# Numerical Optimization

Chris Cornwell

Sept. 25, 2025

# General approach to numerical optimization

Approach to minimization of function $g$.

# General approach to numerical optimization

Approach to minimization of function $g$.

1. Start the process from some initial point $\mathbf{w}^{(0)}$.

# General approach to numerical optimization

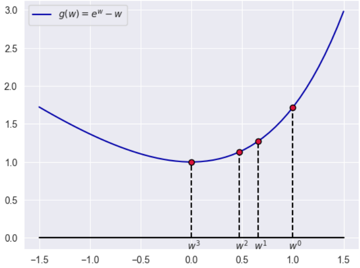
Approach to minimization of function $g$.

1. Start the process from some initial point $\mathbf{w}^{(0)}$.
2. After $t$ steps, with $\mathbf{w}^{(t)}$ as current input to $g$, update it to some $\mathbf{w}^{(t)}$, "going downhill" toward a stationary point.

# General approach to numerical optimization

Approach to minimization of function $g$.

1. Start the process from some initial point $\mathbf{w}^{(0)}$.
2. After $t$ steps, with $\mathbf{w}^{(t)}$ as current input to $g$, update it to some $\mathbf{w}^{(t)}$, "going downhill" toward a stationary point.
3. Repeat Step (2) – converging to a stationary point, in good scenario – until you meet a **stopping condition** at some step $T$. The approximate stationary point (potentially a minimizer) is $\mathbf{w}^{(T)}$.



# Outline

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called “step size”).[1]

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called "step size").[1]
- For each $t \geq 0$, iteratively assign

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_{t+1} \nabla \ell(\mathbf{w}^{(t)}).$$

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called "step size").[1]
- For each $t \geq 0$, iteratively assign

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_{t+1}\nabla\ell(\mathbf{w}^{(t)}).$$

- As in the general optimization description, proceed until a **stopping condition** is met.

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called "step size").[1]
- For each $t \geq 0$, iteratively assign

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_{t+1} \nabla \ell(\mathbf{w}^{(t)}).$$

- As in the general optimization description, proceed until a **stopping condition** is met.

More on stopping conditions later. For now, ...

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called "step size").[1]
- For each $t \geq 0$, iteratively assign

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_{t+1} \nabla \ell\big(\mathbf{w}^{(t)}\big).$$

- As in the general optimization description, proceed until a **stopping condition** is met.

More on stopping conditions later. For now, ...

- choose a ("small") threshhold value $\varepsilon$. Stop when, as part of the last update, the <u>change</u> in every parameter <u>divided by its size</u> is not more than $\varepsilon$.

[1] For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

# Gradient Descent - a first-order method

Looking to minimize a loss (cost) function $\ell : \mathbb{R}^N \to \mathbb{R}$. Want to approximate stationary point for $\ell$ in $\mathbb{R}^N$, in order to minimize $\ell$ (hopefully).

- In addition to choosing initial $\mathbf{w}^{(0)} \in \mathbb{R}^N$, pick a **learning rate** $\alpha_{t+1} > 0$ for each step in the process (also called “step size”).[1]
- For each $t \geq 0$, iteratively assign

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha_{t+1} \nabla \ell(\mathbf{w}^{(t)}).$$

- As in the general optimization description, proceed until a **stopping condition** is met.

More on stopping conditions later. For now, ...

- choose a (“small”) threshhold value $\varepsilon$. Stop when, as part of the last update, the change in every parameter divided by its size is not more than $\varepsilon$. That is, stop when

$$\frac{|\omega_j^{(t)} - \omega_j^{(t-1)}|}{|\omega_j^{(t-1)}|} \leq \varepsilon, \qquad \forall 1 \leq j \leq p.$$

[1]For us, we will usually use $\alpha_{t+1}$ that is constant in $t$, meaning it is the same for all

## Relation to Linear approximations

The update step in Gradient Descent uses the linear approximation to the loss function in order to make an update.

# Relation to Linear approximations

The update step in Gradient Descent uses the linear approximation to the loss function in order to make an update.
Note: linear approximation is

$$h(\mathbf{w}) = \ell(\mathbf{w}^{(t)}) + \nabla \ell(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}).$$

How does one change $\mathbf{w}^{(t)}$ to make $h(\mathbf{w})$ decrease the most? Want some (fixed length vector) $\Delta \mathbf{w}$ so that $h(\mathbf{w}^{(t)}) - h(\mathbf{w}^{(t)} + \Delta \mathbf{w})$ is as large as possible.

# Relation to Linear approximations

The update step in Gradient Descent uses the linear approximation to the loss function in order to make an update.
Note: linear approximation is

$$h(\mathbf{w}) = \ell(\mathbf{w}^{(t)}) + \nabla \ell(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}).$$

How does one change $\mathbf{w}^{(t)}$ to make $h(\mathbf{w})$ decrease the most? Want some (fixed length vector) $\Delta \mathbf{w}$ so that $h(\mathbf{w}^{(t)}) - h(\mathbf{w}^{(t)} + \Delta \mathbf{w})$ is as large as possible.
Since

$$h(\mathbf{w}^{(t)}) - h(\mathbf{w}^{(t)} + \Delta \mathbf{w}) = -\nabla \ell(\mathbf{w}^{(t)})^T \Delta \mathbf{w}$$

this occurs when $\nabla \ell(\mathbf{w}^{(t)})^T \Delta \mathbf{w}$ is as negative as possible, which is when $\Delta \mathbf{w}$ is in the opposite direction of $\nabla \ell(\mathbf{w}^{(t)})$; and so, our update is $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta \mathbf{w} = \mathbf{w}^{(t)} - \alpha_{t+1} \nabla \ell(\mathbf{w}^{(t)})$.

# Convergence

Does gradient descent converge to point at which loss function is minimized? Is it even guaranteed to converge?

# Convergence

Does gradient descent converge to point at which loss function is minimized? Is it even guaranteed to converge?

Short answer: No, not necessarily.

...so, *in what cases* can we guarantee such a thing?

# Issues with convergence

To demonstrate the difficulty, imagine a “toy” loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each $w$ we have $\nabla \ell = \left( \frac{d\ell}{dw} \right) = (2w)$.

# Issues with convergence

To demonstrate the difficulty, imagine a “toy” loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each $w$ we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\alpha_{t+1} > 1$ for all $t$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\alpha_{t+1} w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $w^{(t+1)} < -w^{(t)}$ which means that $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$.

## Issues with convergence

To demonstrate the difficulty, imagine a “toy” loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each $w$ we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\alpha_{t+1} > 1$ for all $t$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\alpha_{t+1} w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $w^{(t+1)} < -w^{(t)}$ which means that $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$. Likewise, if $w^{(t)} < 0$ then $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$.

# Issues with convergence

To demonstrate the difficulty, imagine a "toy" loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each $w$ we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\alpha_{t+1} > 1$ for all $t$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\alpha_{t+1}w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $w^{(t+1)} < -w^{(t)}$ which means that $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$. Likewise, if $w^{(t)} < 0$ then $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$.

So, get divergence when $\alpha_{t+1} > 1$. However, if $0 < \alpha_{t+1} < 1$, then (for the function $\ell(w) = w^2$, at least) it will converge to minimizer $w = 0$.

# Issues with convergence

To demonstrate the difficulty, imagine a "toy" loss function: $\ell : \mathbb{R} \to \mathbb{R}$ with $\ell(w) = w^2$. At each $w$ we have $\nabla \ell = \left(\frac{d\ell}{dw}\right) = (2w)$.

Say learning rate: $\alpha_{t+1} > 1$ for all $t$. Then, at any $w^{(t)} > 0$, we get

$$w^{(t+1)} = w^{(t)} - 2\alpha_{t+1} w^{(t)} < w^{(t)} - 2w^{(t)} = -w^{(t)},$$

and so $w^{(t+1)} < -w^{(t)}$ which means that $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$. Likewise, if $w^{(t)} < 0$ then $\left|w^{(t+1)}\right| > \left|w^{(t)}\right|$.

So, get divergence when $\alpha_{t+1} > 1$. However, if $0 < \alpha_{t+1} < 1$, then (for the function $\ell(w) = w^2$, at least) it will converge to minimizer $w = 0$.

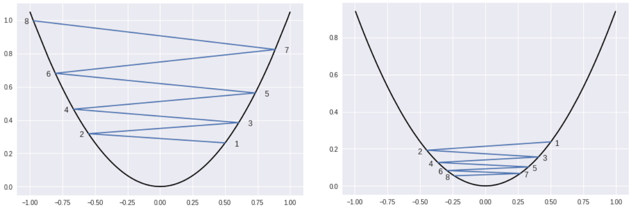



Figure: Gradient descent on $\ell(w) = w^2$. Left: $\alpha = 1.05$; right: $\alpha = 0.95$.

# Outline

# Newton's method

As before, we want to use approximations to get closer to a stationary point of $\ell : \mathbb{R}^N \to \mathbb{R}$, however we will now use a second order approximation:

$$h(\mathbf{w}) = \ell(\mathbf{w}^{(t)}) + \nabla \ell(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2} (\mathbf{w} - \mathbf{w}^{(t)})^T \nabla^2 \ell(\mathbf{w}^{(t)}) (\mathbf{w} - \mathbf{w}^{(t)}).$$

# Newton's method

As before, we want to use approximations to get closer to a stationary point of $\ell : \mathbb{R}^N \to \mathbb{R}$, however we will now use a second order approximation:

$$h(\mathbf{w}) = \ell(\mathbf{w}^{(t)}) + \nabla \ell(\mathbf{w}^{(t)})^T (\mathbf{w} - \mathbf{w}^{(t)}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^{(t)})^T \nabla^2 \ell(\mathbf{w}^{(t)})(\mathbf{w} - \mathbf{w}^{(t)}).$$

As a consequence from the recent homework, this has a stationary point at a solution to the linear equation $\nabla^2 \ell(\mathbf{w}^{(t)})\mathbf{w} = \nabla^2 \ell(\mathbf{w}^{(t)})\mathbf{w}^{(t)} - \nabla \ell(\mathbf{w}^{(t)})$. Call that solution $\mathbf{w}^{(t+1)}$. Repeat the above step until some stopping condition is met.

# Newton's method - advantages and disadvantages

**Advantage:** When converging to a minimum of the function, Newton's method will converge in fewer steps.

**Disadvantages:** One has to compute the Hessian – more computation time and more storage needed. Solving the linear system can also have numerical issues and take more time.

Also, if $\ell$ is not convex (and $\mathbf{w}^{(t)}$ is not in a convex region for the function), then it may approach a maximum instead of minimum.

# Outline

Gradient Descent

Newton's method - a second-order method

Examples and some convergence guarantees

# Example with single-variable Linear regression

Least Squares cost function for Linear regression is determined by sample data $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{P}$: given the data, the cost function is $g_{\mathcal{S}}(b, w) = \sum_{i=1}^{P}(b + wx_i - y_i)^2$. In homework you found,

$$\nabla g_{\mathcal{S}}(b, w) = \left( 2\sum_{i=1}^{P}(b + wx_i - y_i), \quad 2\sum_{i=1}^{P}(b + wx_i - y_i)x_i \right).$$

## Example with single-variable Linear regression

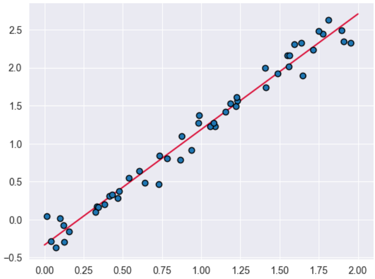
Least Squares cost function for Linear regression is determined by sample data $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{P}$: given the data, the cost function is $g_{\mathcal{S}}(b, w) = \sum_{i=1}^{P}(b + wx_i - y_i)^2$. In homework you found,

$$\nabla g_{\mathcal{S}}(b, w) = \left(2\sum_{i=1}^{P}(b + wx_i - y_i),\quad 2\sum_{i=1}^{P}(b + wx_i - y_i)x_i\right).$$

Example: batch gradient descent working on the 'Example1.csv' data.

The LSR line, using closed form.

- $w^\star \approx 1.520$, $b^\star = -0.3346$:



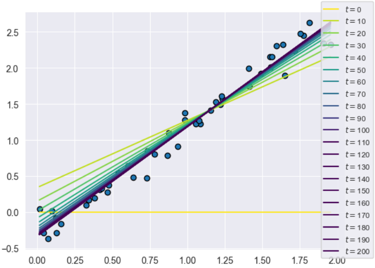
# Example with Linear regression

Least Squares cost function for Linear regression is determined by sample data $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{P}$: given the data, the cost function is $g_{\mathcal{S}}(b, w) = \sum_{i=1}^{P}(b + wx_i - y_i)^2$. In homework you found,

$$\nabla g_{\mathcal{S}}(b, w) = \left( 2\sum_{i=1}^{P}(b + wx_i - y_i), \quad 2\sum_{i=1}^{P}(b + wx_i - y_i)x_i \right).$$

Example: batch gradient descent working on the 'Example1.csv' data.

Plot of selected lines found during batch GD updates; starting parameters $m = 0$, $b = 0$;



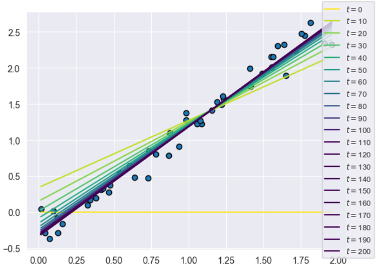
# Example with Linear regression

Least Squares cost function for Linear regression is determined by sample data $\mathcal{S} = \{(x_i, y_i)\}_{i=1}^{P}$: given the data, the cost function is $g_{\mathcal{S}}(b, w) = \sum_{i=1}^{P}(b + wx_i - y_i)^2$. In homework you found,

$$\nabla g_{\mathcal{S}}(b, w) = \left(2\sum_{i=1}^{P}(b + wx_i - y_i),\ \ 2\sum_{i=1}^{P}(b + wx_i - y_i)x_i\right).$$

Example: batch gradient descent working on the 'Example1.csv' data.

Plot of selected lines found during batch GD updates; starting parameters $m = 0$, $b = 0$; learning rate set to 0.1. Parameter values on iteration 208: $m = 1.519$, $b = -0.334$.



# Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial deriv. of $g_{\mathcal{S}}$, can compute it in one line of code.

# Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial deriv. of $g_{\mathcal{S}}$, can compute it in one line of code. Presuming `x`, `y` are arrays of coordinates for data, current parameter values are `b`, `w`, the following gives partial derivatives:

```
partial_w = 2*np.sum( (w*x + b - y)*x )
partial_b = 2*np.sum( (w*x + b - y) )
```

# Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial deriv. of $g_{\mathcal{S}}$, can compute it in one line of code. Presuming `x`, `y` are arrays of coordinates for data, current parameter values are `b`, `w`, the following gives partial derivatives:

```
partial_w = 2*np.sum( (w*x + b - y)*x )
partial_b = 2*np.sum( (w*x + b - y) )
```

2. To implement GD, want more than one function – at the least, one to compute the gradient (given current parameters); another that performs update and checks for stopping. *Roughly*...

# Implementing Gradient Descent

Some notes on implementation of the gradient descent updates.

1. Each partial deriv. of $g_{\mathcal{S}}$, can compute it in one line of code. Presuming `x`, `y` are arrays of coordinates for data, current parameter values are `b`, `w`, the following gives partial derivatives:

```
partial_w = 2*np.sum( (w*x + b - y)*x )
partial_b = 2*np.sum( (w*x + b - y) )
```

2. To implement GD, want more than one function – at the least, one to compute the gradient (given current parameters); another that performs update and checks for stopping. *Roughly*...

```
## lr is learning rate; threshhold is for stopping;
input: X, y, lr, threshhold
params ← initial array of parameters
while (max of last_update > threshhold){
    grad ← compute_grad(params, X, y)
    last_update ← | grad / params | ## entrywise array division
    # handle params[i] near 0
    params ← params − lr*grad
}
return params
```

# Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some "control" on size of the gradient then, by choosing $\alpha_{t+1}$ small enough, can guarantee convergence.

[2] Meaning: $\exists$ a constant $C$ s.t. for all $\tilde{w}_1, \tilde{w}_2$, $|\nabla g_{\mathcal{S}}(\tilde{w}_1) - \nabla g_{\mathcal{S}}(\tilde{w}_2)| \leq C|\tilde{w}_1 - \tilde{w}_2|$.

# Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some "control" on size of the gradient then, by choosing $\alpha_{t+1}$ small enough, can guarantee convergence.

## Theorem

Suppose a loss function $g_{\mathcal{S}} : \mathbb{R}^{N+1} \to \mathbb{R}$ is differentiable and convex, and suppose that $\nabla g_{\mathcal{S}}$ is *Lipschitz continuous*[2] with some constant $C > 0$ and that $\alpha_{t+1} = \alpha \leq 1/C$ (constant learning rate). Then, for a minimizer $\tilde{w}^\star$ of $g_{\mathcal{S}}$,

$$g_{\mathcal{S}}(\tilde{w}^{(t)}) - g_{\mathcal{S}}(\tilde{w}^\star) \leq \frac{|\tilde{w}^{(0)} - \tilde{w}^\star|^2}{2\alpha t}.$$

[2] Meaning: $\exists$ a constant $C$ s.t. for all $\tilde{w}_1, \tilde{w}_2$, $|\nabla g_{\mathcal{S}}(\tilde{w}_1) - \nabla g_{\mathcal{S}}(\tilde{w}_2)| \leq C|\tilde{w}_1 - \tilde{w}_2|$.

# Guarantees of convergence

If your loss function is differentiable and a **convex function**, and if have some "control" on size of the gradient then, by choosing $\alpha_{t+1}$ small enough, can guarantee convergence.

### Theorem

Suppose a loss function $g_{\mathcal{S}} : \mathbb{R}^{N+1} \to \mathbb{R}$ is differentiable and convex, and suppose that $\nabla g_{\mathcal{S}}$ is *Lipschitz continuous*[2] with some constant $C > 0$ and that $\alpha_{t+1} = \alpha \leq 1/C$ (constant learning rate). Then, for a minimizer $\tilde{w}^{\star}$ of $g_{\mathcal{S}}$,

$$g_{\mathcal{S}}(\tilde{w}^{(t)}) - g_{\mathcal{S}}(\tilde{w}^{\star}) \leq \frac{|\tilde{w}^{(0)} - \tilde{w}^{\star}|^2}{2\alpha t}.$$

- The difference between $g_{\mathcal{S}}(\tilde{w}^{(t)})$ and the minimum of $g_{\mathcal{S}}$ is bounded by a constant times $1/t$.

[2] Meaning: $\exists$ a constant $C$ s.t. for all $\tilde{w}_1, \tilde{w}_2$, $|\nabla g_{\mathcal{S}}(\tilde{w}_1) - \nabla g_{\mathcal{S}}(\tilde{w}_2)| \leq C|\tilde{w}_1 - \tilde{w}_2|$.

# Guarantees of convergence

Previous theorem requires using actual gradient of $g_{\mathcal{S}}$ in each update step. Here is a convergence guarantee that allows for a random vector $\mathbf{D}_t$, in place of $\nabla g_{\mathcal{S}}$, as long as $\mathbb{E}[\mathbf{D}_t | \tilde{w}^{(t)}] = \nabla g_{\mathcal{S}}(\tilde{w}^{(t)})$.

- e.g., use this in alternate versions of gradient descent, *e.g.*, "mini-batch gradient descent."

# Guarantees of convergence

Previous theorem requires using actual gradient of $g_{\mathcal{S}}$ in each update step. Here is a convergence guarantee that allows for a random vector $\mathbf{D}_t$, in place of $\nabla g_{\mathcal{S}}$, as long as $\mathbb{E}[\mathbf{D}_t | \tilde{w}^{(t)}] = \nabla g_{\mathcal{S}}(\tilde{w}^{(t)})$.

- e.g., use this in alternate versions of gradient descent, *e.g.*, "mini-batch gradient descent."

## Theorem

Suppose that $g_{\mathcal{S}}$ is differentiable and convex, that $\tilde{w}^{(0)} = \mathbf{0}$, and that for all $t$, $\alpha_{t+1} = \frac{1}{\sqrt{K}}$ for an integer $K > 0$. Finally, suppose that $|\mathbf{D}_t| \leq 1$ for all $1 \leq t \leq K$. Then, for a minimizer $\tilde{w}^\star$ of $g_{\mathcal{S}}$,

$$\mathbb{E}[g_{\mathcal{S}}(\bar{w})] - g_{\mathcal{S}}(\tilde{w}^\star) \leq \frac{1}{\sqrt{K}}$$

where $\bar{w}$ is the average of $\tilde{w}^{(1)}, \tilde{w}^{(2)}, \ldots, \tilde{w}^{(K)}$.