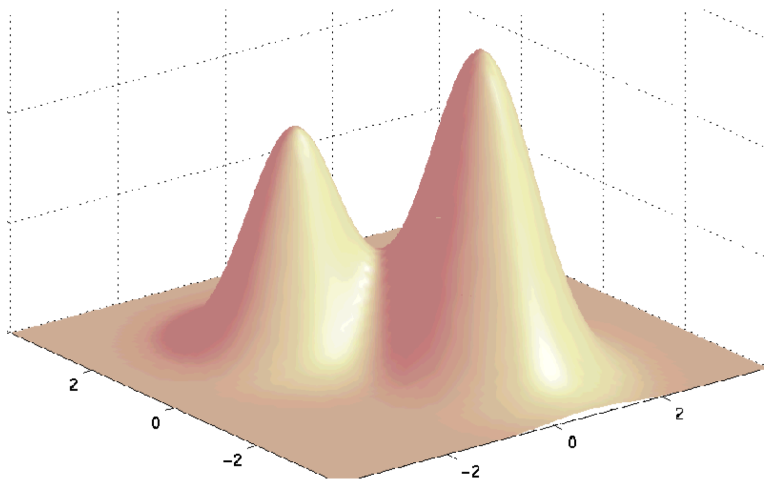
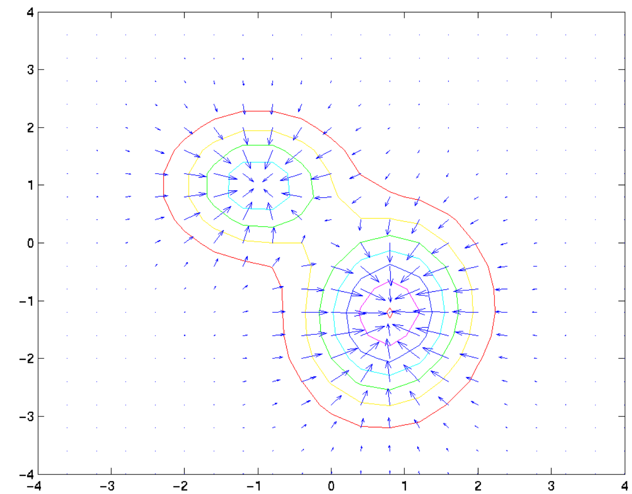
Advanced Machine Learning

Optimisation



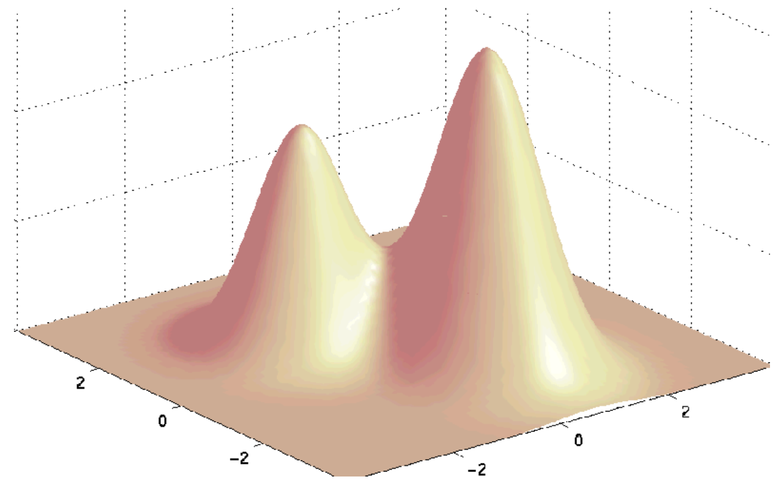


$$z = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$

Gradient descent, quadratic minima, differing length scales

Outline

1. **Motivation**
2. Gradient Descent
3. Why Gradient Descent is Difficult



ML = Optimisation

- Many learning machines can be thought of as functions of the form
$$\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$$
(or more generally $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$)

- Given an input pattern (set of features) $\boldsymbol{x}$ the learning machine makes a prediction $\hat{y}$

- We try to choose the parameters $\boldsymbol{w}$ so that the predictions are good

- In practice training a learning machine comes down to optimising some loss function

ML = Optimisation

- Many learning machines can be thought of as functions of the form
$$\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$$
(or more generally $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$)

- Given an input pattern (set of features) $\boldsymbol{x}$ the learning machine makes a prediction $\hat{y}$

- We try to choose the parameters $\boldsymbol{w}$ so that the predictions are good

- In practice training a learning machine comes down to optimising some loss function

ML = Optimisation

- Many learning machines can be thought of as functions of the form
$$\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$$
(or more generally $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$)

- Given an input pattern (set of features) $\boldsymbol{x}$ the learning machine makes a prediction $\hat{y}$

- We try to choose the parameters $\boldsymbol{w}$ so that the predictions are good

- In practice training a learning machine comes down to optimising some loss function

ML = Optimisation

- Many learning machines can be thought of as functions of the form
$$\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$$
(or more generally $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$)

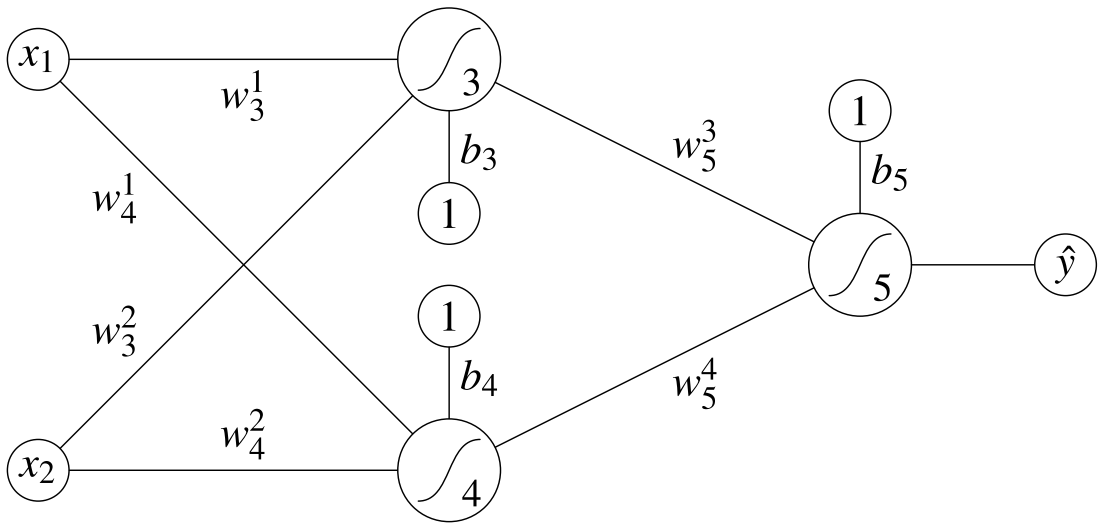
- Given an input pattern (set of features) $\boldsymbol{x}$ the learning machine makes a prediction $\hat{y}$

- We try to choose the parameters $\boldsymbol{w}$ so that the predictions are good

- In practice training a learning machine comes down to optimising some loss function

MLP

- We can depict a neural network such as an MLP by a diagram



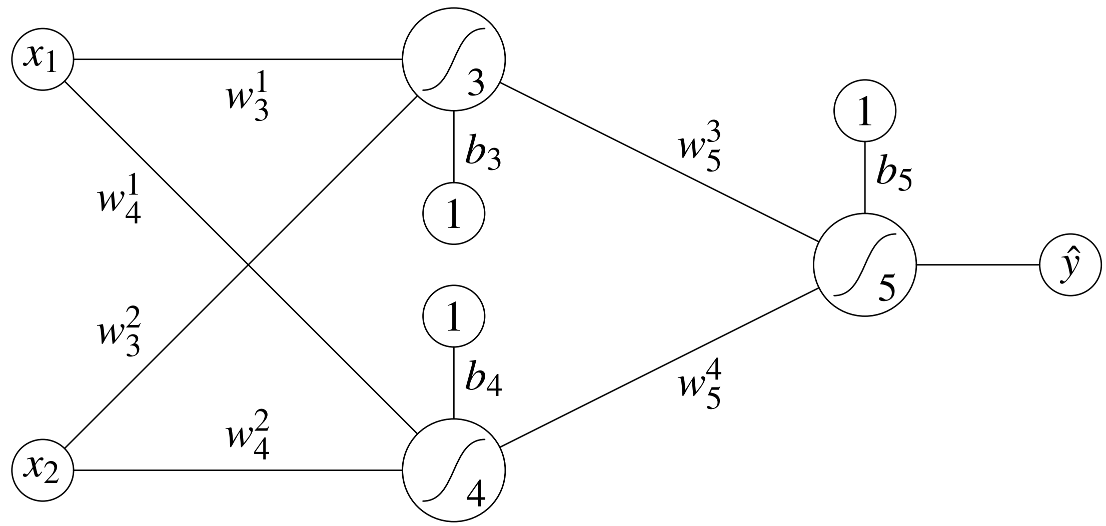
- Stands for the function $(\hat{y} = f(\boldsymbol{x}|\boldsymbol{w}))$

$$\hat{y} = g\big(w_5^3 g(w_3^1 x_1 + w_3^2 x_2 + b_3) + w_5^4 g(w_4^1 x_1 + w_4^2 x_2 + b_4) + b_5\big)$$

where, for example, $g(V) = \frac{1}{1+\mathrm{e}^{-V}}$

MLP

- We can depict a neural network such as an MLP by a diagram



- Stands for the function $(\hat{y} = f(\boldsymbol{x}|\boldsymbol{w}))$

$$\hat{y} = g\big(w_5^3 g(w_3^1 x_1 + w_3^2 x_2 + b_3) + w_5^4 g(w_4^1 x_1 + w_4^2 x_2 + b_4) + b_5\big)$$

where, for example, $g(V) = \frac{1}{1+\mathrm{e}^{-V}}$

Training

- Given a (labelled) training dataset

$$\mathcal{D} = \left\{ (\boldsymbol{x}_k, y_k) \middle| k = 1, \ldots, m \right\}$$

- We define an error or loss function that we want to minimise

$$L(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \frac{1}{m} \sum_{k=1}^{m} \big(f(\boldsymbol{x}_k|\boldsymbol{w}) - y_k \big)^2$$

- We then use the machine with the weights $\boldsymbol{w}^*$ which minimise $L(\boldsymbol{w}|\mathcal{D})$

Training

- Given a (labelled) training dataset

$$\mathcal{D} = \left\{ (\boldsymbol{x}_k, y_k) \middle| k = 1, \ldots, m \right\}$$

- We define an error or loss function that we want to minimise

$$L(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \frac{1}{m} \sum_{k=1}^{m} \big(f(\boldsymbol{x}_k|\boldsymbol{w}) - y_k \big)^2$$

- We then use the machine with the weights $\boldsymbol{w}^*$ which minimise $L(\boldsymbol{w}|\mathcal{D})$

Training

- Given a (labelled) training dataset

$$\mathcal{D} = \left\{(\boldsymbol{x}_k, y_k) \middle| k = 1, \ldots, m\right\}$$

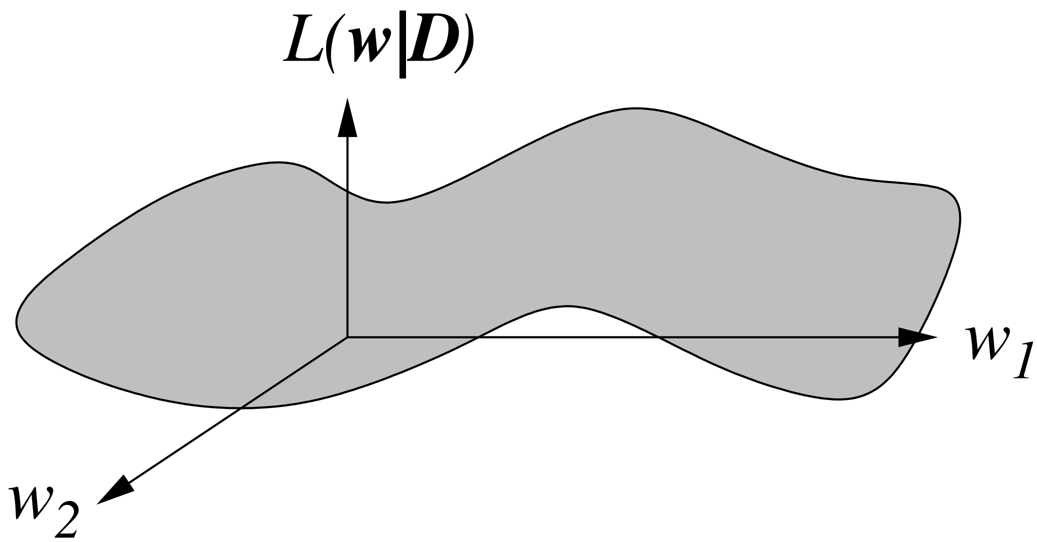
- We define an error or loss function that we want to minimise

$$L(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \frac{1}{m}\sum_{k=1}^{m}\big(f(\boldsymbol{x}_k|\boldsymbol{w}) - y_k\big)^2$$

- We then use the machine with the weights $\boldsymbol{w}^*$ which minimise $L(\boldsymbol{w}|\mathcal{D})$

Computing Gradients

- $L(\boldsymbol{w}|\mathcal{D})$ is a complex function of the weights $\boldsymbol{w}$

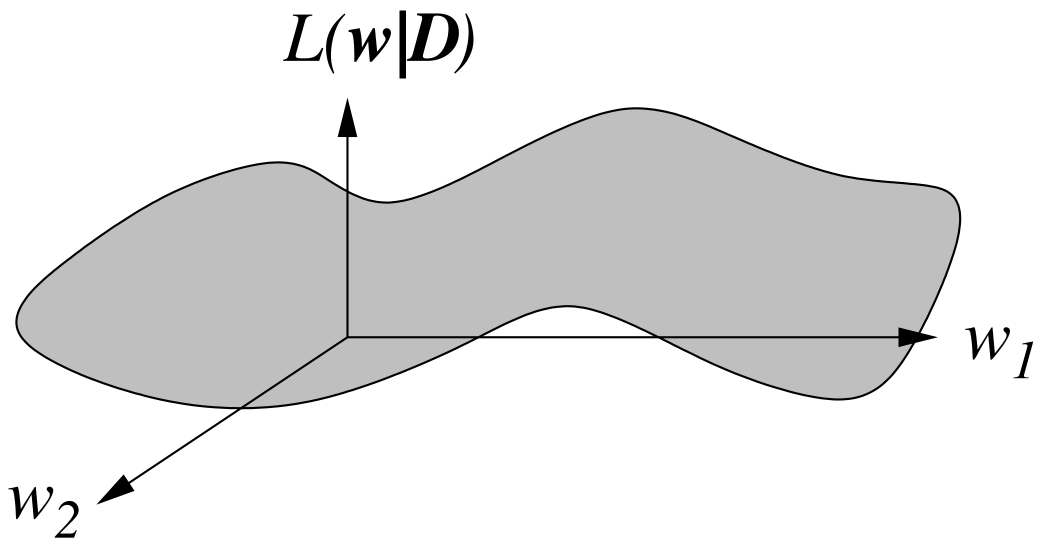


- To minimise we $L(\boldsymbol{w}|\mathcal{D})$ we compute the gradient $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

- In MLP an efficient algorithm for computing the gradient is known as back-prop

Computing Gradients

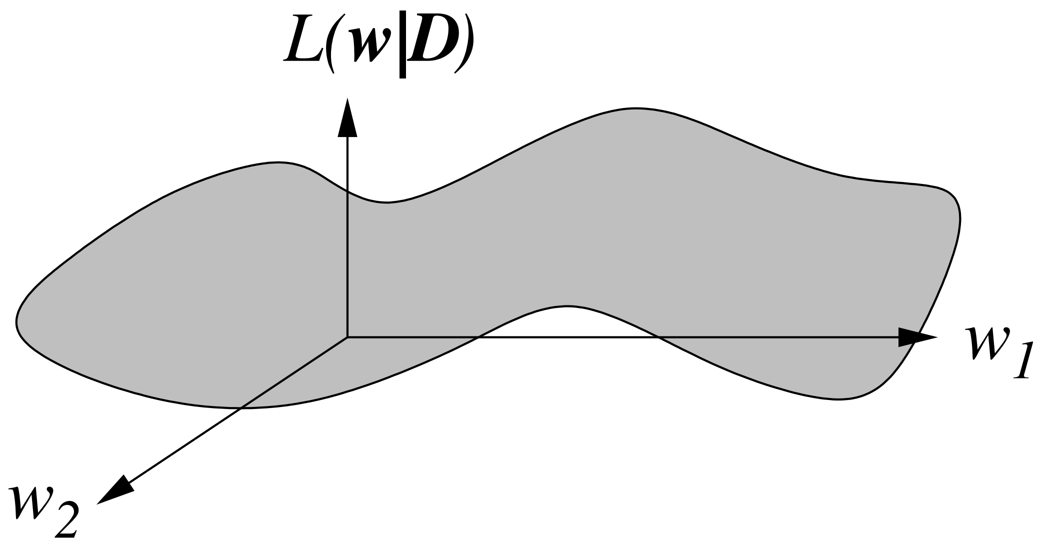
- $L(\boldsymbol{w}|\mathcal{D})$ is a complex function of the weights $\boldsymbol{w}$



- To minimise we $L(\boldsymbol{w}|\mathcal{D})$ we compute the gradient $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
- In MLP an efficient algorithm for computing the gradient is known as back-prop

Computing Gradients

- $L(\boldsymbol{w}|\mathcal{D})$ is a complex function of the weights $\boldsymbol{w}$

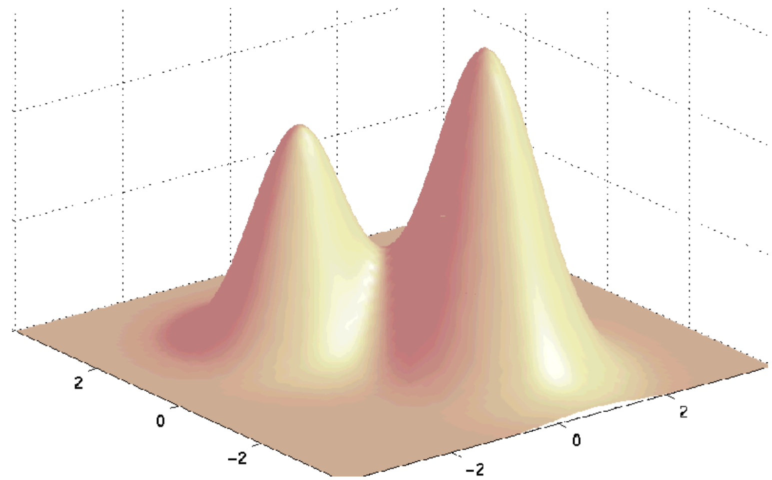


- To minimise we $L(\boldsymbol{w}|\mathcal{D})$ we compute the gradient $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

- In MLP an efficient algorithm for computing the gradient is known as back-prop

Outline

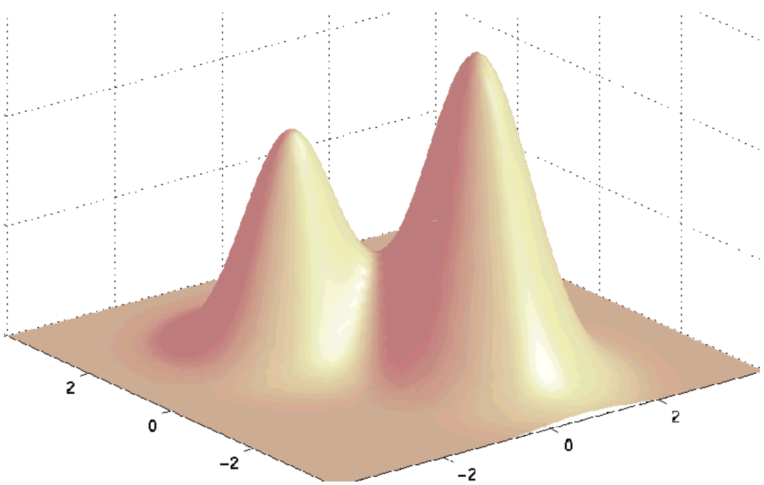
1. Motivation
2. **Gradient Descent**
3. Why Gradient Descent is Difficult

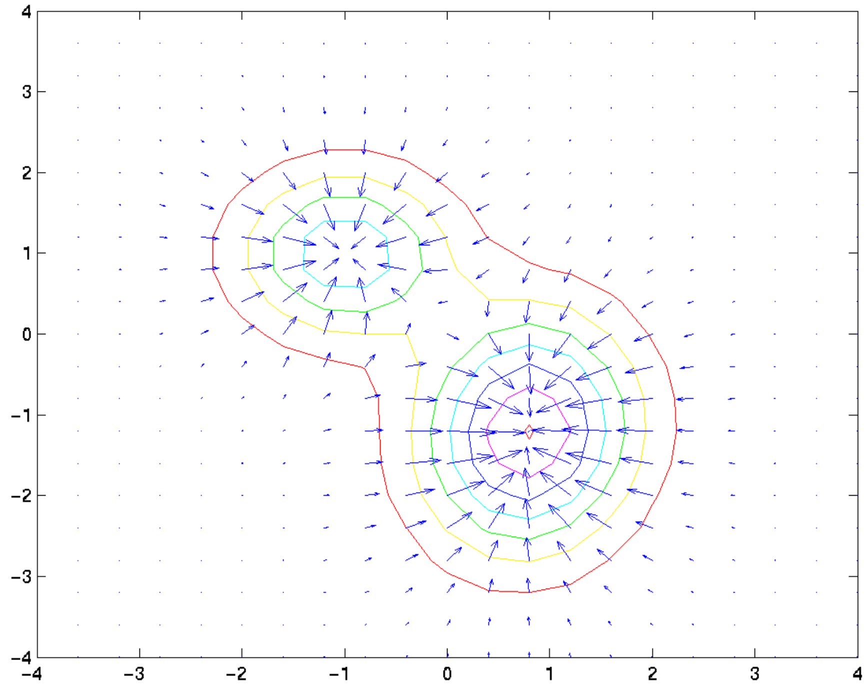


Gradient Optimisation

- A maximum or minimum occurs when $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \mathbf{0}$
- E.g.

$$L = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$

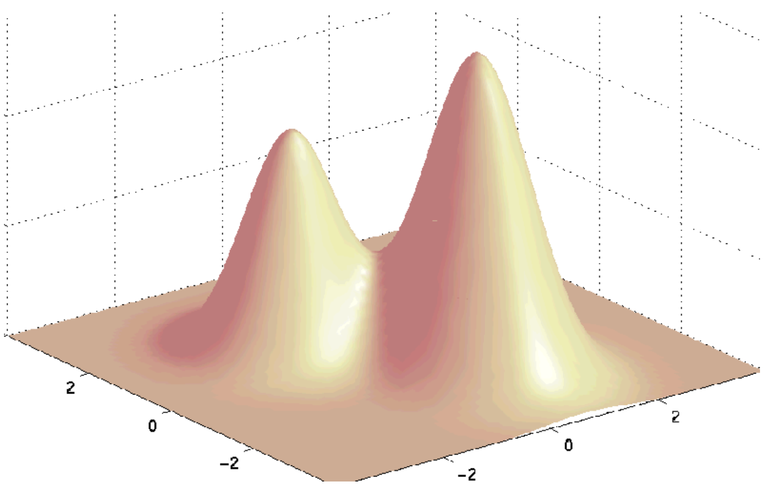


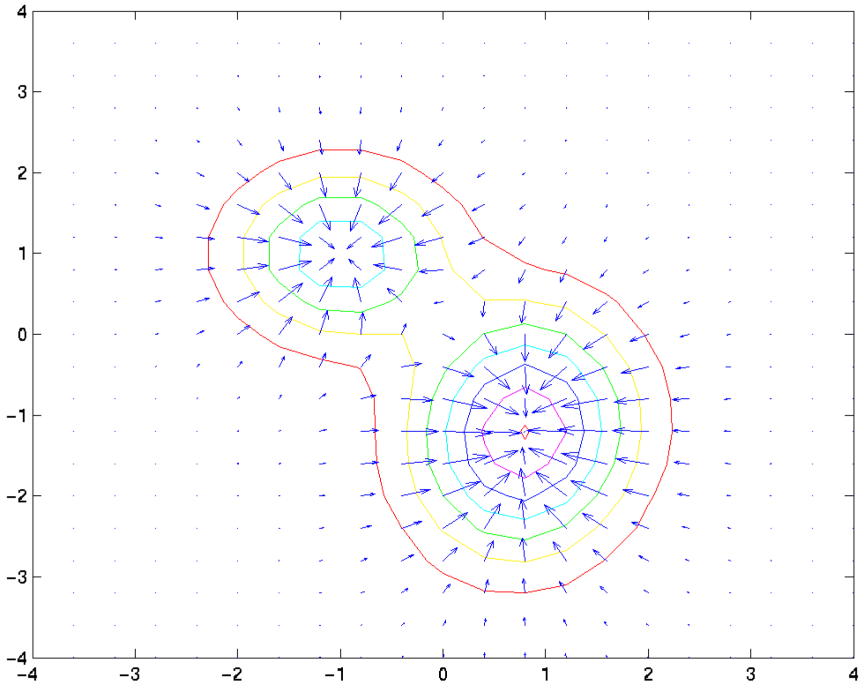


Gradient Optimisation

- A maximum or minimum occurs when $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$
- E.g.

$$L = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$





Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron

- For a non-linear functions we usually can't solve this set of simultaneous equations

- We can find a maximum or minimum **iteratively**

- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} + r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} - r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \mathbf{0}$ explicitly. E.g. the linear perceptron

- For a non-linear functions we usually can't solve this set of simultaneous equations

- We can find a maximum or minimum **iteratively**

- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} + r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} - r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron
- For a non-linear functions we usually can't solve this set of simultaneous equations
- We can find a maximum or minimum **iteratively**
- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} + r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} - r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron

- For a non-linear functions we usually can't solve this set of simultaneous equations

- We can find a maximum or minimum **iteratively**

- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} + r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} - r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron

- For a non-linear functions we usually can't solve this set of simultaneous equations

- We can find a maximum or minimum **iteratively**

- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} + r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} - r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron
- For a non-linear functions we usually can't solve this set of simultaneous equations
- We can find a maximum or minimum **iteratively**
- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} + r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} - r\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

Gradient Descent

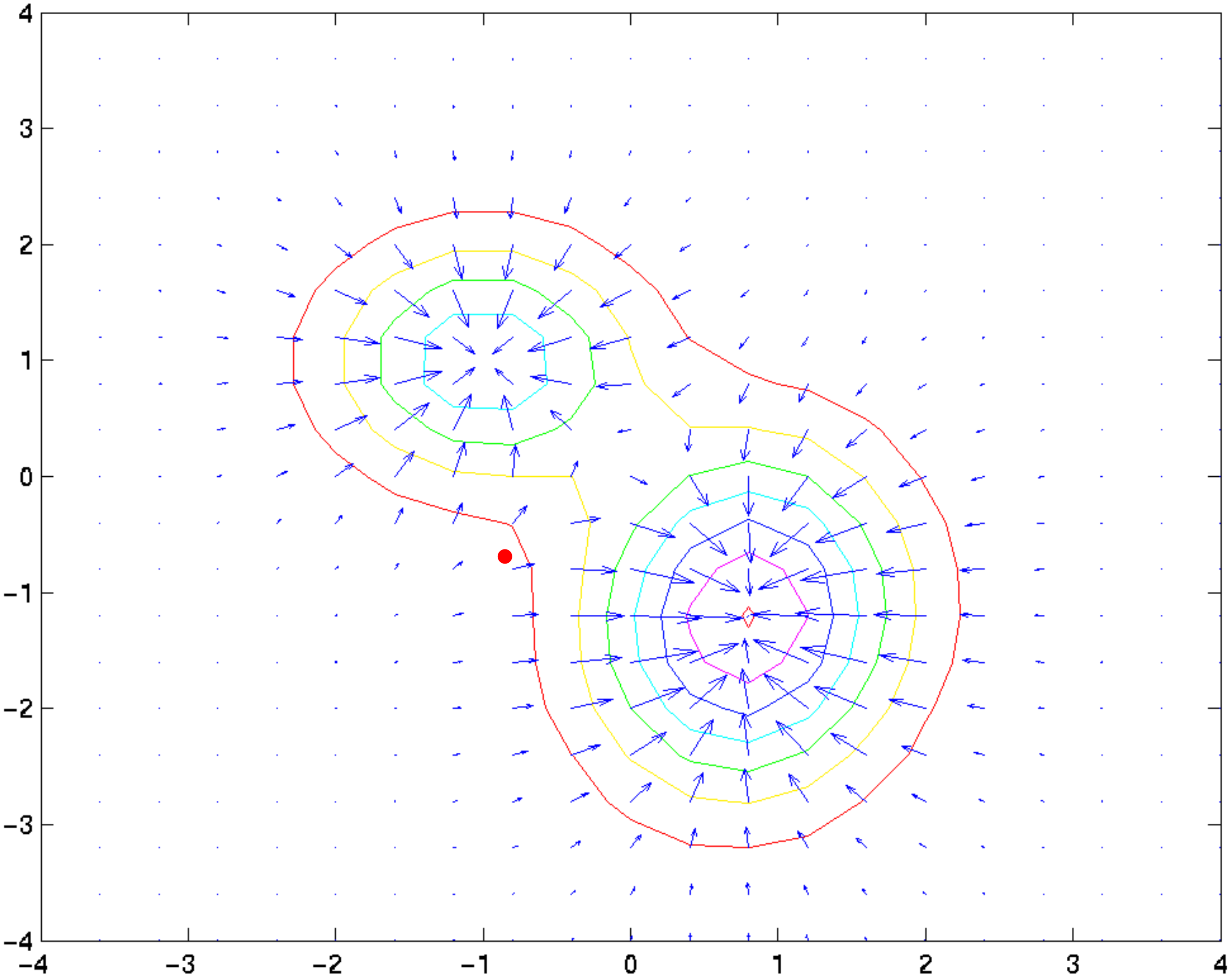
- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$ explicitly. E.g. the linear perceptron

- For a non-linear functions we usually can't solve this set of simultaneous equations

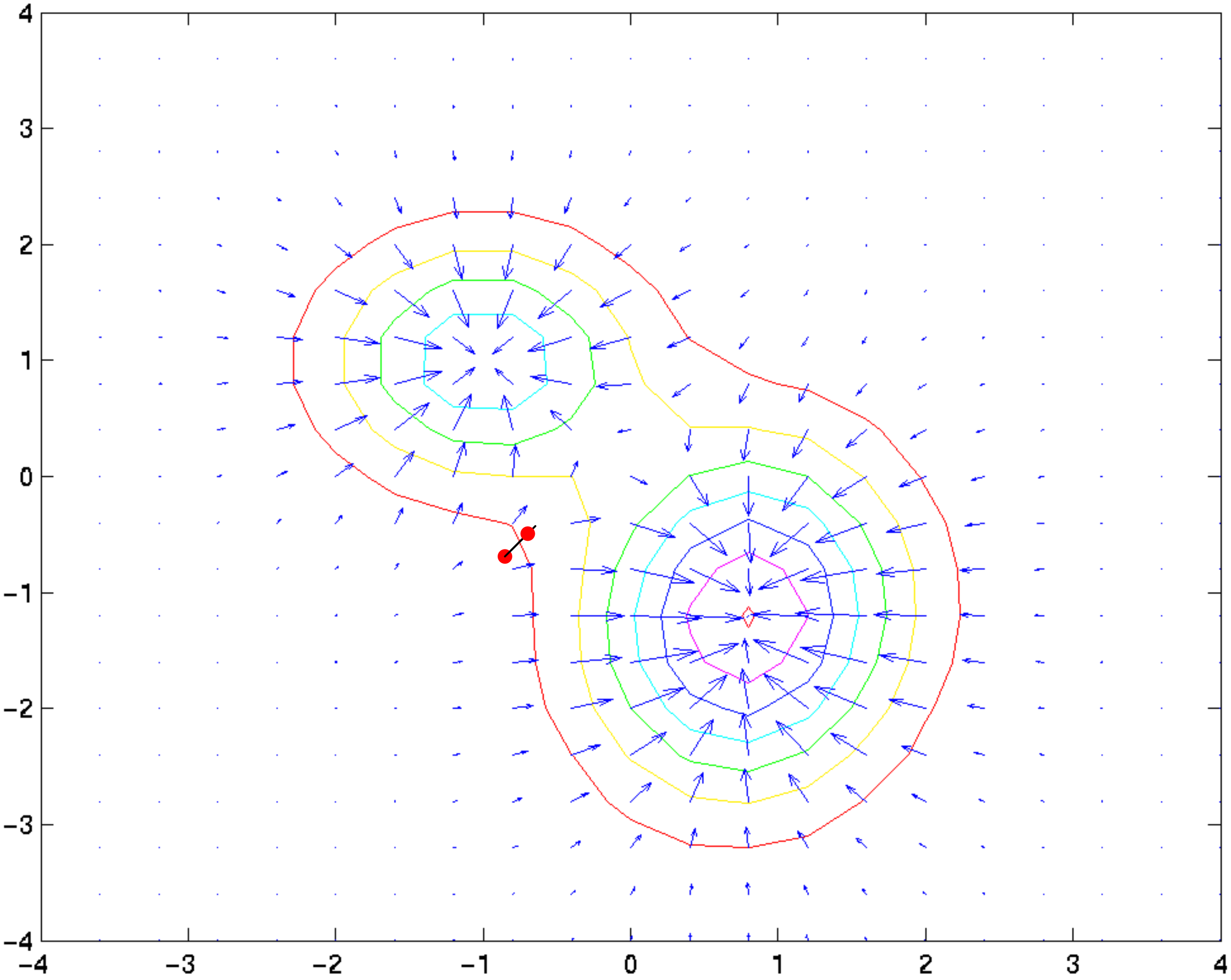
- We can find a maximum or minimum **iteratively**

- If we know the gradient then we can follow the gradient
  - ⋆ Maximisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} + r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
  - ⋆ Minimisation: $\boldsymbol{w} \to \boldsymbol{w}' = \boldsymbol{w} - r \boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$

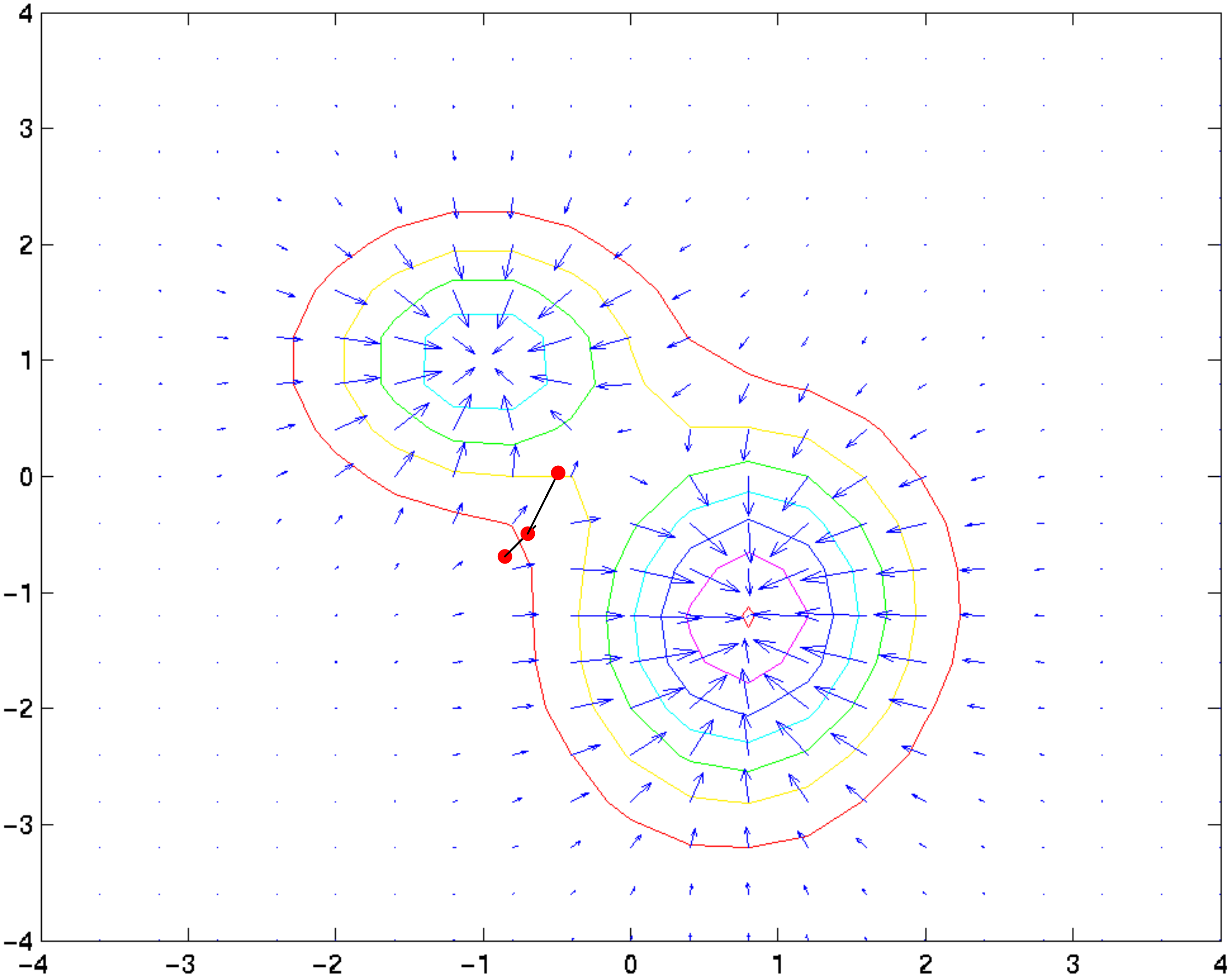
Hill-Climbing



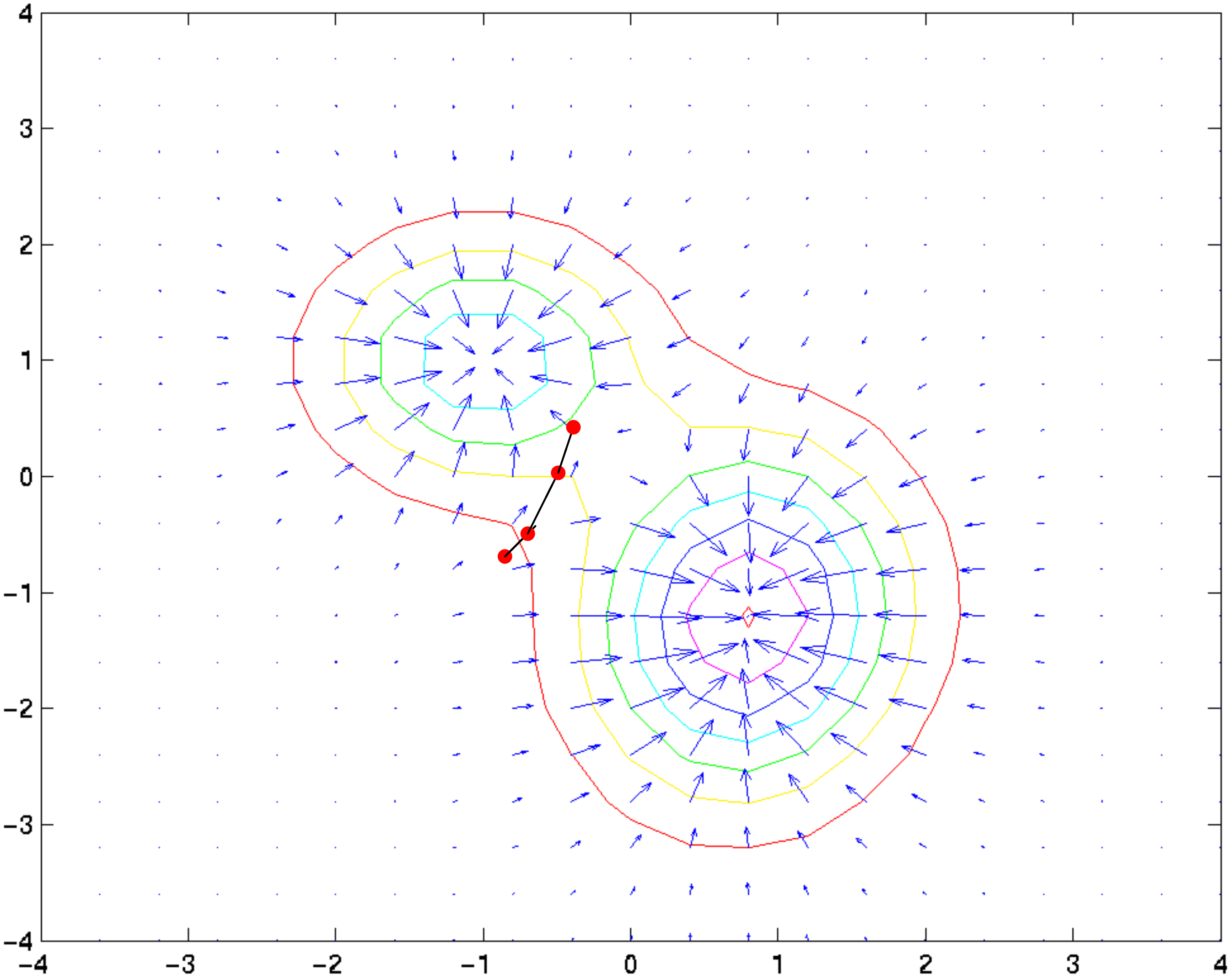
Hill-Climbing



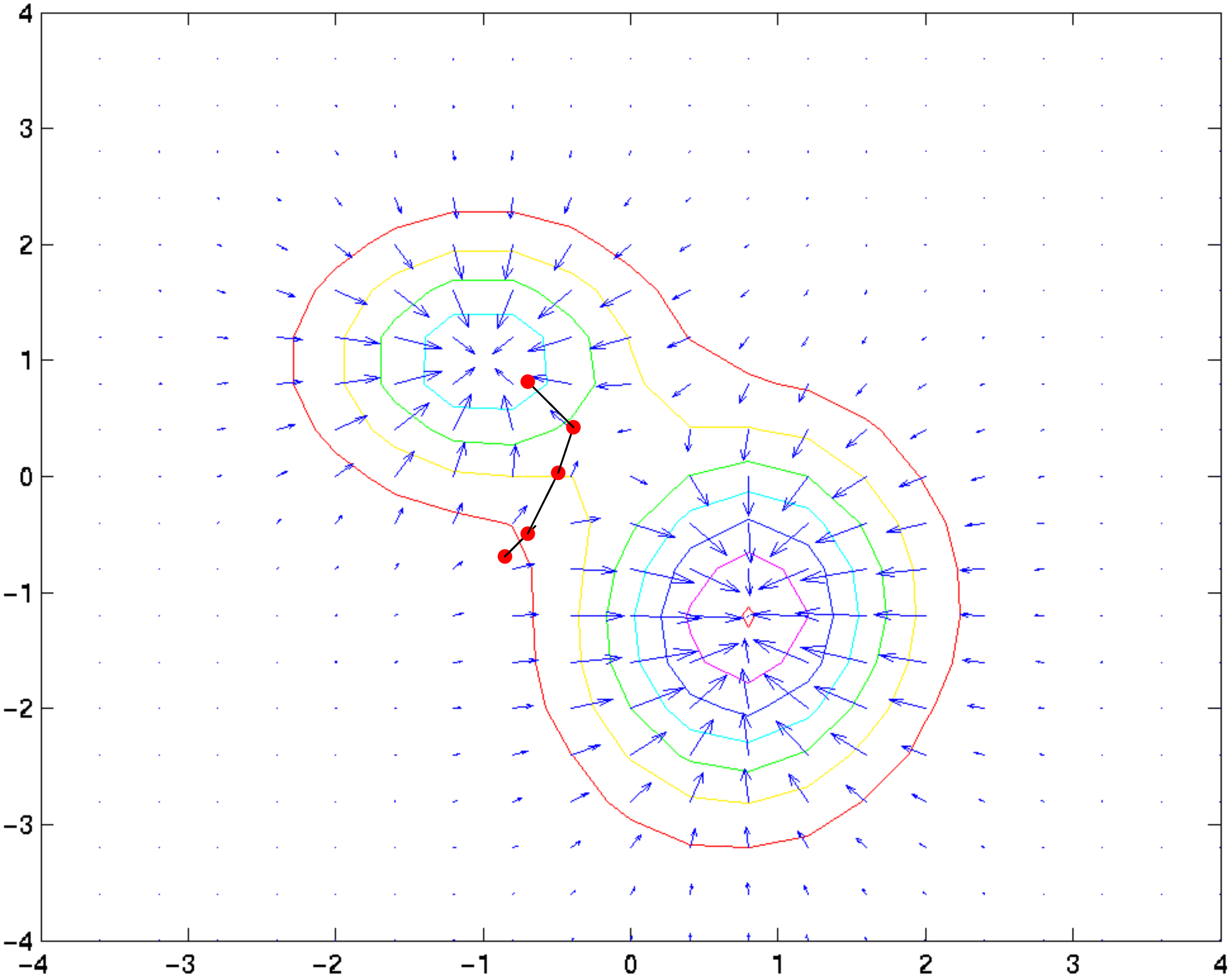
Hill-Climbing



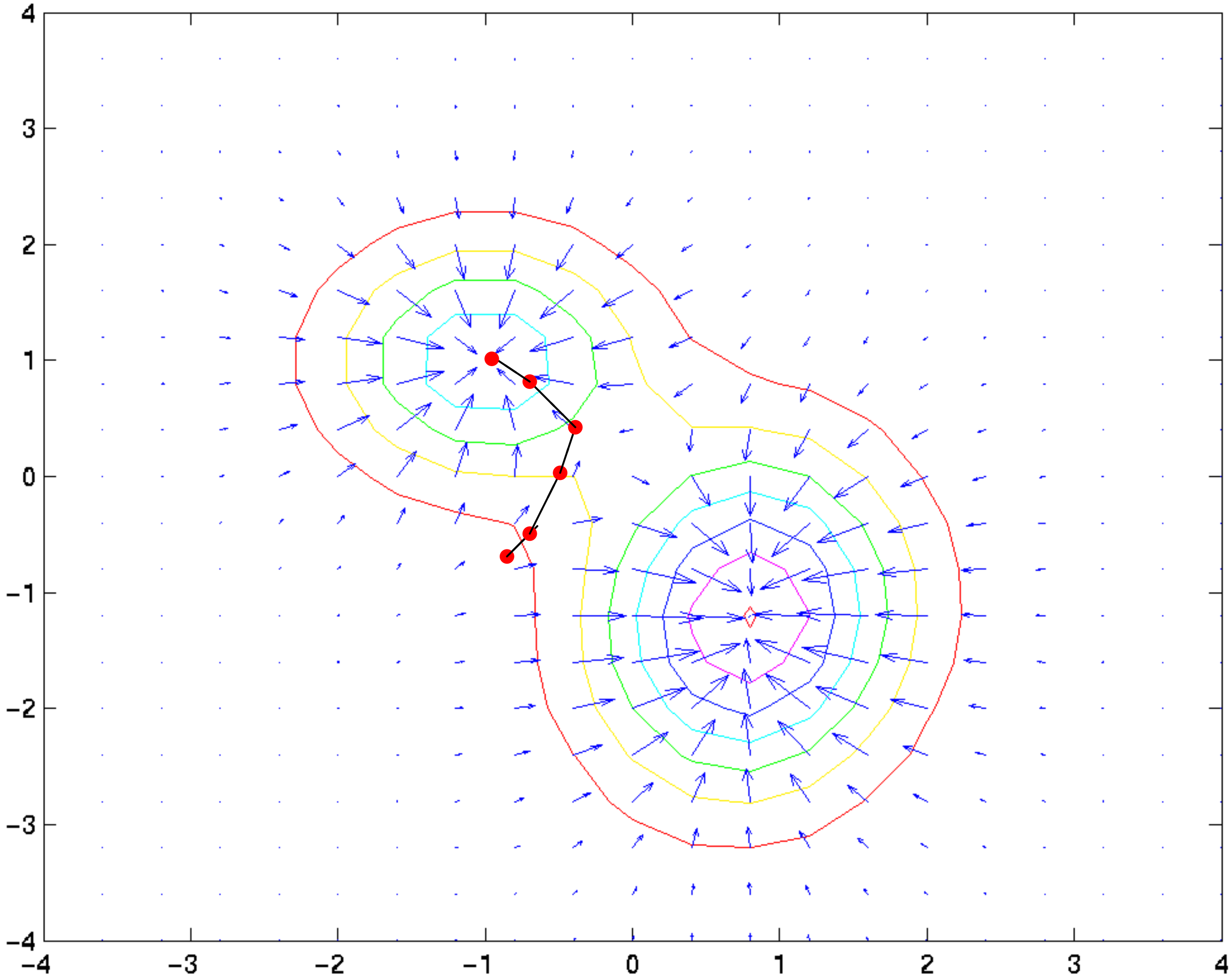
Hill-Climbing



Hill-Climbing

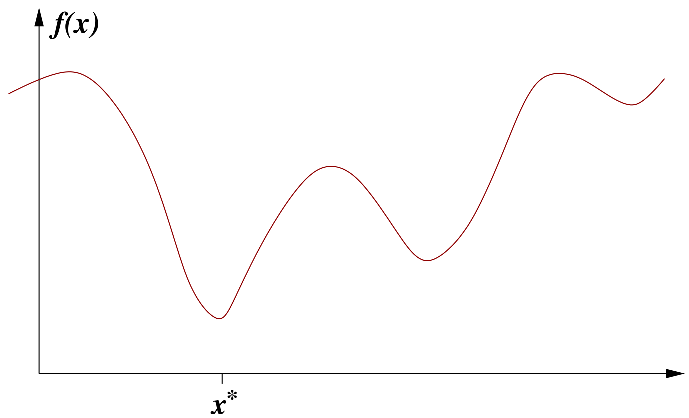


Hill-Climbing



What Goes Right

- Almost all minima are quadratic (Morse's theorem)



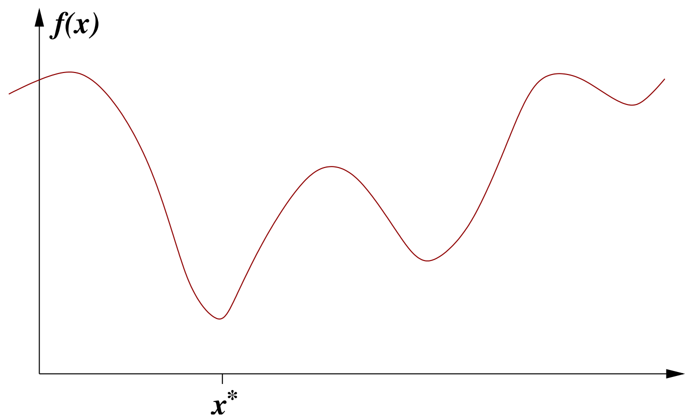
- Taylor expanding around a minimum x^*

$$\begin{aligned} f(x) &= f(x^*) + (x - x^*) f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \cdots \\ &= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \cdots \end{aligned}$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

What Goes Right

- Almost all minima are quadratic (Morse's theorem)



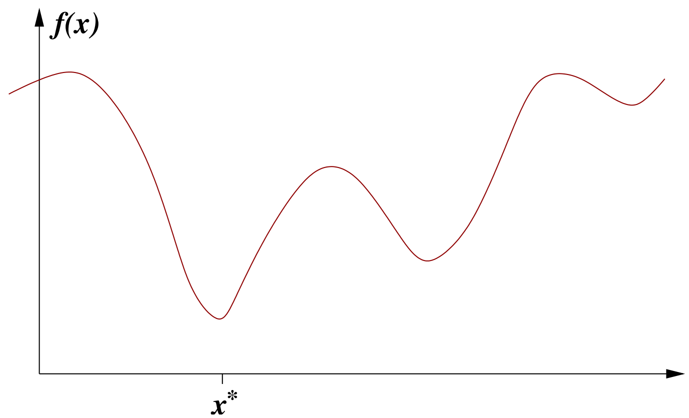
- Taylor expanding around a minimum x^*

$$\begin{aligned} f(x) &= f(x^*) + (x - x^*)\, f'(x^*) + \frac{1}{2}(x - x^*)^2\, f''(x^*) + \cdots \\ &= f(x^*) + \frac{1}{2}(x - x^*)^2\, f''(x^*) + \frac{1}{3!}(x - x^*)^3\, f'''(x^*) + \cdots \end{aligned}$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

What Goes Right

- Almost all minima are quadratic (Morse's theorem)



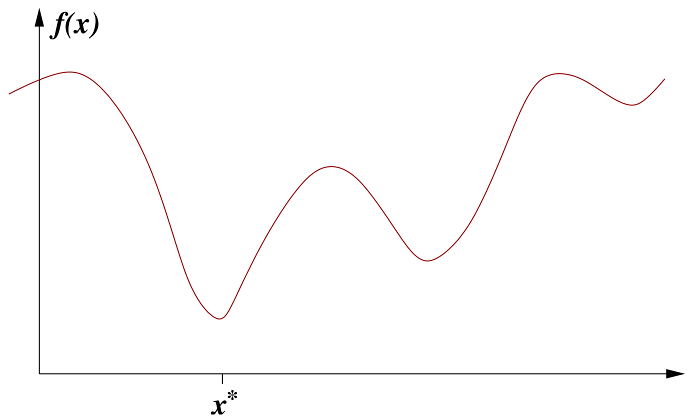
- Taylor expanding around a minimum x^*

$$\begin{aligned} f(x) &= f(x^*) + (x - x^*) f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \cdots \\ &= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \cdots \end{aligned}$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

What Goes Right

- Almost all minima are quadratic (Morse's theorem)



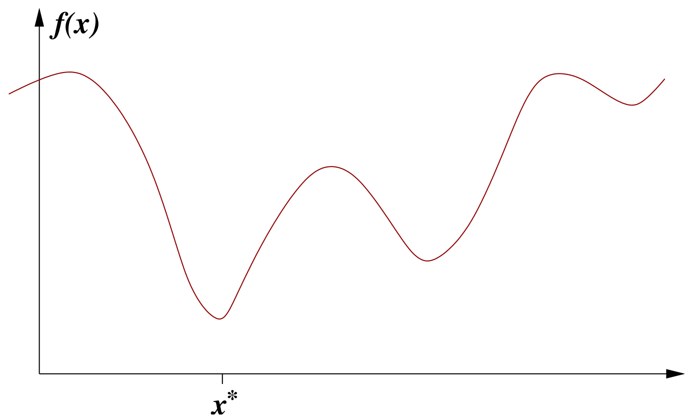
- Taylor expanding around a minimum x^*

$$
\begin{aligned}
f(x) &= f(x^*) + (x - x^*) f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \cdots \\
&= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \cdots
\end{aligned}
$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

What Goes Right

- Almost all minima are quadratic (Morse's theorem)



- Taylor expanding around a minimum x^*

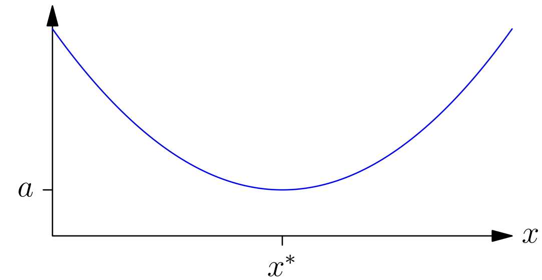
$$
\begin{aligned}
f(x) &= f(x^*) + (x - x^*) f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \cdots \\
&= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \cdots
\end{aligned}
$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x - x^*)^2$$



- then

$$f'(x) = b(x - x^*), \qquad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

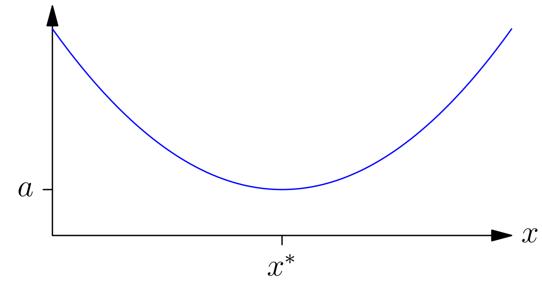
- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x - x^*)^2$$



- then

$$f'(x) = b\,(x - x^*), \qquad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

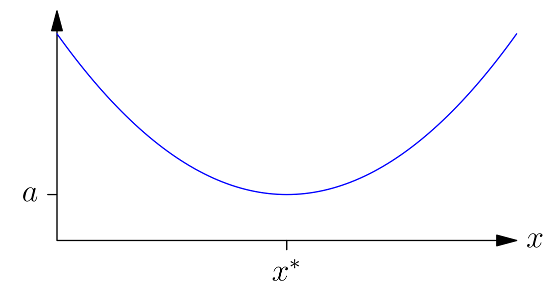
- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x - x^*)^2$$



- then

$$f'(x) = b(x - x^*), \qquad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

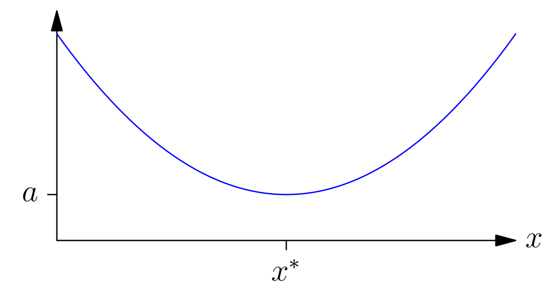
- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x - x^*)^2$$



- then

$$f'(x) = b(x - x^*), \qquad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Newton's Method

- This is Newton's methods
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions
- The Taylor expansion of a function $f(\boldsymbol{x})$ about $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \mathbf{H}(\boldsymbol{x} - \boldsymbol{x}_0) + \cdots$$

where $\mathbf{H}$ is the **Hessian** matrix with elements

$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i \partial x_j}$$

- Newton's method in high dimension is

$$\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions
- The Taylor expansion of a function $f(\boldsymbol{x})$ about $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \mathbf{H} (\boldsymbol{x} - \boldsymbol{x}_0) + \cdots$$

where $\mathbf{H}$ is the **Hessian** matrix with elements

$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i \partial x_j}$$

- Newton's method in high dimension is

$$\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions
- The Taylor expansion of a function $f(\boldsymbol{x})$ about $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \mathbf{H} (\boldsymbol{x} - \boldsymbol{x}_0) + \cdots$$

where $\mathbf{H}$ is the **Hessian** matrix with elements

$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i \partial x_j}$$

- Newton's method in high dimension is

$$\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions
- The Taylor expansion of a function $f(\boldsymbol{x})$ about $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^{\mathsf{T}} \mathbf{H} (\boldsymbol{x} - \boldsymbol{x}_0) + \cdots$$

where $\mathbf{H}$ is the **Hessian** matrix with elements

$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i \partial x_j}$$

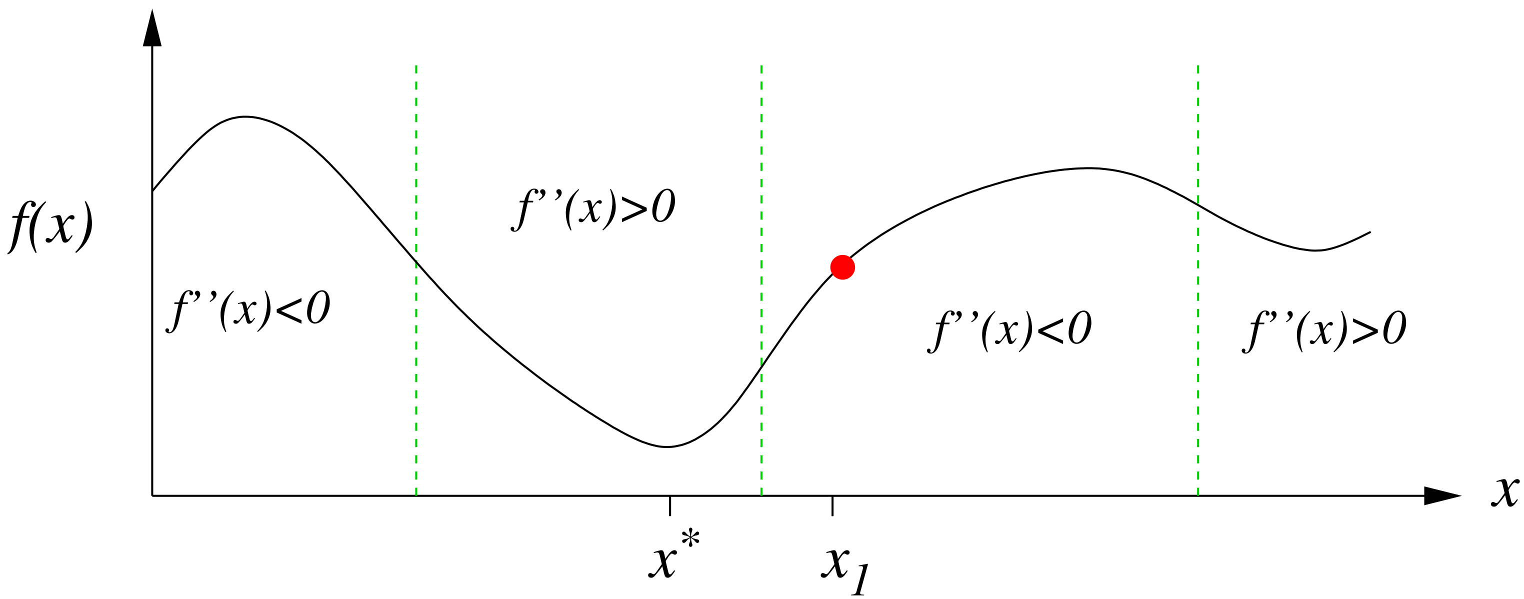
- Newton's method in high dimension is

$$\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1} \boldsymbol{\nabla} f(\boldsymbol{x})$$

Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding)

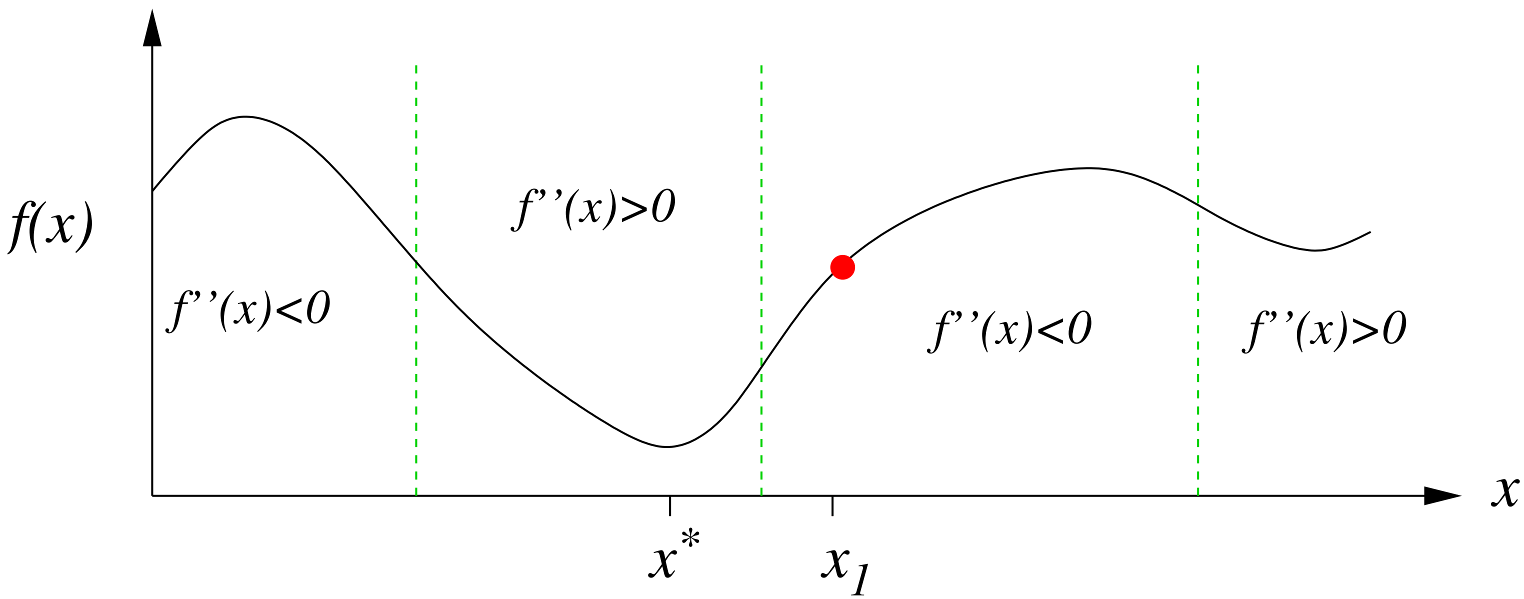
- Away from minima they can be misleading



Using the Second Derivative

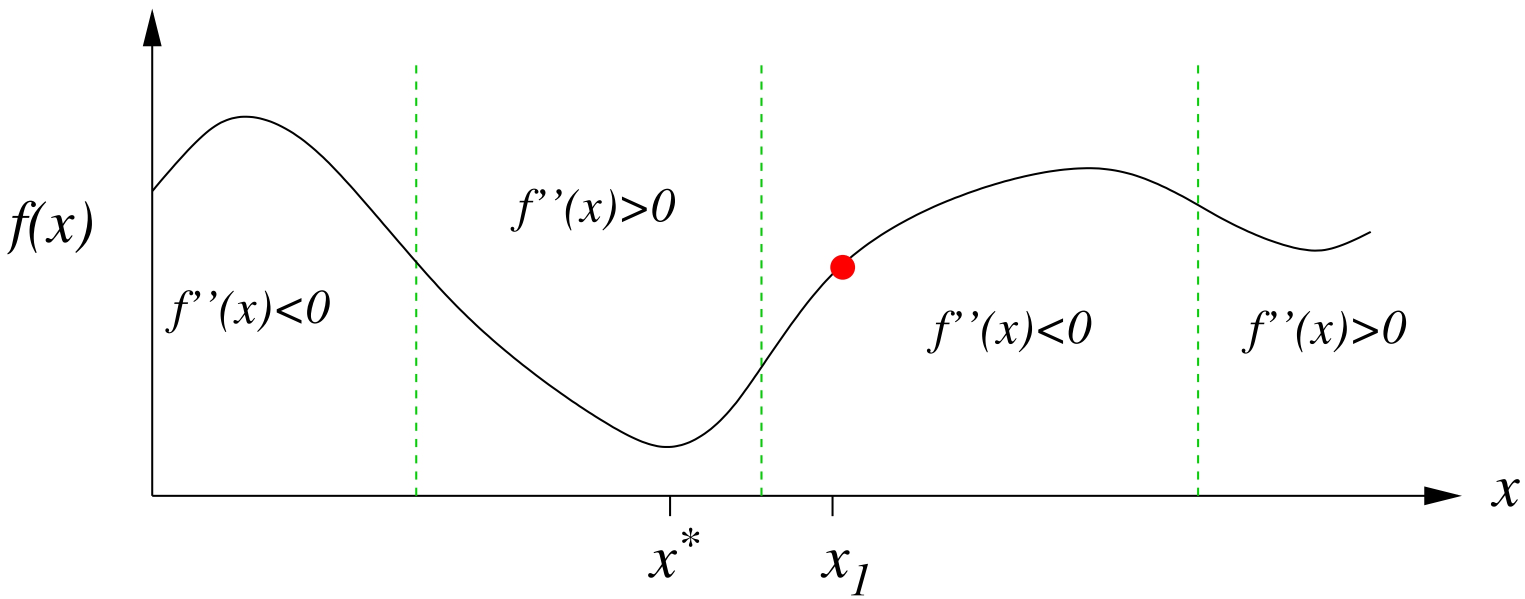
- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding)

- Away from minima they can be misleading



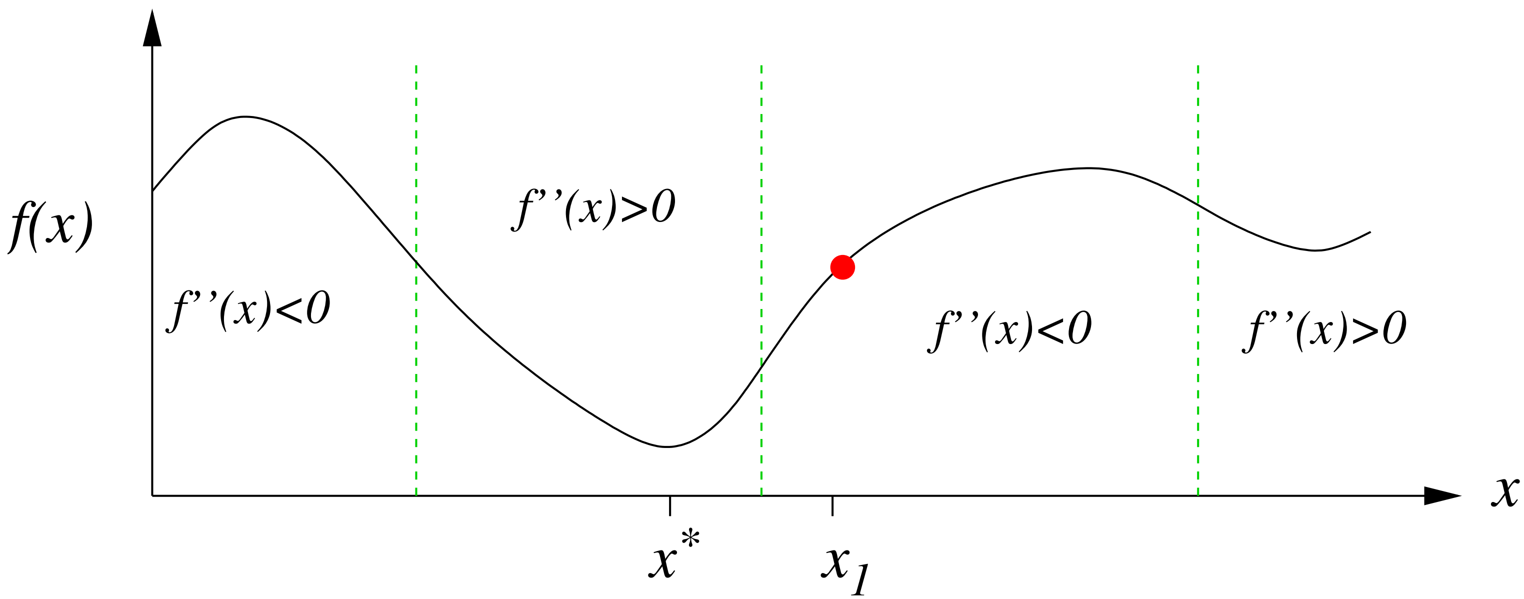
Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding)
- Away from minima they can be misleading



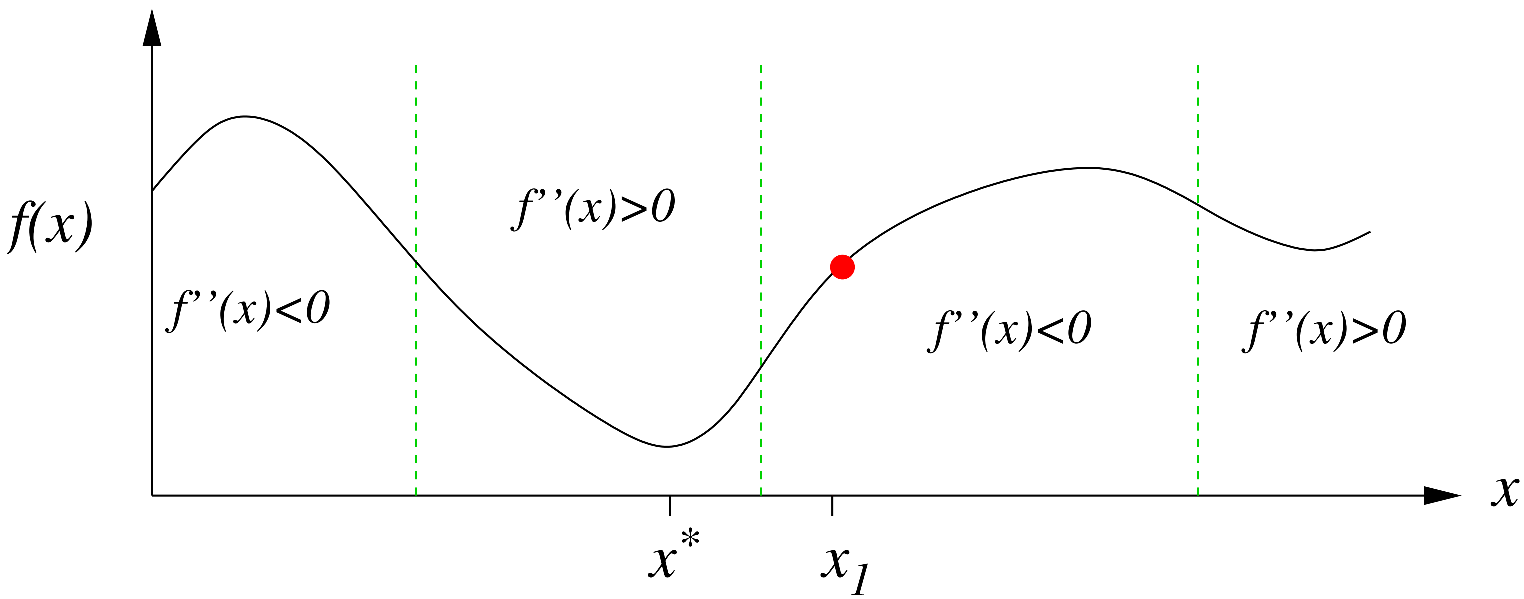
Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding)—for deep learning it is impossible even to store the Hessian
- Away from minima they can be misleading



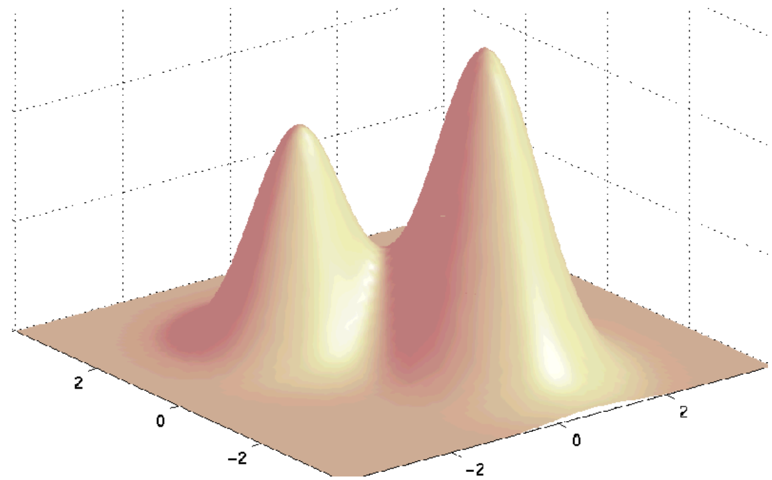
Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix
- It is time-consuming to compute (and prone to errors when coding)—for deep learning it is impossible even to store the Hessian
- Away from minima they can be misleading



Outline

1. Motivation
2. Gradient Descent
3. **Why Gradient Descent is Difficult**



Step Size

- Gradient descent

$$\boldsymbol{x}' = \boldsymbol{x} - r\,\boldsymbol{\nabla} f(\boldsymbol{x})$$

- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

Step Size

- Gradient descent

$$\boldsymbol{x}' = \boldsymbol{x} - r\,\boldsymbol{\nabla} f(\boldsymbol{x})$$

- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

Step Size

- Gradient descent

$$\boldsymbol{x}' = \boldsymbol{x} - r\,\boldsymbol{\nabla} f(\boldsymbol{x})$$

- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

Step Size

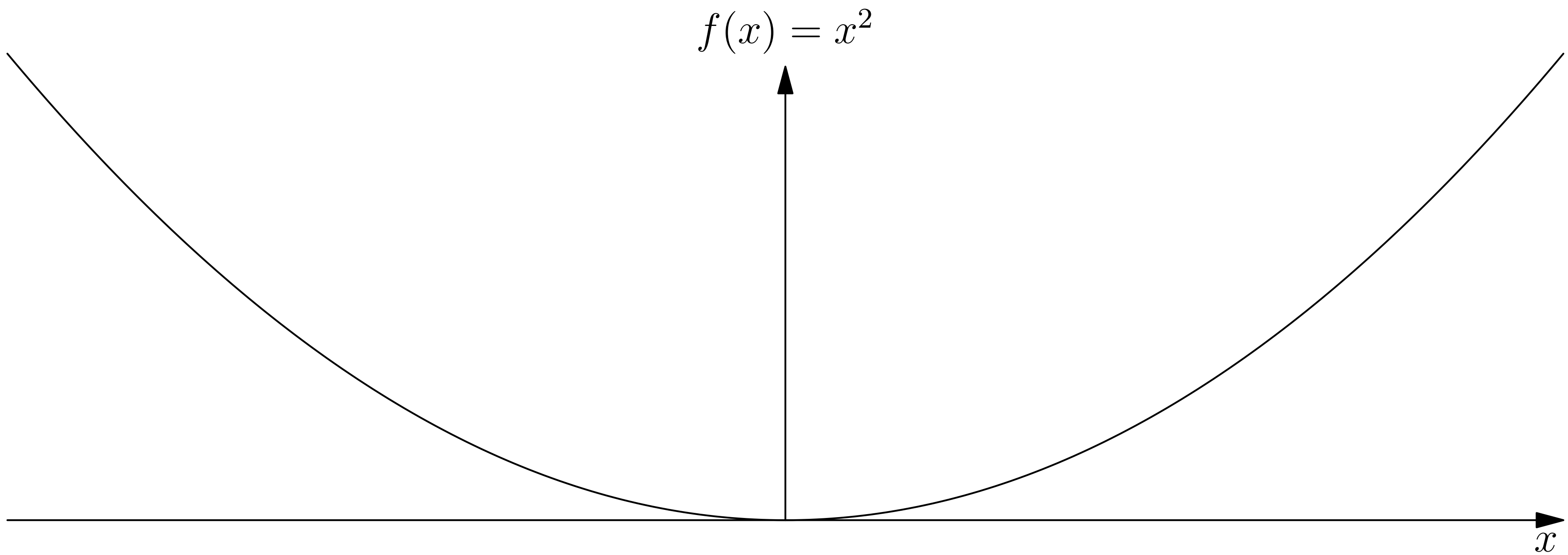
- Gradient descent

$$\boldsymbol{x}' = \boldsymbol{x} - r\,\boldsymbol{\nabla} f(\boldsymbol{x})$$

- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

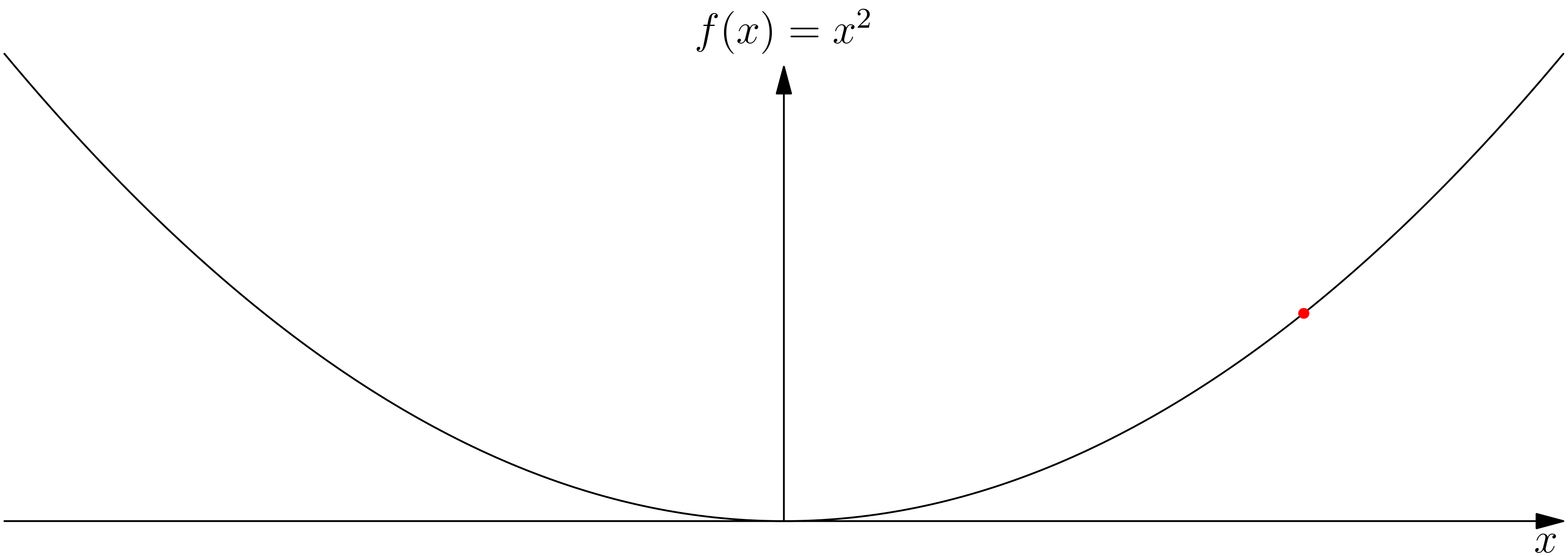
Step Size

$x \leftarrow x - r\, f'(x)$



Step Size

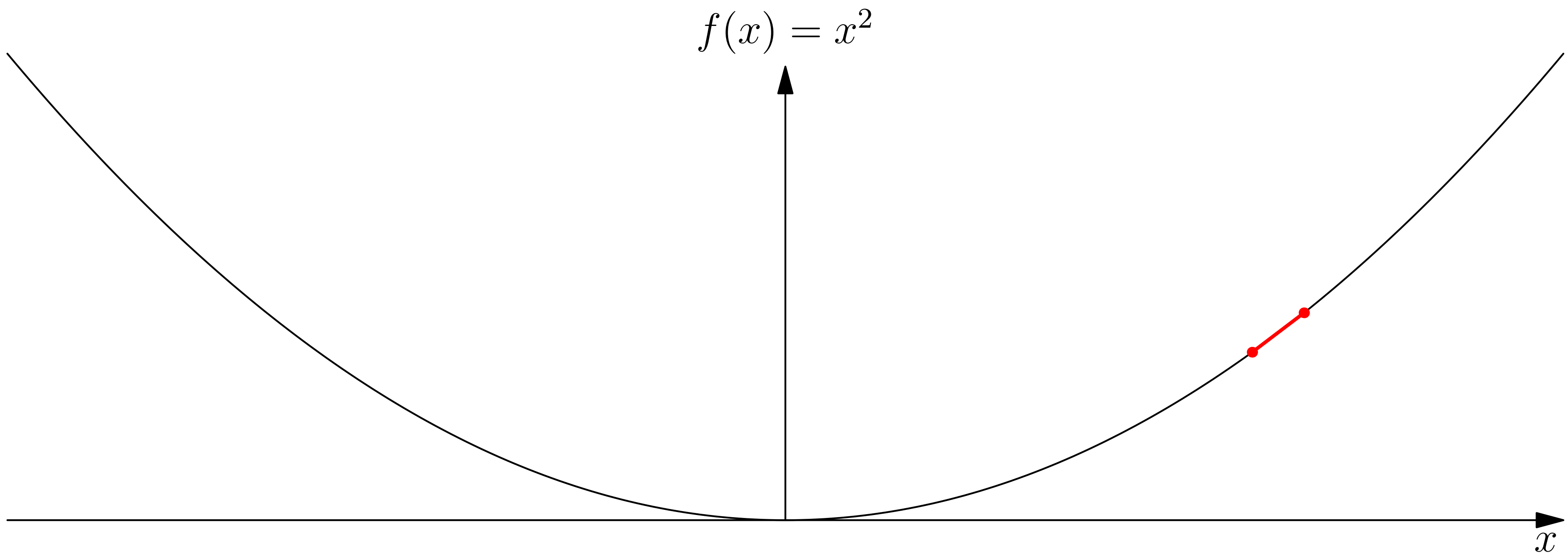
$x \leftarrow x - r\, f'(x)$



r = 0.05

Step Size

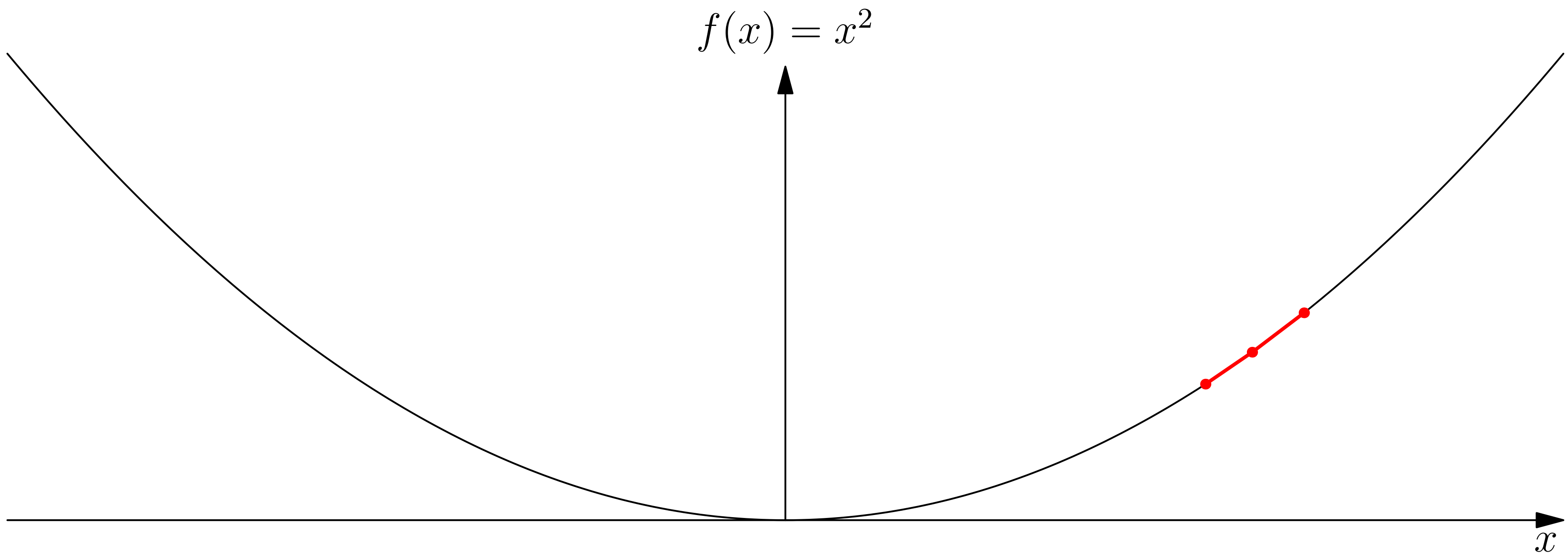
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$

Step Size

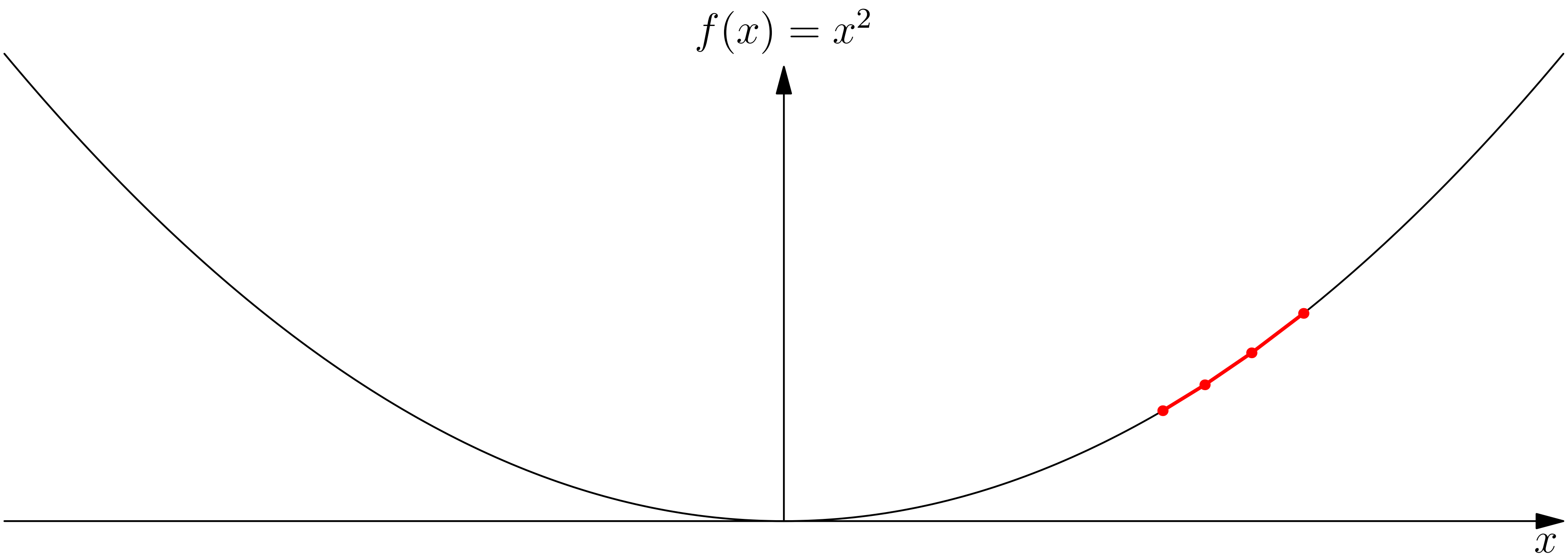
$x \leftarrow x - r\, f'(x)$



r = 0.05

Step Size

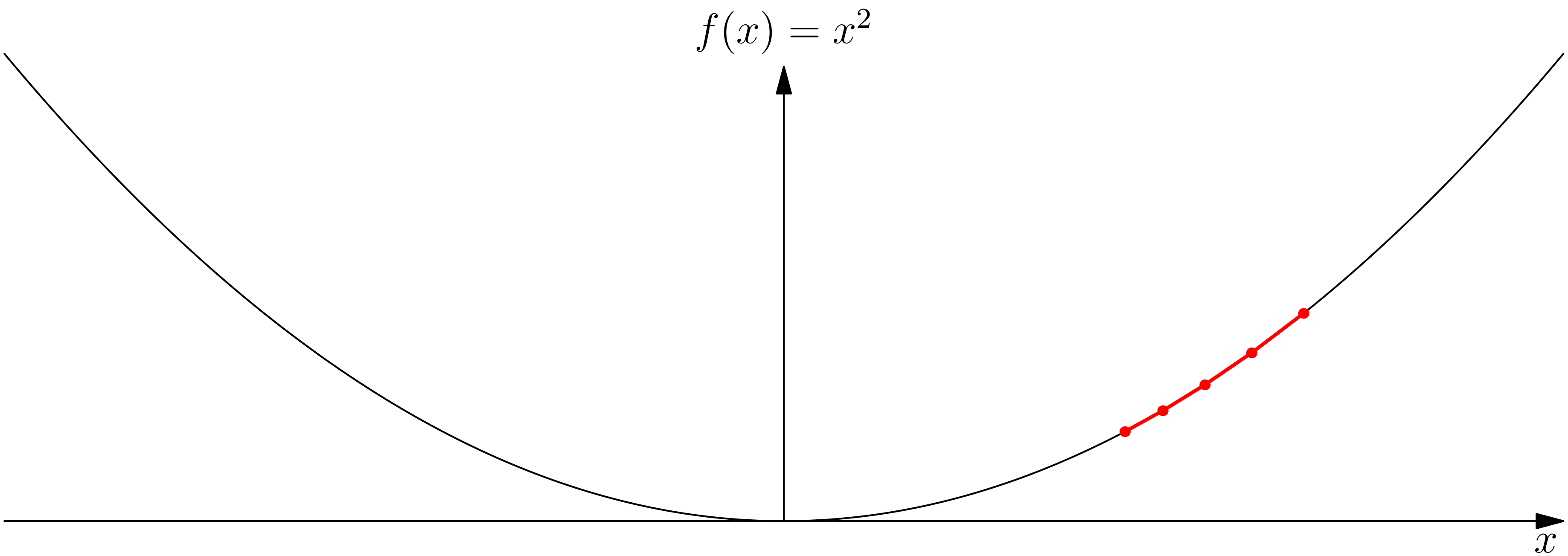
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$

Step Size

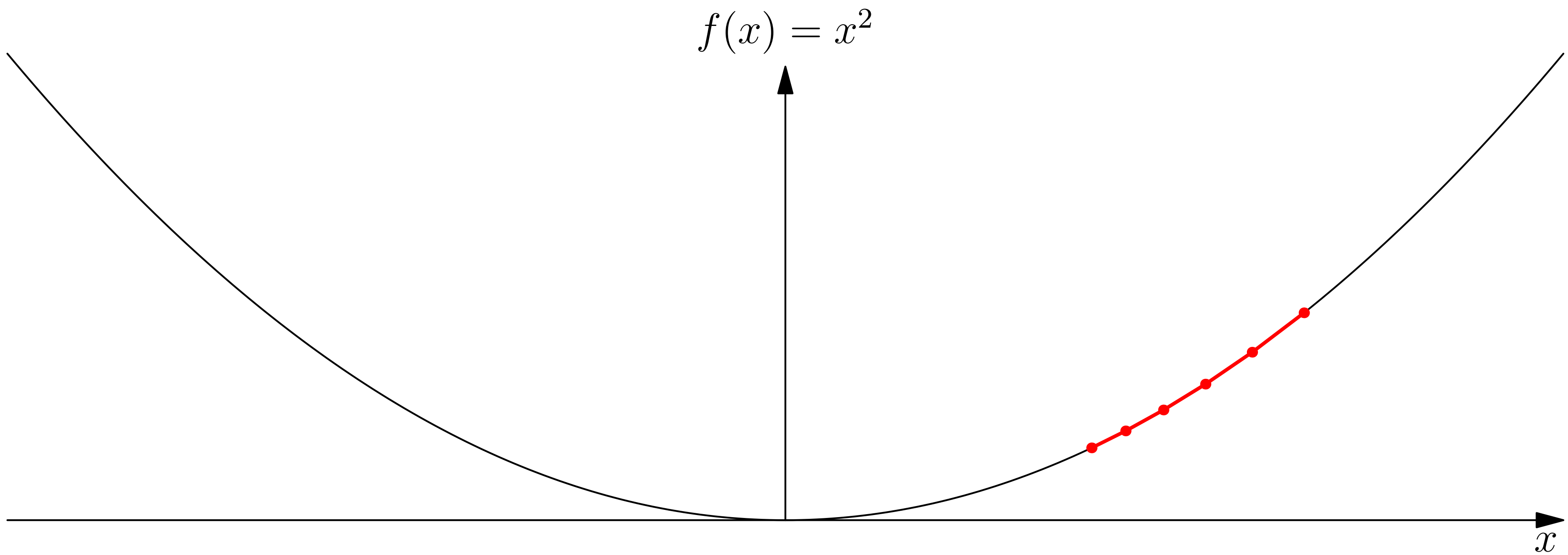
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$

Step Size

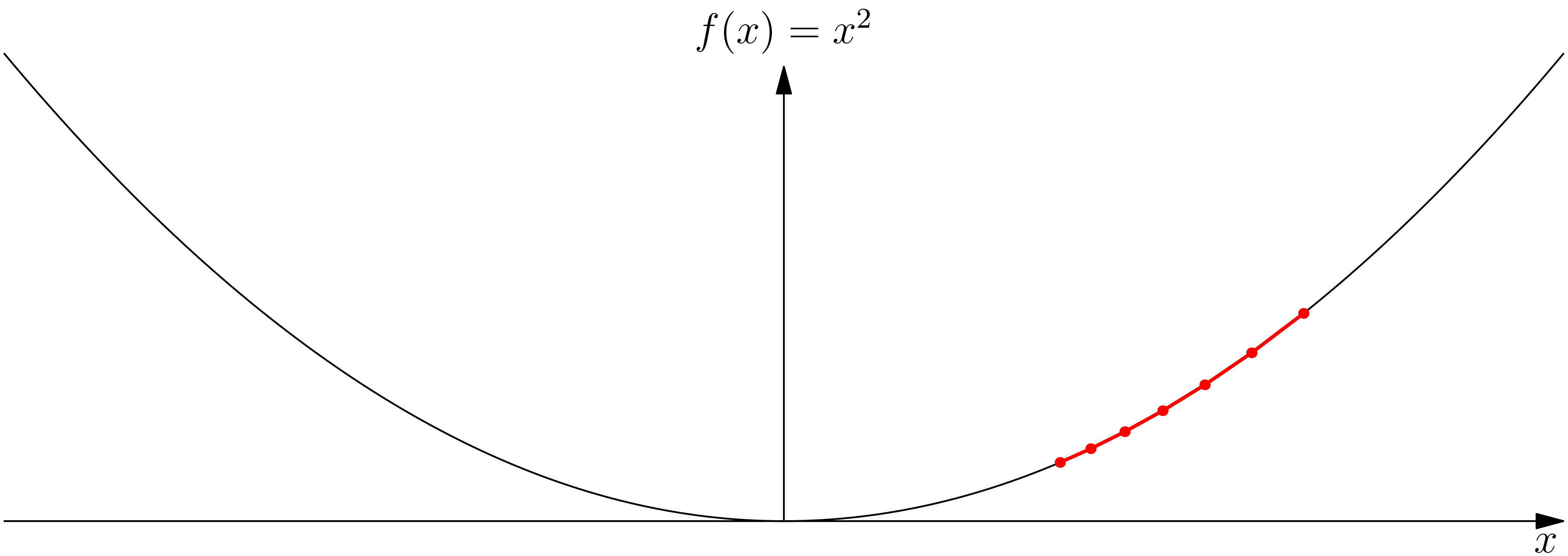
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$

Step Size

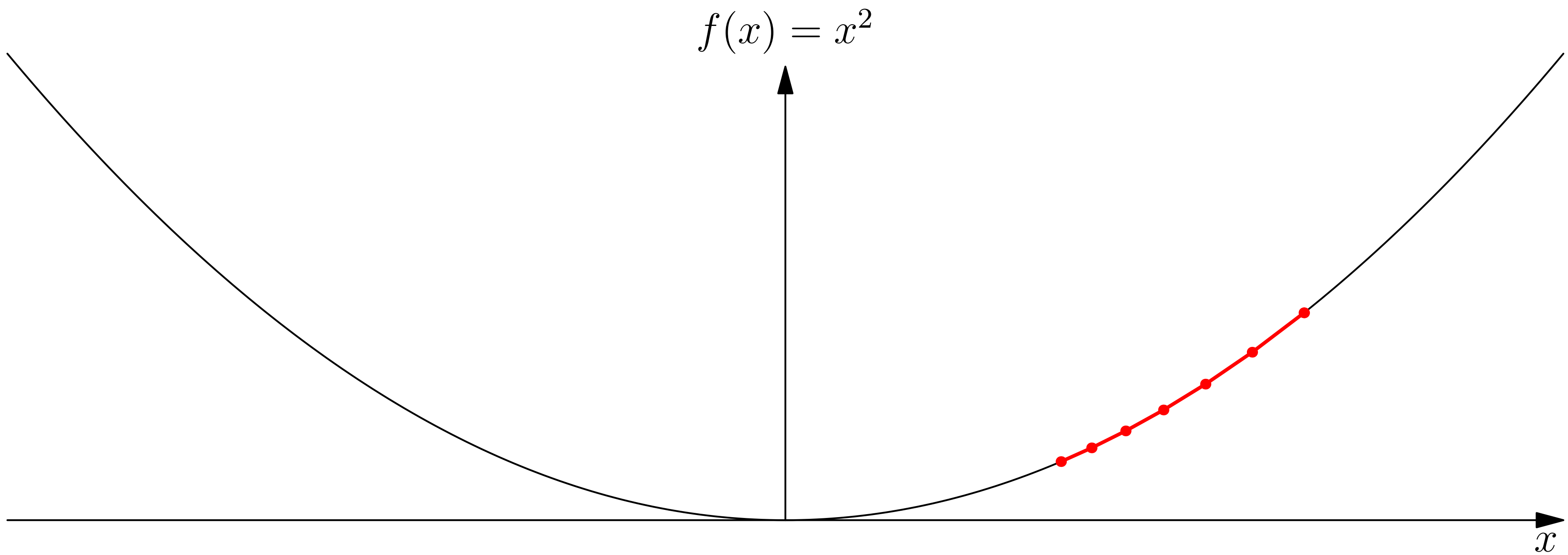
$x \leftarrow x - r\, f'(x)$



r = 0.05

Step Size

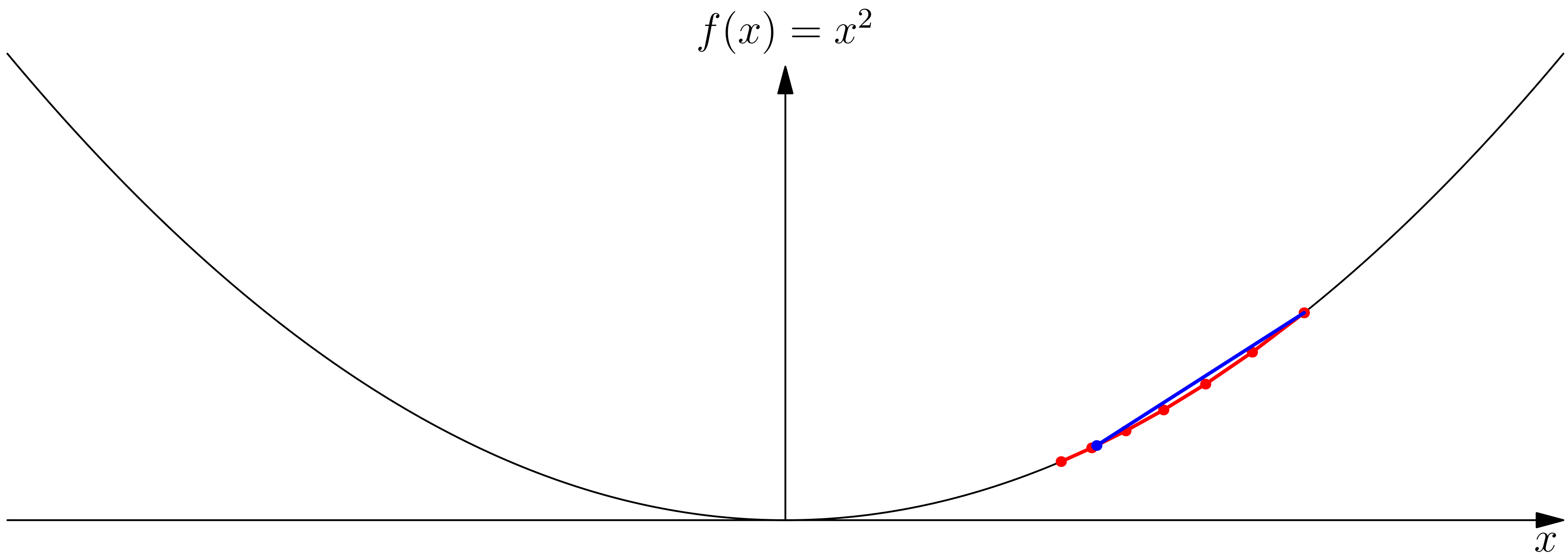
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$ $\mathrm{r} = 0.2$

Step Size

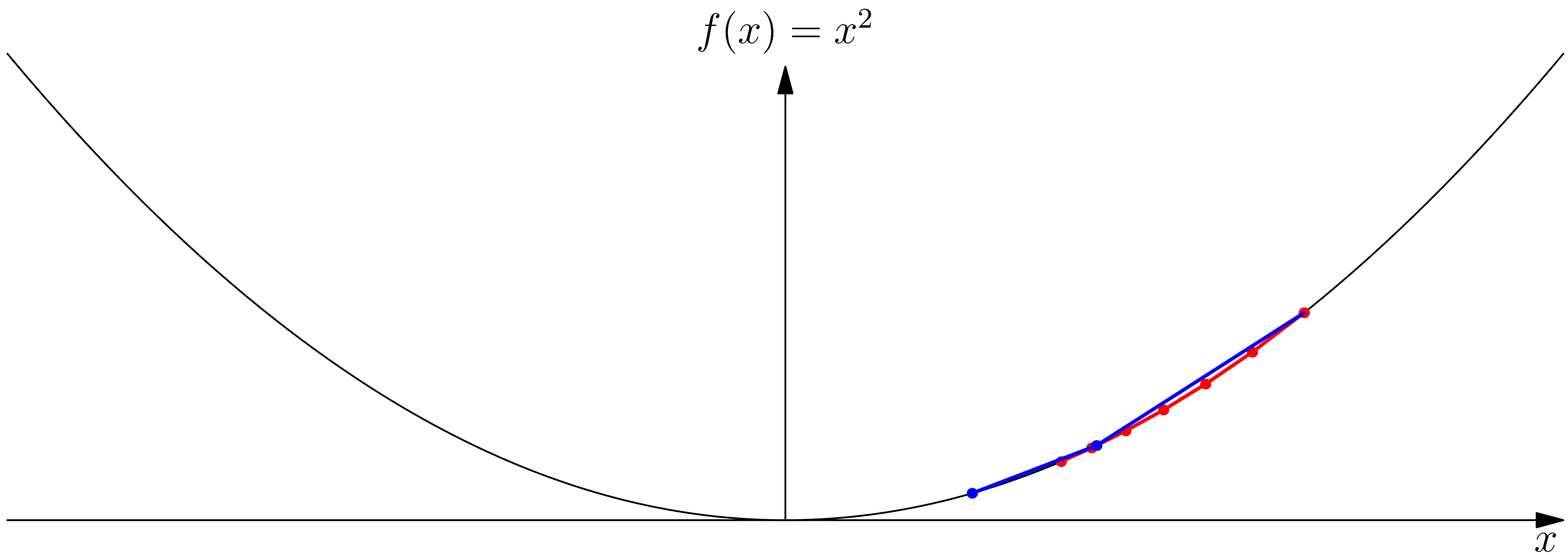
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$ $\mathrm{r} = 0.2$

Step Size

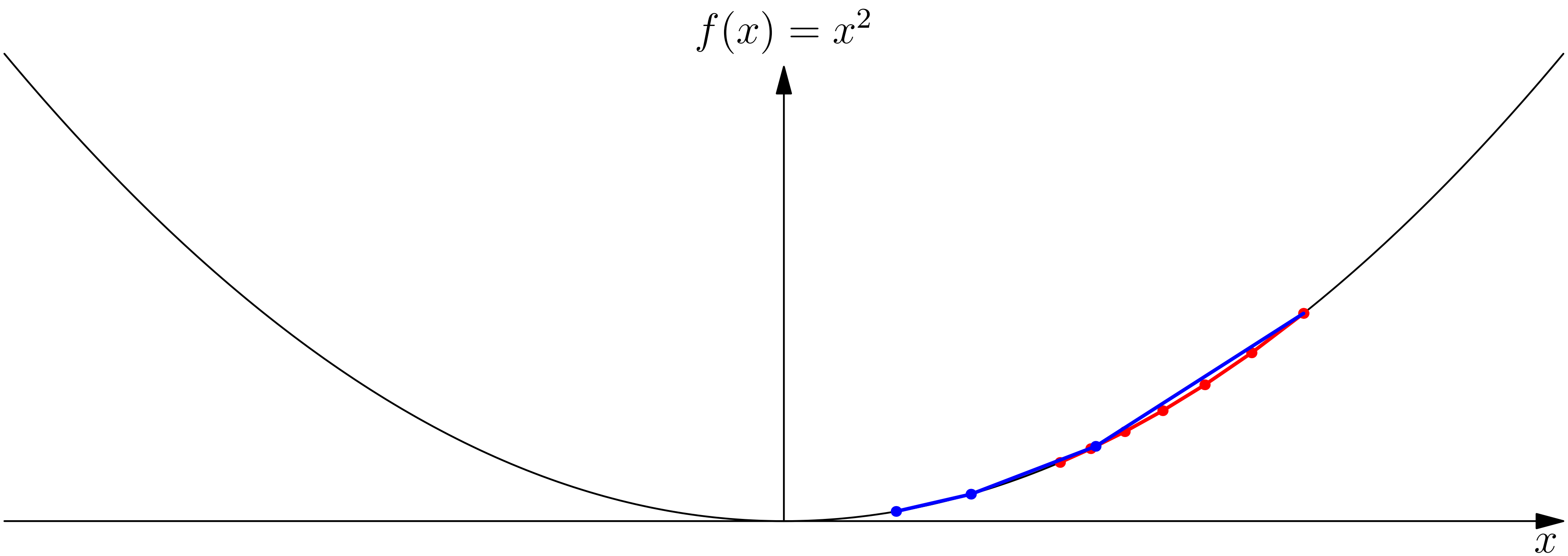
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$ $\mathrm{r} = 0.2$

Step Size

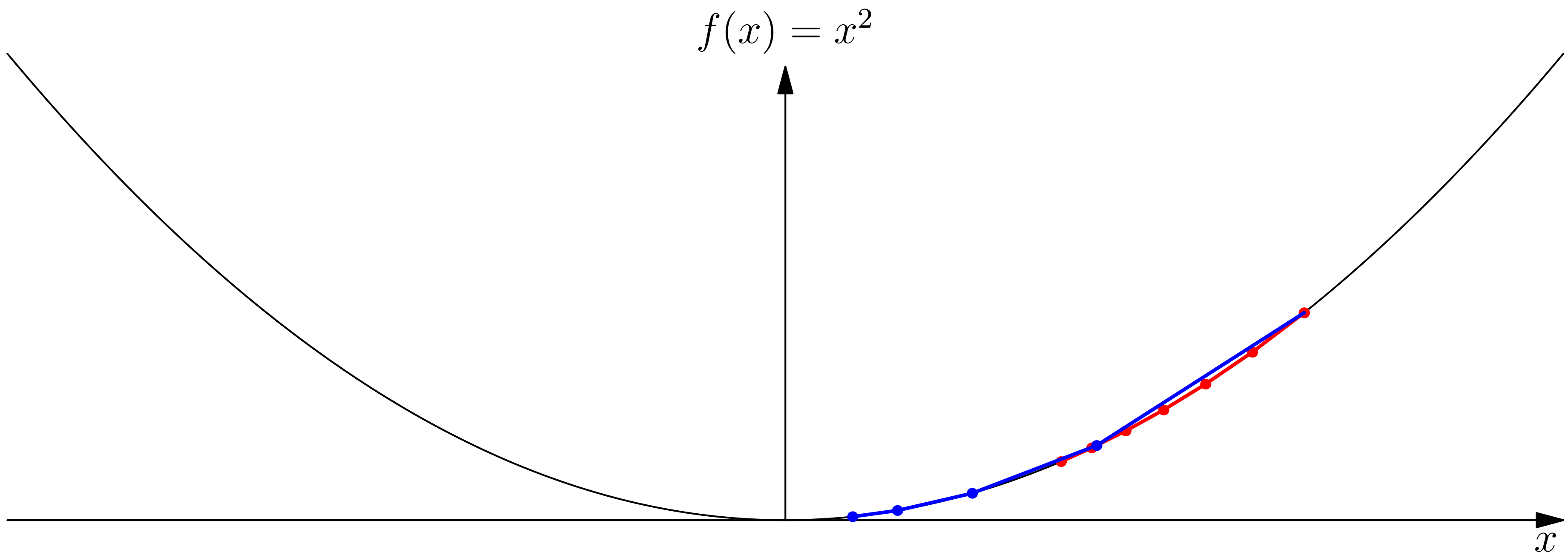
$x \leftarrow x - r\,f'(x)$



$\mathrm{r} = 0.05$ $\mathrm{r} = 0.2$

Step Size

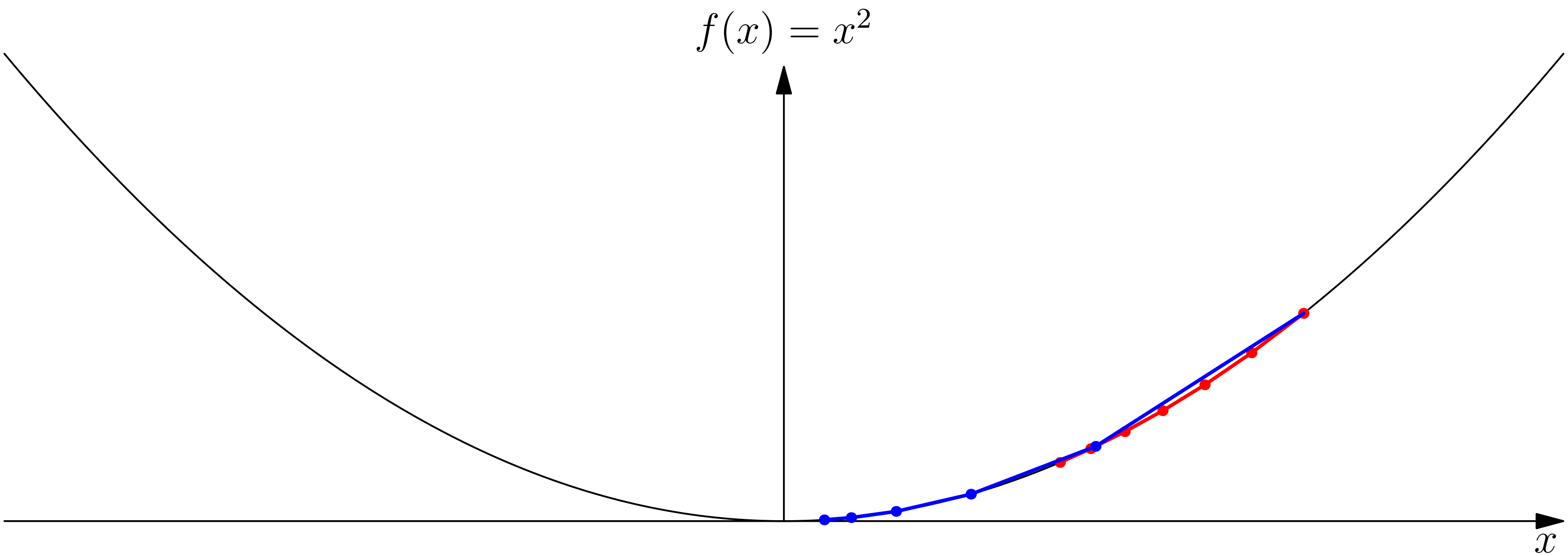
$$x \leftarrow x - r\, f'(x)$$



$\color{red}{\mathrm{r} = 0.05}$ $\color{blue}{\mathrm{r} = 0.2}$

Step Size

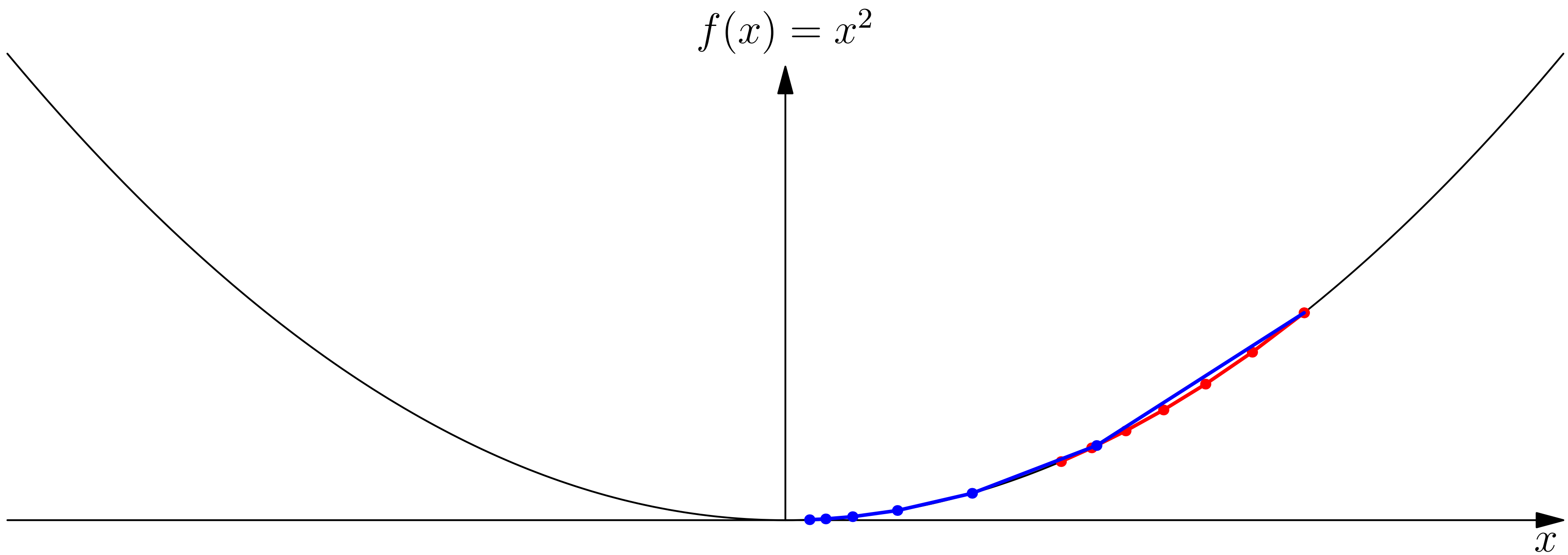
$x \leftarrow x - r\, f'(x)$



r = 0.05   r = 0.2

Step Size

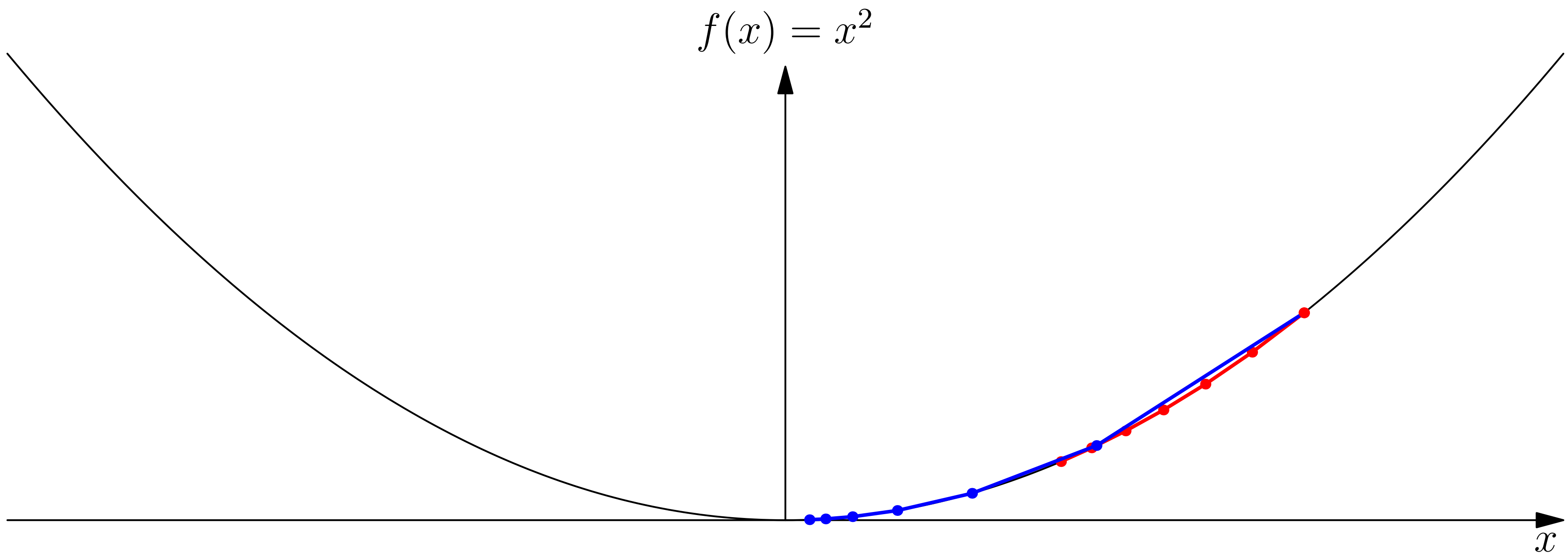
$x \leftarrow x - r\, f'(x)$



$\color{red}{\mathrm{r} = 0.05}$ $\color{blue}{\mathrm{r} = 0.2}$

Step Size

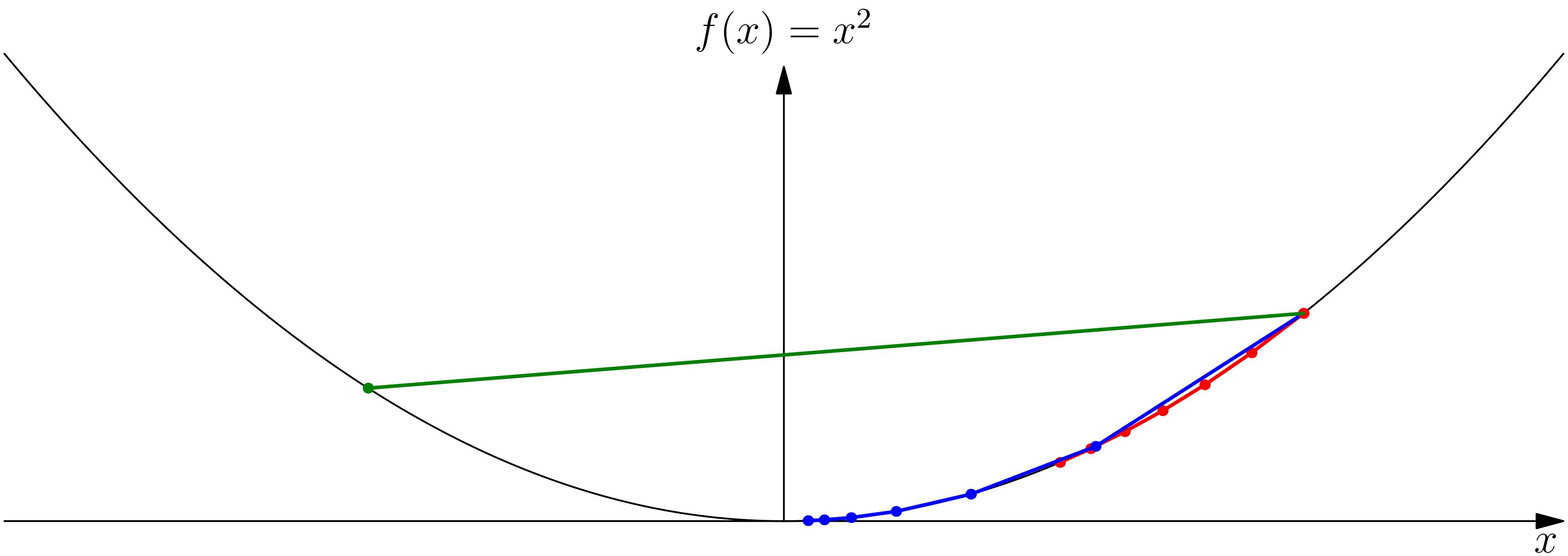
$x \leftarrow x - r\, f'(x)$



r = 0.05 r = 0.2 r = 0.9

Step Size

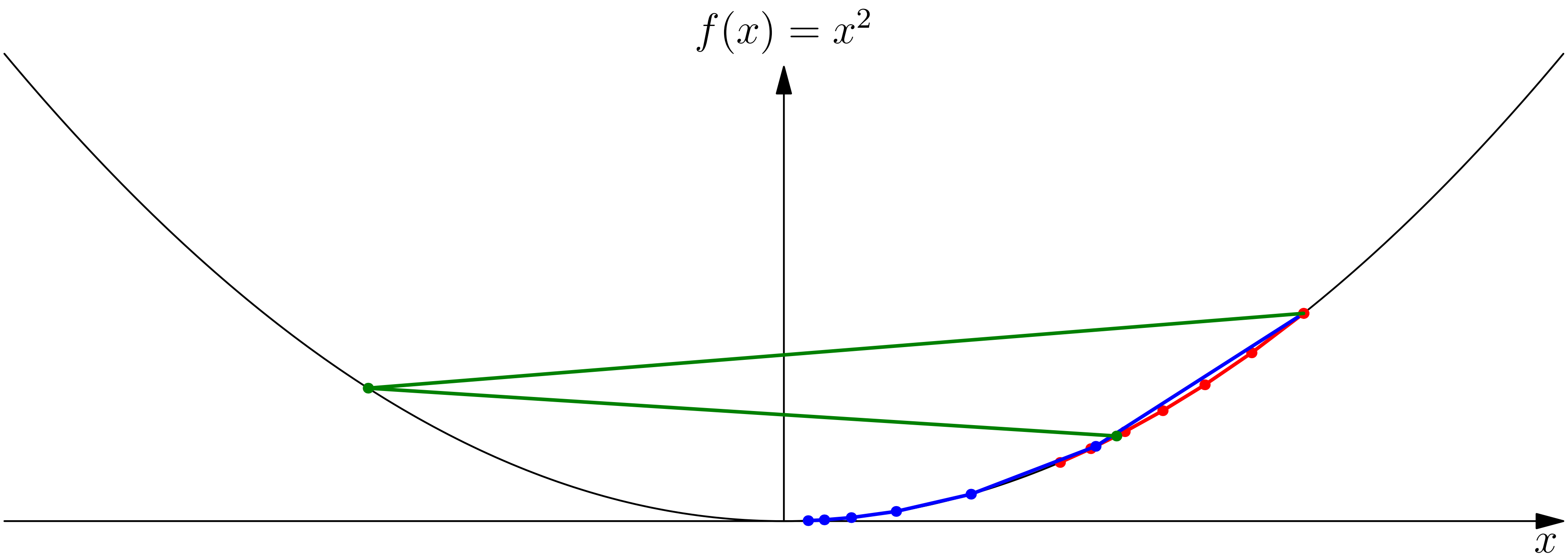
$x \leftarrow x - r\, f'(x)$



r = 0.05 r = 0.2 r = 0.9

Step Size

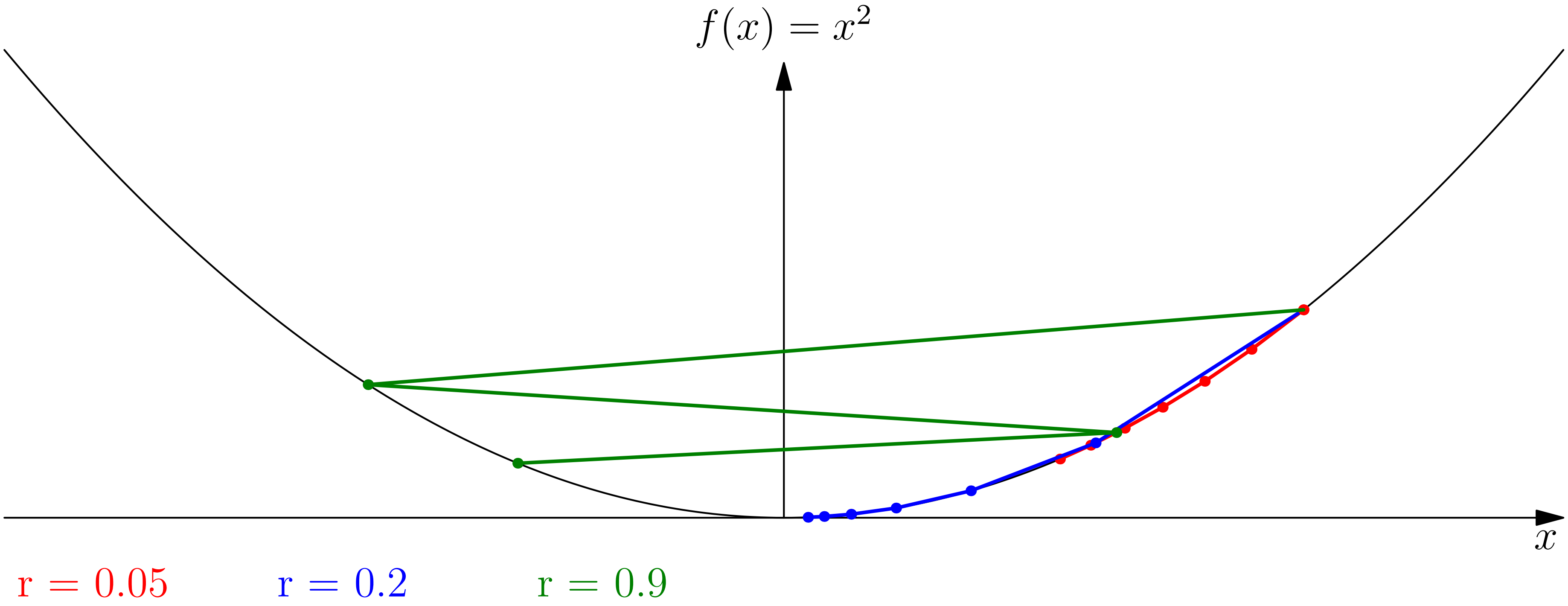
$x \leftarrow x - r\, f'(x)$



$\color{red}{\mathrm{r} = 0.05}$ $\color{blue}{\mathrm{r} = 0.2}$ $\color{green}{\mathrm{r} = 0.9}$

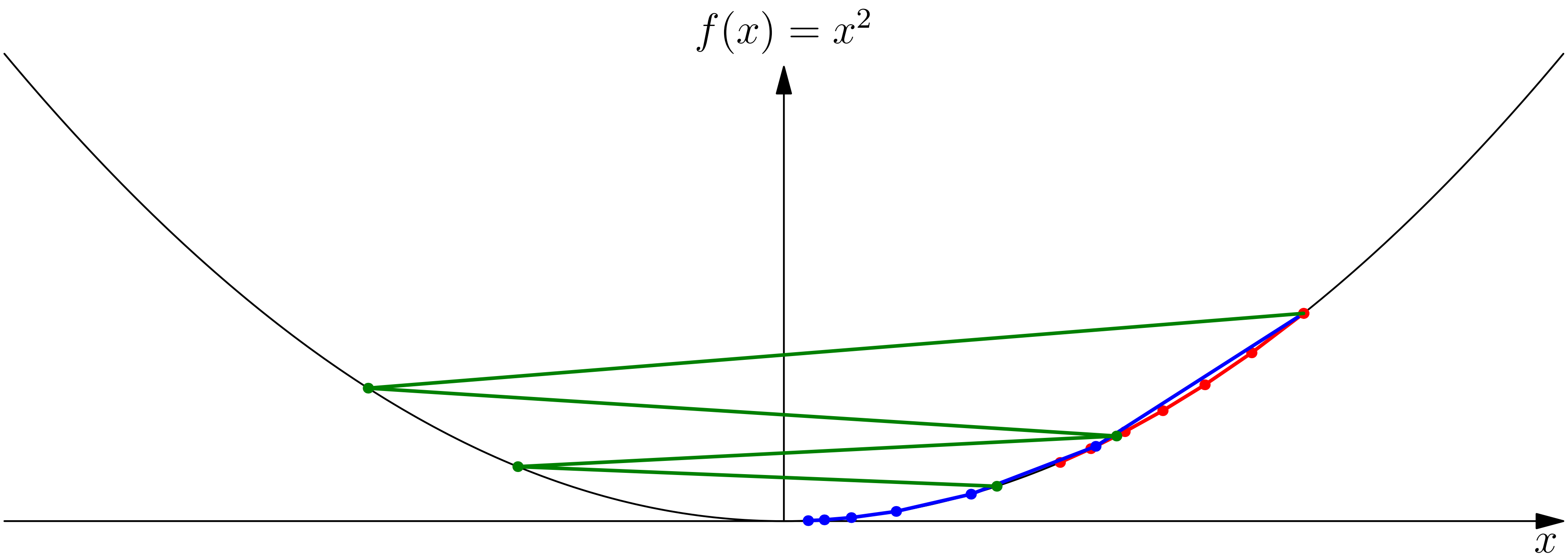
Step Size

$x \leftarrow x - r\, f'(x)$



Step Size

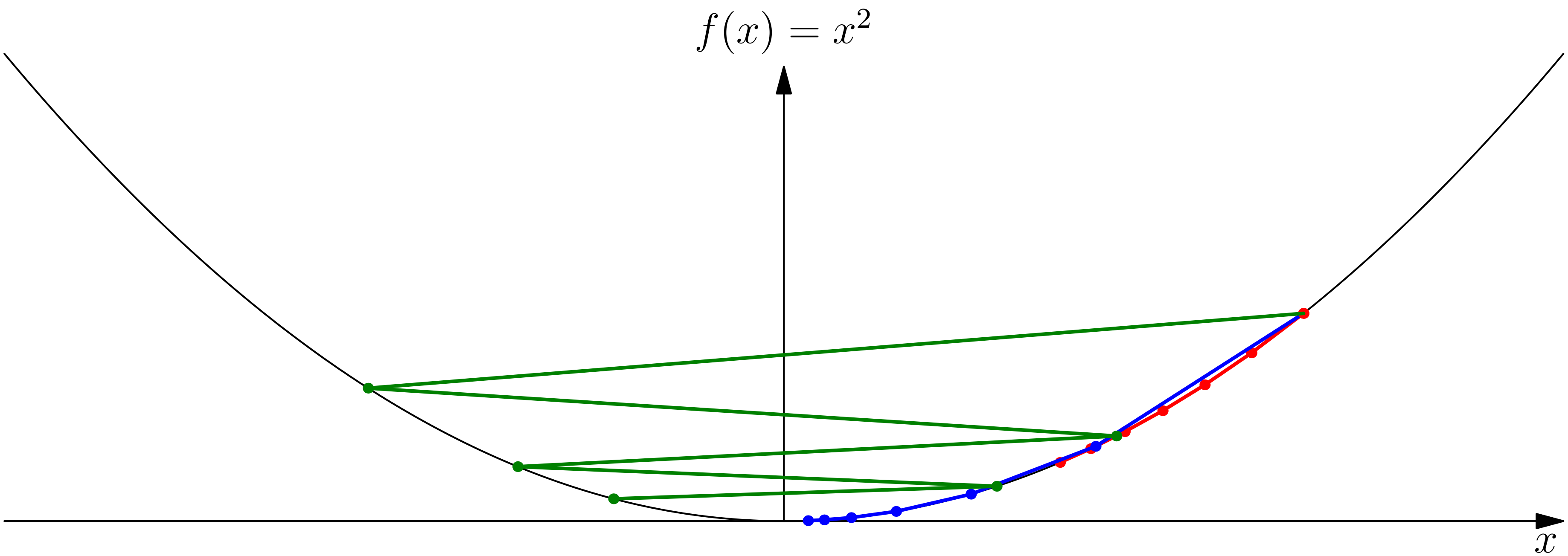
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05 \qquad \mathrm{r} = 0.2 \qquad \mathrm{r} = 0.9$

Step Size

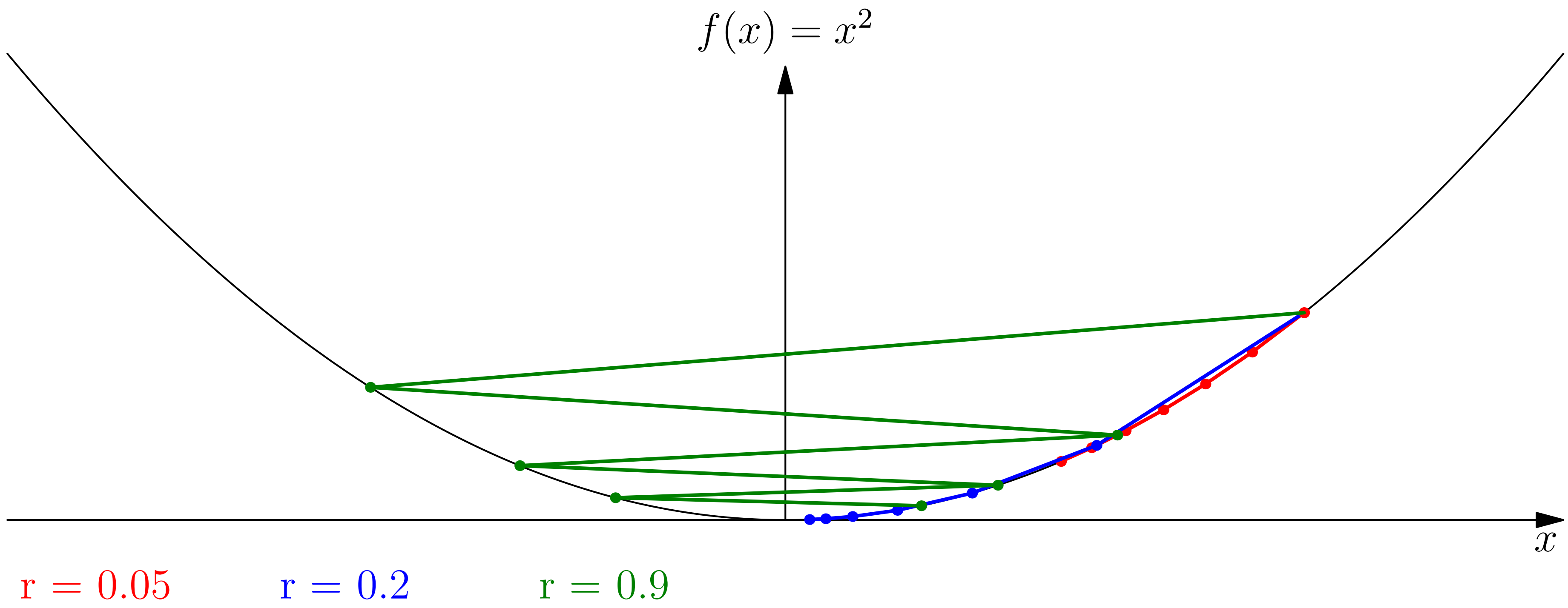
$x \leftarrow x - r\, f'(x)$



$\mathrm{r} = 0.05$ $\mathrm{r} = 0.2$ $\mathrm{r} = 0.9$

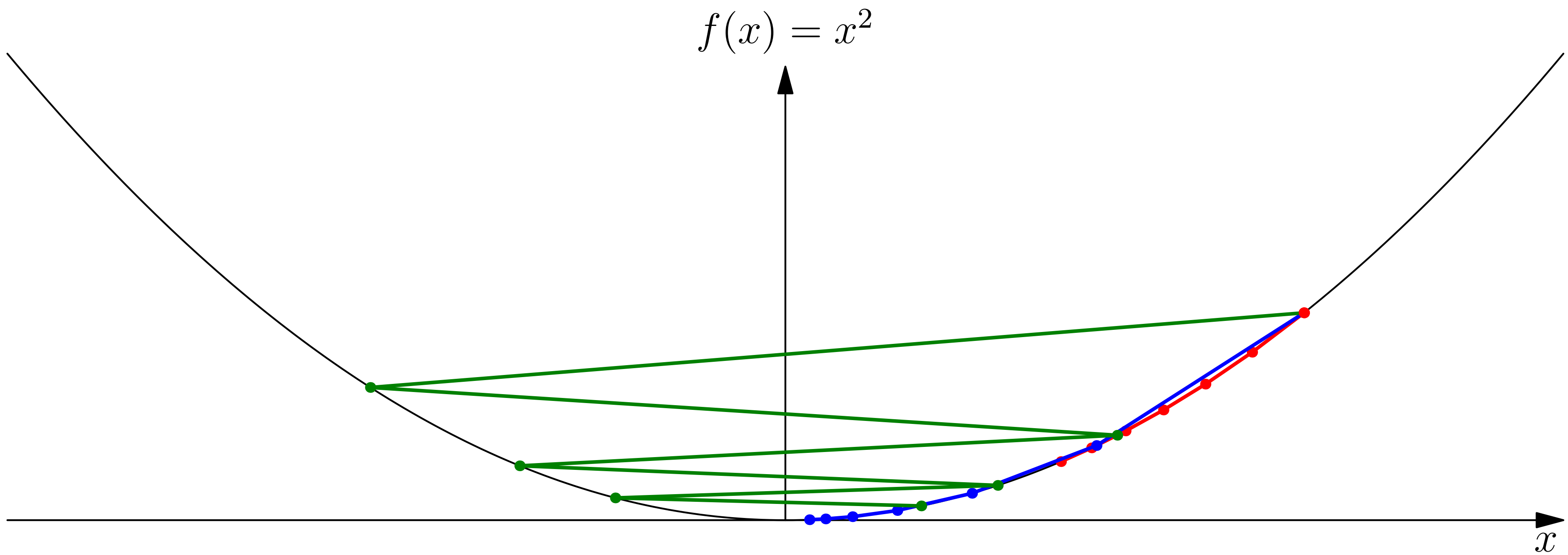
Step Size

$x \leftarrow x - r\, f'(x)$



Step Size

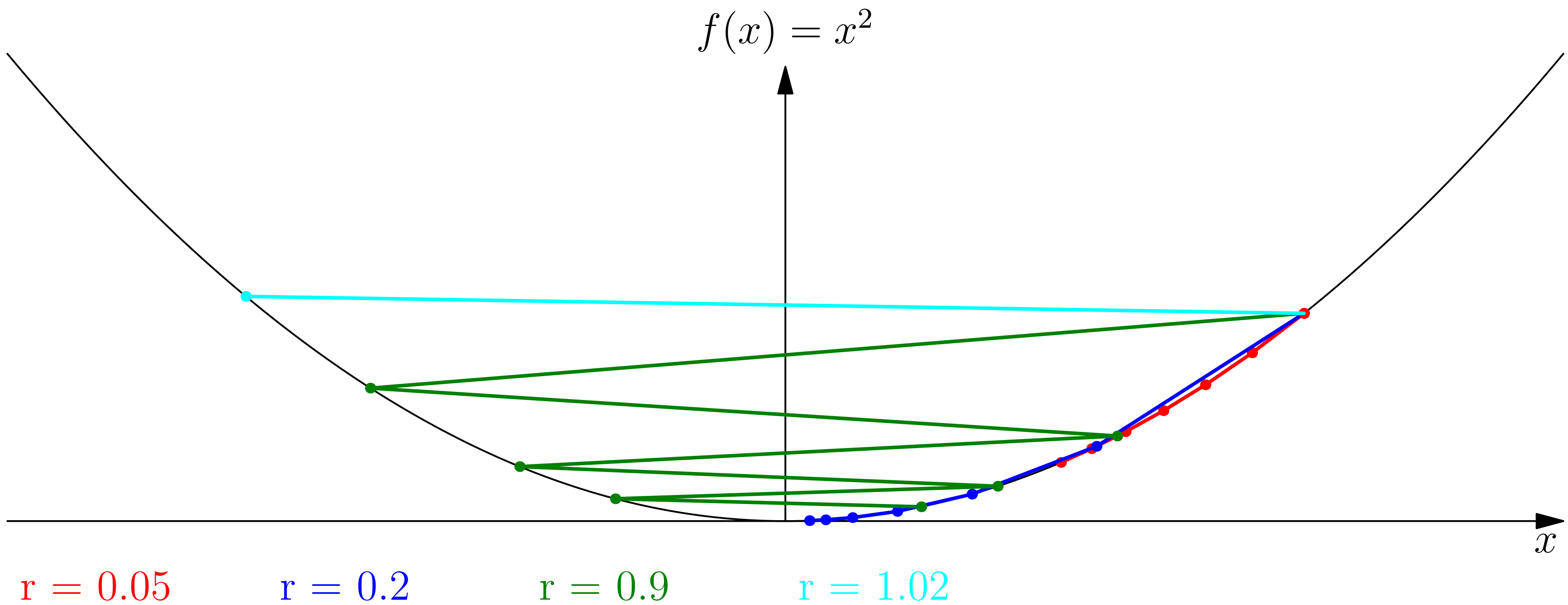
$x \leftarrow x - r\, f'(x)$



r = 0.05   r = 0.2   r = 0.9   r = 1.02

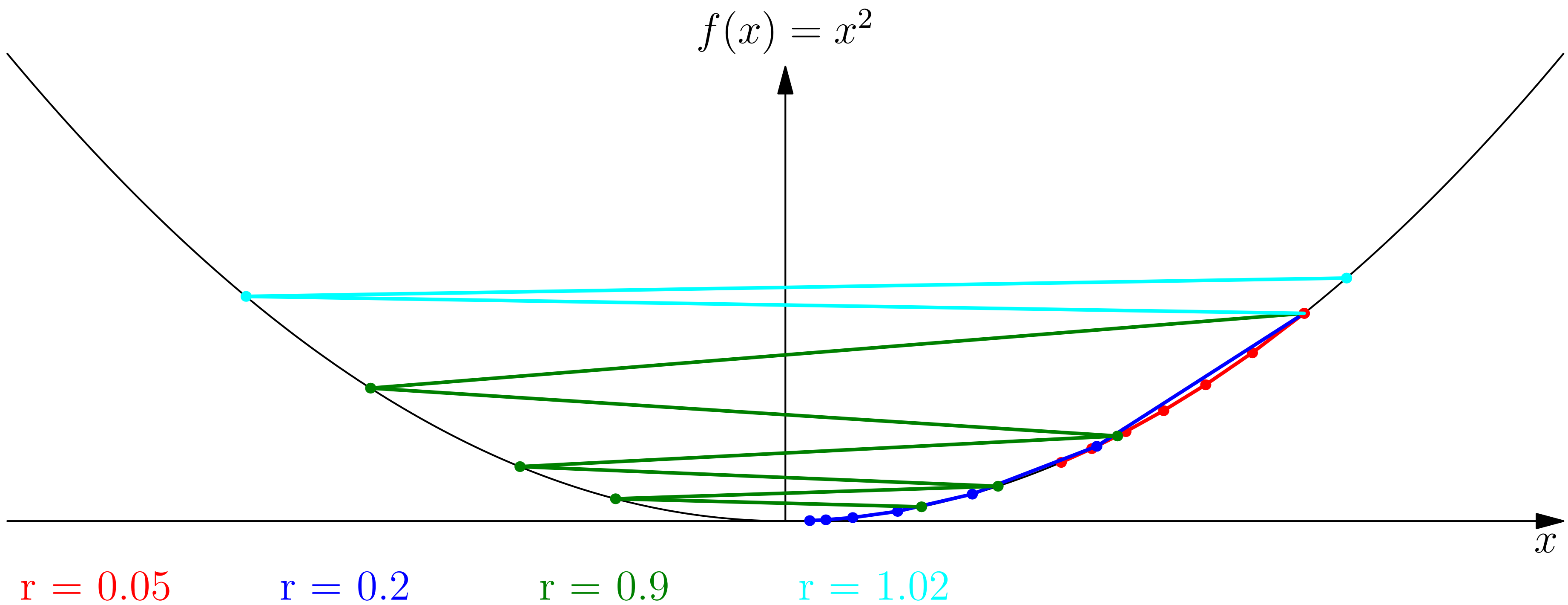
Step Size

$x \leftarrow x - r\, f'(x)$



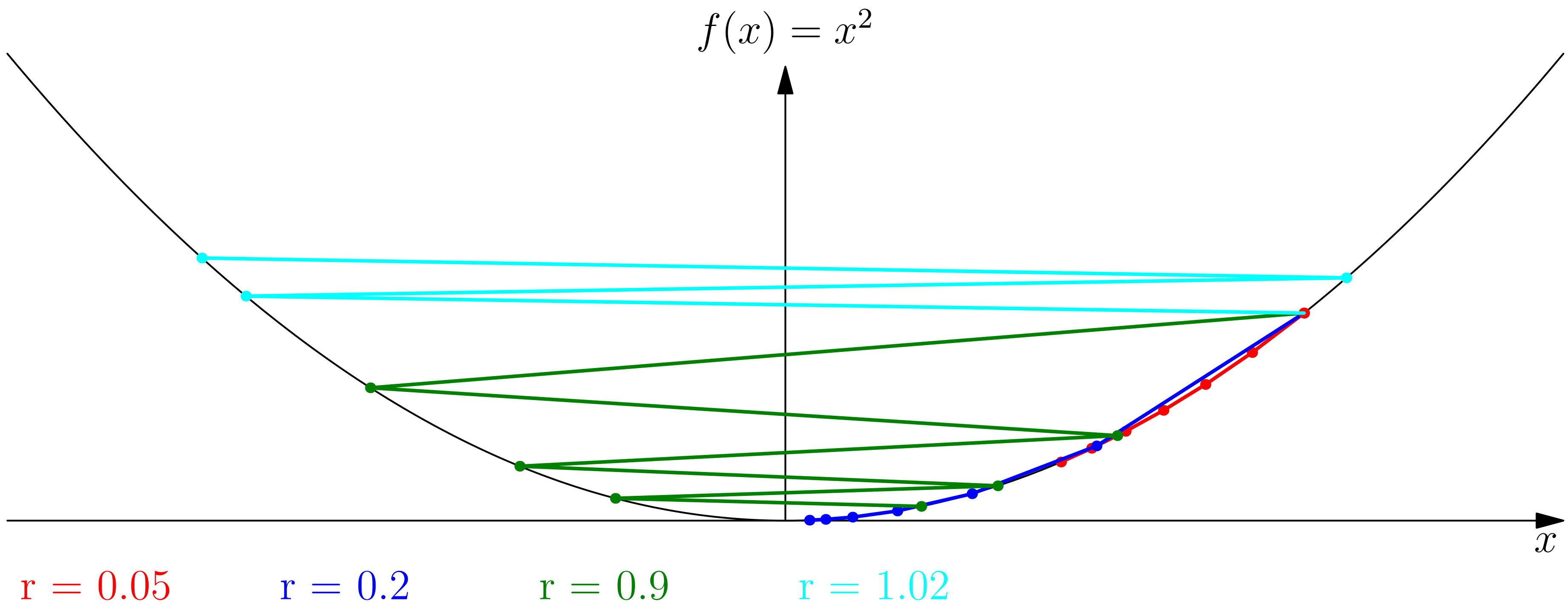
Step Size

$x \leftarrow x - r\, f'(x)$



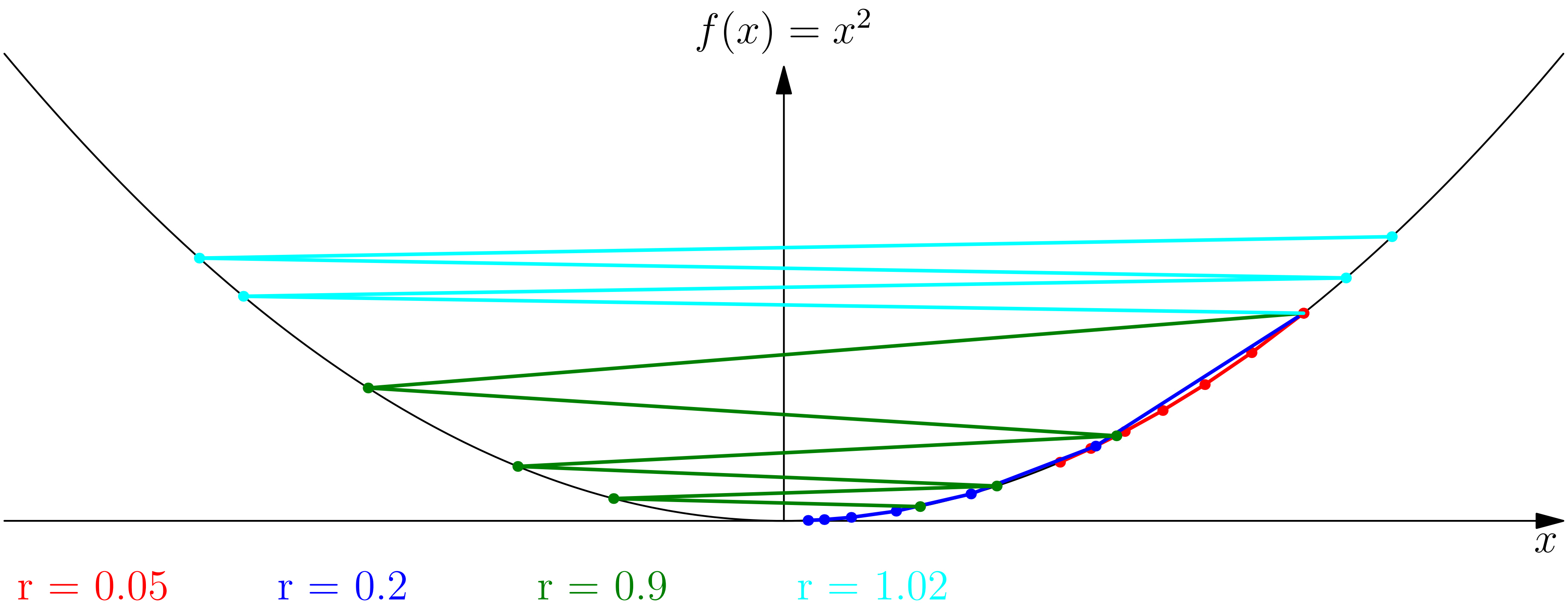
Step Size

$x \leftarrow x - r\, f'(x)$



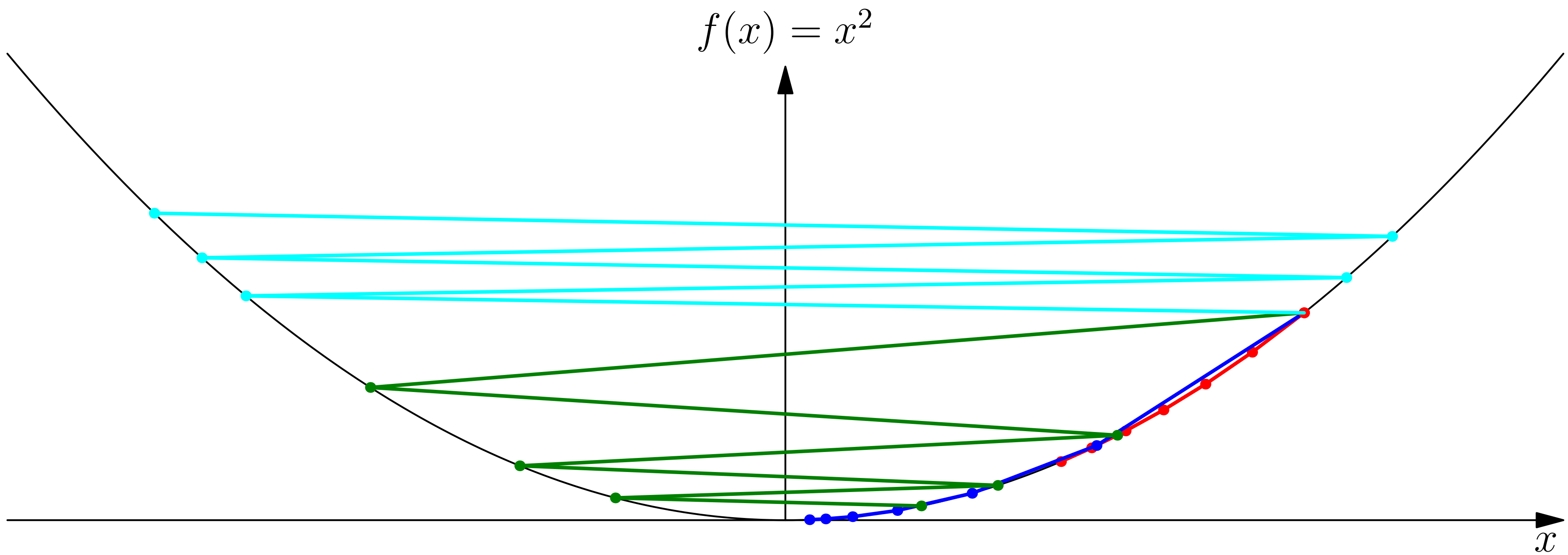
Step Size

$$x \leftarrow x - r\, f'(x)$$



Step Size

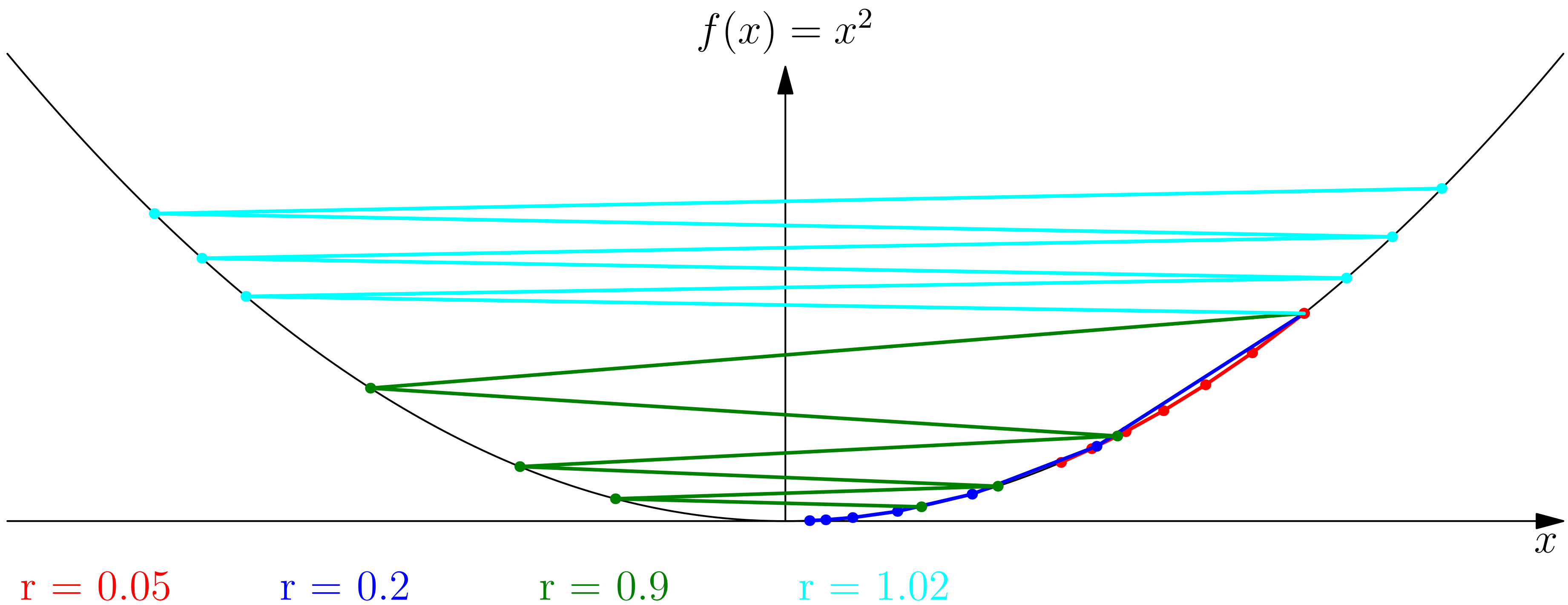
$x \leftarrow x - r\, f'(x)$



r = 0.05   r = 0.2   r = 0.9   r = 1.02

Step Size

$x \leftarrow x - r\, f'(x)$

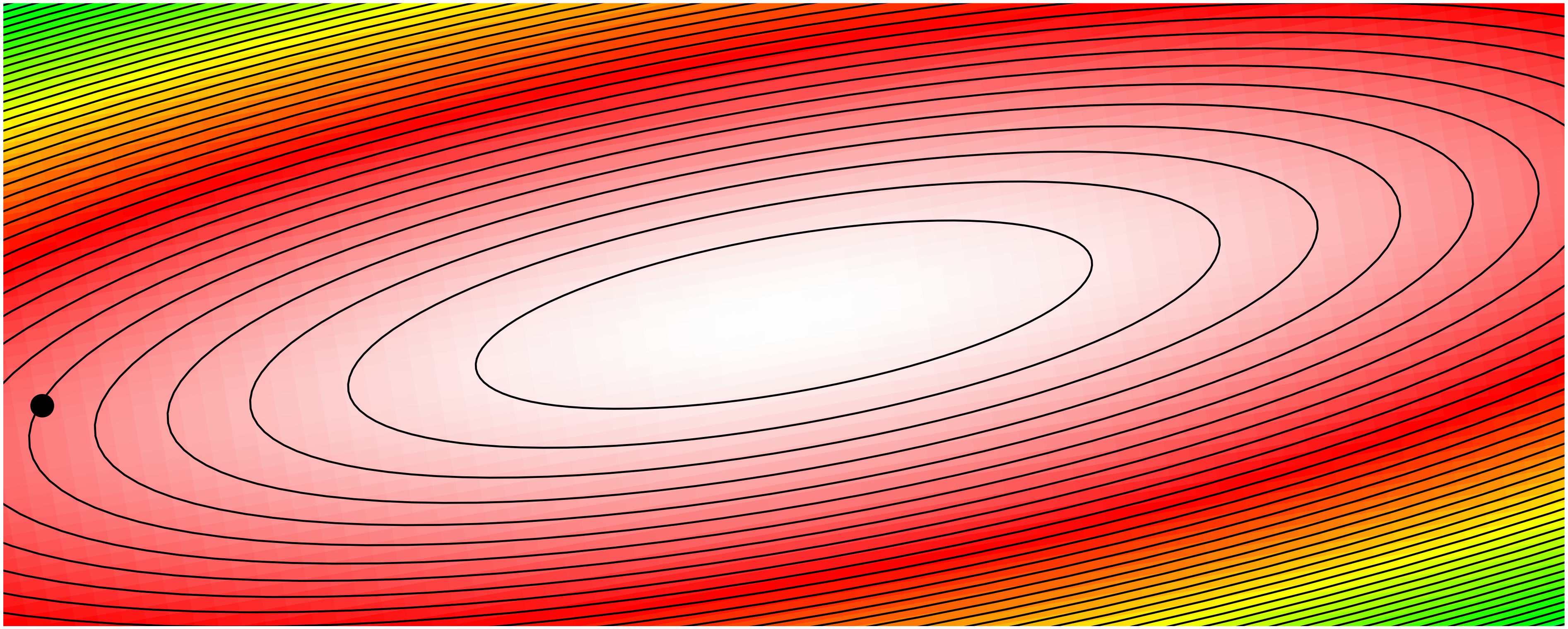


Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way

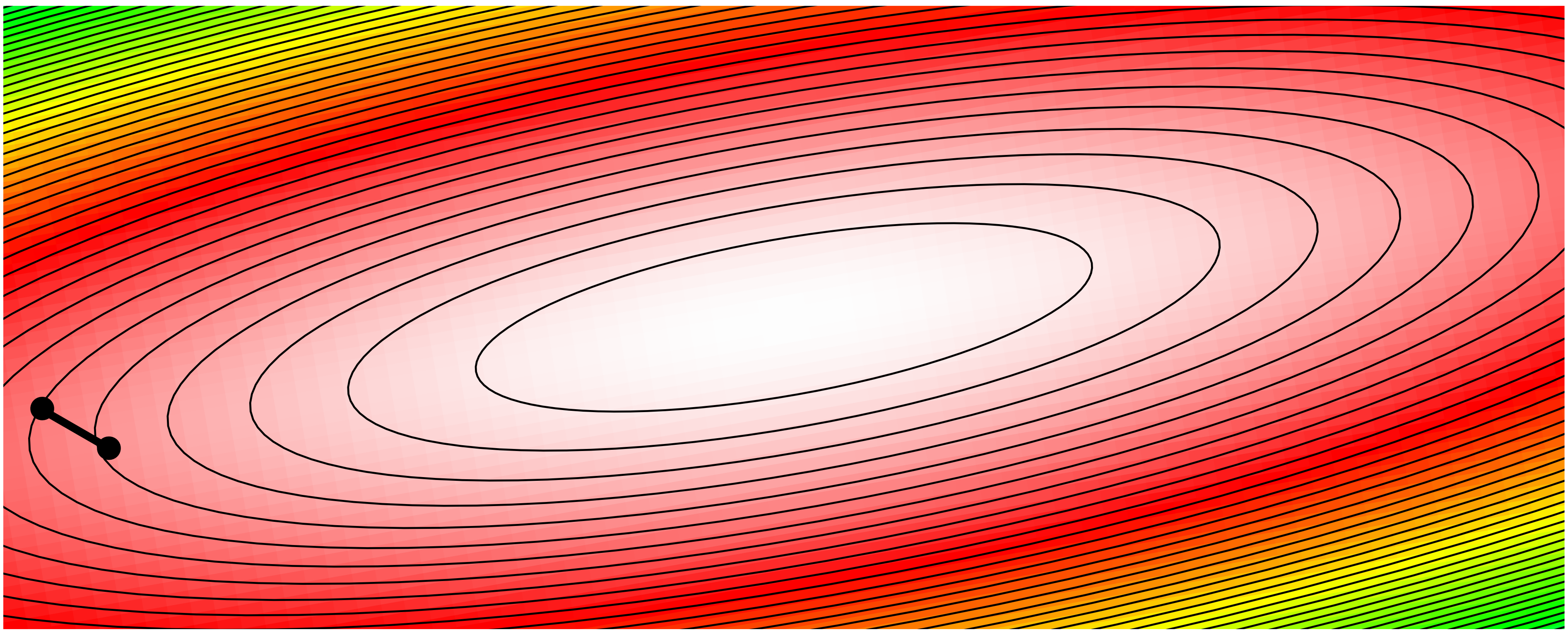
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



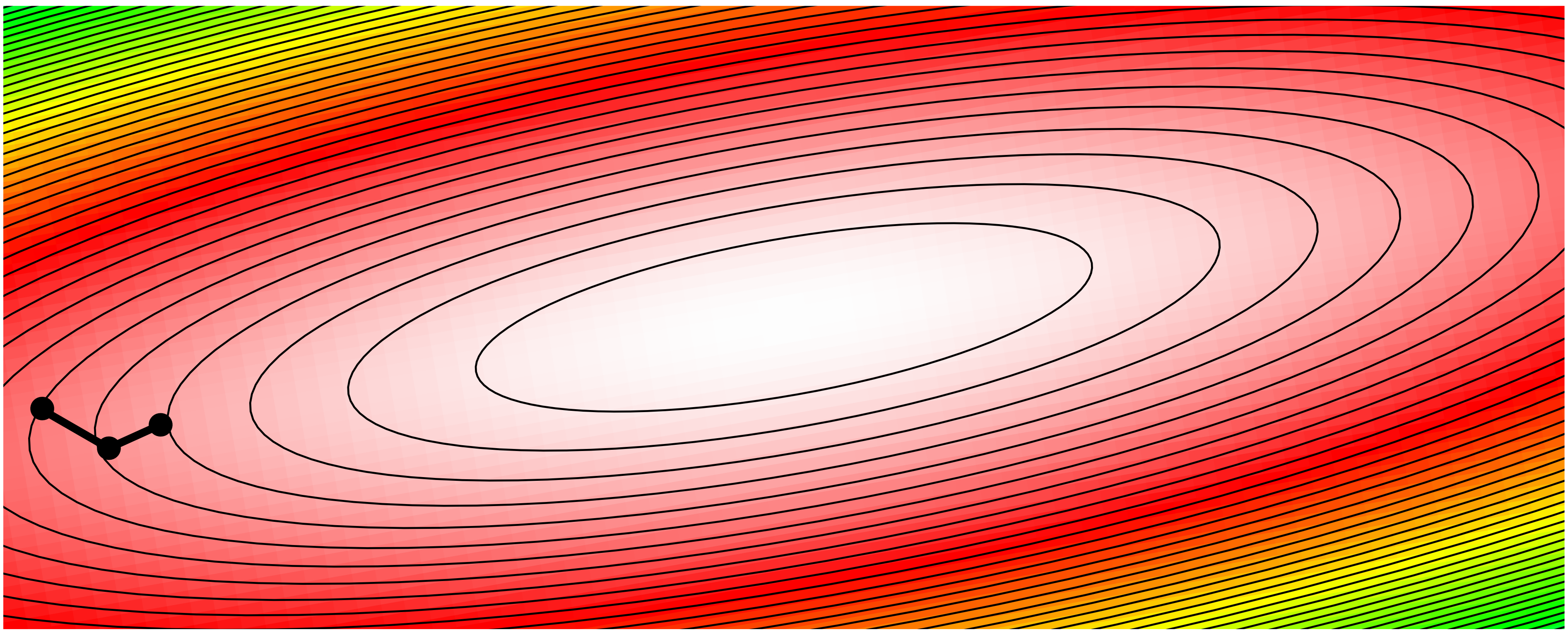
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



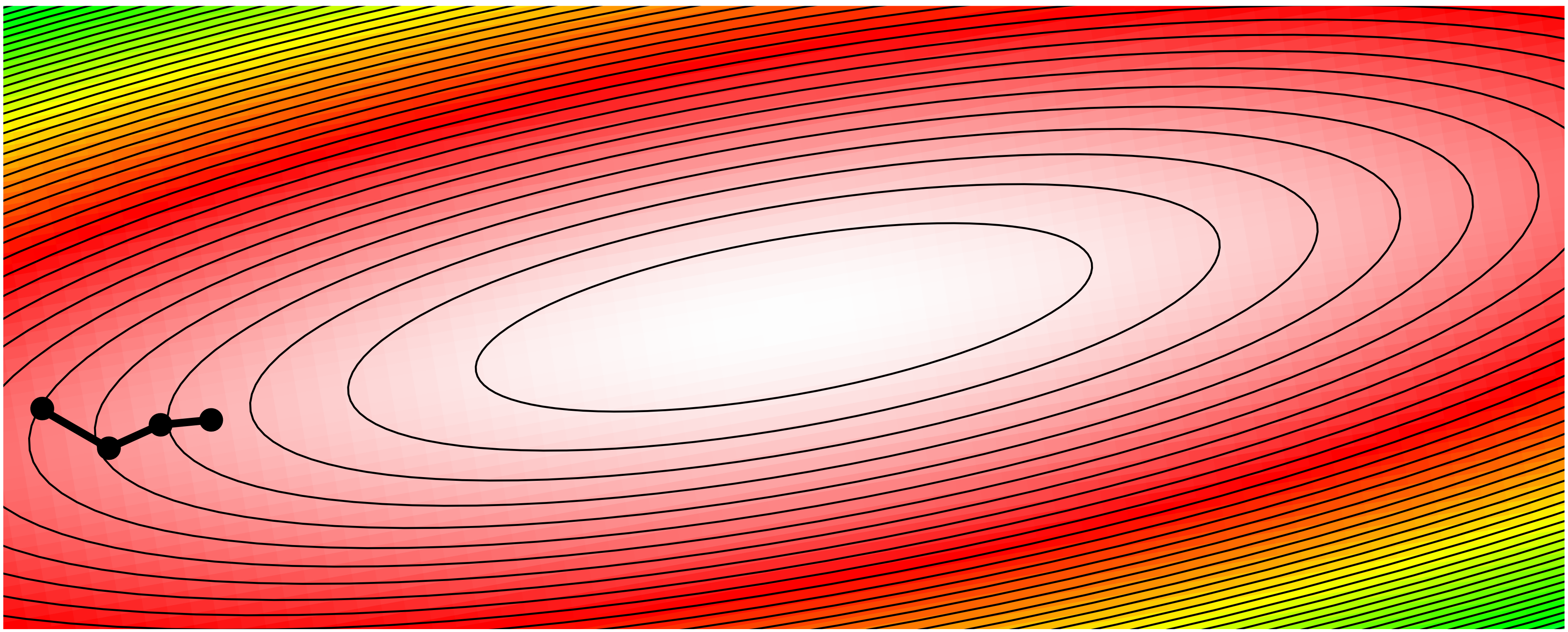
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



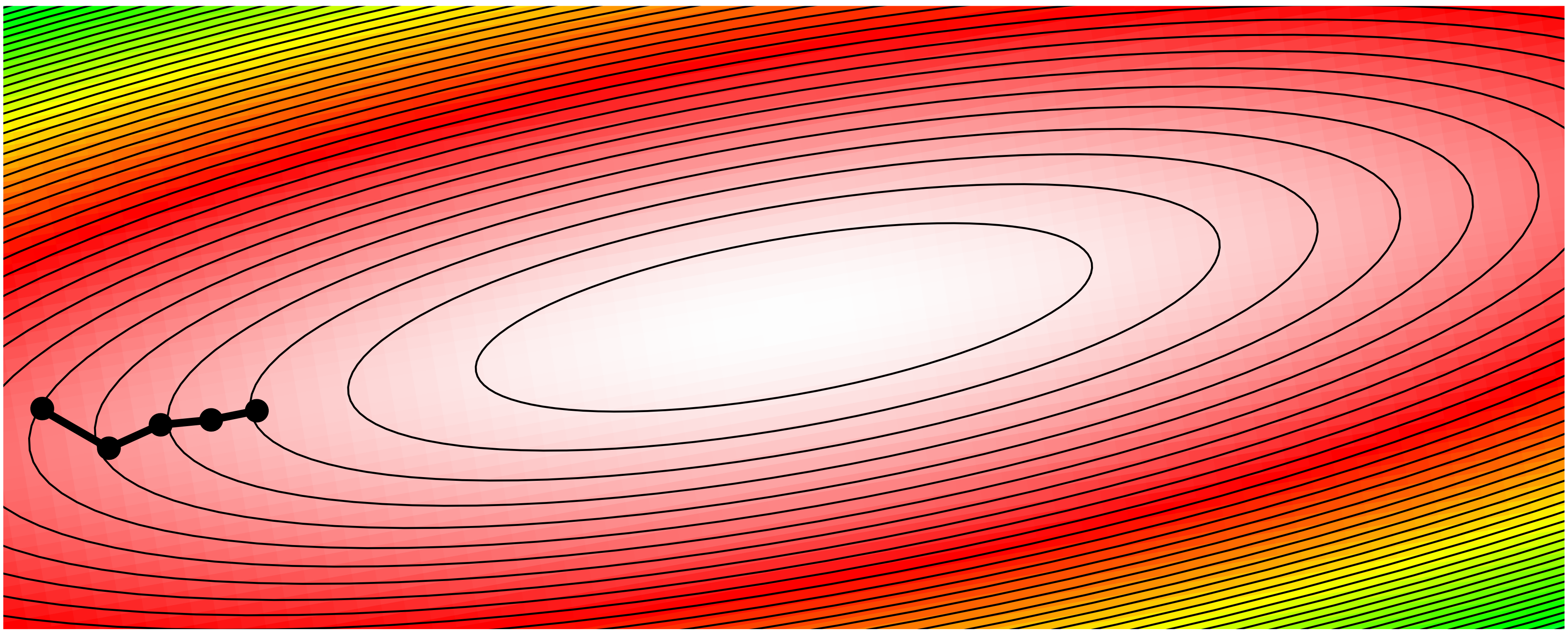
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



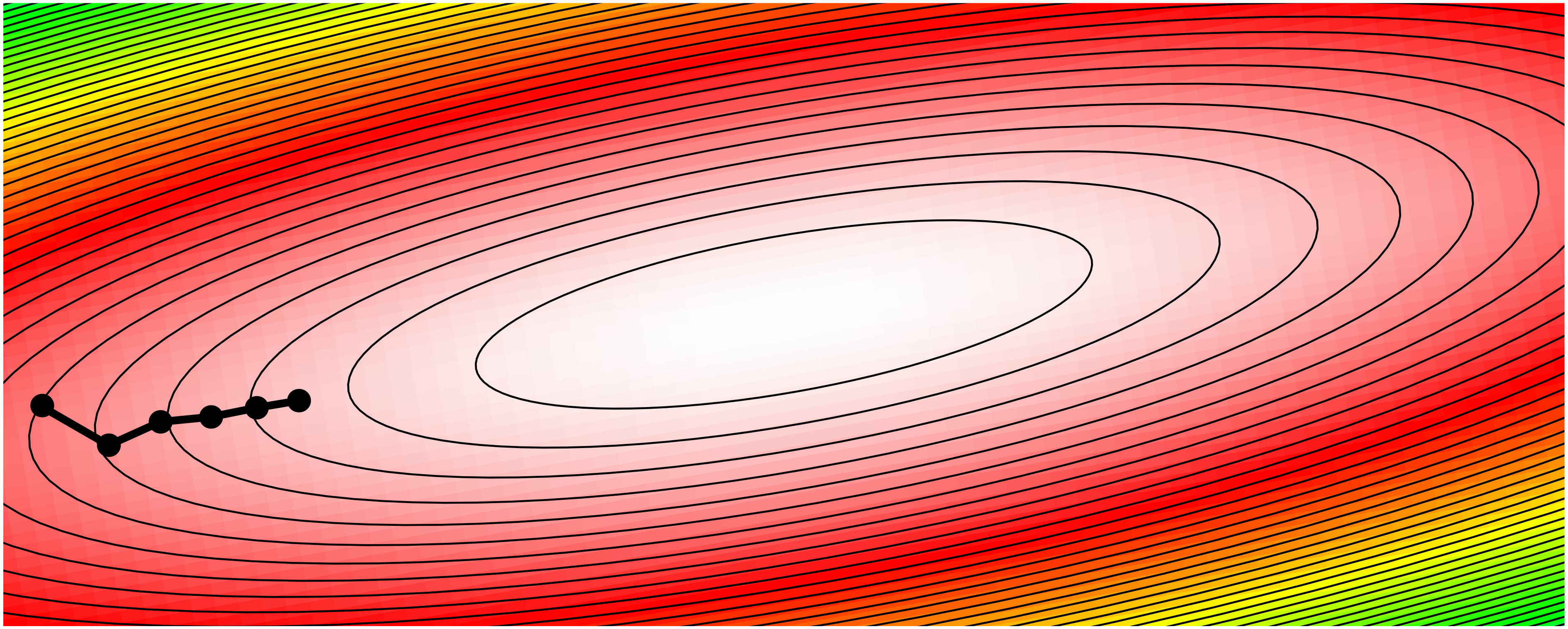
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



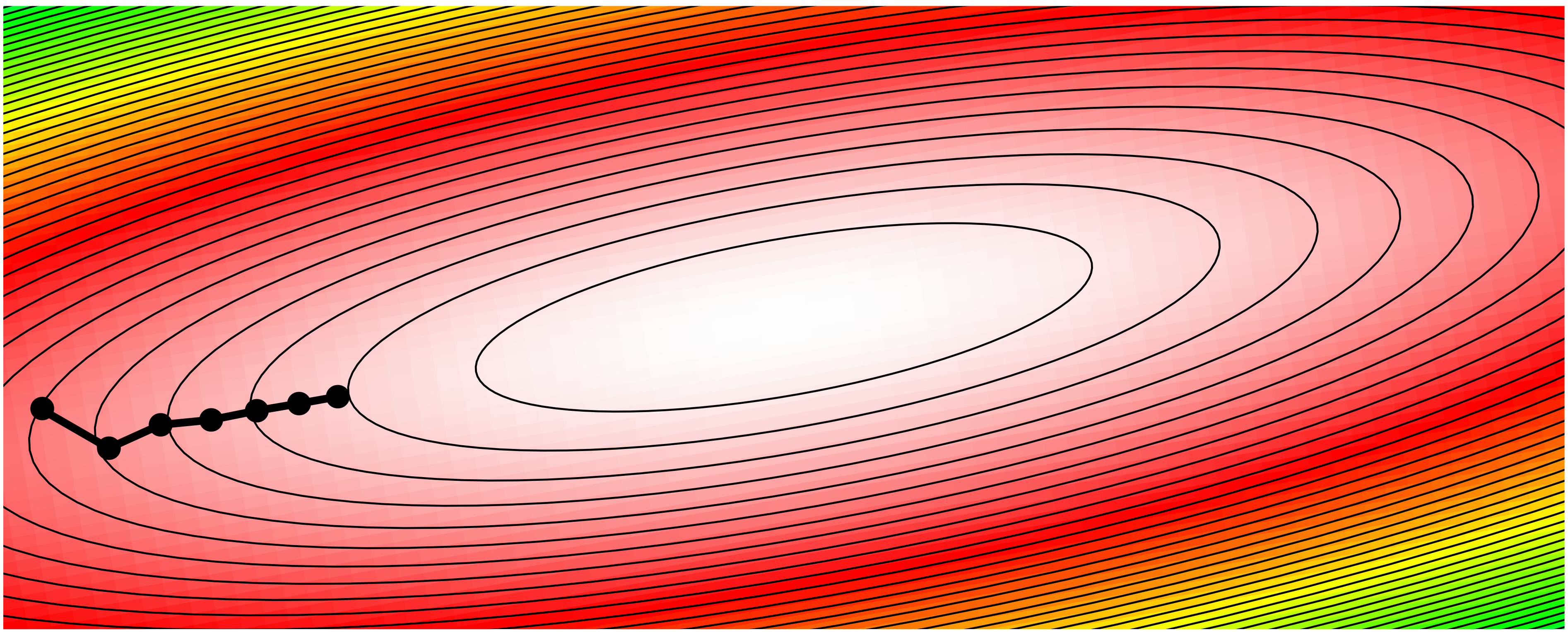
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



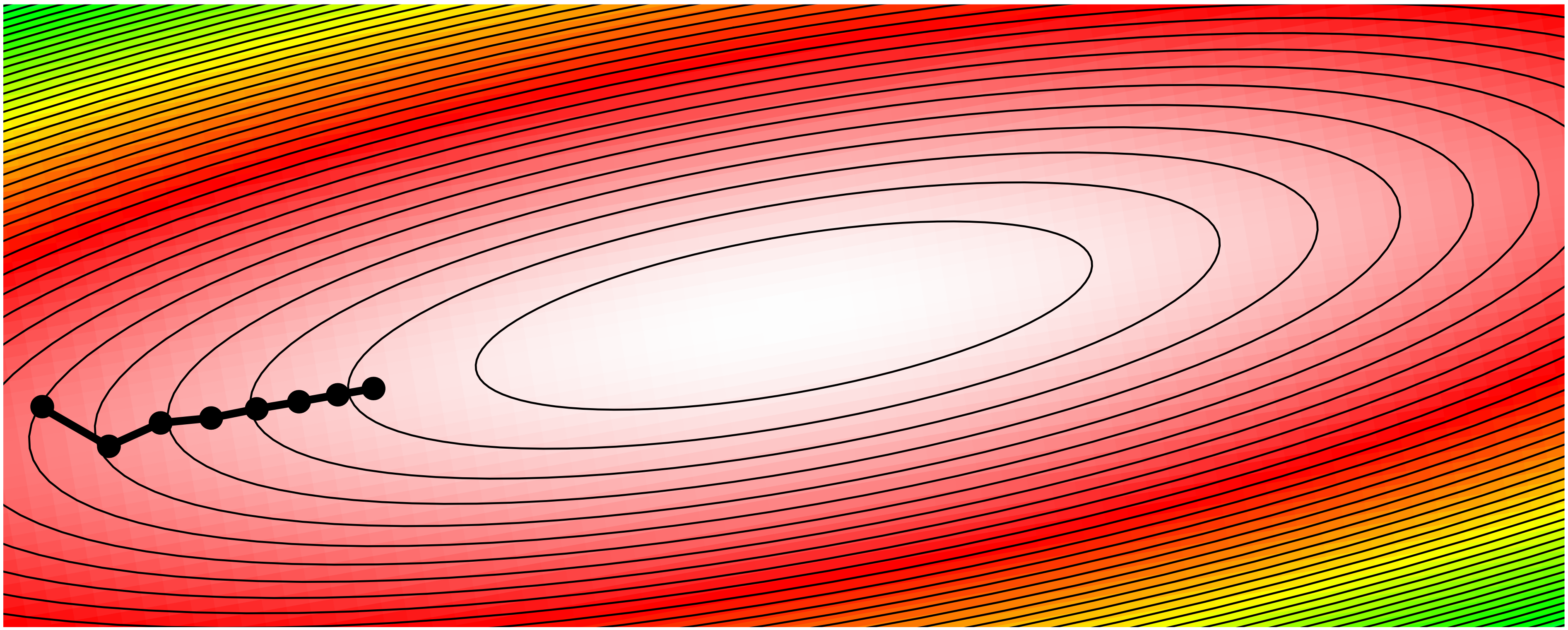
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



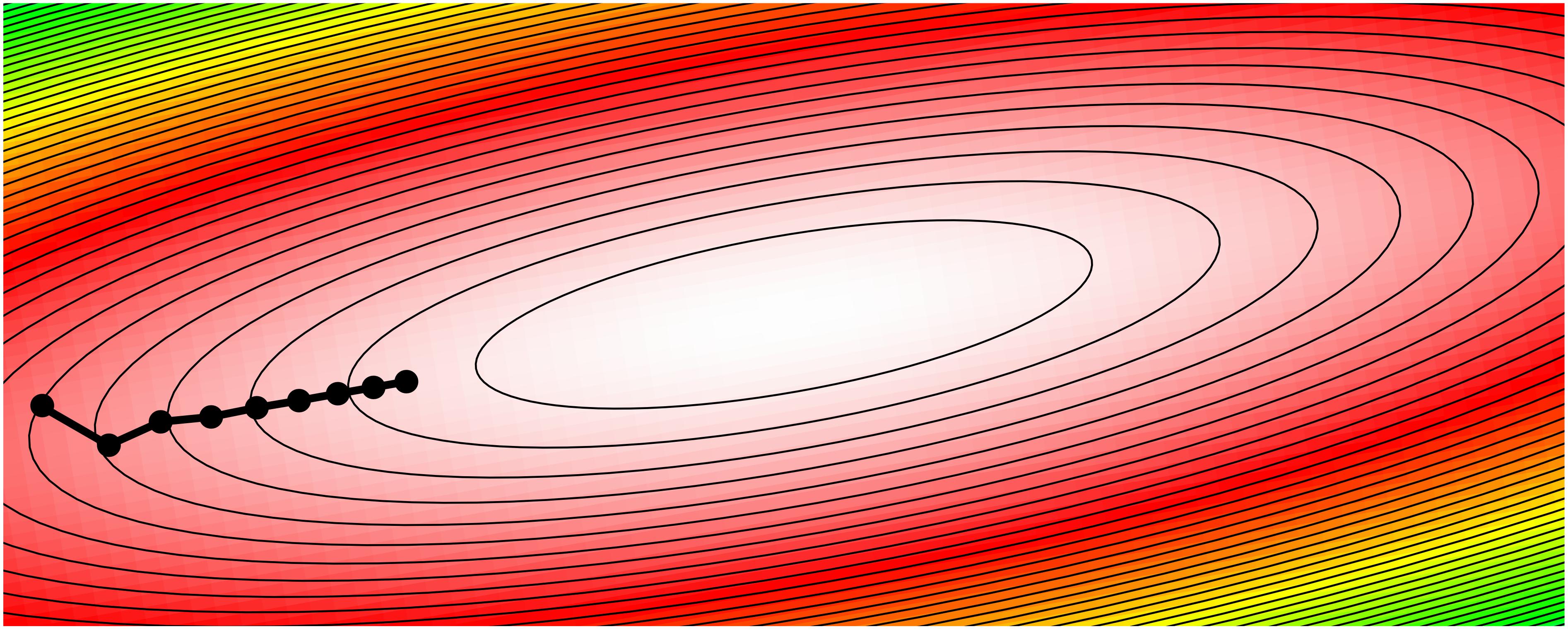
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



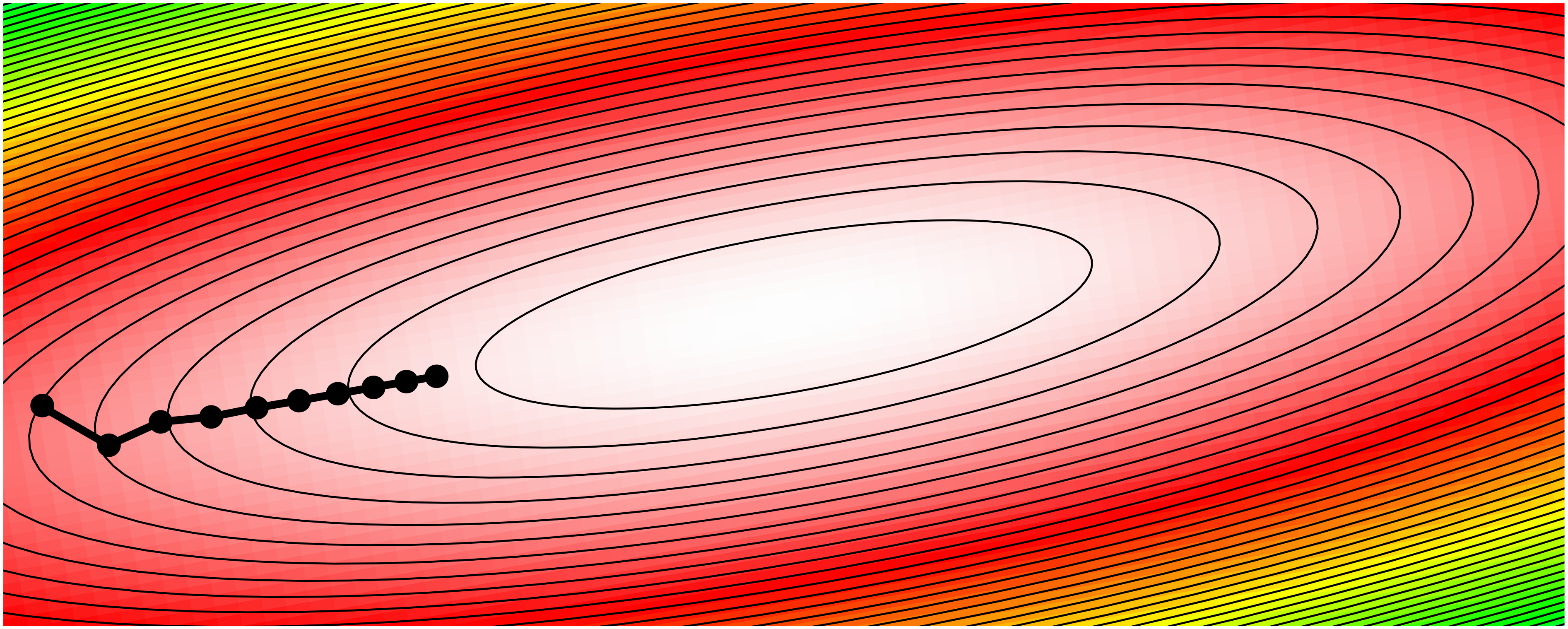
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



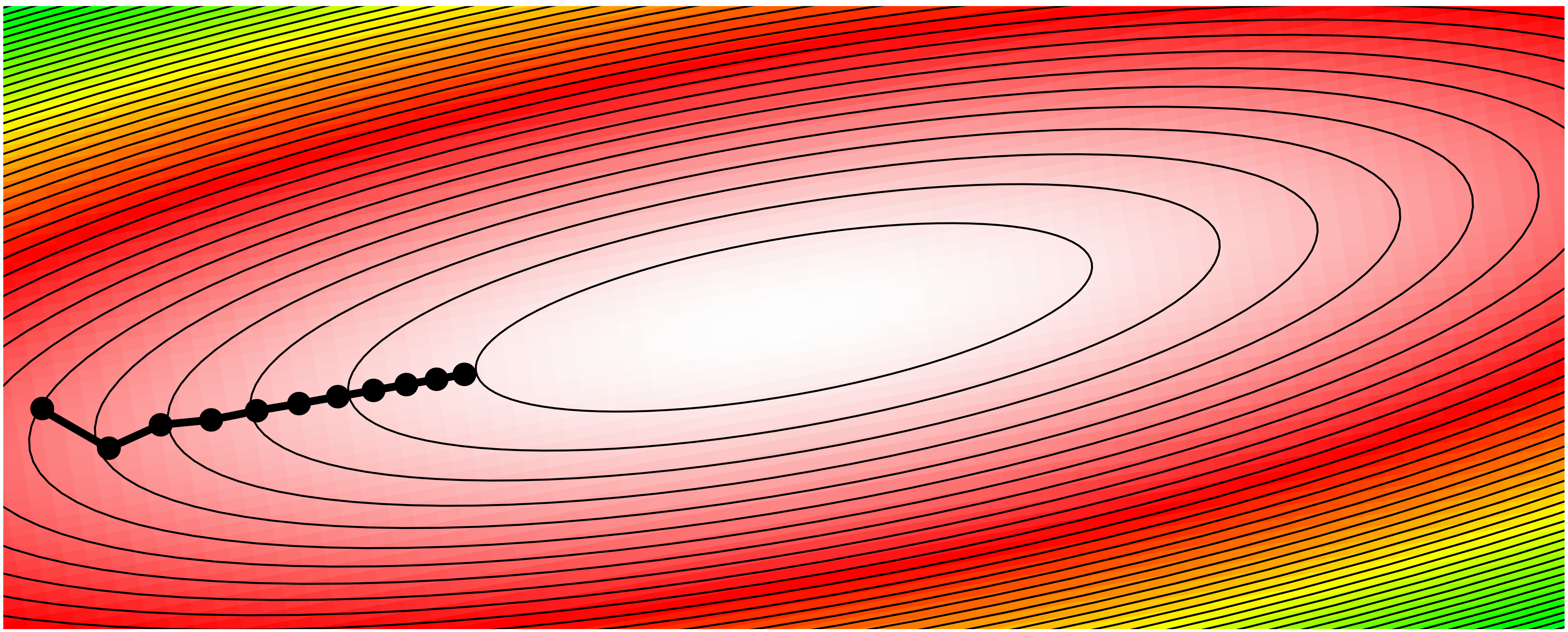
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way

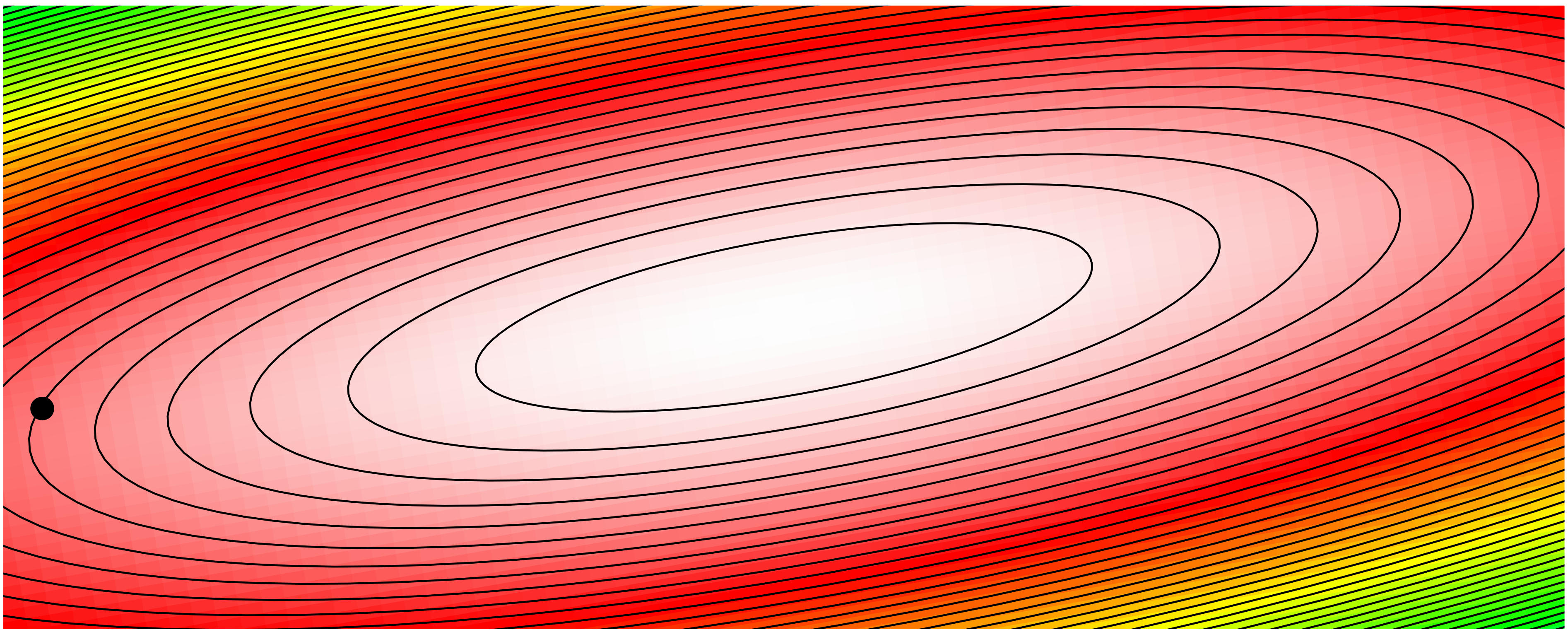


Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum

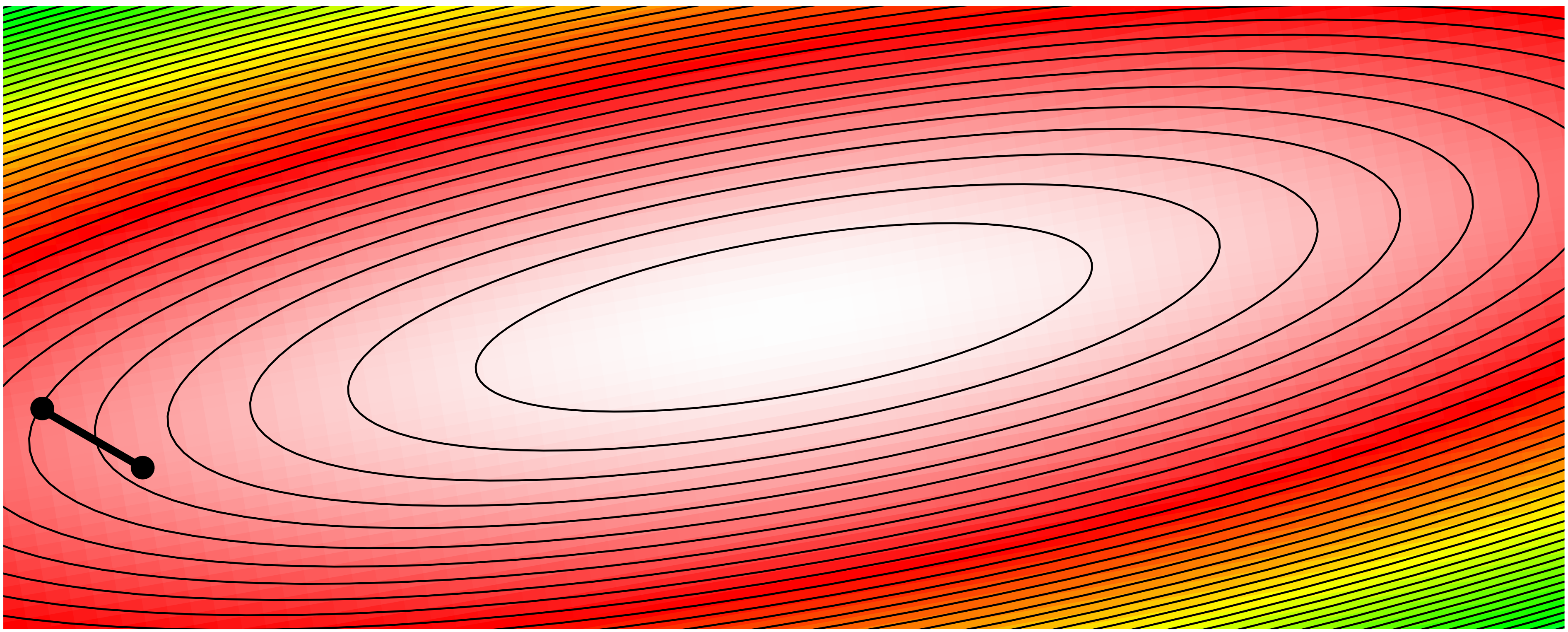
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



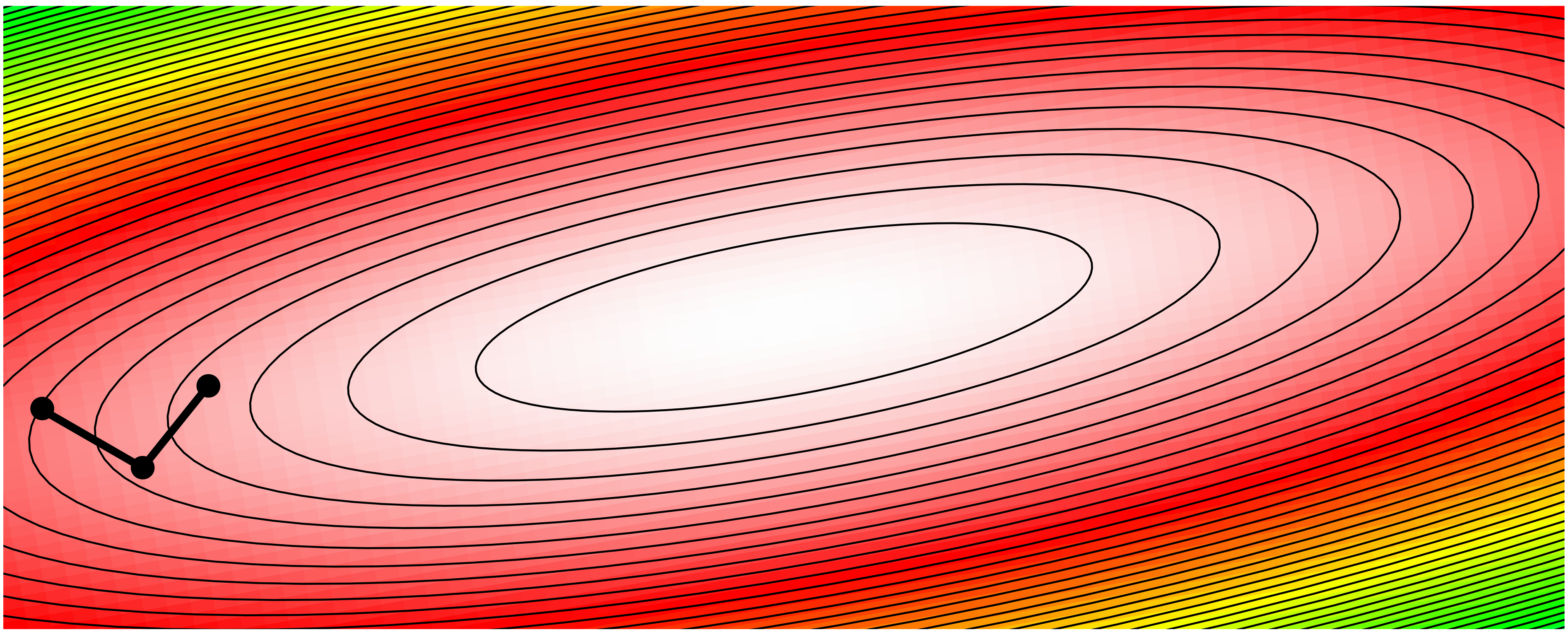
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



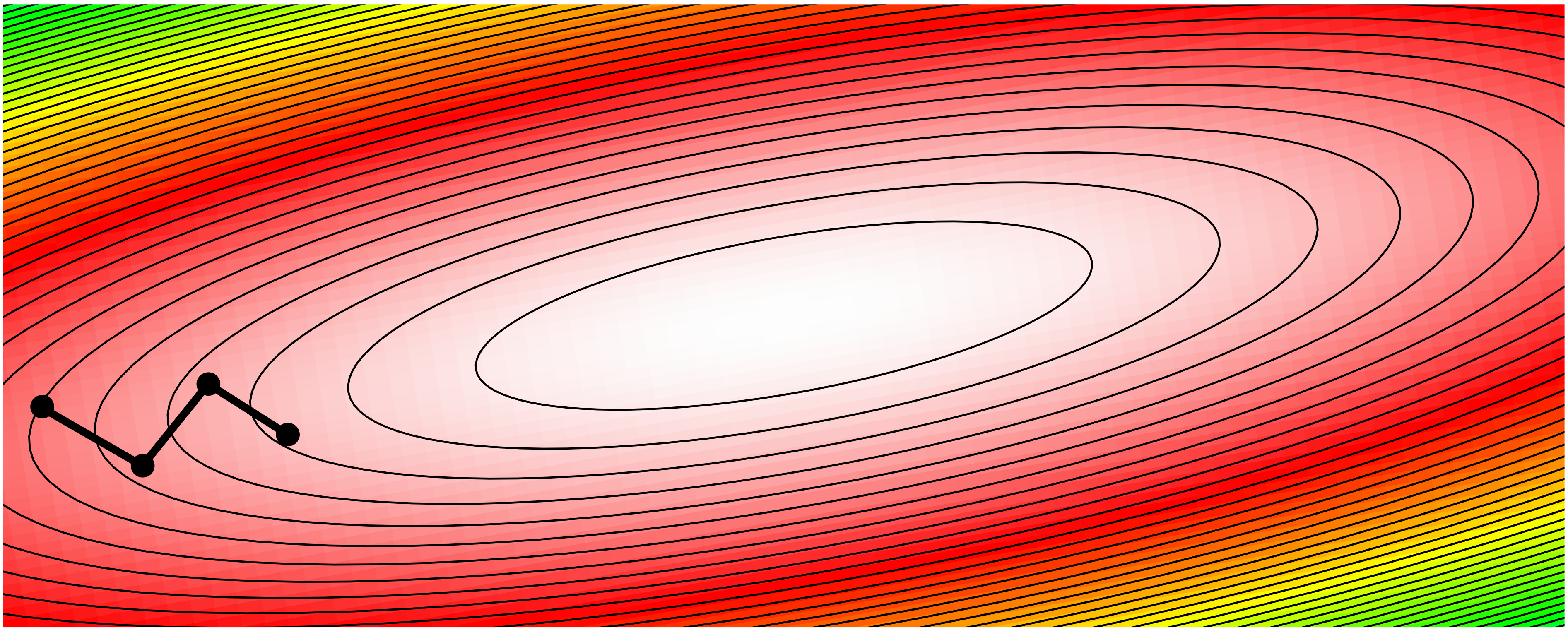
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



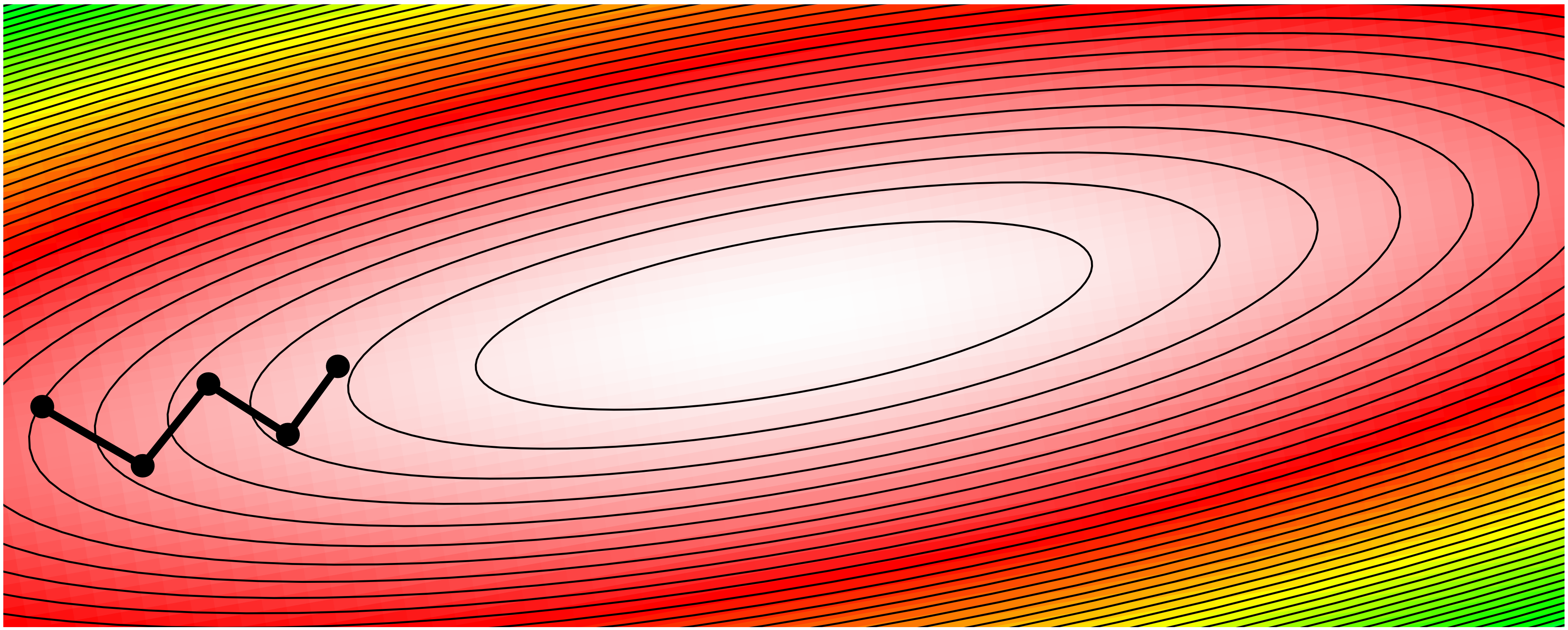
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



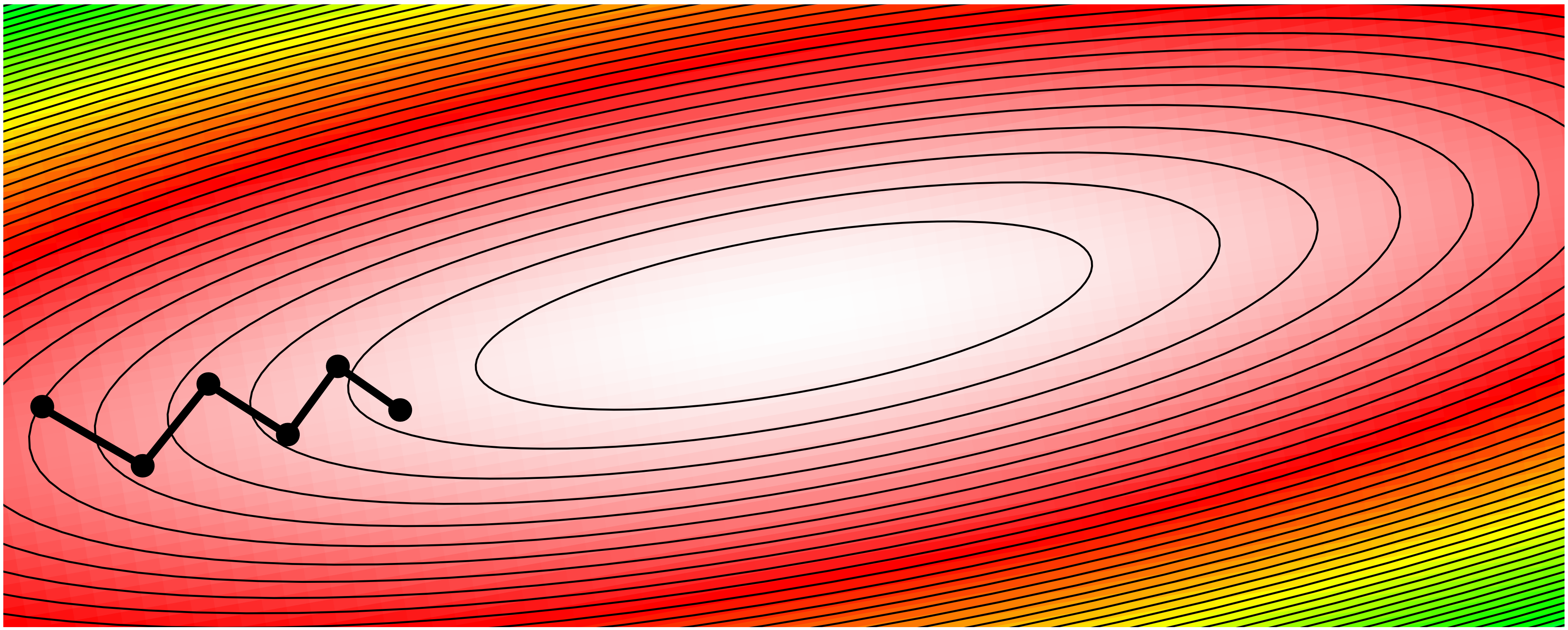
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



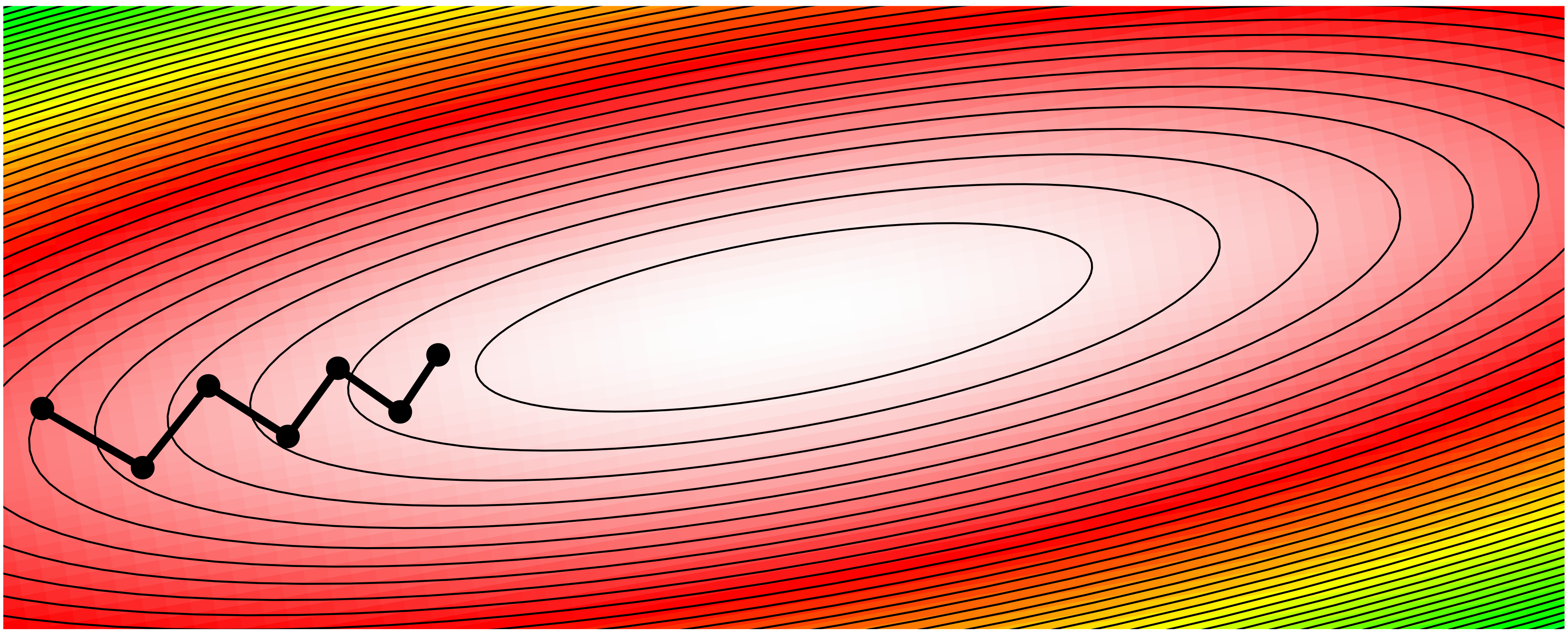
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



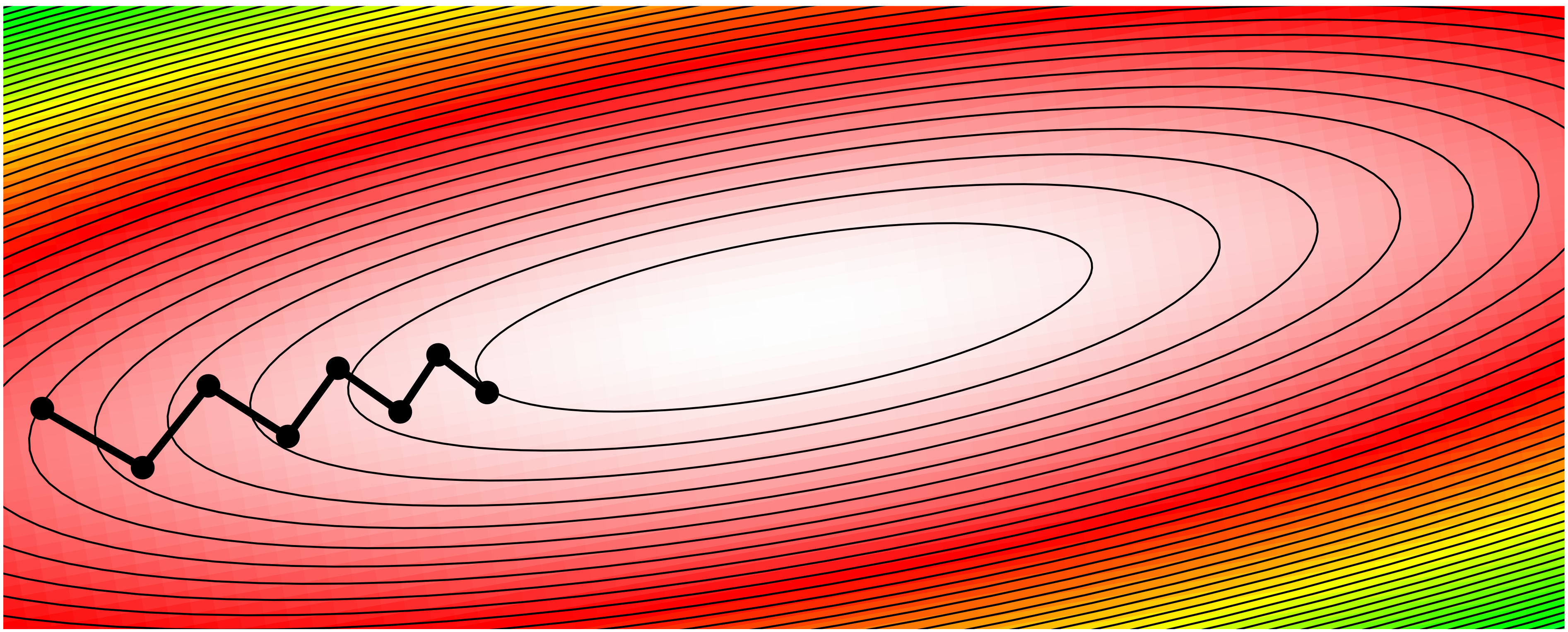
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



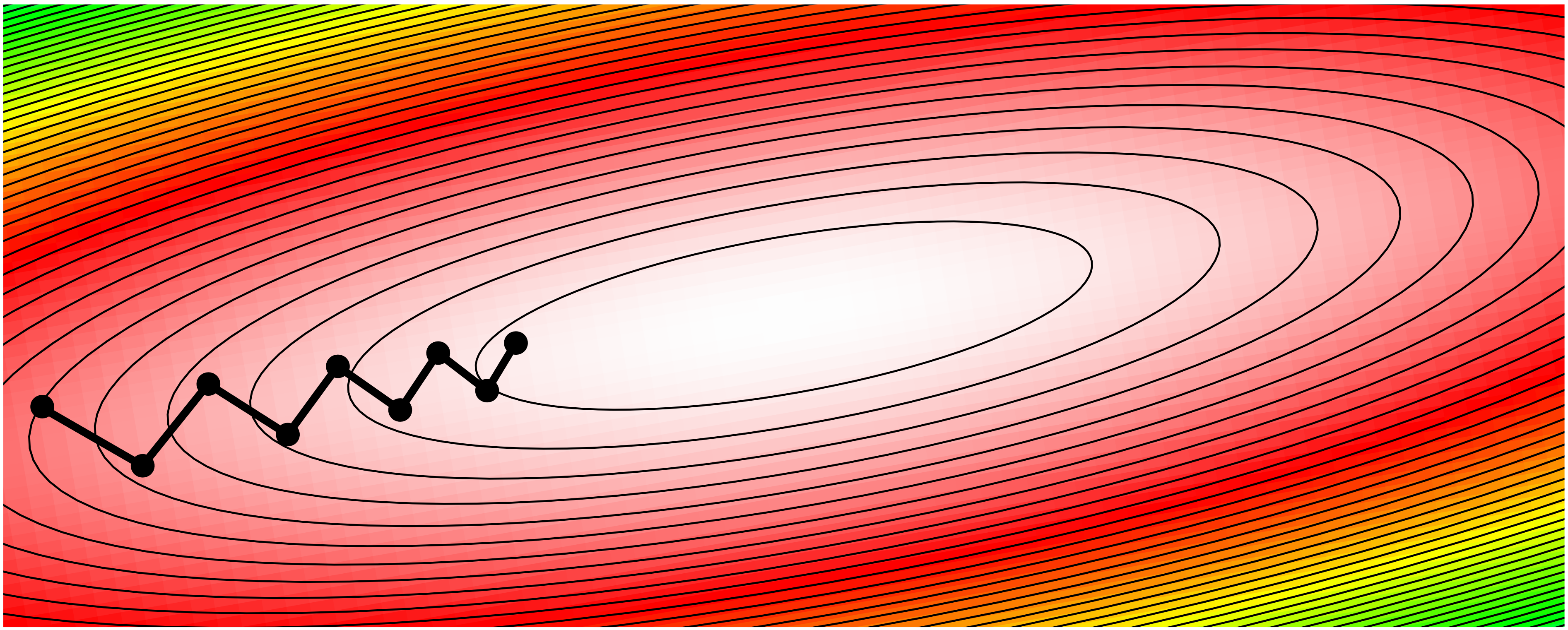
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



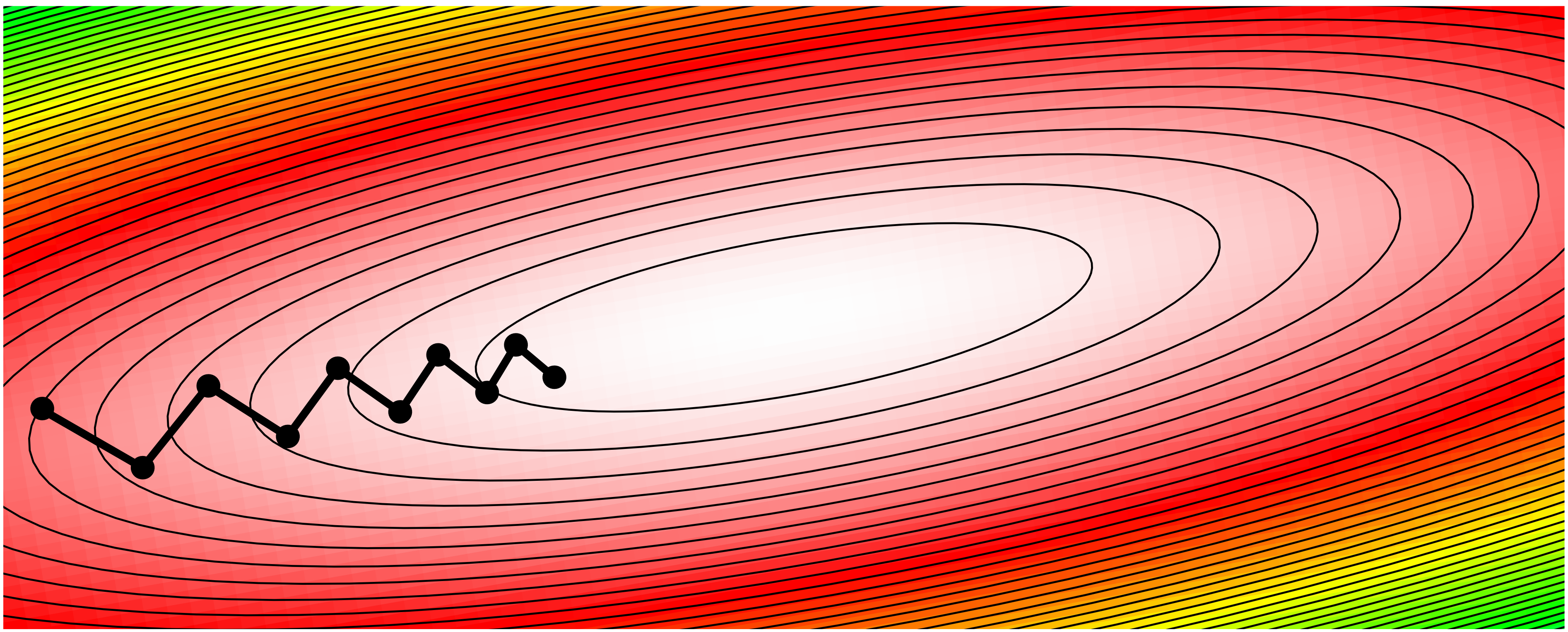
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



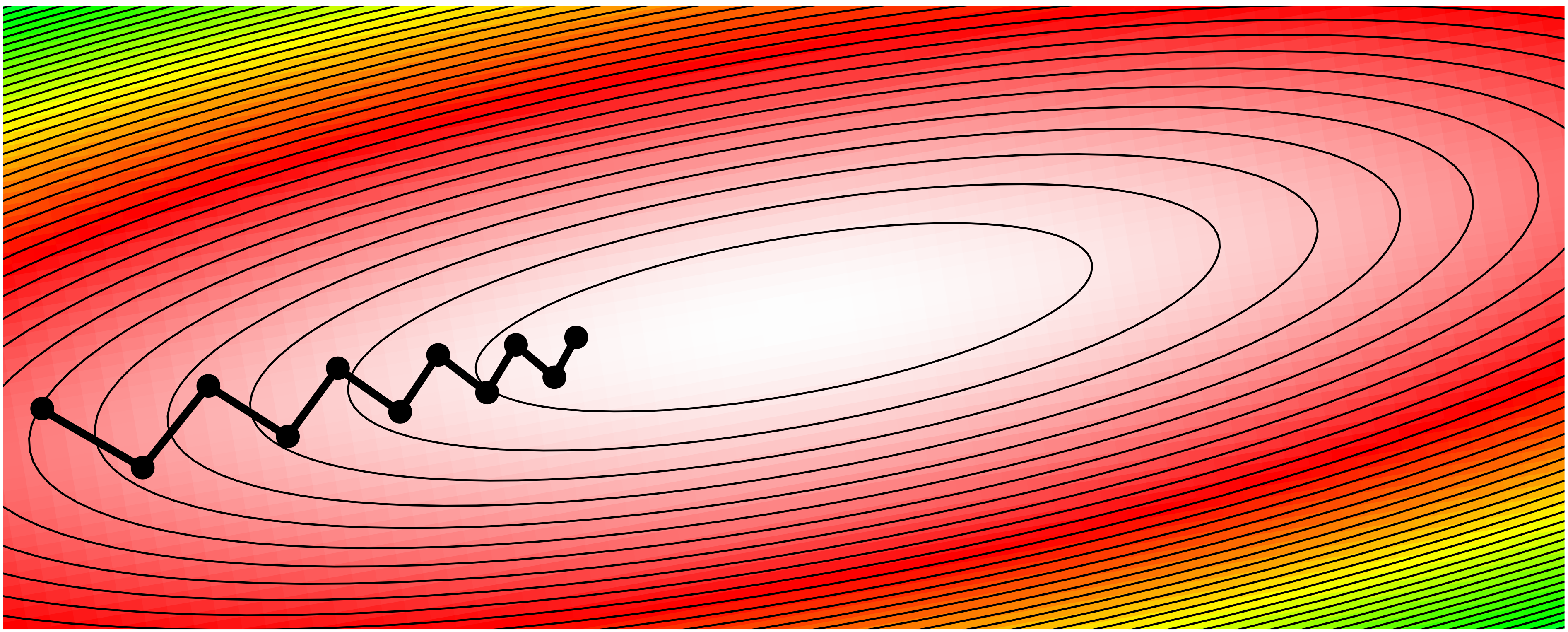
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum

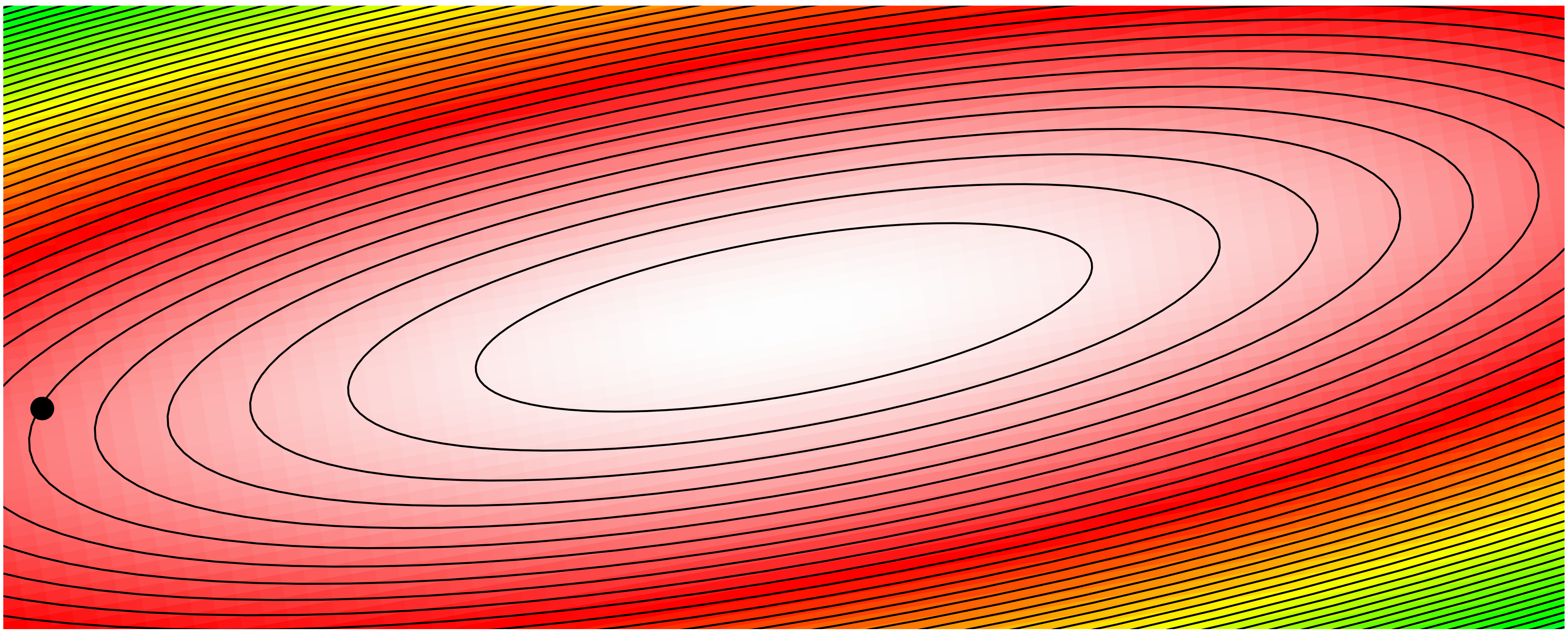


More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .

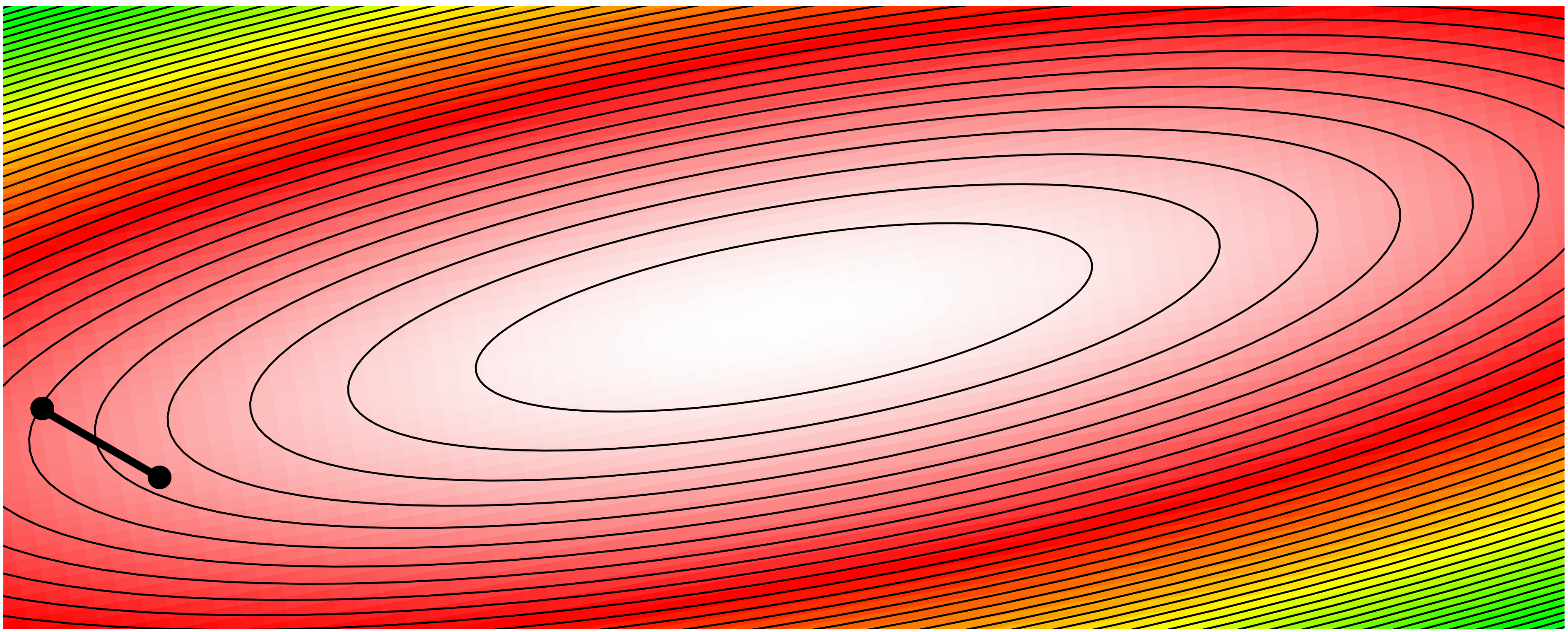
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



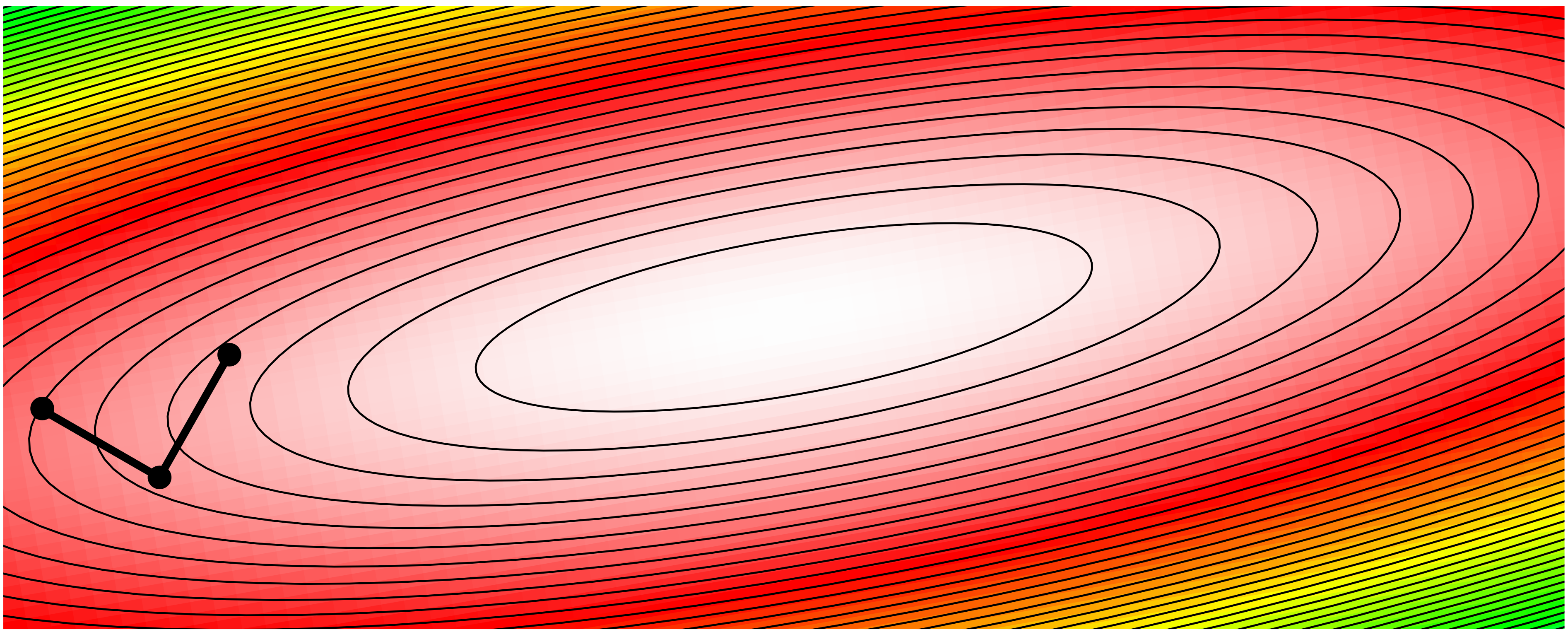
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



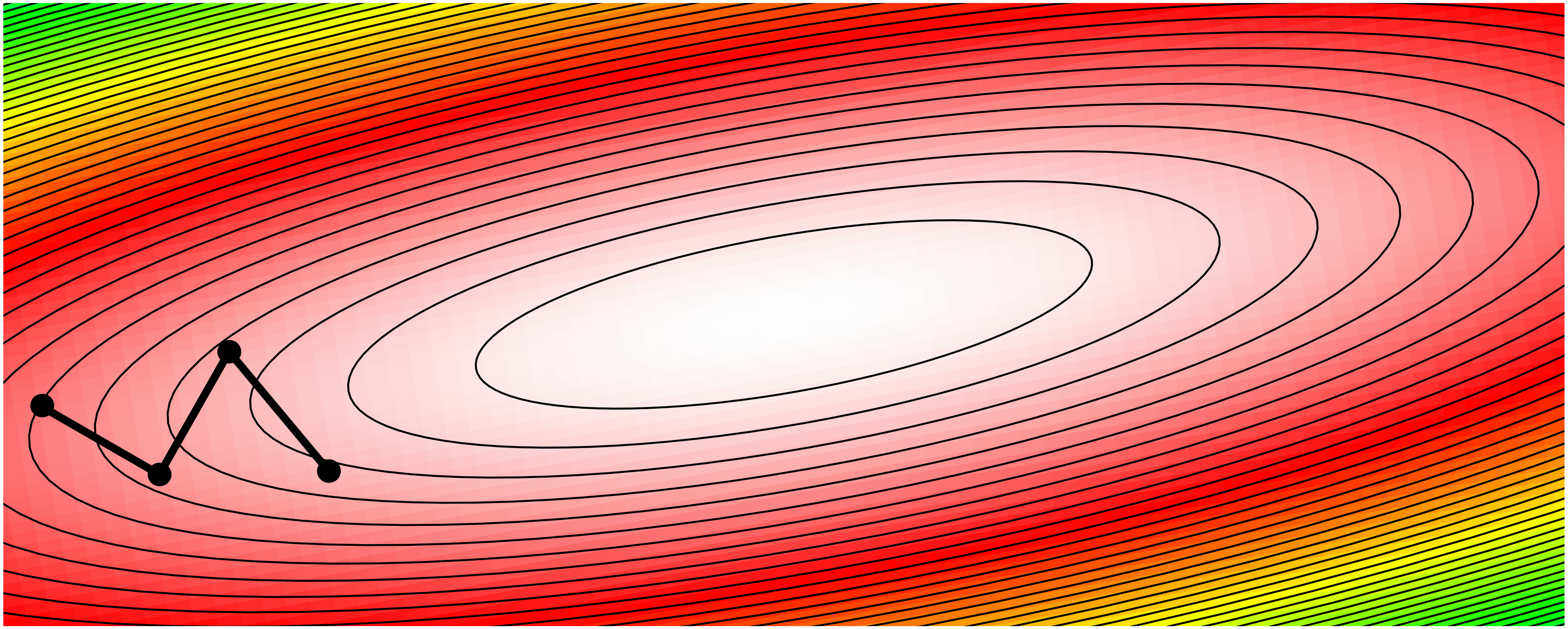
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



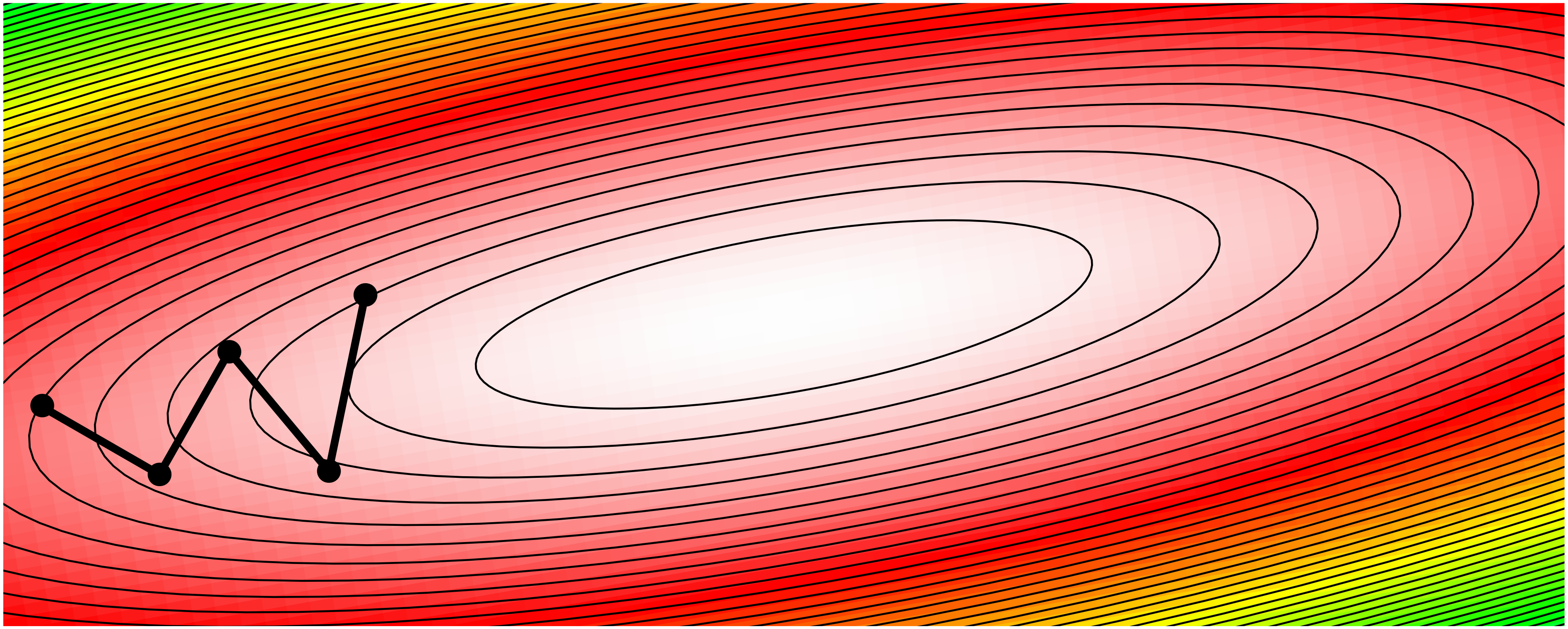
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



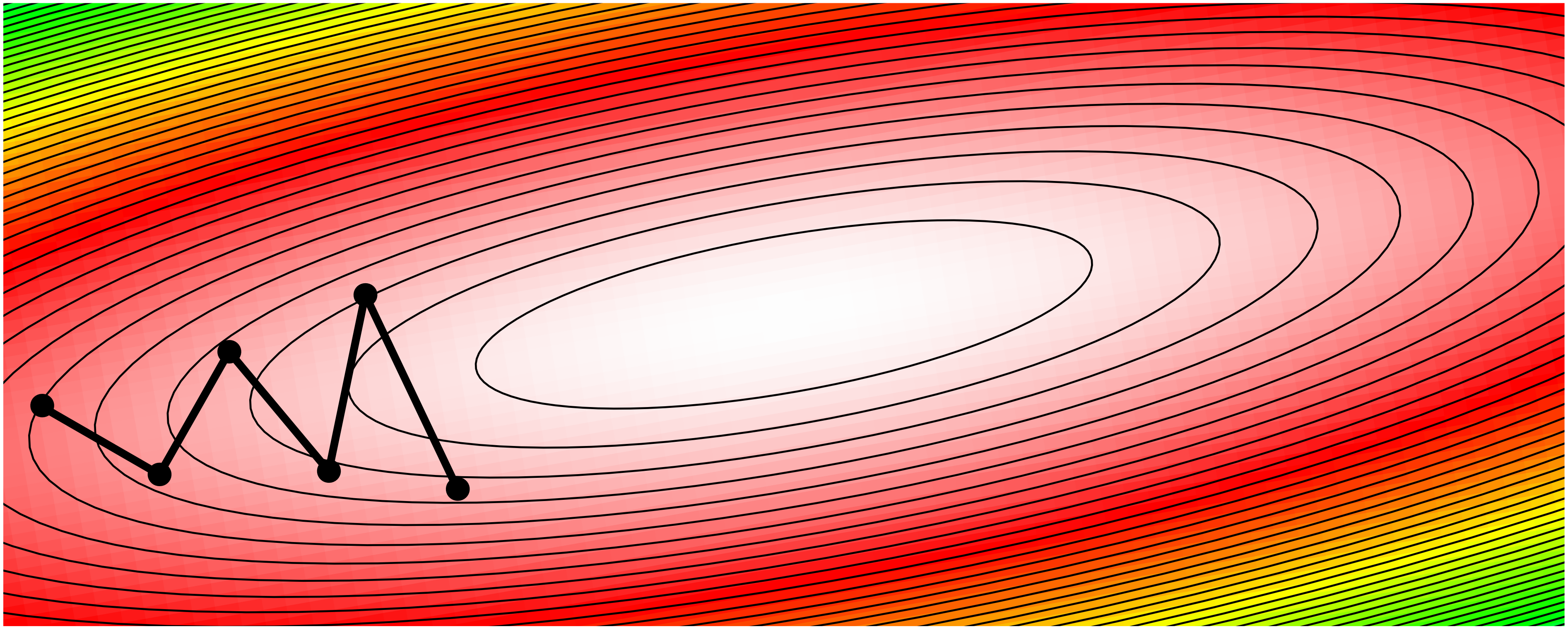
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



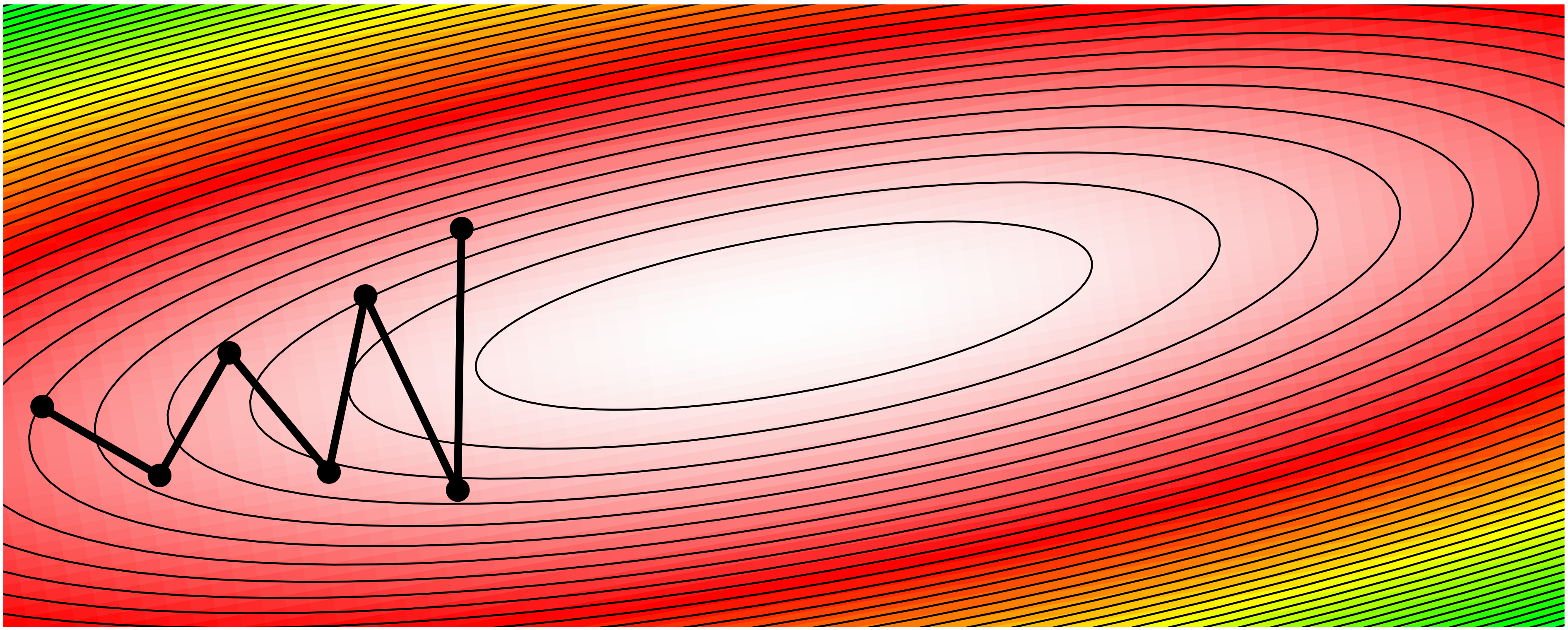
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



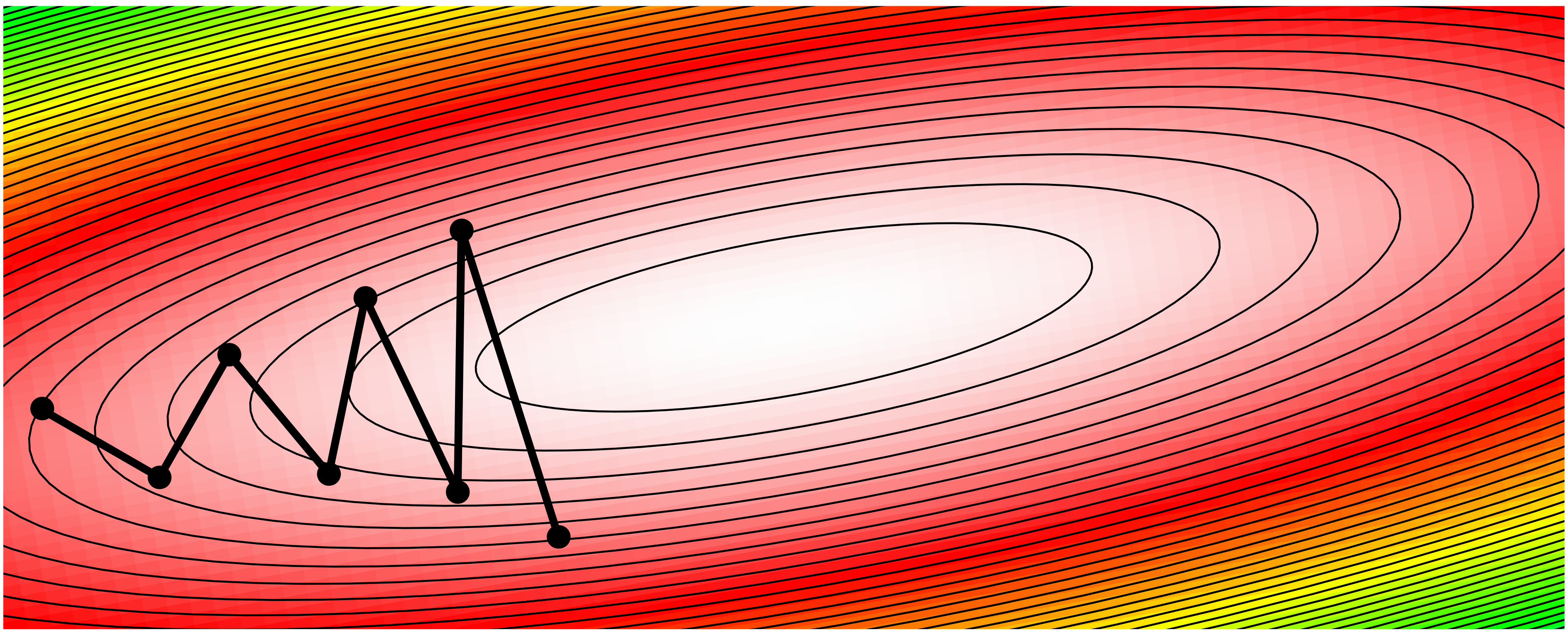
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



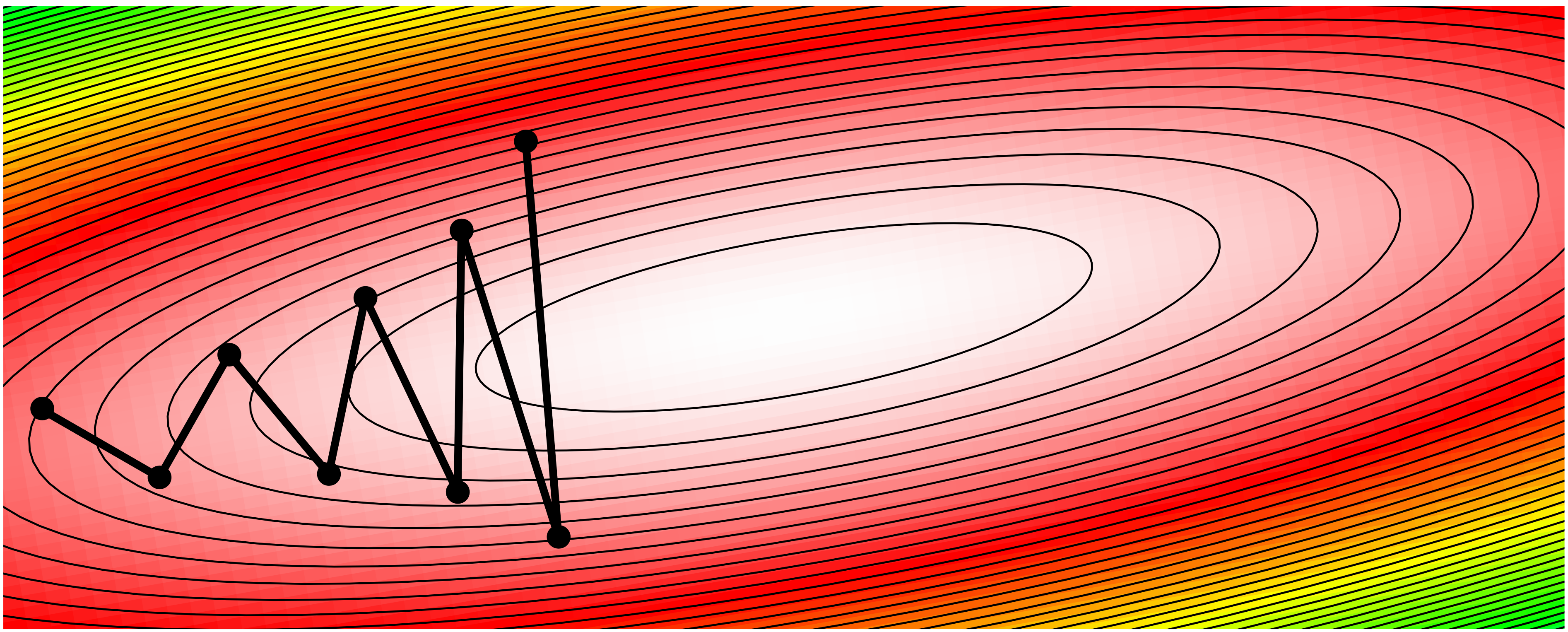
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



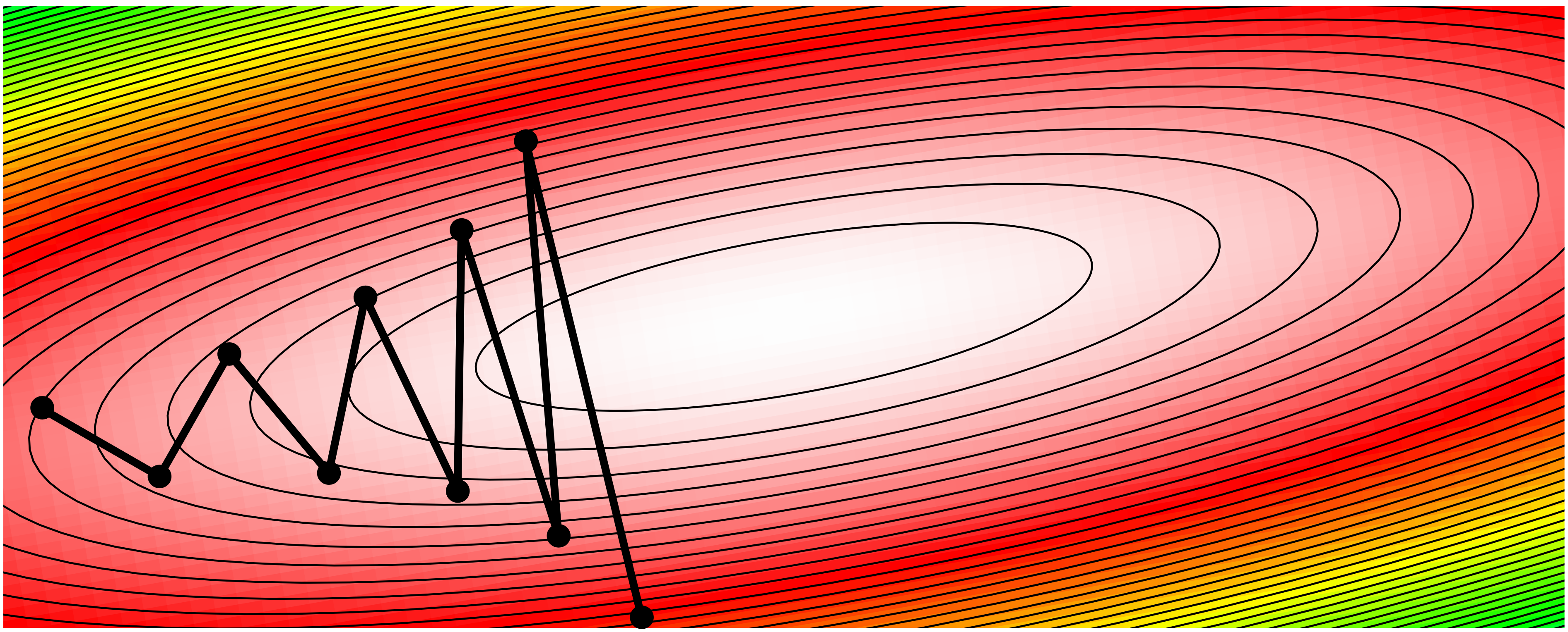
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



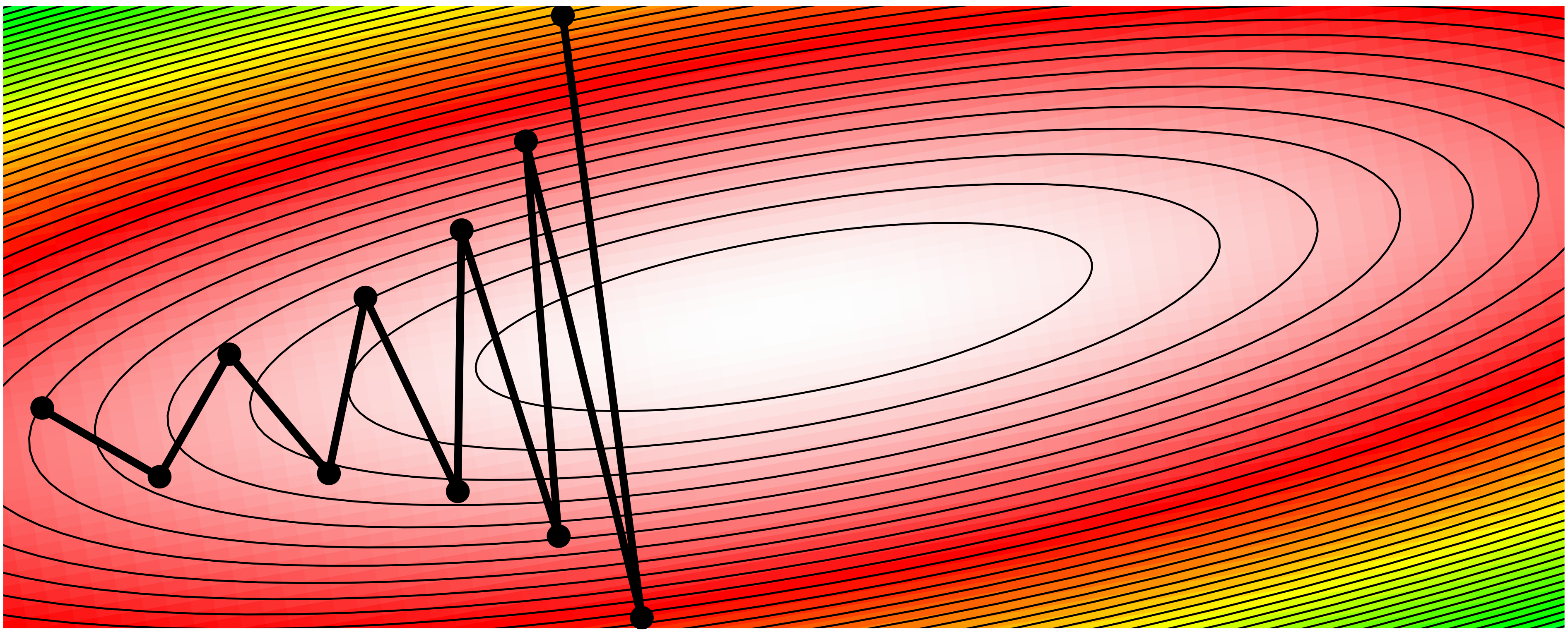
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .

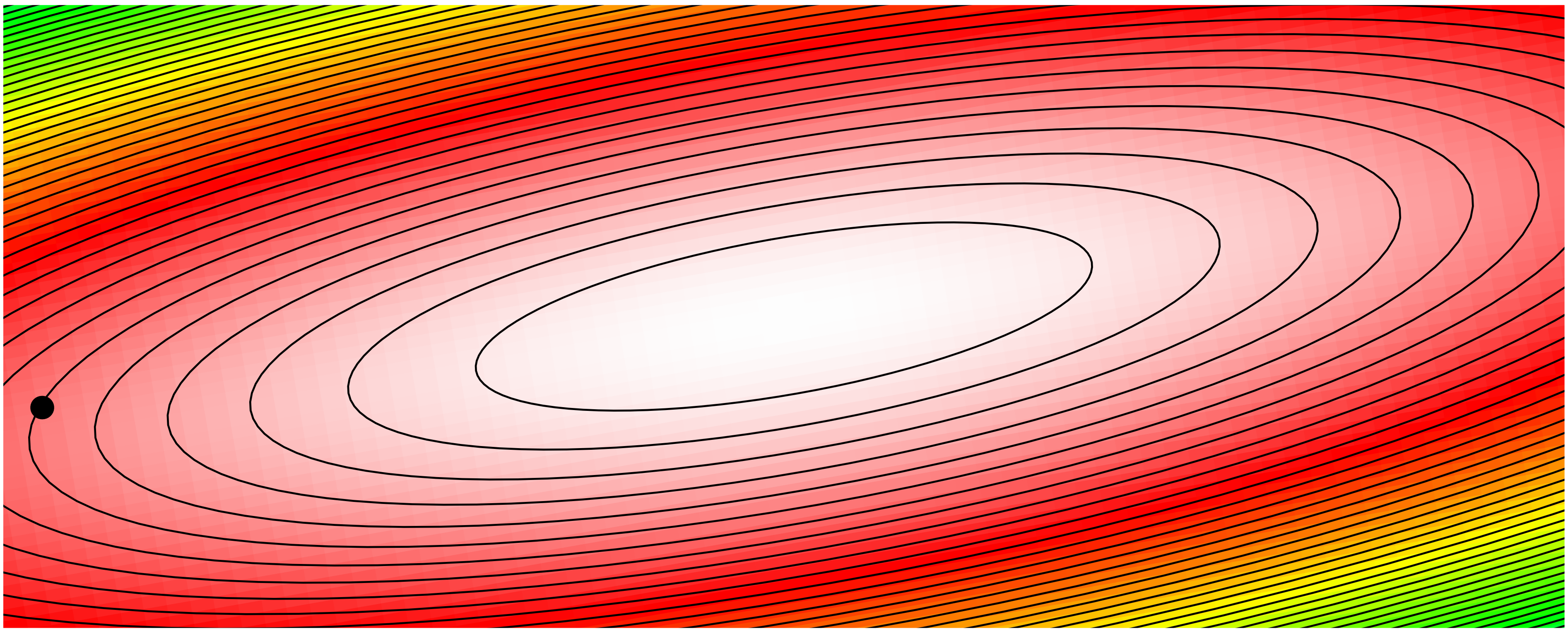


Line Minimisation

- We can systematically seek the minimum along a line of the gradient

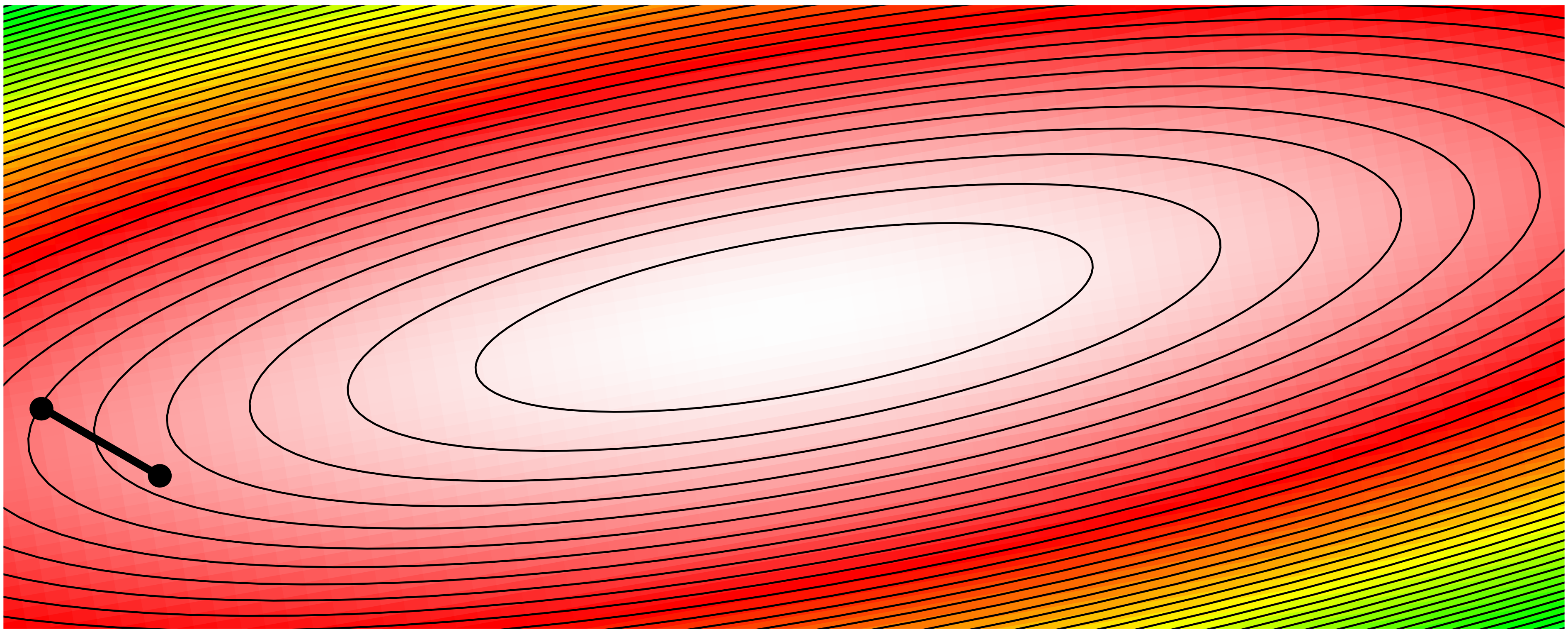
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



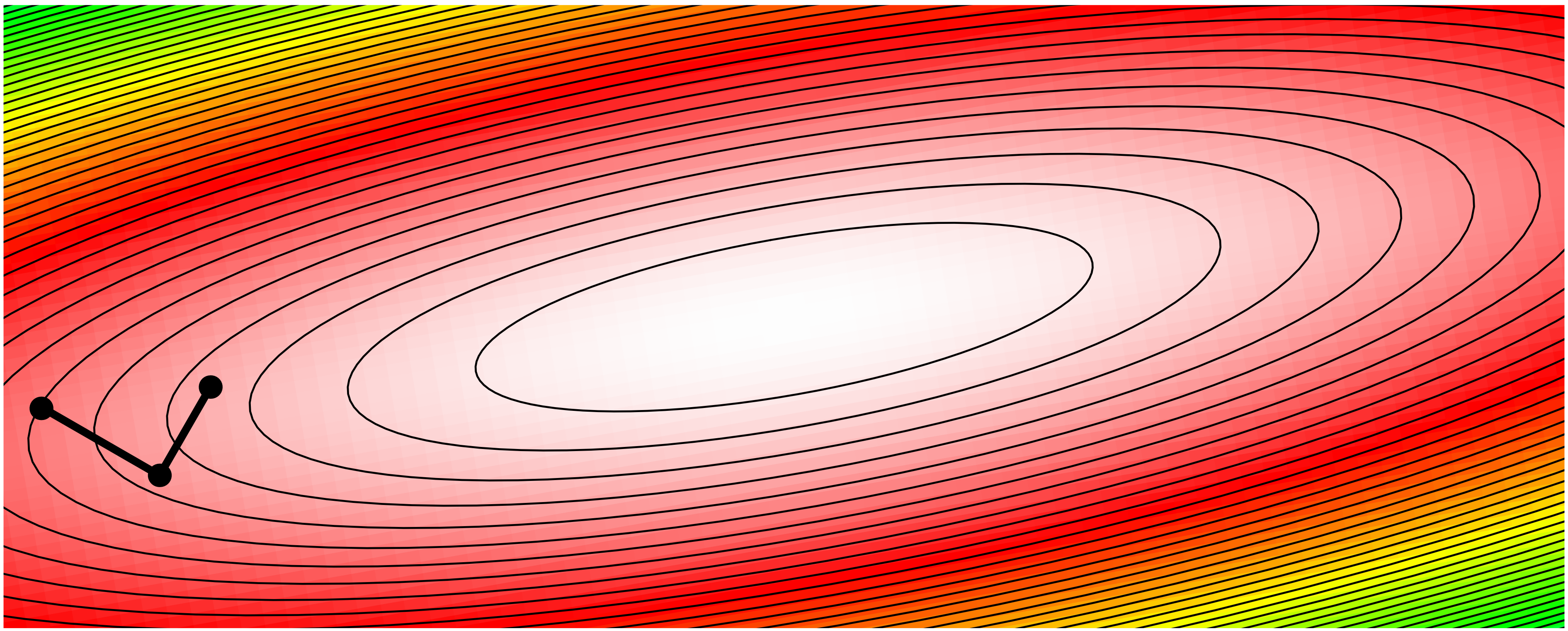
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



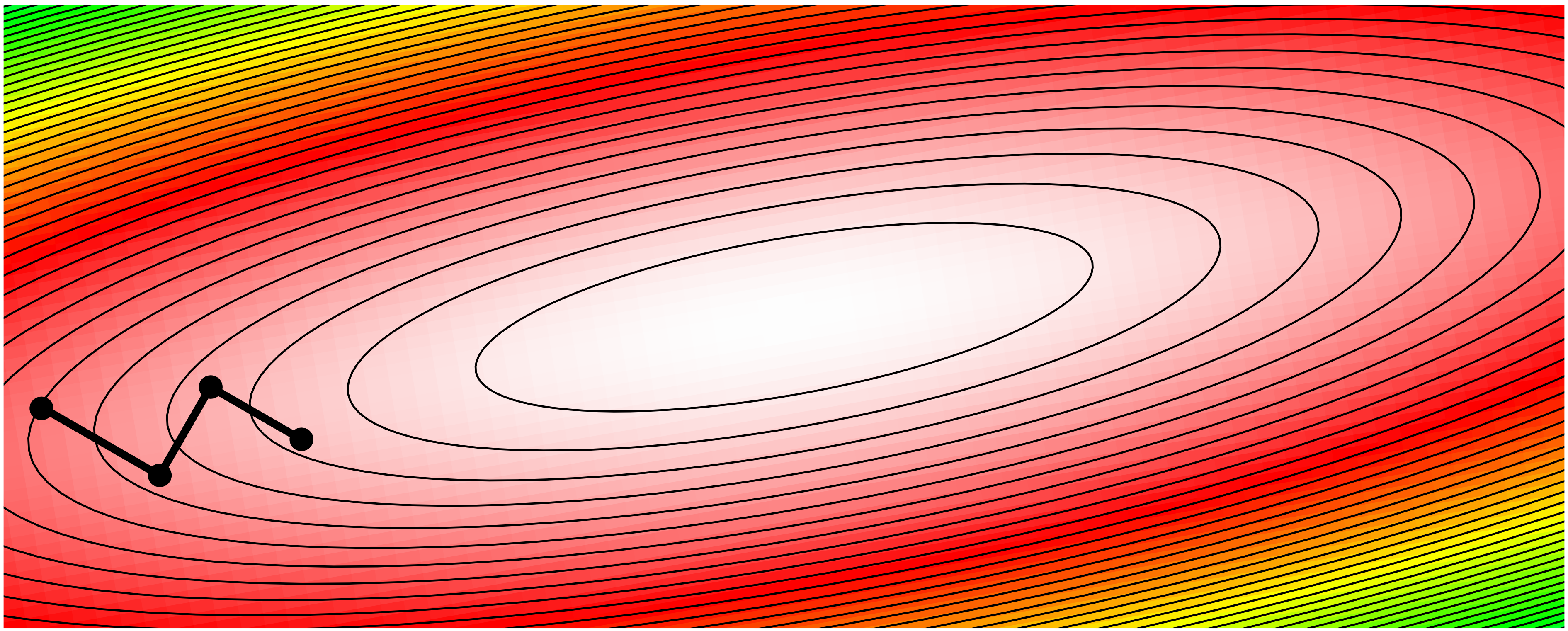
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



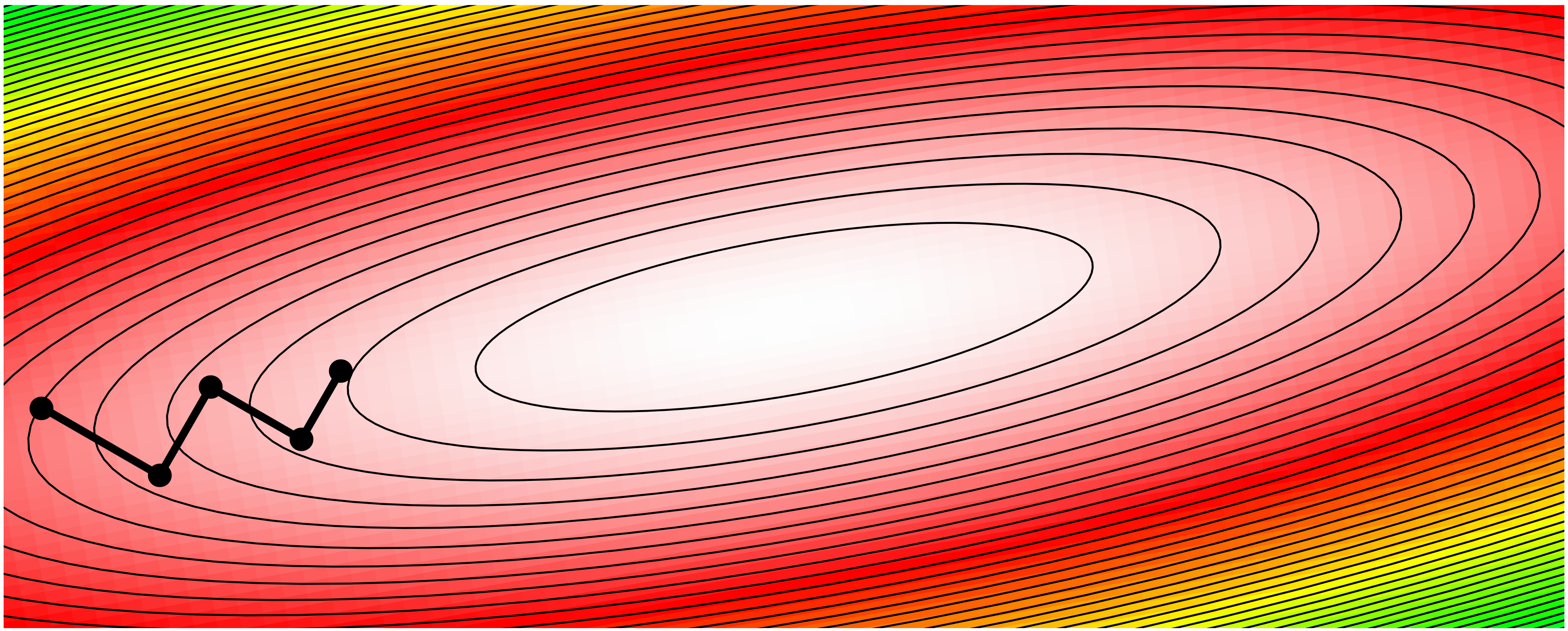
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



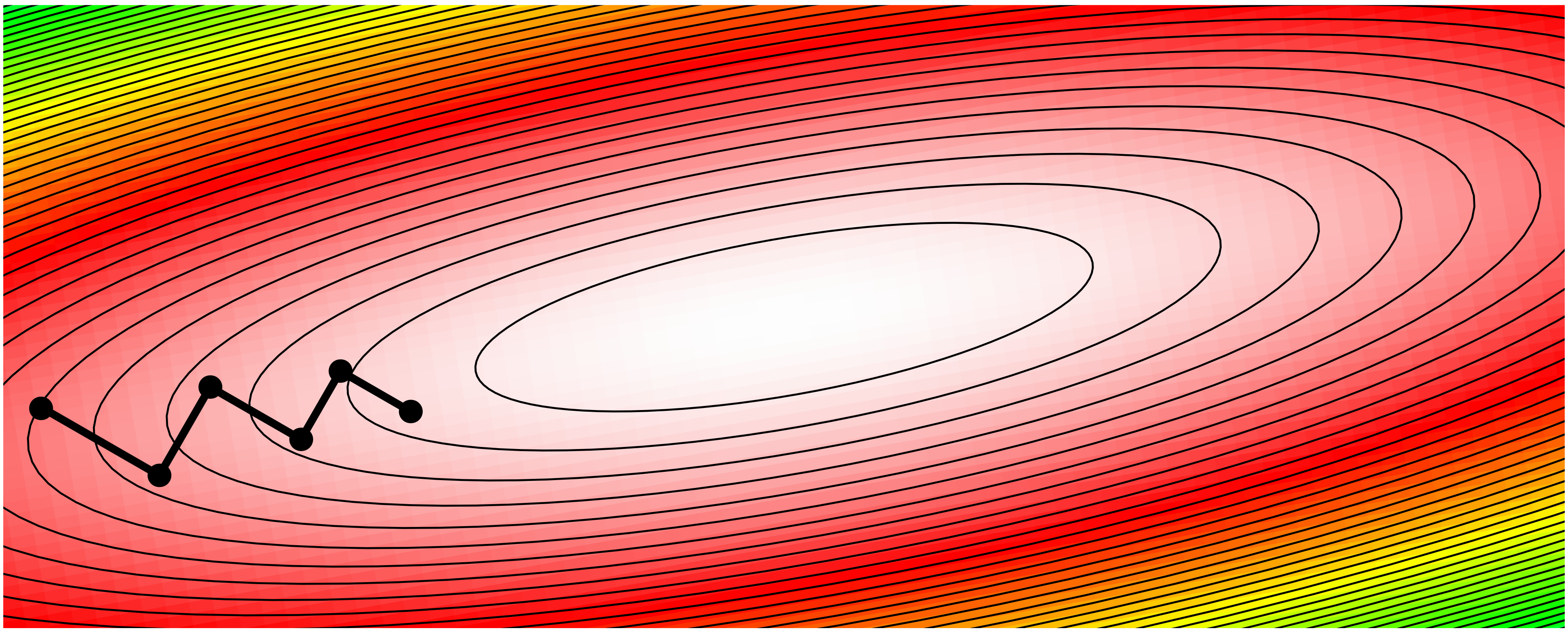
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



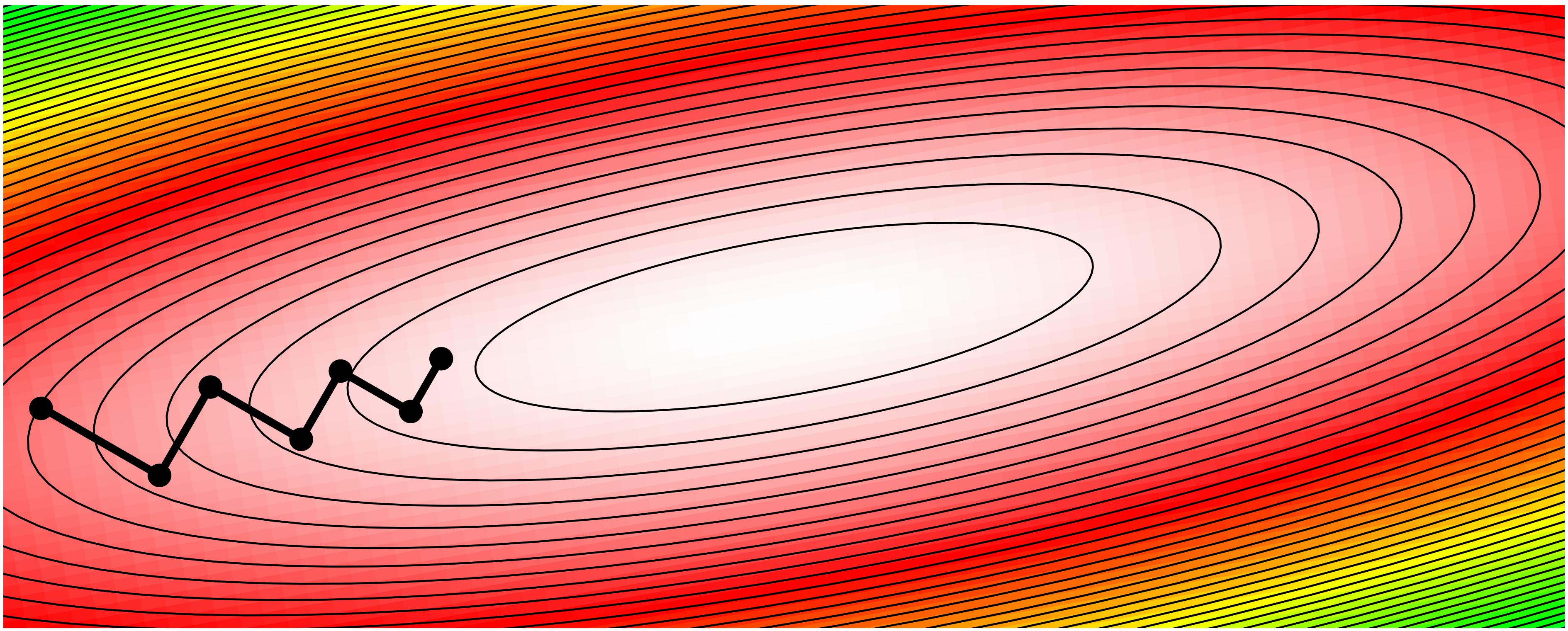
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



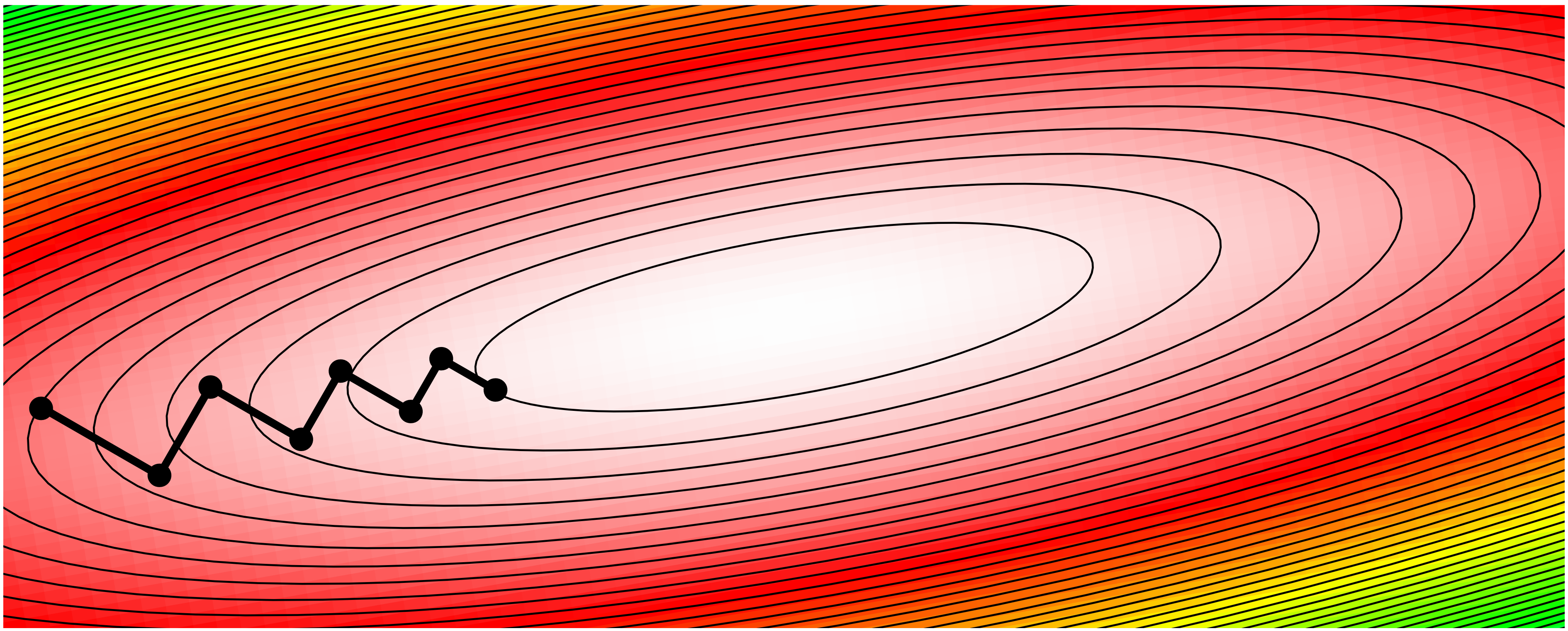
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



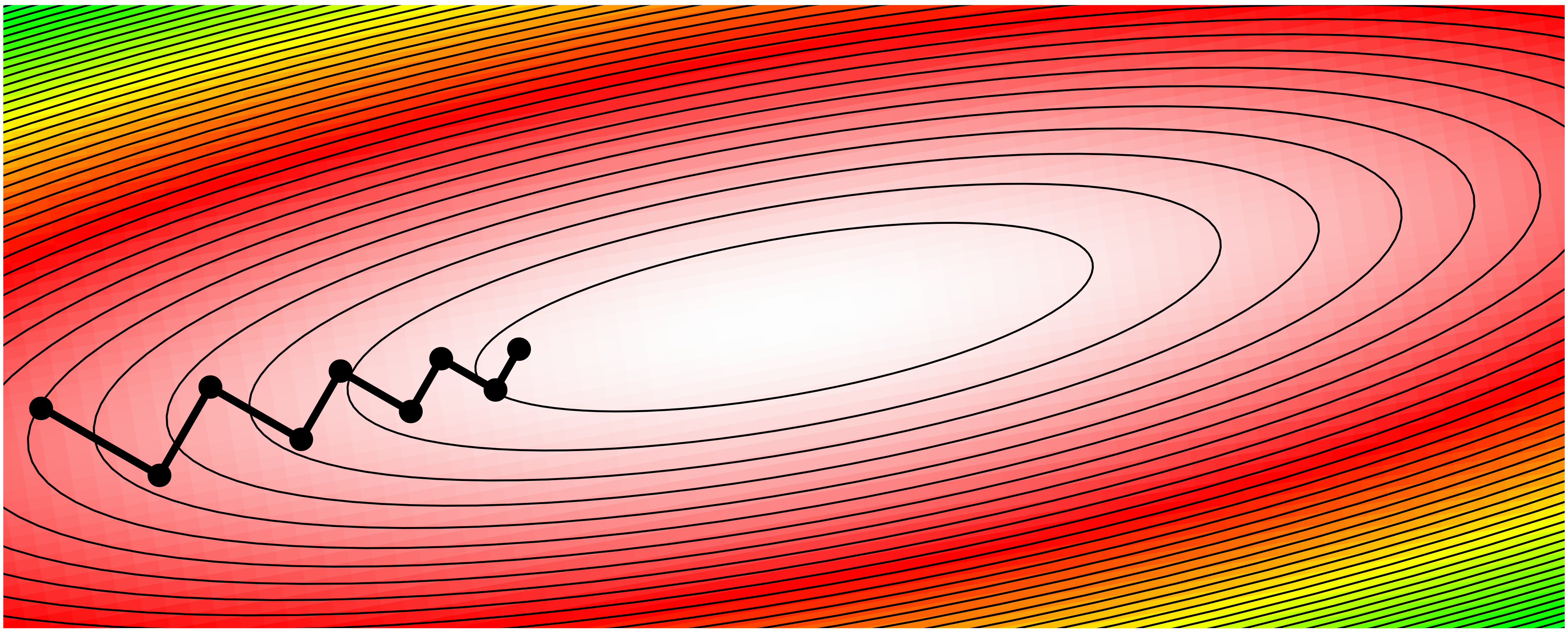
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



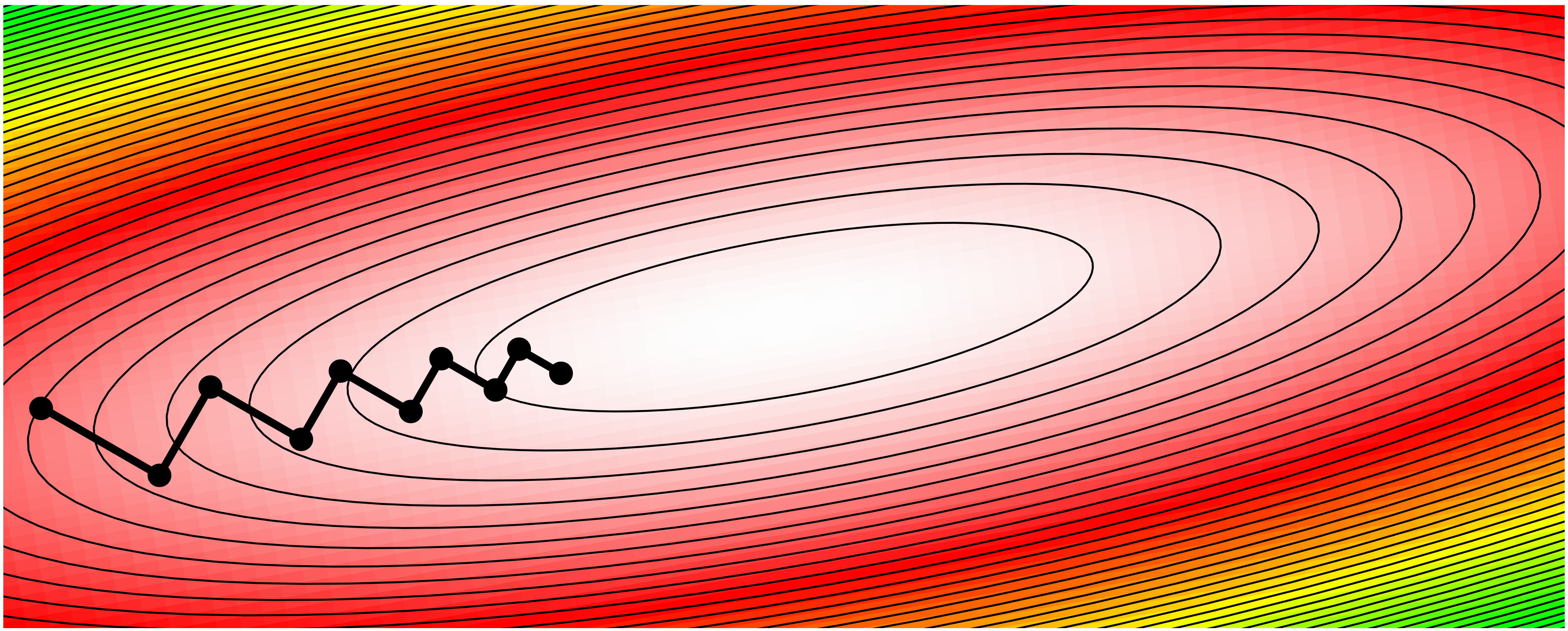
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



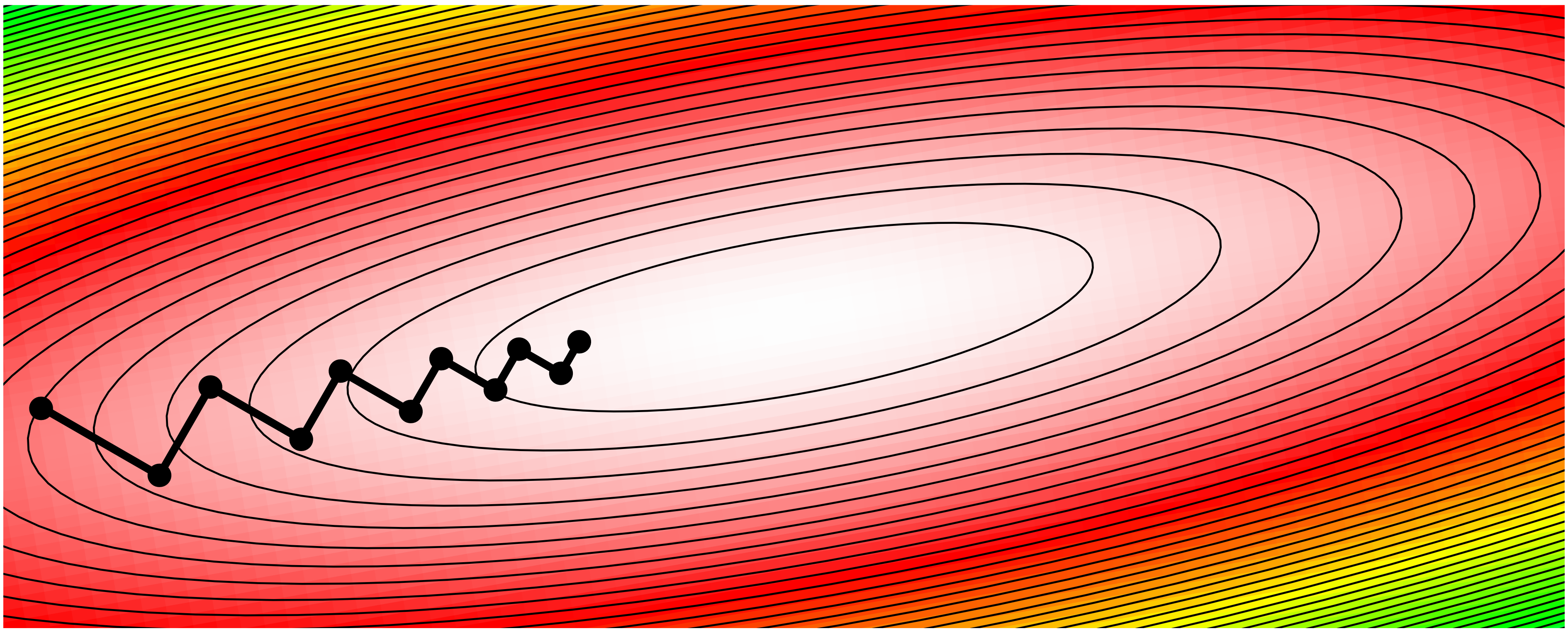
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



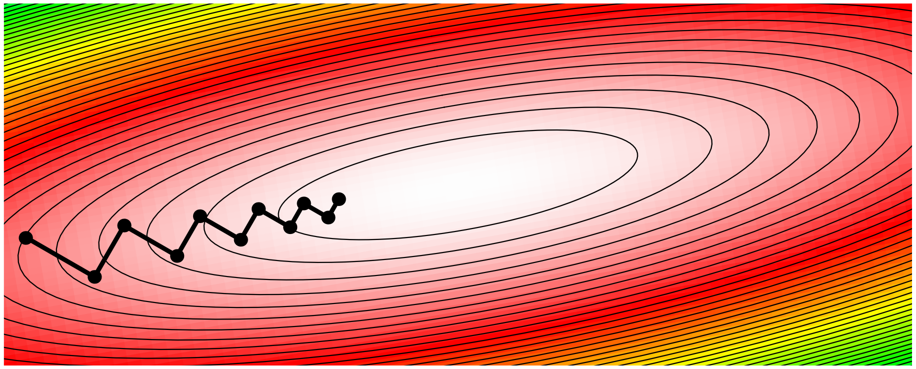
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag

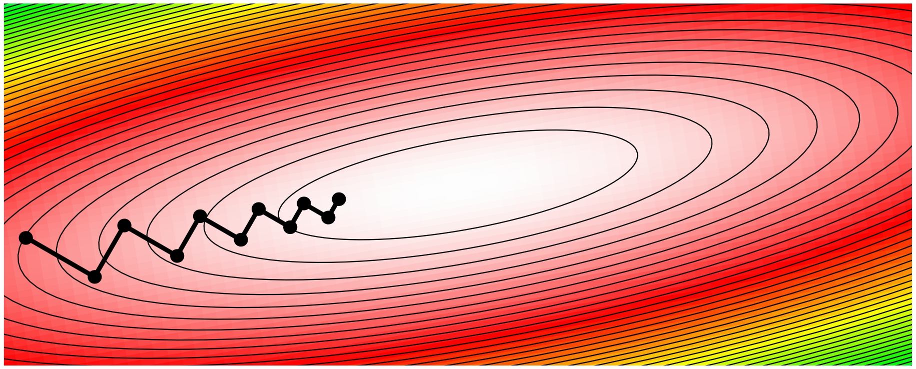


- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential

- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag

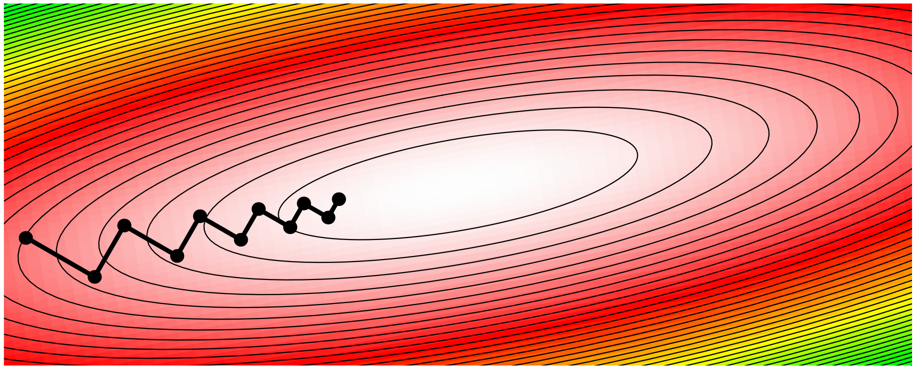


- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential

- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag

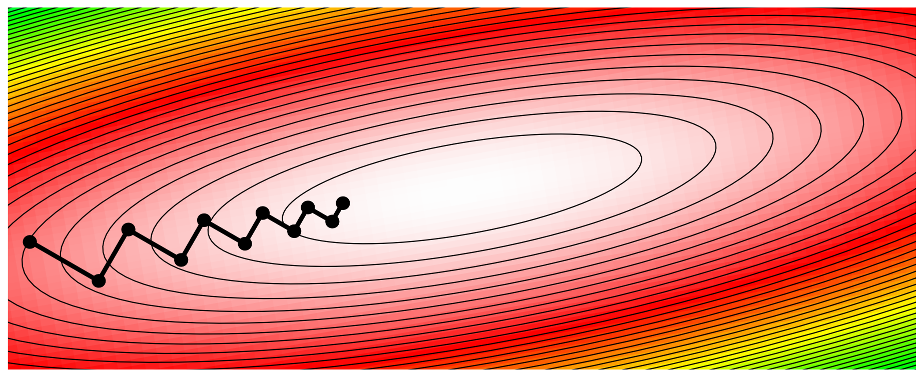


- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential

- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag



- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential

- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian

- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps

- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Levenberg-Marquardt

- Want to minimise $\|\boldsymbol{\epsilon}(\boldsymbol{w})\|^2$ where $\epsilon_i(\boldsymbol{w}) = f(\boldsymbol{x}_i|\boldsymbol{w}) - y_i$

- Use linear approximation

$$\epsilon_i(\boldsymbol{w}) \approx \epsilon_i(\boldsymbol{w}^{(k)}) + (\boldsymbol{w} - \boldsymbol{w}^{(k)})\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)})$$

with $\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)}) = \boldsymbol{\nabla}f(\boldsymbol{x}_i|\boldsymbol{w}^{(k)})$

- Solve quadratic minimisation of approximate error $\operatorname{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w})$ with $\mathbf{J} = \boldsymbol{\nabla}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$

$$\begin{aligned}
L_{approx}(\boldsymbol{w}) &= \|\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + \mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)})\|^2 \\
&= \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + 2(\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) \\
&\quad + (\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)})
\end{aligned}$$

Levenberg-Marquardt

- Want to minimise $\|\boldsymbol{\epsilon}(\boldsymbol{w})\|^2$ where $\epsilon_i(\boldsymbol{w}) = f(\boldsymbol{x}_i|\boldsymbol{w}) - y_i$

- Use linear approximation

$$\epsilon_i(\boldsymbol{w}) \approx \epsilon_i(\boldsymbol{w}^{(k)}) + (\boldsymbol{w} - \boldsymbol{w}^{(k)}) \boldsymbol{\nabla} \epsilon_i(\boldsymbol{w}^{(k)})$$

  with $\boldsymbol{\nabla} \epsilon_i(\boldsymbol{w}^{(k)}) = \boldsymbol{\nabla} f(\boldsymbol{x}_i|\boldsymbol{w}^{(k)})$

- Solve quadratic minimisation of approximate error $\mathrm{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w})$ with $\mathbf{J} = \boldsymbol{\nabla}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$

$$\begin{aligned} L_{approx}(\boldsymbol{w}) &= \|\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + \mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)})\|^2 \\ &= \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})^{\mathsf{T}} \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + 2(\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}} \mathbf{J}^{\mathsf{T}} \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) \\ &\quad + (\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}} \mathbf{J}^{\mathsf{T}} \mathbf{J} (\boldsymbol{w} - \boldsymbol{w}^{(k)}) \end{aligned}$$

Levenberg-Marquardt

- Want to minimise $\|\boldsymbol{\epsilon}(\boldsymbol{w})\|^2$ where $\epsilon_i(\boldsymbol{w}) = f(\boldsymbol{x}_i|\boldsymbol{w}) - y_i$
- Use linear approximation

$$\epsilon_i(\boldsymbol{w}) \approx \epsilon_i(\boldsymbol{w}^{(k)}) + (\boldsymbol{w} - \boldsymbol{w}^{(k)})\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)})$$

with $\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)}) = \boldsymbol{\nabla} f(\boldsymbol{x}_i|\boldsymbol{w}^{(k)})$

- Solve quadratic minimisation of approximate error $\mathrm{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w})$ with $\mathbf{J} = \boldsymbol{\nabla}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$

$$\begin{aligned} L_{approx}(\boldsymbol{w}) &= \|\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + \mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)})\|^2 \\ &= \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + 2(\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) \\ &\quad + (\boldsymbol{w} - \boldsymbol{w}^{(k)})^{\mathsf{T}}\mathbf{J}^{\mathsf{T}}\mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)}) \end{aligned}$$

Trust Region

- Solution given by $\boldsymbol{\nabla}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1} \mathbf{J}^{\mathsf{T}} \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction

- Instead use $\boldsymbol{w}^{(k+1)} = \operatorname{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu \|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \nu \mathbf{I}\right)^{-1} \mathbf{J}^{\mathsf{T}} \boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size

- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

Trust Region

- Solution given by $\boldsymbol{\nabla}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction

- Instead use $\boldsymbol{w}^{(k+1)} = \operatorname{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu\|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \nu\mathbf{I}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size

- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

Trust Region

- Solution given by $\boldsymbol{\nabla}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction

- Instead use $\boldsymbol{w}^{(k+1)} = \operatorname{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu\|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \nu\mathbf{I}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size

- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

Trust Region

- Solution given by $\boldsymbol{\nabla}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction

- Instead use $\boldsymbol{w}^{(k+1)} = \operatorname{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu\|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \nu\mathbf{I}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size

- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

Trust Region

- Solution given by $\nabla_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

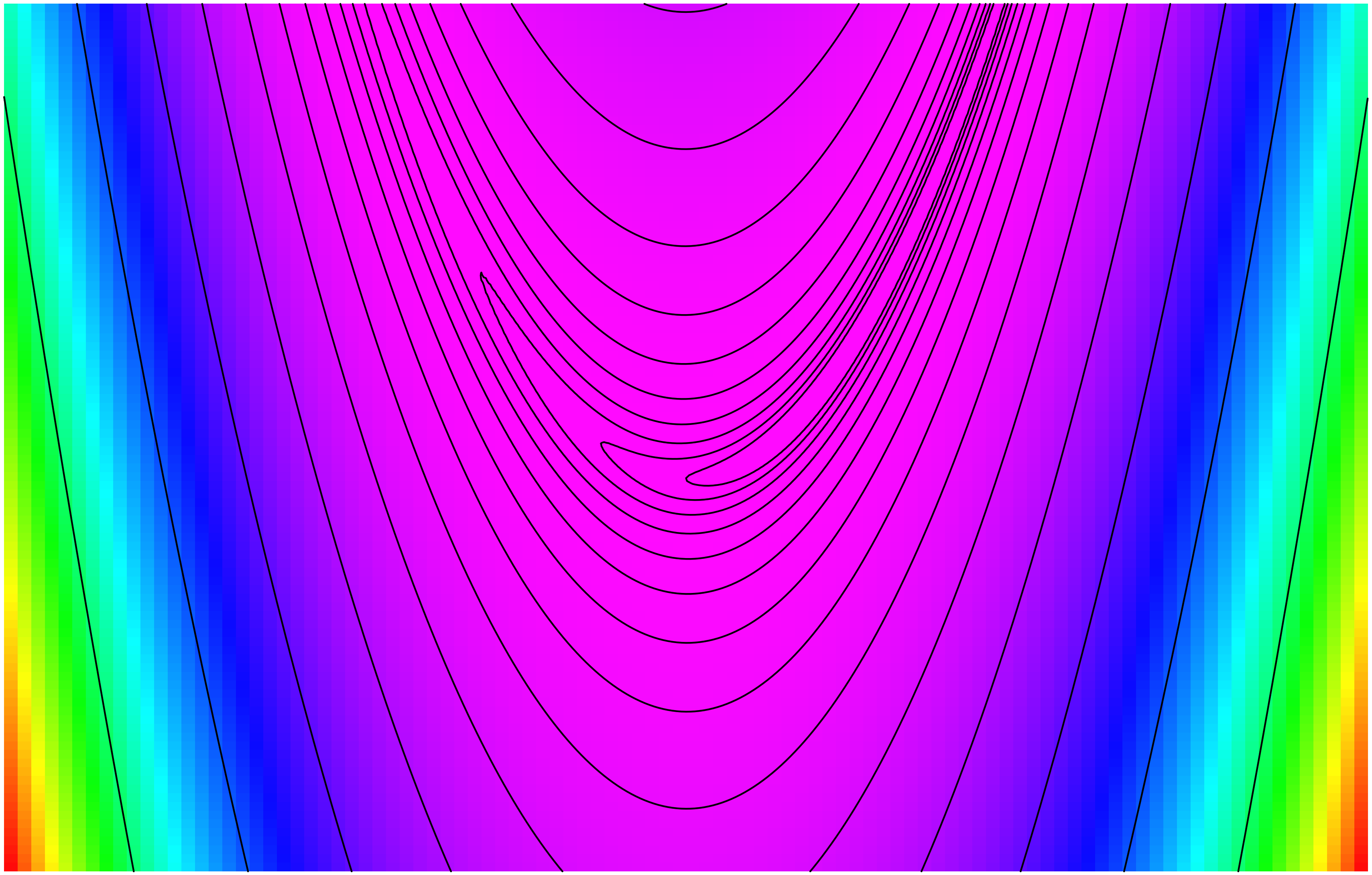
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction
- Instead use $\boldsymbol{w}^{(k+1)} = \mathrm{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu\|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

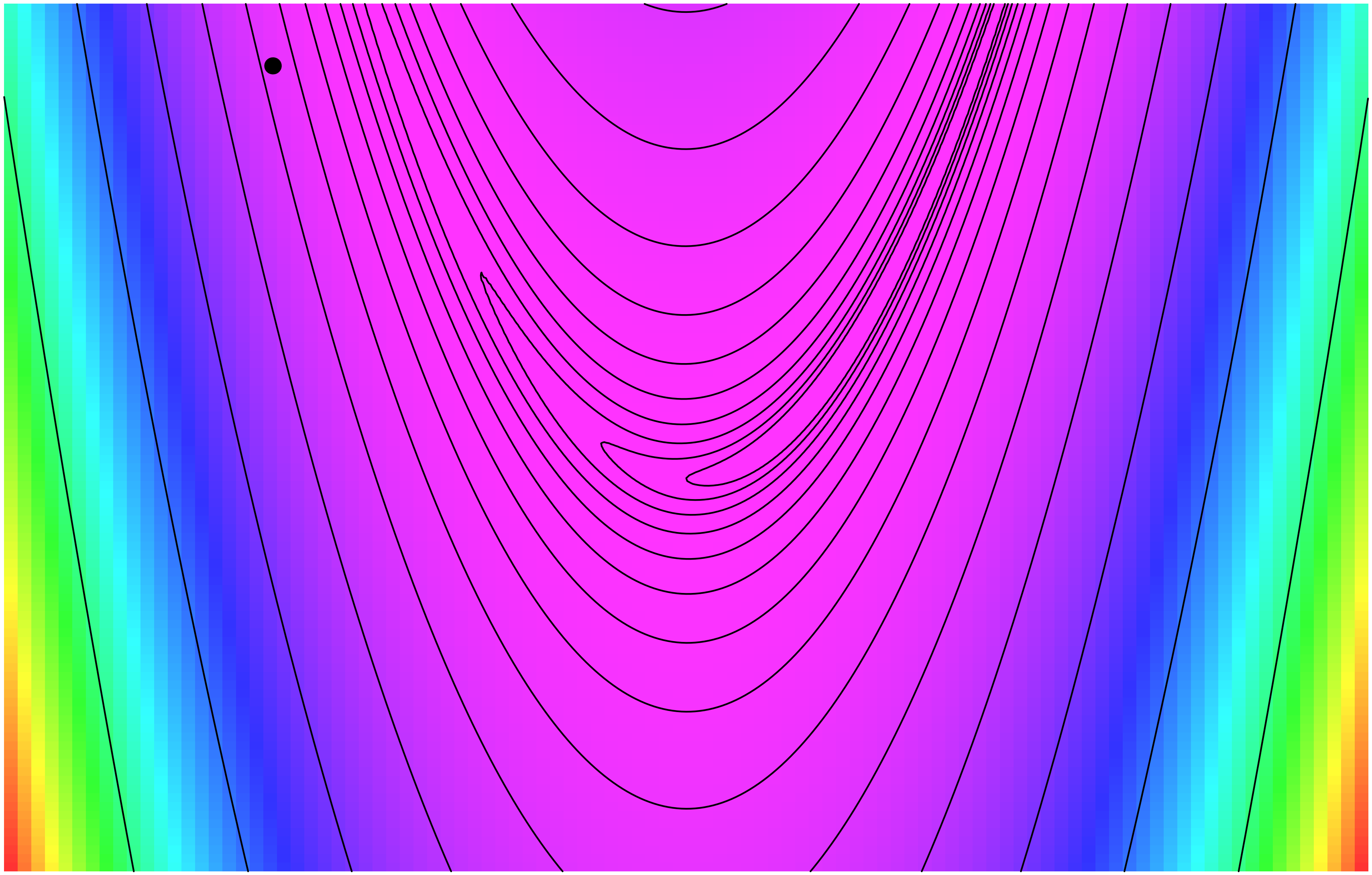
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J} + \nu\mathbf{I}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size
- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

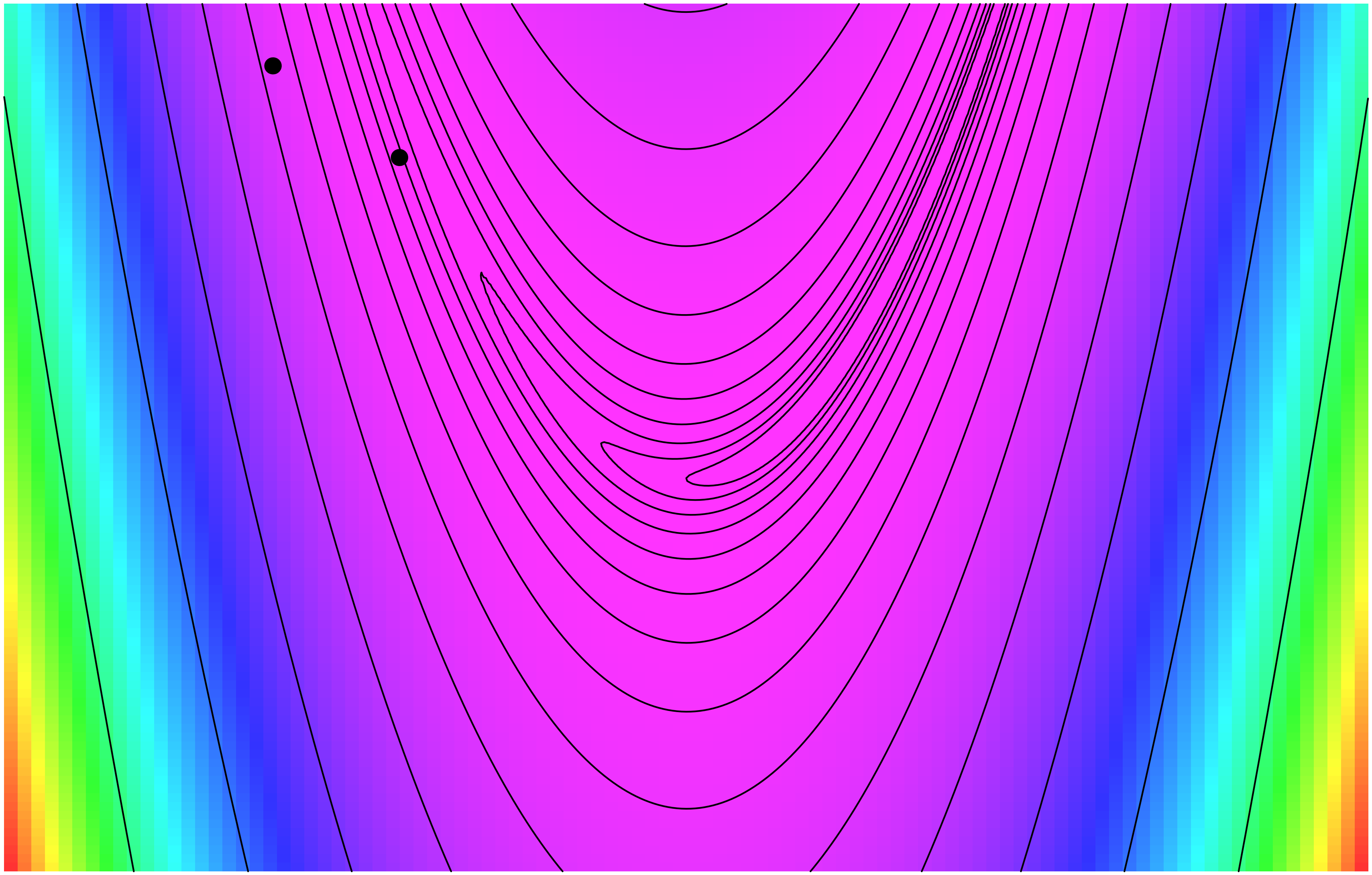
$\epsilon_1 = 10(x_2 - x_1^2)$ and $\epsilon_2 = 1 - x_1$



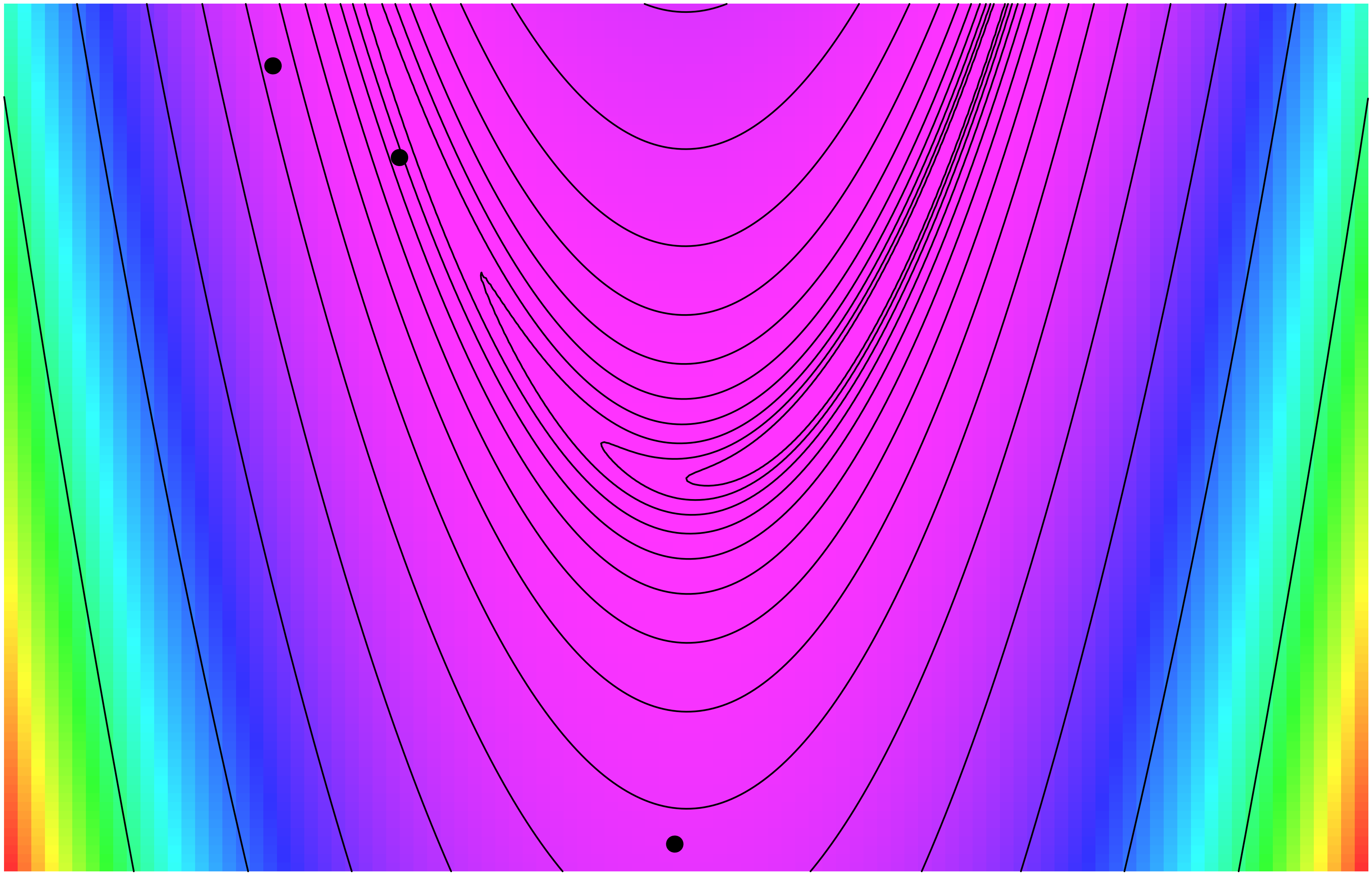
$$\epsilon_1 = 10(x_2 - x_1^2) \textbf{ and } \epsilon_2 = 1 - x_1$$



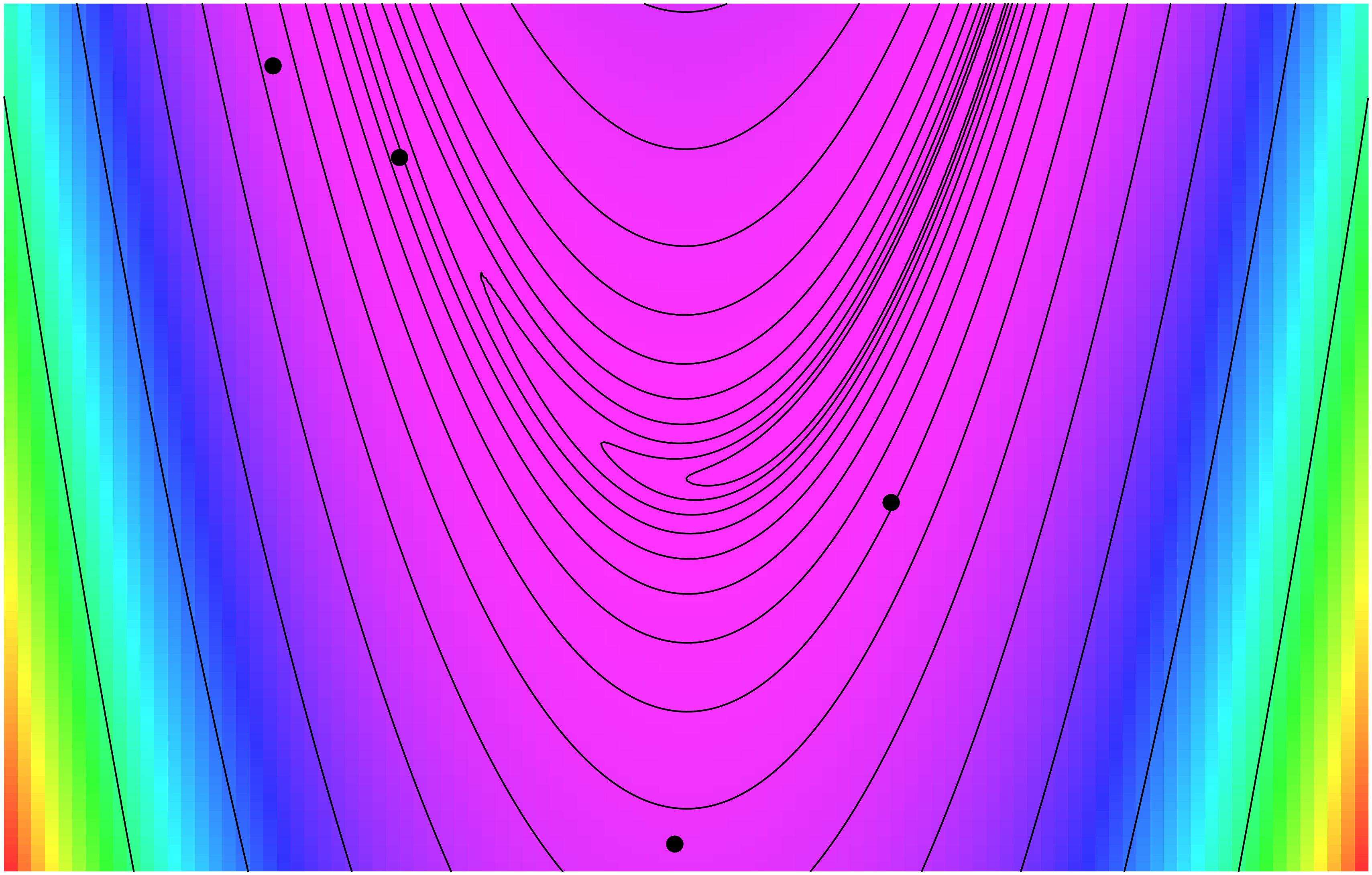
$$\epsilon_1 = 10(x_2 - x_1^2) \text{ and } \epsilon_2 = 1 - x_1$$



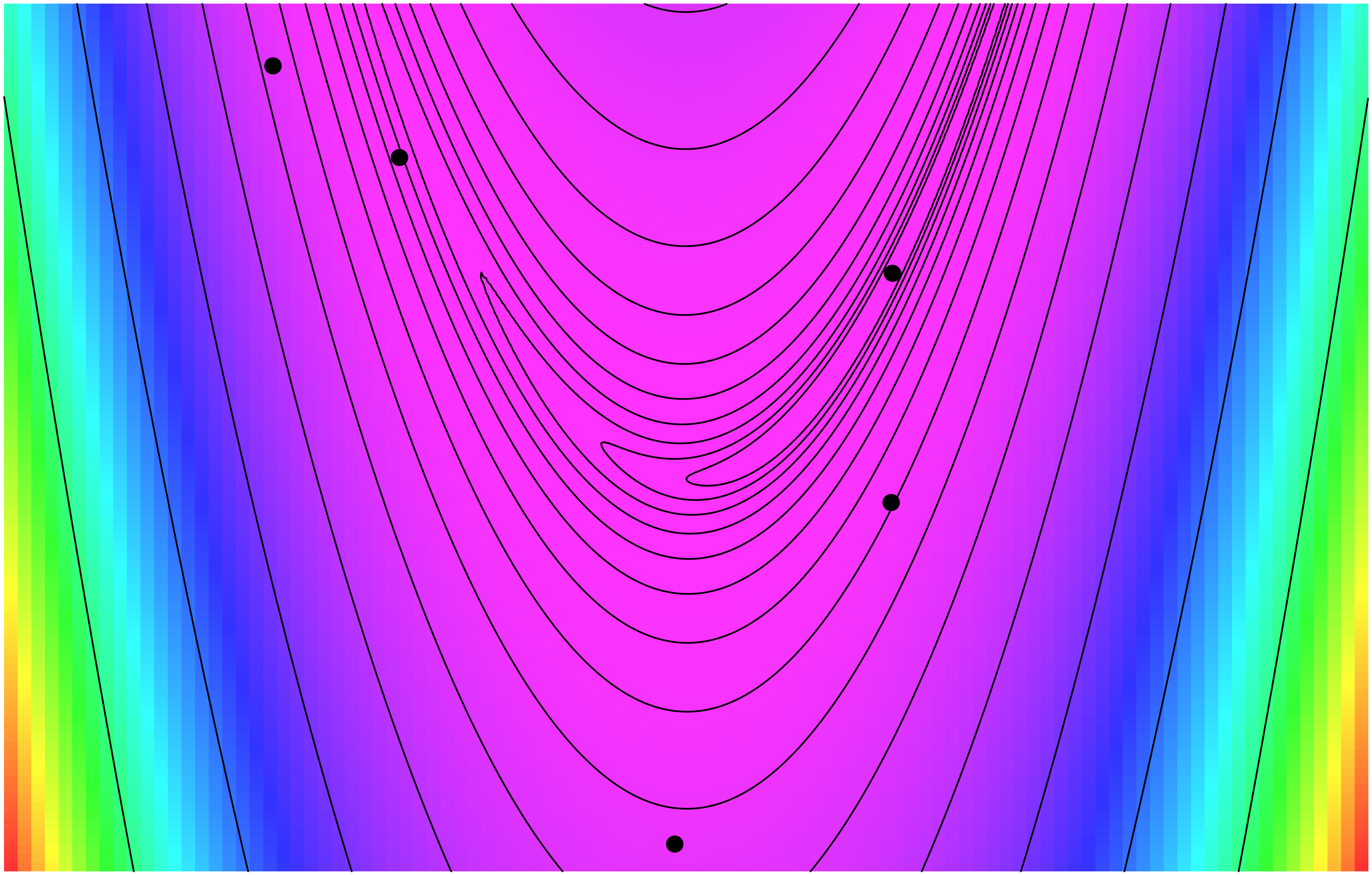
$\epsilon_1 = 10(x_2 - x_1^2)$ **and** $\epsilon_2 = 1 - x_1$



$\epsilon_1 = 10(x_2 - x_1^2)$ and $\epsilon_2 = 1 - x_1$



$$\epsilon_1 = 10(x_2 - x_1^2) \textbf{ and } \epsilon_2 = 1 - x_1$$



Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian

Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian

Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian

Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian