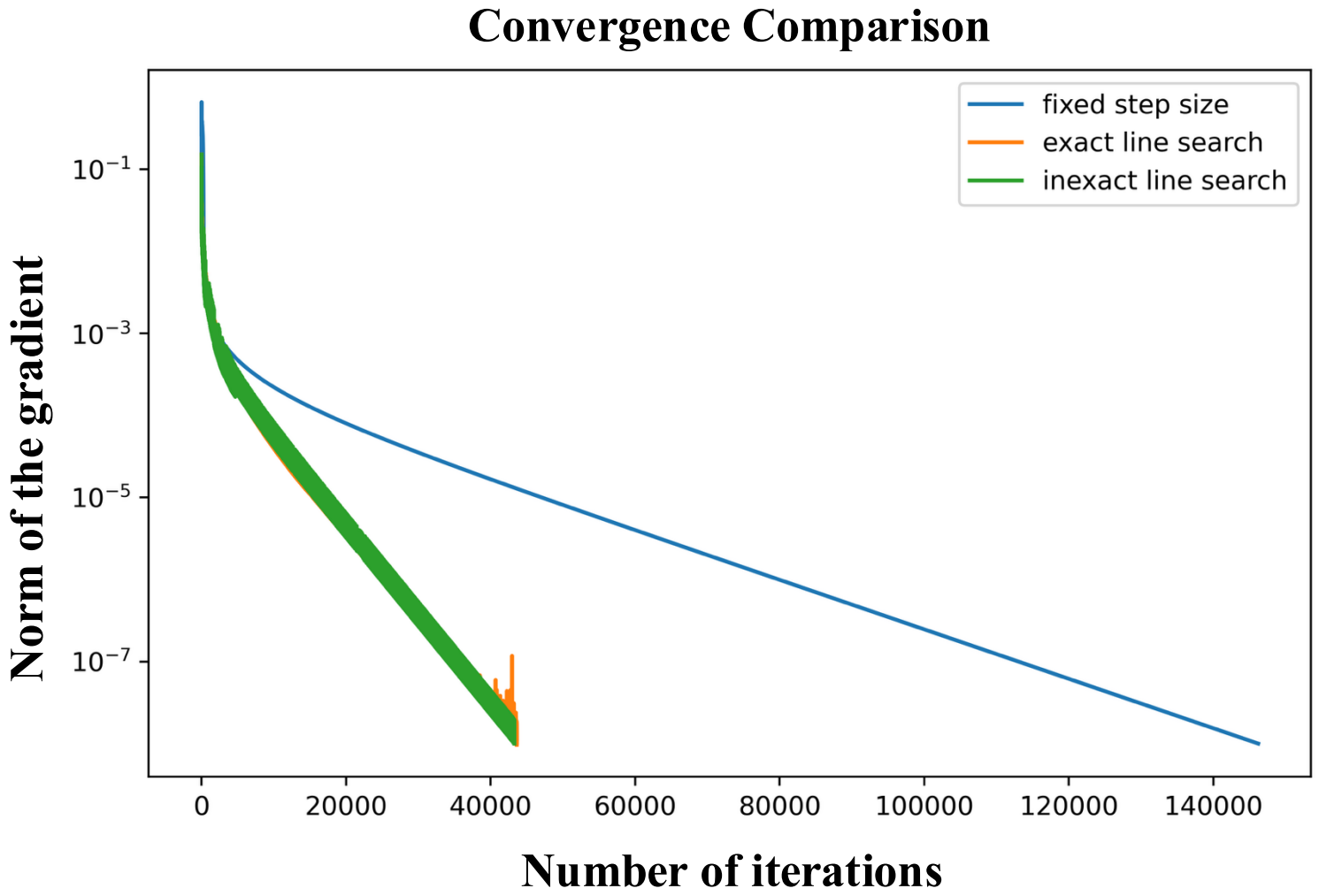

Set $\alpha$ to be sufficiently large

While $g(\alpha) > f(x^k) - \alpha \cdot \gamma \cdot ||\nabla f(x^k)||_2^2$:

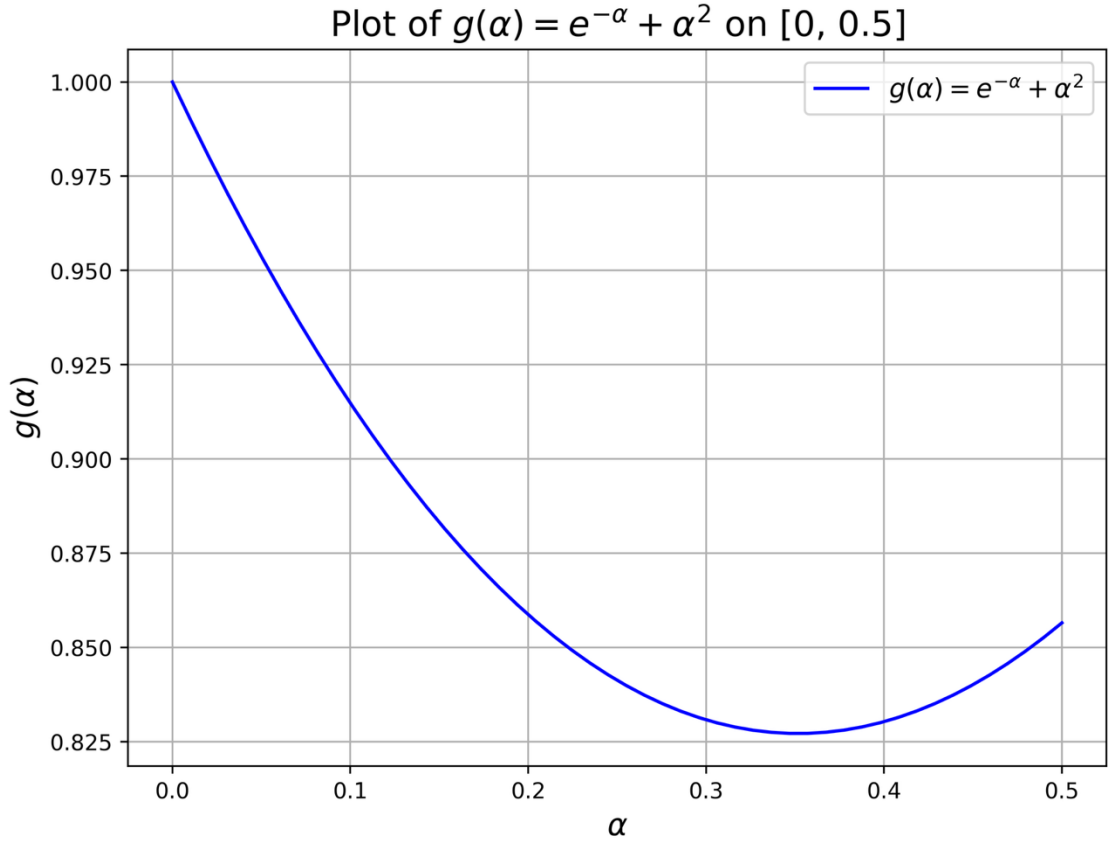
Set $\alpha = \beta \cdot \alpha$

where $\beta \in (0, 1)$

# Steepest Descent



# Newton's Method




**1. Write down the second-order approximation of $g(\alpha)$ at $\alpha = 0.1$**

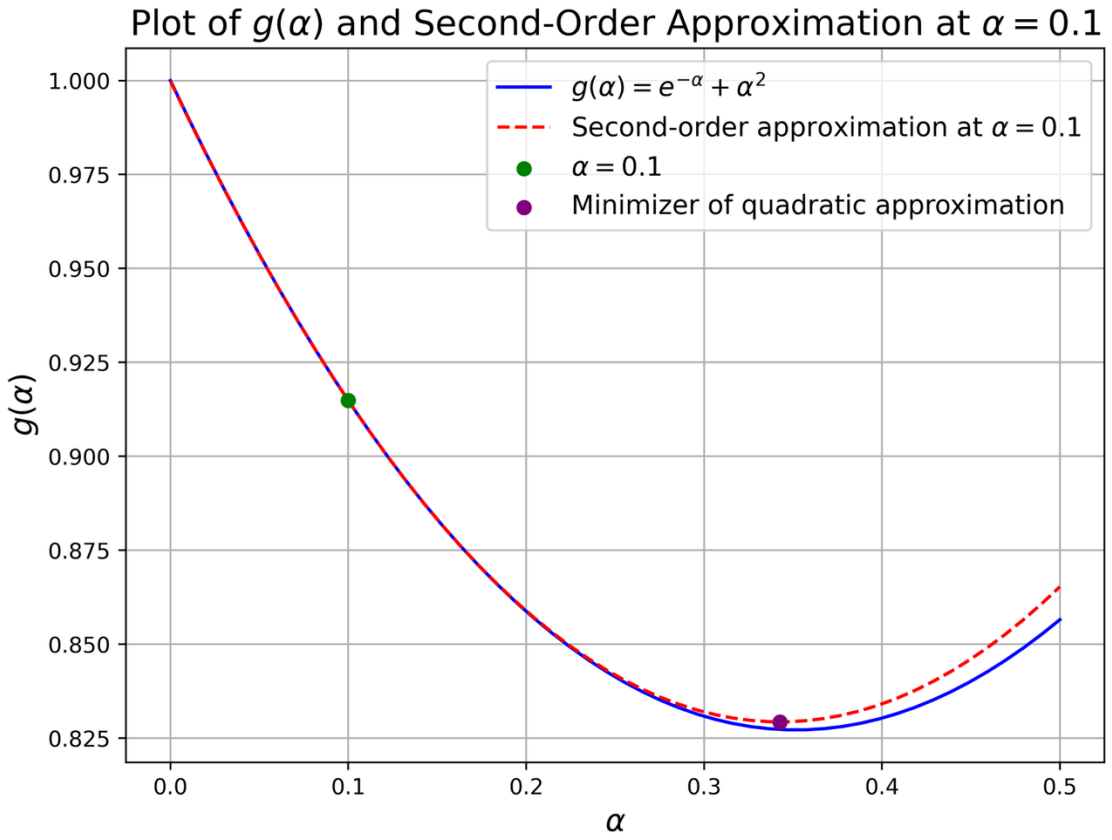
$$g(0.1) = e^{-0.1} + 0.1^2$$

$$g'(0.1) = -e^{-0.1} + 2 \cdot 0.1$$

$$g''(0.1) = e^{-0.1} + 2$$

$$g(0.1) + g'(0.1) \cdot (\alpha - 0.1) + \frac{1}{2} \cdot g''(0.1) \cdot (\alpha - 0.1)^2$$

# Newton's Method




**1. Write down the second-order approximation of $g(\alpha)$ at $\alpha = 0.1$**

$$g(0.1) = e^{-0.1} + 0.1^2$$

$$g'(0.1) = -e^{-0.1} + 2 \cdot 0.1$$

$$g''(0.1) = e^{-0.1} + 2$$

$$g(0.1) + g'(0.1) \cdot (\alpha - 0.1) + \frac{1}{2} \cdot g''(0.1) \cdot (\alpha - 0.1)^2$$

**2. Minimize the second-order approximation**

$$\alpha^* - 0.1 = -\, g''(0.1)^{-1} \cdot g'(0.1)$$

**Repeat the process at $\alpha^*$**

# Newton's Method

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &x \in \mathbb{R}^n \end{aligned}$$

**Newton's method:**

For $k = 0, 1, \ldots$

Set $x^{k+1} = \text{argmin}\, f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{2} \cdot (x - x^k)^T \cdot \nabla^2 f(x^k) \cdot (x - x^k)$

If $||\nabla f(x^k)||_2 < \epsilon$, break

$$x^{k+1} = x^k - \nabla^2 f(x^k)^{-1} \cdot \nabla f(x^k)$$

# Newton's Method

## Coding

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

```
step_size= 1
num_iterations = 1000000
w = np.zeros(X.shape[1])

for i in range(num_iterations):
    ……
```

**Try Newton's method**

$$x^{k+1} = x^k - \nabla^2 f(x^k)^{-1} \cdot \nabla f(x^k)$$

Gradient: $X^{\mathrm{T}}(p - y)$

Hessian $X^{\mathrm{T}}\mathrm{diag}(p \circ (1 - p))X$

# Newton's Method

## Coding

```
import numpy as np

np.random.seed(0)

N = 1000
GPA = np.random.normal(80, 10, N).clip(30, 100) / 100
PAPER = np.random.choice([0, 1], size=N, p=[0.8, 0.2])
One = np.ones(N)
X = np.column_stack((GPA, PAPER, One))
weight_real = np.array([100, 10, -85])
z = X @ weight_real
p = 1 / (1 + np.exp(-z))
y = np.random.binomial(1, p)
```

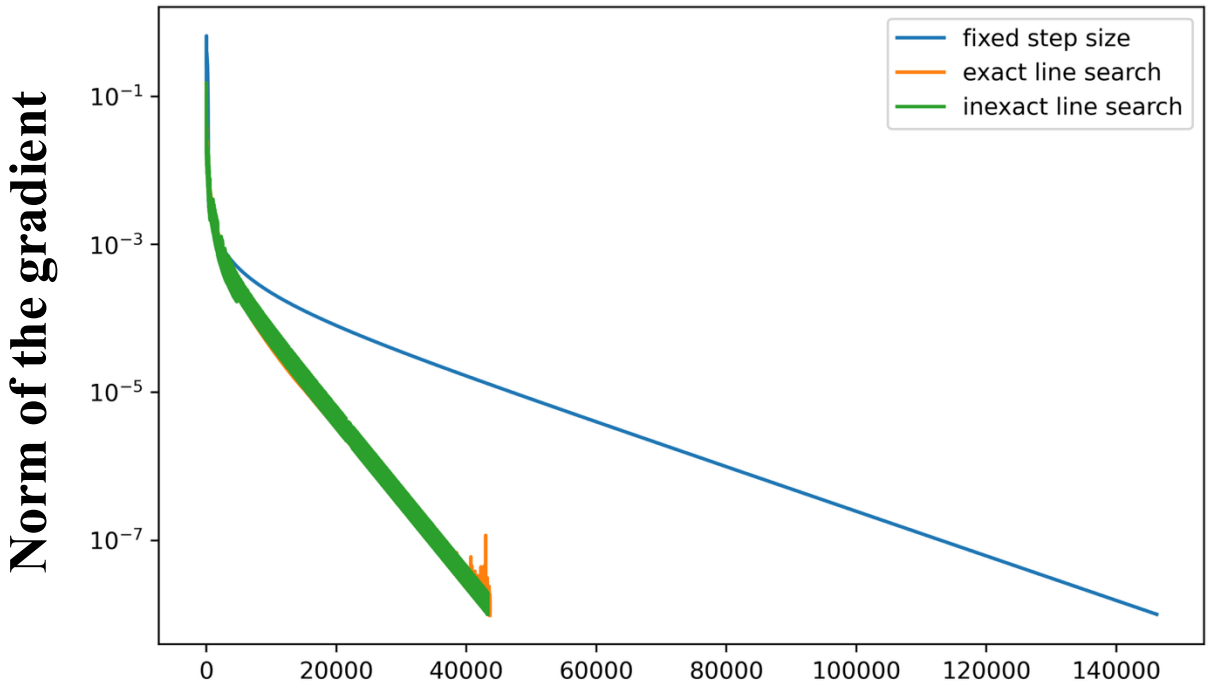
```
step_size= 1
num_iterations = 1000000
w = np.zeros(X.shape[1])

for i in range(num_iterations):
    z = X @ w
    y_pred = 1 / (1 + np.exp(-z))

    gradient = X.T @ (y_pred - y)
    D = np.diag(y_pred * (1 - y_pred))
    H = X.T @ D @ X
    H_inv = np.linalg.inv(H)
    w -= H_inv @ gradient
```

# Newton's Method

## Convergence Comparison




Iteration 0: Gradient norm = 151.76568317302826
Iteration 1: Gradient norm = 41.67300966838135
Iteration 2: Gradient norm = 17.27174782734874
Iteration 3: Gradient norm = 7.5802186450459255
Iteration 4: Gradient norm = 2.9991763766206017
Iteration 5: Gradient norm = 0.822293681097574
Iteration 6: Gradient norm = 0.10800930800203702
Iteration 7: Gradient norm = 0.003367907087056753
Iteration 8: Gradient norm = 7.52212952102895e-06
Iteration 9: Gradient norm = 8.788063428786474e-11

**Newton's method is god**