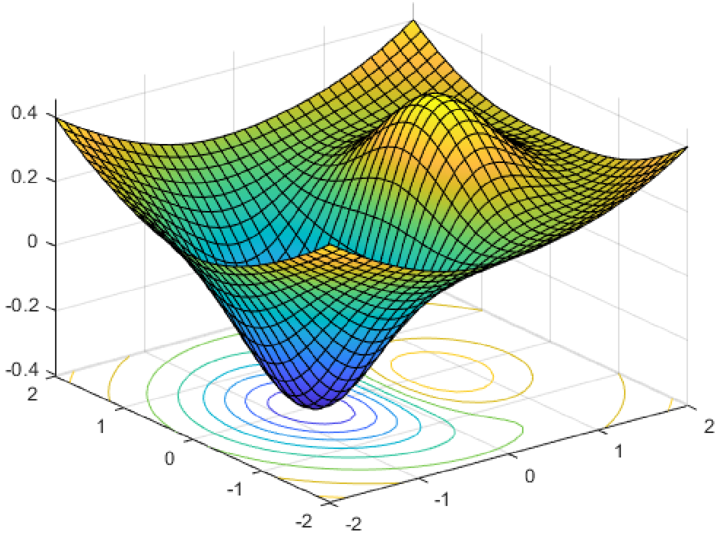
STT 1000 - STATISTIQUES

ARTHUR CHARPENTIER

Optimization



Optimisation: continuous (differentiable) case

The problem is to solve $\min_{y\in\mathbb{R}}\{f(y)\}$

Note: $\min_{y\in\mathbb{R}}\{f(y)\} = \max_{y\in\mathbb{R}}\{-f(y)\}$

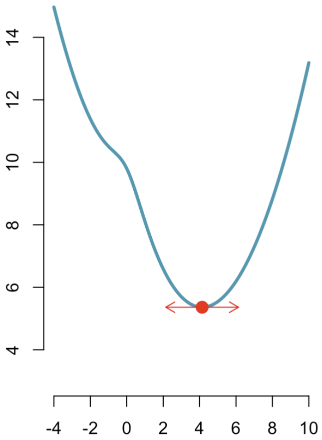
Note: $y^\star \in \text{argmin}_{y\in\mathbb{R}}\{f(y)\}$

and $\min_{y\in\mathbb{R}}\{f(y)\} = f(y^\star)$.

First order condition

$$f'(y^\star) = \left.\frac{\partial f(y)}{\partial y}\right|_{y=y^\star} = 0$$

(necessary condition)



Optimisation: continuous (differentiable) case

First order condition

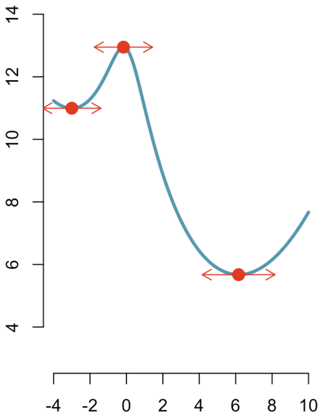
$$f'(y^\star) = \left.\frac{\partial f(y)}{\partial y}\right|_{y=y^\star} = 0$$

might be not sufficient

$$f''(y^\star) = \left.\frac{\partial^2 f}{\partial y^2}\right|_{y=y^\star} > 0 : \text{ minimum}$$

$$f''(y^\star) = \left.\frac{\partial^2 f}{\partial y^2}\right|_{y=y^\star} < 0 : \text{ maximum}$$

can be a local minimum...



Optimisation: continuous (differentiable) case

Example : $\{y_1, \cdots, y_n\}$ in $\mathbb{R}$, let

$$f(y) = \sum_{i=1}^{n} (y_i - y)^2$$

$$\frac{\partial f(y)}{\partial y} = \frac{\partial}{\partial y} \sum_{i=1}^{n} (y_i - y)^2 = \sum_{i=1}^{n} \frac{\partial (y_i - y)^2}{\partial y} = \sum_{i=1}^{n} -2(y_i - y)$$

so

$$\left.\frac{\partial f(y)}{\partial y}\right|_{y=y^\star} = 0 \text{ if and only if } \sum_{i=1}^{n} (y_i - y^\star) = 0 \text{ or } \sum_{i=1}^{n} y_i = ny^\star$$

i.e. $y^\star = \dfrac{1}{n} \displaystyle\sum_{i=1}^{n} y_i = \overline{y}$.

Optimisation: continuous (differentiable) case

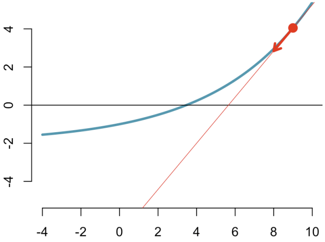
Solving $f'(y^\star) = 0$ numerically
Newton's method: solve $g(y^\star) = 0$

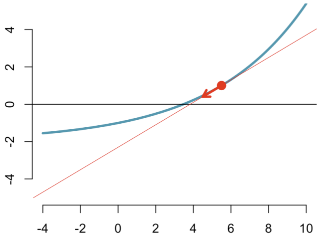
$$g(y) \simeq g(y_0) + g'(y_0)(y - y_0)$$

If $g(y) \simeq 0$, $g(y_0) + g'(y_0)(y - y_0) \simeq 0$

Start from y_0, then

$$y_{k+1} = y_k - \frac{g(y_k)}{g'(y_k)}$$



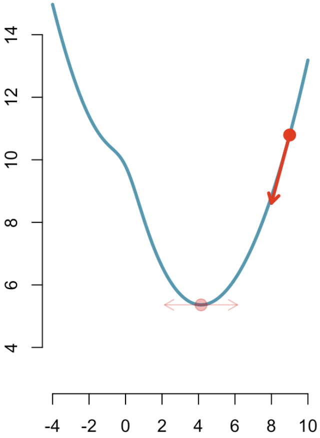


Optimisation: continuous (differentiable) case

To solve $f'(y^\star) = 0$ numerically
Start from y_0, then

$$y_{k+1} = y_k - \frac{f'(y_k)}{f''(y_k)}$$

```
> v = c(0.89367,-1.04729,1.97133,
  -0.38363,1.65414)
> f = function(x) sum((v-x)^2)
> df = function(x) -2*sum((v-x))
> d2f = function(x) 2*length(v)
> (x0 = 2)
[1] 2
> (x1 = x0-df(x0)/d2f(x0))
[1] 0.617644
> (x2 = x1-df(x1)/d2f(x1))
[1] 0.617644
```



Optimisation: continuous (differentiable) case

In python, we can solve $\text{argmin}\{f(a)\}$ for $a \in [-1, +1]$

```
> import statistics as stat
> v=[0.89367,-1.04729,1.97133,-0.38363,1.65414]
> stat.mean(v)
0.617644
> import numpy as np
> def f0(x):
...   return np.sum((np.array(v)-x)**2)
> f = np.vectorize(f0)
> from scipy.optimize import fminbound
> fminbound(f, -1, 1)
0.61764399999999992
```

or using a gradient descent, starting from $a_0 = 0$,

```
> from scipy.optimize import minimize
> minimize(f, 0, method='nelder-mead')
 final_simplex: (array([[0.617625 ],
        [0.6176875]]), array([6.75753495, 6.75753495]))
           fun: 6.7575349465249999
       message: 'Optimization terminated successfully.'
```

Optimisation: continuous (differentiable) case

or

```
> def f(x):
...   s=[0]*len(v)
...   for i in range(len(v)):
...     s[i]=((v[i]-x)**2)
...   return sum(s)
> fminbound(f, -1, 1)
0.617643999999992
```

and in R, using a gradient descent, starting from $a_0 = 0$,

```
> v = c(0.89367,-1.04729,1.97133,-0.38363,1.65414)
> mean(v)
[1] 0.617644
> f = function(x) sum((v-x)^2)
> optim(0, f)
$par
[1] 0.6175781
$value
[1] 6.757535
```

Optimisation: continuous (differentiable) case

The problem is $\min_{\boldsymbol{y}\in\mathbb{R}^p}\{f(\boldsymbol{y})\}$

or $\min_{(y_1,\cdots,y_p)\in\mathbb{R}^p}\{f(y_1,\cdots,y_p)\}$

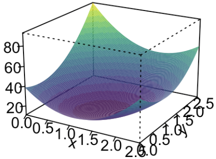
First order conditions: $\nabla f(\boldsymbol{y}^\star)=\boldsymbol{0}$,

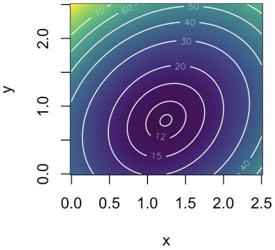
$$\left.\frac{\partial f(y_1,y_2,\cdots,y_p)}{\partial y_1}\right|_{\boldsymbol{y}=\boldsymbol{y}^\star}=0$$

$$\left.\frac{\partial f(y_1,y_2,\cdots,y_p)}{\partial y_2}\right|_{\boldsymbol{y}=\boldsymbol{y}^\star}=0$$

$$\vdots$$

$$\left.\frac{\partial f(y_1,y_2,\cdots,y_p)}{\partial y_p}\right|_{\boldsymbol{y}=\boldsymbol{y}^\star}=0$$





Optimisation: continuous (differentiable) case

To solve $\nabla f(\boldsymbol{y}^\star) = \boldsymbol{0}$ numerically
Start from $\boldsymbol{y}_0$, then

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k - \boldsymbol{H}_k \nabla f(\boldsymbol{y}_k)$$

$\nabla f(\boldsymbol{y}_k)$ gives the direction
$\boldsymbol{H}_k$ gives the speed of convergence
$\boldsymbol{H}_k$ is the inverse of the Hessian matrix

One can also consider some numerical tricks, see coordinate descent
where we iterate on the dimension
(univariate optimisation problems)