MS-E2122 - Nonlinear Optimization
Lecture V

Fernando Dias

Department of Mathematics and Systems Analysis

Aalto University
School of Science

September 26, 2023

Outline of this lecture

Line search methods - univariate (single variable) optimisation

- Line searches without derivatives
- Line searches with derivatives

Methods for unconstrained optimisation

- Coordinate descent
- Gradient method
- Newton's method

- Optimality conditions;
- KKT conditions

Outline of this lecture

Line search methods

This pseudocode represents the general concept of optimization methods:

Algorithm Conceptual optimisation algorithm

```
initialise. iteration count k = 0, starting point x_0
while stopping criteria are not met do
    compute direction d_k
    compute step size λ_k
    x_{k+1} = x_k + λ_k d_k
    k = k + 1
end while
return x_k
```

where

- k is an iteration counter;
- λ_k is a suitable step size;
- d_k is a direction vector;

Line search methods

Finding an optimal step size λ_k is in itself an optimisation problem called line search due to its **unidimensional** nature.

Line searches are the backbone of most optimisation methods.

Let $\theta(\lambda) = f(x + \lambda d)$. If f is **differentiable**, a straightforward approach is to find an optimal setup size λ is

$$\theta'(\lambda) = d^\top \nabla f(x + \lambda d) = 0$$

Line search methods

Finding an optimal step size λ_k is in itself an optimisation problem called line search due to its **unidimensional** nature.

Line searches are the backbone of most optimisation methods.

Let $\theta(\lambda) = f(x + \lambda d)$. If f is **differentiable**, a straightforward approach is to find an optimal setup size λ is

$$\theta'(\lambda) = d^\top \nabla f(x + \lambda d) = 0$$

However, one must bear in mind that:

- $d^\top f(x + \lambda d)$ is often nonlinear in λ;
- $\overline{\lambda} = \mathrm{argmin}_\lambda\, d^\top \nabla f(x + \lambda d) = 0$ is not necessarily optimal.

Line search methods

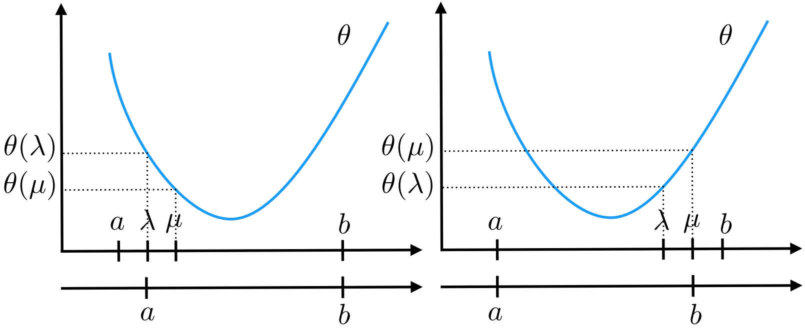
Theorem 1 (Line search reduction)

Let $\theta : \mathbb{R} \to \mathbb{R}$ *be strictly quasiconvex over the interval* $[a, b]$*, and let* $\lambda, \mu \in [a, b]$ *such that* $\lambda < \mu$*. If* $\theta(\lambda) > \theta(\mu)$*, then* $\theta(z) \geq \theta(\mu)$ *for all* $z \in [a, \lambda]$*. If* $\theta(\lambda) \leq \theta(\mu)$*, then* $\theta(z) \geq \theta(\lambda)$ *for all* $z \in [\mu, b]$*.*

Line search methods

Theorem 1 (Line search reduction)

Let $\theta : \mathbb{R} \to \mathbb{R}$ be strictly quasiconvex over the interval $[a, b]$, and let $\lambda, \mu \in [a, b]$ such that $\lambda < \mu$. If $\theta(\lambda) > \theta(\mu)$, then $\theta(z) \geq \theta(\mu)$ for all $z \in [a, \lambda]$. If $\theta(\lambda) \leq \theta(\mu)$, then $\theta(z) \geq \theta(\lambda)$ for all $z \in [\mu, b]$.



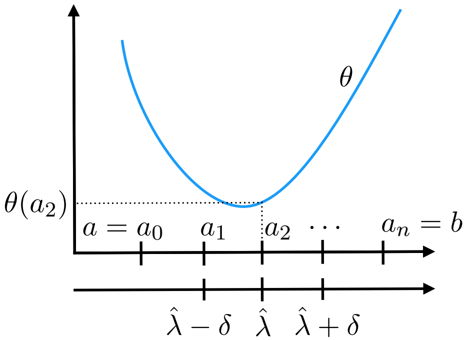
Applying Theorem 1 allows to iteratively reduce the search space.

Line search methods - uniform search

Break $[a, b]$ into n uniform intervals of size δ, which leads to $n + 1$ grid points $a_k = a_0 + k\delta$, with $a = a_0, b = a_n$, and $k = 0, \ldots, n$.

Line search methods - uniform search

Break $[a, b]$ into n uniform intervals of size δ, which leads to $n + 1$ grid points $a_k = a_0 + k\delta$, with $a = a_0, b = a_n$, and $k = 0, \dots, n$.

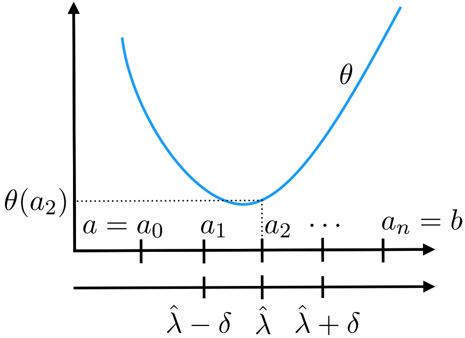


Grid search with 5 points; Note that $\theta(a_2) = \min_{i=0,\dots,n} \theta(a_i)$.

Let $\hat{\lambda} = \operatorname{argmin}_{i=0,\dots,n} \theta(a_i)$.
We know that the optimal $\overline{\lambda} \in [\hat{\lambda} - \delta, \hat{\lambda} + \delta]$.

Line search methods - uniform search

Break $[a, b]$ into n uniform intervals of size δ, which leads to $n + 1$ grid points $a_k = a_0 + k\delta$, with $a = a_0, b = a_n$, and $k = 0, \ldots, n$.



Grid search with 5 points; Note that $\theta(a_2) = \min_{i=0,\ldots,n} \theta(a_i)$.

Let $\hat{\lambda} = \operatorname{argmin}_{i=0,\ldots,n} \theta(a_i)$. We know that the optimal $\overline{\lambda} \in [\hat{\lambda} - \delta, \hat{\lambda} + \delta]$.

Remarks:

- The search can be repeated making $a = \hat{\lambda} - \delta$ and $b = \hat{\lambda} + \delta$.
- The number of grid points can increase dynamically, saving function evaluations.

Line search methods - sequential searches

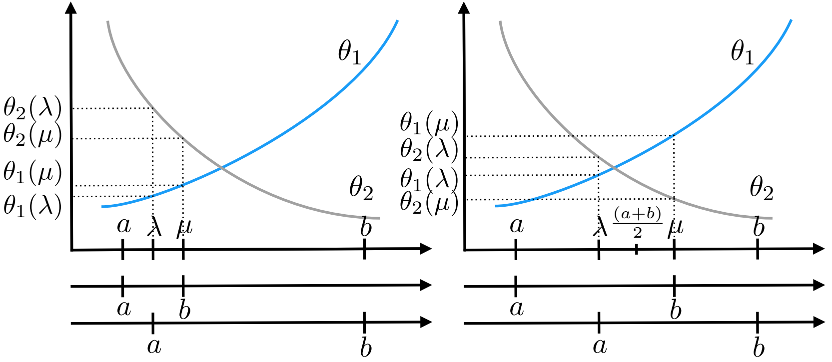
More efficient methods can be devised by using information of the **previous** evaluation of θ. These are known as sequential searches.

1. Dichotomous search: we place two points, λ and μ, around the midpoint of $[a, b]$ at a small distance ϵ.

Line search methods - sequential searches

More efficient methods can be devised by using information of the **previous** evaluation of θ. These are known as sequential searches.

1. Dichotomous search: we place two points, λ and μ, around the midpoint of $[a, b]$ at a small distance ϵ.



Using the midpoint $(a + b)/2$ and Theorem 1 to reduce the search space.

Line search methods - sequential searches

Algorithm Dichotomous search

initialise. distance $\epsilon > 0$, tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
while $b_k - a_k > l$ **do**
  $\lambda_k = \frac{a_k+b_k}{2} - \epsilon$, $\mu_k = \frac{a_k+b_k}{2} + \epsilon$
  if $\theta(\lambda_k) < \theta(\mu_k)$ **then**
    $a_{k+1} = a_k$, $b_{k+1} = \mu_k$
  else
    $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
  end if
  $k = k + 1$
end while
return $\overline{\lambda} = \frac{a_k+b_k}{2}$

Line search methods - sequential searches

Algorithm Dichotomous search

initialise. distance $\epsilon > 0$, tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
while $b_k - a_k > l$ **do**
    $\lambda_k = \frac{a_k + b_k}{2} - \epsilon$, $\mu_k = \frac{a_k + b_k}{2} + \epsilon$
    if $\theta(\lambda_k) < \theta(\mu_k)$ **then**
        $a_{k+1} = a_k$, $b_{k+1} = \mu_k$
    else
        $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$
    end if
    $k = k + 1$
end while
return $\overline{\lambda} = \frac{a_k + b_k}{2}$

Remark: The number of steps (and evaluations of θ) can be predicted beforehand:

$$b_{k+1} - a_{k+1} = \frac{1}{2^k}(b_0 - a_0) + 2\epsilon\left(1 - \frac{1}{2^k}\right).$$

Bisection method

More than just function evaluations, it also uses derivative information. We assume $\theta(\lambda)$ to be **differentiable** and **convex**.

2. Bisection method: The main idea is

1. if $\theta'(\lambda_k) = 0$, then λ_k is a **minimiser**.

Bisection method

More than just function evaluations, it also uses derivative information. We assume $\theta(\lambda)$ to be **differentiable** and **convex**.

2. Bisection method: The main idea is

1. if $\theta'(\lambda_k) = 0$, then λ_k is a **minimiser**.
2. if $\theta'(\lambda_k) > 0$, then, for $\lambda > \lambda_k$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) \geq 0$, which implies $\theta(\lambda) \geq \theta(\lambda_k)$ since θ is **convex**. Therefore, the new search interval becomes $[a_{k+1}, b_{k+1}] = [a_k, \lambda_k]$.

Bisection method

More than just function evaluations, it also uses derivative information. We assume $\theta(\lambda)$ to be **differentiable** and **convex**.

2. **Bisection method:** The main idea is

   1. if $\theta'(\lambda_k) = 0$, then λ_k is a **minimiser**.
   2. if $\theta'(\lambda_k) > 0$, then, for $\lambda > \lambda_k$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) \geq 0$, which implies $\theta(\lambda) \geq \theta(\lambda_k)$ since θ is **convex**. Therefore, the new search interval becomes $[a_{k+1}, b_{k+1}] = [a_k, \lambda_k]$.
   3. if $\theta'(\lambda_k) < 0$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) \geq 0$ for $\lambda < \lambda_k$. Thus, the new search interval becomes $[\boldsymbol{a_{k+1}}, \boldsymbol{b_{k+1}}] = [\boldsymbol{\lambda_k}, \boldsymbol{b_k}]$.

Bisection method

More than just function evaluations, it also uses derivative information. We assume $\theta(\lambda)$ to be **differentiable** and **convex**.

2. Bisection method: The main idea is

1. if $\theta'(\lambda_k) = 0$, then λ_k is a **minimiser**.
2. if $\theta'(\lambda_k) > 0$, then, for $\lambda > \lambda_k$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) \geq 0$, which implies $\theta(\lambda) \geq \theta(\lambda_k)$ since θ is **convex**. Therefore, the new search interval becomes $[a_{k+1}, b_{k+1}] = [a_k, \lambda_k]$.
3. if $\theta'(\lambda_k) < 0$, we have $\theta'(\lambda_k)(\lambda - \lambda_k) \geq 0$ for $\lambda < \lambda_k$. Thus, the new search interval becomes $\boldsymbol{[a_{k+1}, b_{k+1}] = [\lambda_k, b_k]}$.
4. As in the dichotomous search, to maximise overall interval reduction, we set $\lambda_k = \frac{1}{2}(b_k + a_k)$.

Bisection method

Algorithm Bisection method

initialise. tolerance $l > 0$, $[a_0, b_0] = [a, b]$, $k = 0$
while $b_k - a_k > l$ **do**
  $\lambda_k = \frac{(b_k + a_k)}{2}$ and evaluate $\theta'(\lambda_k)$
  if $\theta'(\lambda_k) = 0$ **then** return λ_k
  else if $\theta'(\lambda_k) > 0$ **then**
    $a_{k+1} = a_k,\ b_{k+1} = \lambda_k$
  else
    $a_{k+1} = \lambda_k,\ b_{k+1} = b_k$
  end if
  $k = k + 1.$
end while
return $\overline{\lambda} = \frac{a_k + b_k}{2}$

Inexact line searches - Armijo rule

Often the use of non-optimal (i.e., inexact) step sizes λ_k is enough to guarantee a good performance.

Armijo's rule: find acceptable step sizes by balancing the trade-off between convergence and numerical performance.

Inexact line searches - Armijo rule

Often the use of non-optimal (i.e., inexact) step sizes λ_k is enough to guarantee a good performance.

Armijo's rule: find acceptable step sizes by balancing the trade-off between convergence and numerical performance. A step size $\overline{\lambda}$ is considered acceptable if

$$f(\overline{x} + d\overline{\lambda}) - f(\overline{x}) \leq \alpha\overline{\lambda}\nabla f(\overline{x})^\top d$$

which, at $\lambda = 0$, is the same as

$$\begin{aligned} &\theta(\overline{\lambda}) - \theta(0) \leq \alpha\overline{\lambda}\theta'(0) \\ &\theta(\overline{\lambda}) \leq \theta(0) + \alpha\overline{\lambda}\theta'(0) : \text{ Armijo's rule (AR)} \end{aligned}$$

Inexact line searches - Armijo rule

Often the use of non-optimal (i.e., inexact) step sizes λ_k is enough to guarantee a good performance.

Armijo's rule: find acceptable step sizes by balancing the trade-off between convergence and numerical performance. A step size $\overline{\lambda}$ is considered acceptable if

$$f(\overline{x} + d\overline{\lambda}) - f(\overline{x}) \leq \alpha\overline{\lambda}\nabla f(\overline{x})^\top d$$

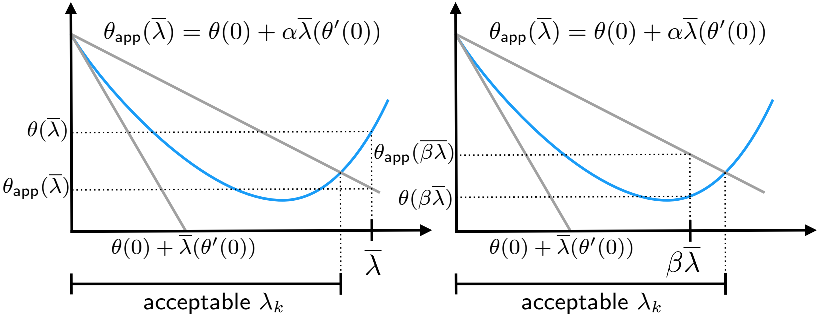
which, at $\lambda = 0$, is the same as

$$\theta(\overline{\lambda}) - \theta(0) \leq \alpha\overline{\lambda}\theta'(0)$$
$$\theta(\overline{\lambda}) \leq \theta(0) + \alpha\overline{\lambda}\theta'(0) : \text{ Armijo's rule (AR)}$$

If $\overline{\lambda}$ does not satisfy AR, $\overline{\lambda}$ is reduced by a factor $\beta \in (0, 1)$ and the test is repeated until (AR) is satisfied.

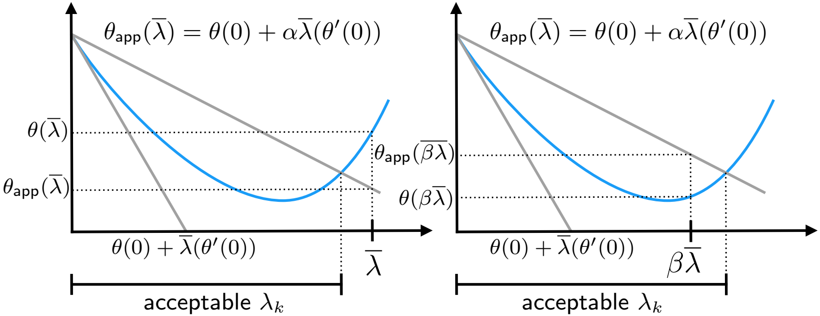
Inexact line searches - Armijo's rule

Armijo's rule has a nice graphical interpretation: $\overline{\lambda}$ is accepted if it is in an interval where the function $\theta(\lambda)$ is below a deflected linear extrapolation (from 0).



At first $\lambda_0 = \overline{\lambda}$ is not acceptable; after reducing the step size to $\lambda_1 = \beta\overline{\lambda}$, it enters the acceptable range where $\theta(\lambda_k) \leq \theta_{\text{app}}(\lambda_k) = \theta(0) + \alpha\lambda_k(\theta'(0))$.

Inexact line searches - Armijo's rule



Remarks:

1. Some variants also consider rules to guarantee that $\overline{\lambda}$ is not too small, such as $\theta(\delta\overline{\lambda}) \leq \theta(0) + \alpha\delta\overline{\lambda}\theta'(0)$, with $\delta > 1$.
2. Also known in the literature as backtracking.
3. Typical values: $\alpha \in [0.1, 0.5]$ and $\beta \in [0.6, 0.99]$. Very small α, e.g. 10^{-4} is often used as well.

Outline of this lecture

Coordinate descent

Next, we focus on optimizing functions $f : \mathbb{R}^n \to \mathbb{R}$ with **more** than one dimension.

The main difference is that we need to specify search directions d at each point $x \in \mathbb{R}^n$.

Coordinate descent

Next, we focus on optimizing functions $f : \mathbb{R}^n \to \mathbb{R}$ with **more** than one dimension.

The main difference is that we need to specify search directions d at each point $x \in \mathbb{R}^n$.

1. Coordinate descent: the search direction is one coordinate axis per iteration. That is, $d_i = 1$ for coordinate i and $d_{j \neq i} = 0$, for $i, j \in \{1, \ldots, n\}$.

Coordinate descent

Next, we focus on optimizing functions $f : \mathbb{R}^n \to \mathbb{R}$ with **more** than one dimension.

The main difference is that we need to specify search directions d at each point $x \in \mathbb{R}^n$.

1. Coordinate descent: the search direction is one coordinate axis per iteration. That is, $d_i = 1$ for coordinate i and $d_{j \neq i} = 0$, for $i, j \in \{1, \ldots, n\}$.

Several variants:

1. **Cyclic:** coordinates are considered in order $1, \ldots, n$;
2. **Double-sweep:** swap the coordinate order at each iteration;
3. **Gauss-Southwell:** choose components with largest $\frac{\partial f(x)}{\partial x_i}$;
4. **Stochastic:** coordinates are selected at random

Coordinate descent

Algorithm Coordinate descent method (cyclic)

initialise. tolerance $\epsilon > 0$, initial point x^0, iteration count $k = 0$
while $||x^{k+1} - x^k|| > \epsilon$ **do**
  for $j = 1, \dots n$ **do**
    $d = \{d_i = 1, \text{ if } i = j; d_i = 0, \text{ if } i \neq j\}$
    $\overline{\lambda}_j = \text{argmin}_{\lambda \in \mathbb{R}} \left\{ f(x_j^k + \lambda d_j) \right\}$
    $x_j^{k+1} = x_j^k + \overline{\lambda}_j d_j$
  end for
  $k = k + 1$
end while
return x^k

Coordinate descent

Algorithm Coordinate descent method (cyclic)

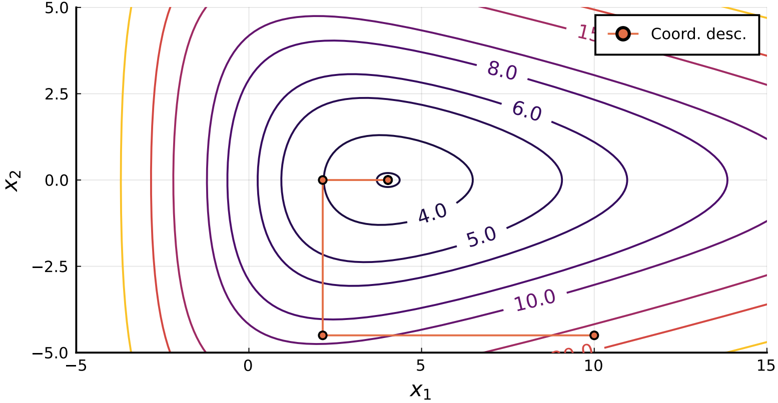
initialise. tolerance $\epsilon > 0$, initial point x^0, iteration count $k = 0$
while $||x^{k+1} - x^k|| > \epsilon$ **do**
  for $j = 1, \dots n$ **do**
    $d = \{d_i = 1, \text{ if } i = j; d_i = 0, \text{ if } i \neq j\}$
    $\overline{\lambda}_j = \text{argmin}_{\lambda \in \mathbb{R}} \left\{ f(x_j^k + \lambda d_j) \right\}$
    $x_j^{k+1} = x_j^k + \overline{\lambda}_j d_j$
  end for
  $k = k + 1$
end while
return x^k

Remarks:

1. The one-dimensional minimisation is called Gauss-Seidel step;
2. Block-coordinate methods use subgroups (blocks) of coordinates to define directions.

Coordinate descent

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



Coordinate descent method applied to f. Convergence is observed in 4 steps for a tolerance $\epsilon = 10^{-4}$

Gradient method

Recall that if d is a **descent direction**, there exists $\delta > 0$ such that $f(x + \lambda d) < f(x)$ for all $\lambda \in (0, \delta)$. The following result provides directions of steepest descent.

Lemma 2 (Steepest descent direction)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ *is differentiable at* $x \in \mathbb{R}^n$ *and* $\nabla f(x) \neq 0$. *Then* $\overline{d} = -\frac{\nabla f(x)}{||\nabla f(x)||}$ *is the direction of steepest descent of* f *at* x.

Gradient method

Recall that if d is a **descent direction**, there exists $\delta > 0$ such that $f(x + \lambda d) < f(x)$ for all $\lambda \in (0, \delta)$. The following result provides directions of steepest descent.

Lemma 2 (Steepest descent direction)

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ *is differentiable at* $x \in \mathbb{R}^n$ *and* $\nabla f(x) \neq 0$. *Then* $\overline{d} = -\frac{\nabla f(x)}{||\nabla f(x)||}$ *is the direction of steepest descent of* f *at* x.

Proof.

From differentiability of f, we have

$$f'(x; d) = \lim_{\lambda \to 0^+} \frac{f(x + \lambda d) - f(x)}{\lambda} = \nabla f(x)^\top d.$$

Thus, $\overline{d} = \operatorname{argmin}_{||d|| \leq 1} \left\{ \nabla f(x)^\top d \right\} = -\frac{\nabla f(x)}{||\nabla f(x)||}$. $\square$

Gradient method

Algorithm Gradient method

1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$.
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $\quad d = -\frac{\nabla f(x_k)}{||\nabla f(\overline{x})||}$
4: $\quad \overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
5: $\quad x_{k+1} = x_k + \overline{\lambda} d_j$
6: $\quad k = k + 1$
7: **end while**
8: **return** x_k

Gradient method

Algorithm Gradient method

```
initialise. tolerance ϵ > 0, initial point x_0, iteration count k = 0.
while ||∇f(x_k)|| > ϵ do
    d = −∇f(x_k) / ||∇f(x̄)||
    λ̄ = argmin_{λ∈ℝ} {f(x_k + λd)}
    x_{k+1} = x_k + λ̄ d_j
    k = k + 1
end while
return x_k
```

Remarks:

1. Steepest descent and gradient methods are different. When $||d|| \leq 1$ uses 2-norm (in Lemma 2), they are equivalent;

Gradient method

Algorithm Gradient method

1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$.
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $\quad d = -\frac{\nabla f(x_k)}{||\nabla f(\overline{x})||}$
4: $\quad \overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
5: $\quad x_{k+1} = x_k + \overline{\lambda} d_j$
6: $\quad k = k + 1$
7: **end while**
8: **return** x_k

Remarks:

1. Steepest descent and gradient methods are different. When $||d|| \leq 1$ uses 2-norm (in Lemma 2), they are equivalent;
2. Poor convergence and zigzagging can be observed due to imprecise linear approximations (more on this later);

Gradient method

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$

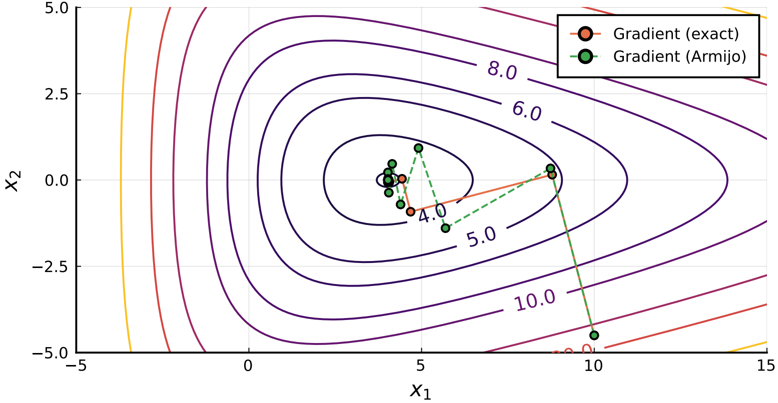


Figure: Gradient method applied to f. Convergence is observed in 9 steps using exact line search and 15 using Armijo's rule ($\epsilon = 10^{-4}$)

Newton's method

Same idea as in the **univariate** case. Can also be seen as deflected steepest descent.

Deflection is achieved using the Hessian, which is equivalent to relying on quadratic approximations (rather than linear).

Newton's method

Same idea as in the **univariate** case. Can also be seen as deflected steepest descent.

Deflection is achieved using the Hessian, which is equivalent to relying on quadratic approximations (rather than linear).

Consider the 2$^{\text{nd}}$-order approximation of f at x_k:

$$q(x) = f(x_k) + \nabla f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k),$$

where $H(x_k)$ is the Hessian at x_k.

Newton's method

Same idea as in the **univariate** case. Can also be seen as deflected steepest descent.

Deflection is achieved using the Hessian, which is equivalent to relying on quadratic approximations (rather than linear).

Consider the 2nd-order approximation of f at x_k:

$$q(x) = f(x_k) + \nabla f(x_k)^\top (x - x_k) + \frac{1}{2}(x - x_k)^\top H(x_k)(x - x_k),$$

where $H(x_k)$ is the Hessian at x_k. We require that $\nabla q(x_{k+1}) = 0$, which leads to

$$\nabla f(x_k) + H(x_k)(x - x_k) = 0.$$

Assuming that $H^{-1}(x_k)$ exists, we obtain the update rule

$$x_{k+1} = x_k - H^{-1}(x_k)\nabla f(x_k).$$

Newton's method

Algorithm Newton's method

initialise. tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$
while $||\nabla f(x_k)|| > \epsilon$ **do**
    $d = -H^{-1}(x_k)\nabla f(x_k)$
    $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
    $x_{k+1} = x_k + \overline{\lambda} d$
    $k = k + 1$
end while
return x_k

Newton's method

Algorithm Newton's method

initialise. tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$
while $||\nabla f(x_k)|| > \epsilon$ **do**
    $d = -H^{-1}(x_k)\nabla f(x_k)$
    $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
    $x_{k+1} = x_k + \overline{\lambda} d$
    $k = k + 1$
end while
return x_k

Remarks:

1. Setting $\overline{\lambda} = 1$ recovers the "pure" Newton's method;

Newton's method

Algorithm Newton's method

1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $\quad d = -H^{-1}(x_k)\nabla f(x_k)$
4: $\quad \overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
5: $\quad x_{k+1} = x_k + \overline{\lambda} d$
6: $\quad k = k + 1$
7: **end while**
8: **return** x_k

Remarks:

1. Setting $\overline{\lambda} = 1$ recovers the "pure" Newton's method;
2. As $\nabla f(x_k)$ gets close to 0, $H^{-1}(x_k)$ becomes singular;

Newton's method

Algorithm Newton's method

initialise. tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$
while $||\nabla f(x_k)|| > \epsilon$ **do**
    $d = -H^{-1}(x_k)\nabla f(x_k)$
    $\overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
    $x_{k+1} = x_k + \overline{\lambda} d$
    $k = k + 1$
end while
return x_k

Remarks:

1. Setting $\overline{\lambda} = 1$ recovers the "pure" Newton's method;
2. As $\nabla f(x_k)$ gets close to 0, $H^{-1}(x_k)$ becomes singular;
3. It might not converge if x_0 is too far from optimal and fixed step size is used;

Newton's method

Algorithm Newton's method

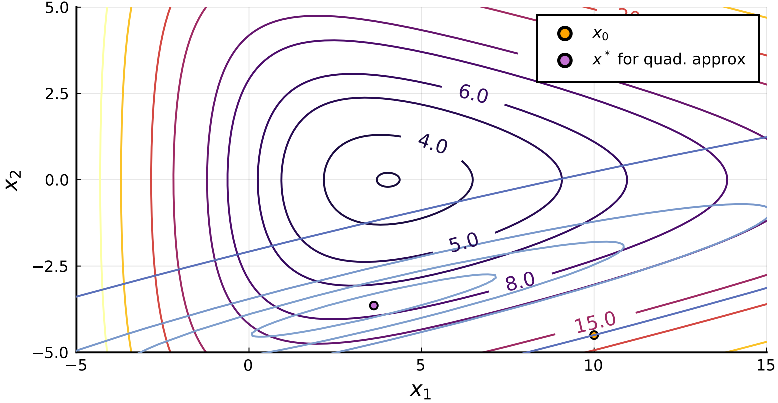
1: **initialise.** tolerance $\epsilon > 0$, initial point x_0, iteration count $k = 0$
2: **while** $||\nabla f(x_k)|| > \epsilon$ **do**
3: $\quad d = -H^{-1}(x_k)\nabla f(x_k)$
4: $\quad \overline{\lambda} = \text{argmin}_{\lambda \in \mathbb{R}} \{f(x_k + \lambda d)\}$
5: $\quad x_{k+1} = x_k + \overline{\lambda} d$
6: $\quad k = k + 1$
7: **end while**
8: **return** x_k

Remarks:

1. Setting $\overline{\lambda} = 1$ recovers the "pure" Newton's method;
2. As $\nabla f(x_k)$ gets close to 0, $H^{-1}(x_k)$ becomes singular;
3. It might not converge if x_0 is too far from optimal and fixed step size is used;
4. Levenberg-Marquardt method and other trust-region method variants also address convergence issues of Newton's method.

Newton's method

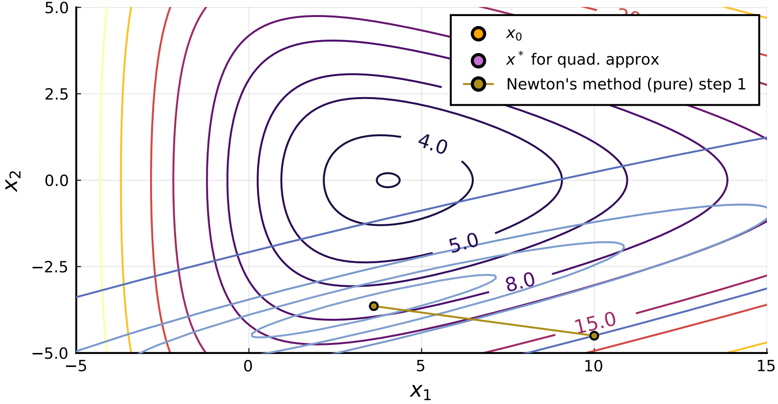
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



The quadratic approximation at x_0 (with level curves in blue) and the optimal point x^*.

Newton's method

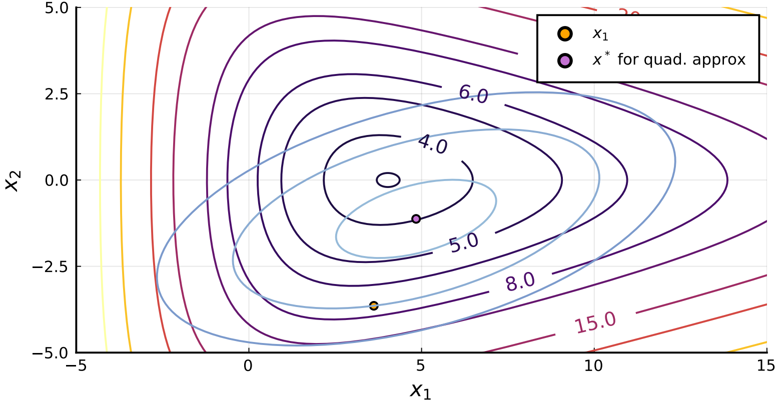
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



The new point x_1 becomes x^*.

Newton's method

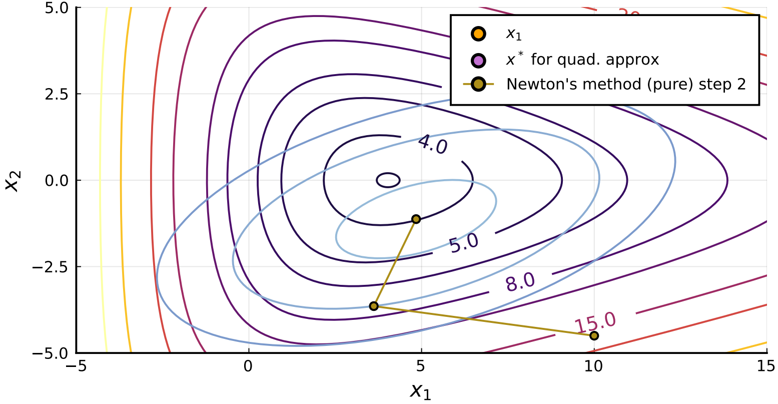
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



The new quadratic approximation at x_1 and new optimal point x^*. Notice how the approximation improved.

Newton's method

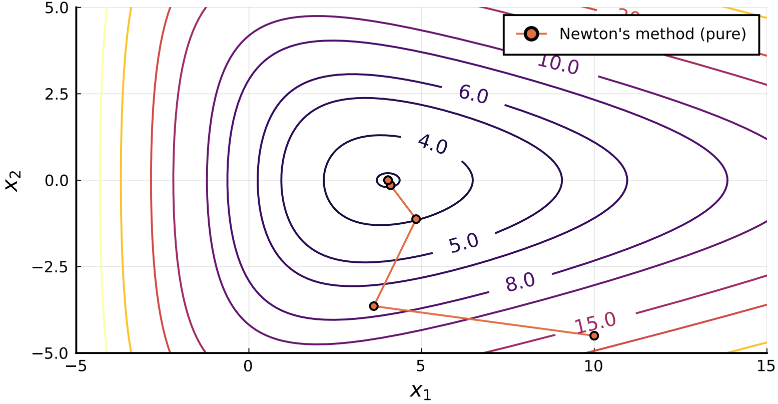
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



The new point x_2 becomes x^*.

Newton's method

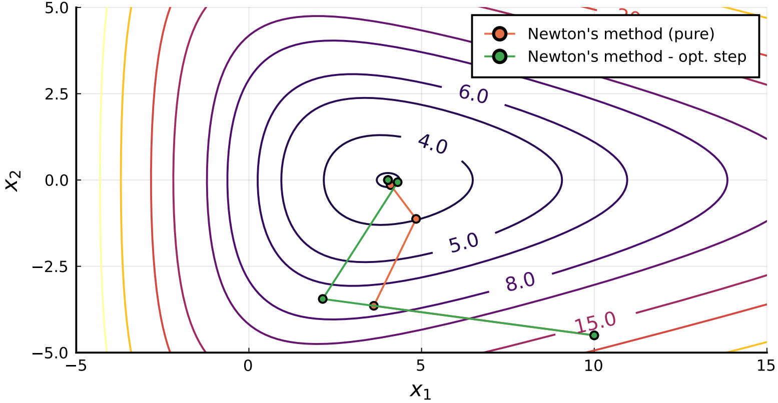
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



The complete trajectory of the pure Newton's method.

Newton's method

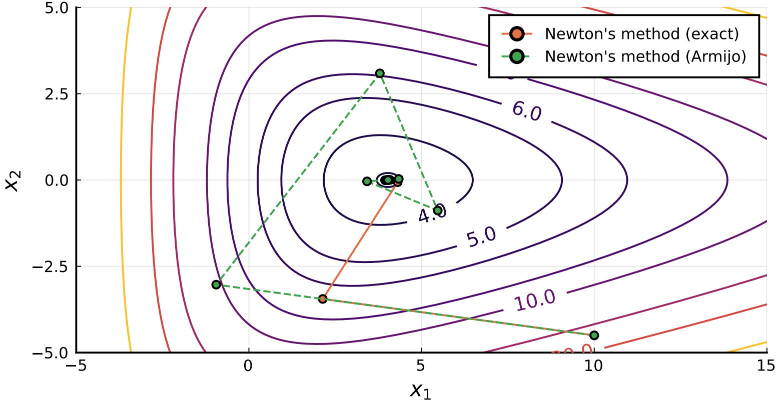
$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



When employing line searches, the direction $x_k - x_{k-1}$ from the pure method is used, but the actual step is optimised.

Newton's method

$$f(x) = e^{(-(x_1-3)/2)} + e^{((4x_2+x_1)/10)} + e^{((-4x_2+x_1)/10)}$$



Newton's method applied to f. Convergence is observed in 4 steps using exact line search and 27 using Armijo's rule ($\epsilon = 10^{-4}$)


THANK YOU