



NTNU

Norwegian University of
Science and Technology

TTK4135 – Lecture 16

Calculating derivatives and Derivative-free optimization

Lecturer: Lars Imsland

Lecture 16: Calculating derivatives (Ch. 8), and Derivative-free optimization (Ch. 9)

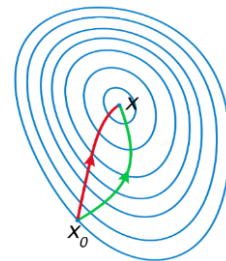
- Brief recap linesearch unconstrained optimization
- Calculating derivatives (gradient/Jacobian and Hessian)
- What can you do when obtaining derivatives is impractical?
 - Derivative-free optimization! For example: Nelder-Mead

Reference: N&W Ch. 8.1, (8.2), Ch. 9.1, 9.5

Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**. Newton's method uses curvature information to take a more direct route. (wikipedia.org)

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

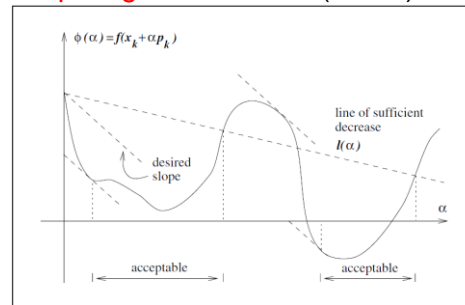
Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$



How to calculate derivatives – Ch. 8

Step length line search (Wolfe):



How many iterations? (Convergence rates)

Quasi-Newton: BFGS method

$$p_k = -B_k^{-1} \nabla f(x_k)$$

$$B_k \approx \nabla^2 f(x_k)$$

$$H_k = B_k^{-1}$$

Algorithm 6.1 (BFGS Method).

Given starting point x_0 , convergence tolerance $\epsilon > 0$,
inverse Hessian approximation H_0 ;

$k \leftarrow 0$;

while $\|\nabla f_k\| > \epsilon$;

 Compute search direction

$$p_k = -H_k \nabla f_k;$$

 Set $x_{k+1} = x_k + \alpha_k p_k$ where α_k is computed from a line search
 procedure to satisfy the Wolfe conditions (3.6);


 Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$;

 Compute H_{k+1} by means of (6.17);

$k \leftarrow k + 1$;

end (while)

← We use only gradient!


$$H_{k+1} = (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho_k s_k s_k^T$$

Example (from book)

- Using steepest descent, BFGS and inexact Newton on Rosenbrock function

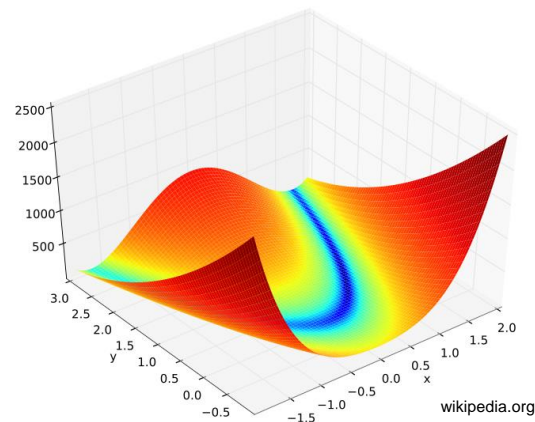
$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

- Iterations from starting point $(-1.2, 1)$:

- Steepest descent: 5264
- BFGS: 34
- Newton: 21

- Last iterations; value of $\|x_k - x^*\|$

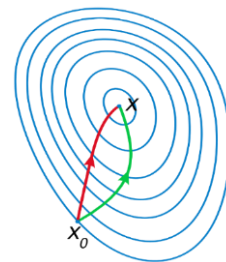
steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08



Learning goal Ch. 2, 3 and 6: Understand this slide

Line-search unconstrained optimization

$$\min_x f(x)$$



A comparison of **steepest descent** and **Newton's method**. Newton's method uses curvature information to take a more direct route. (wikipedia.org)

1. Initial guess x_0
2. While **termination criteria** not fulfilled
 - a) Find **descent direction** p_k from x_k
 - b) Find appropriate **step length** α_k ; set $x_{k+1} = x_k + \alpha_k p_k$
 - c) $k = k+1$
3. $x_M = x^*$? (possibly check sufficient conditions for optimality)

Termination criteria:

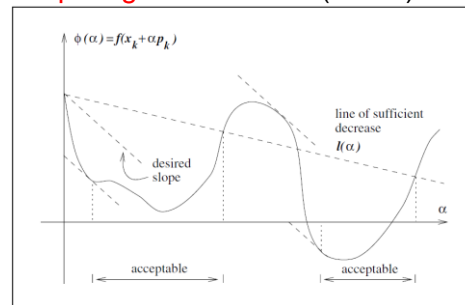
Stop when first of these become true:

- $\|\nabla f(x_k)\| \leq \epsilon$ (necessary condition)
- $\|x_k - x_{k-1}\| \leq \epsilon$ (no progress)
- $\|f(x_k) - f(x_{k-1})\| \leq \epsilon$ (no progress)
- $k \leq k_{\max}$ (kept on too long)

Descent directions:

- Steepest descent
 $p_k = -\nabla f(x_k)$
- Newton
 $p_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k)$
- Quasi-Newton
 $p_k = -B_k^{-1} \nabla f(x_k)$
 $B_k \approx \nabla^2 f(x_k)$

Step length line search (Wolfe):



Need derivatives! How to compute them?

And what if derivatives are not available, or too expensive to compute?

Four ways of finding derivatives

- By hand
 - Time consuming and (very!) error prone for large problems
- Symbolic differentiation
 - Computer algebra systems (CAS)
 - Maple, Mathematica, Matlab symbolic toolbox, ...
 - Typically result in very long code that is expensive to evaluate (and compile!)
- **Numerical differentiation (finite differences)**
 - Easy to implement (do it yourself), but may have low accuracy
- **Automatic (Algorithmic) Differentiation (AD)**
 - Best option!
 - Relatively easy to implement using the right software
 - Exact up to machine precision

Numerical differentiation

Theoretical accuracy of numerical differentiation

Taylor's theorem

$$f : \mathbb{R}^n \rightarrow \mathbb{R}, p \in \mathbb{R}^n$$

- First order: If f is continuously differentiable,

$$f(x + p) = f(x) + \nabla f(x + tp)^\top p, \quad \text{for some } t \in (0, 1)$$

- Second order: If f is twice continuously differentiable

$$f(x + p) = f(x) + \nabla f(x)^\top p + \frac{1}{2} p^\top \nabla^2 f(x + tp) p, \quad \text{for some } t \in (0, 1)$$

Finite differences and accuracy

One-sided difference: $\frac{\partial f}{\partial x_i} = \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} + O(\epsilon)$

Two-sided difference: $\frac{\partial f}{\partial x_i} = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon} + O(\epsilon^2)$

```
>> eps  
ans =  
2.2204e-16
```

Numerical differentiation (finite differences)

- Scalar $f : \mathbb{R} \rightarrow \mathbb{R}$: For some small ϵ ,

$$\frac{df}{dx} \approx \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

- Directional derivative of $f : \mathbb{R}^n \rightarrow \mathbb{R}$

$$\nabla f^\top p \approx \frac{f(x + \epsilon p) - f(x)}{\epsilon}$$

- Full gradient ∇f : Directional derivatives along all axes $p = \epsilon e_i$
($e_1 = (1, 0, 0, \dots)^\top, e_2 = (0, 1, 0, \dots)^\top, \dots$)

– Note: Not necessary to calculate full gradient if you only need directional derivative!
(Also valid for AD!)

- How to choose epsilon?

- Theoretical error proportional to ϵ , but too small ϵ gives numerical noise
- Rule of thumb: $\epsilon = \sqrt{\text{eps}}$, where eps is machine precision (or precision of computing f)
(IEEE double precision: $\epsilon = 10^{-8}$)

Approximating the Hessian

- In many cases, the gradient is available, but not the Hessian. We can then use finite differences on the gradient:

$$\nabla^2 f(x)p \approx \frac{\nabla f(x + \epsilon p) - \nabla f(x)}{\epsilon}$$

- If the gradient is not available, use finite differences “twice” for Hessian:

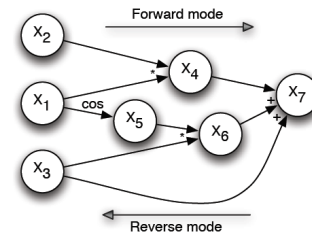
$$\frac{\partial^2 f}{\partial x_i \partial x_j}(x) = \frac{f(x + \epsilon e_i + \epsilon e_j) - f(x + \epsilon e_i) - f(x + \epsilon e_j) + f(x)}{\epsilon^2} + O(\epsilon)$$

(but usually better to use Quasi-Newton then...)

AD – Automatic (algorithmic) differentiation

- Software tools that automatically computes derivatives of your code
- The principle is simple: Extensive/automated use of ‘chain rule’
- Two modes; forward and reverse
- Two (main) implementation variants
 - Source code transformation
 - **Operator overloading** (object oriented language)
- Requires (more or less) that your implementation is differentiable
- Example software: ADOL-C, CppAD, CasADi, ...

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

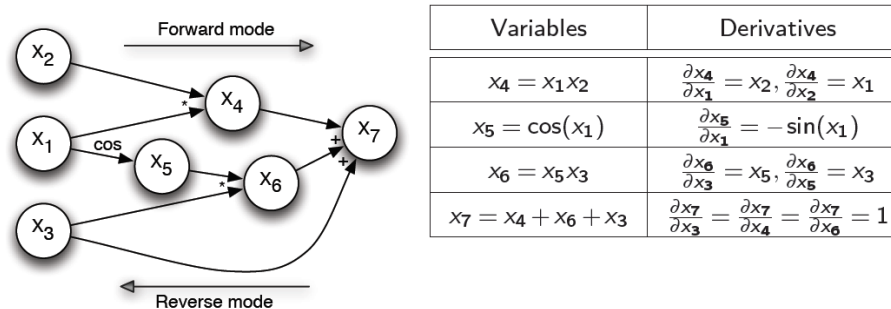


Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward and reverse

- Forward mode
 - Both x_i and ∇x_i are calculated by forward traversing computation graph

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$



- Reverse mode
 - First, calculate x_i by traversing graph forward
 - Then, calculate derivatives by traversing graph backward

AD – forward vs. reverse

- Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
 - Costs of calculating derivatives with AD:
 - Forward mode (one “column”):
 - Forward mode (entire Jacobian):
 - Reverse mode (one “row”):
 - Reverse mode (entire Jacobian):

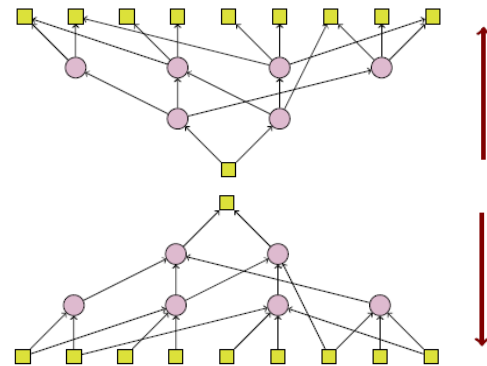
$$\text{cost}(\nabla f^\top p) \leq 2 \text{cost}(f)$$

$$\text{cost}(\nabla f) \leq 2n \text{cost}(f)$$

$$\text{cost}(\lambda^\top \nabla f) \leq 3 \text{cost}(f)$$

$$\text{cost}(\nabla f) \leq 3m \text{cost}(f)$$

- Forward mode: Similar cost as numerical differentiation, but more accurate
- If $m \gg n$, forward mode is fastest
- If $n \gg m$, reverse mode is fastest



How AD software is implemented

- Prototype procedure:
 - Decompose original code into “intrinsic” functions (e.g. x_1x_2 , $\sin(x)$, $\ln(x)$, etc.)
 - Differentiate the intrinsic functions (‘symbolically’, or make a lookup table) ($\sin(x)' = \cos(x)$, etc.)
 - Put everything together according to the chain rule (either forward or reverse mode)
- How to automatically transform your program into a program with derivatives? Two approaches:
 - Source code transformation (Typical: C, Fortran)
 - **Operator overloading** (C++, Fortran 90, Java, Matlab, Python, ...)

Example (C/C++)

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

function.c

```
double f(double x1, double x2, double x3) {  
    double x4, x5, x6, x7;  
  
    x4 = x1*x2;  
    x5 = cos(x1);  
    x6 = x3*x5;  
    x7 = x4 + x6 + x3;  
  
    return x7;  
}
```

function.c

Source code transformation (forward mode)

diff_function.c

```
double* f(double x1, double x2, double x3, double dx1, double dx2, double dx3) {  
    double x4, x5, x6, x7, dx4, dx5, dx7, df[2];  
  
    x4 = x1*x2;  
    dx4 = dx1*x2 + x1*dx2;  
    x5 = cos(x1);  
    dx5 = -sin(x1)*dx1;  
    x6 = x3*x5;  
    dx6 = dx3*x5 + x3*dx5;  
    x7 = x4 + x6 + x3;  
    dx7 = dx4 + dx6 + dx3;  
  
    df[0] = x7;  
    df[1] = dx7;  
  
    return df;  
}
```



Operator overloading example (using CppAD)

- Implement function as you do normally, but with other types (here: using C++ templates):

```
function.cpp

template <class vector>
vector f(vector x) {
    vector ... // possible temporary variables

    return x[1]*x[2] + x[3]*cos(x[1]) + x[3];
}
```

- Record operation sequence when you use the function:

```
...
CppAD::vector<ADdouble> x(3), f_res;
x[1] = pi; x[2] = 4; x[3] = 3;

// declare that x contains the independent variables (and start recording)
CppAD::Independent(x);

f_res = f(x);

// create the AD function object F : x -> f_res (and stop recording)
CppAD::ADFun<double> F(x, f_res);

std::vector<double> jac( NS*NS ) = F.Jacobian;
...
```

function.cpp and user program

AD library

compiler

object file with
derivatives

Software etc.

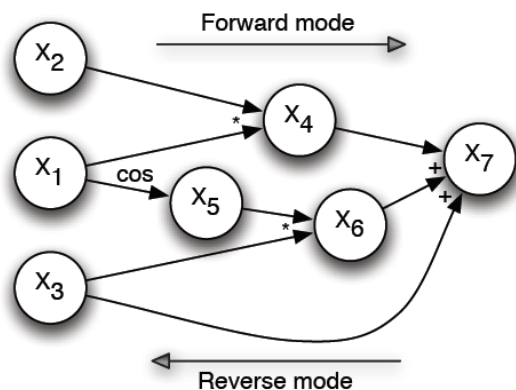
- General information
 - <http://www.autodiff.org/>
 - http://en.wikipedia.org/wiki/Automatic_differentiation
- Many libraries of different maturity/robustness/performance, for different languages and for different applications
- Some mature libraries
 - C++: ADOL-C, CppAD
 - Developed for control&optimization: CasADi (Matlab/Octave, Python, C++)
- Book:
 - A. Griewank, **A. Walther**, “Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation”, 2nd edition. SIAM, 2008.



AD – example (from R. Ringset)

- Calculate gradient of $f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$

at $[\pi \quad 4 \quad 3]^T$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

$$\nabla x_3 = e_3$$

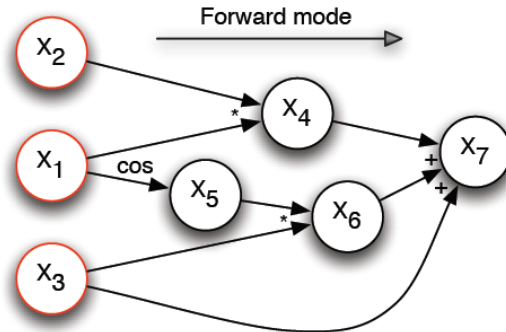
$$\nabla x_4 = ?$$

$$\nabla x_5 = ?$$

$$\nabla x_6 = ?$$

$$\nabla x_7 = ?$$

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^T$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

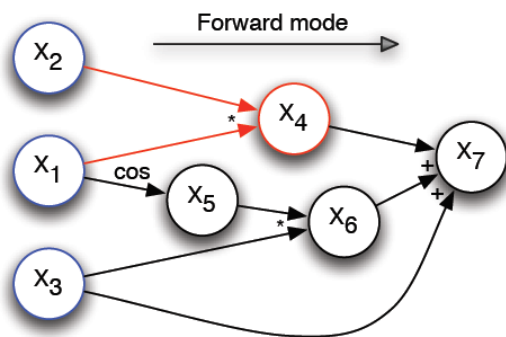
$$\nabla x_3 = e_3$$

$$\nabla x_4 = \frac{\partial x_4}{\partial x_1} \nabla x_1 + \frac{\partial x_4}{\partial x_2} \nabla x_2 = \begin{bmatrix} x_2 & x_1 & 0 \end{bmatrix}^T = \begin{bmatrix} 4 & \pi & 0 \end{bmatrix}^T$$

$$\nabla x_5 = ?$$

$$\nabla x_6 = ?$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

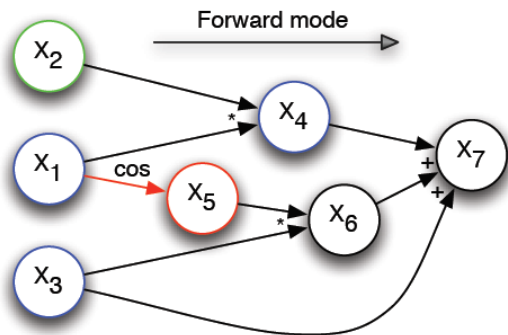
$$\nabla x_3 = e_3$$

$$\nabla x_4 = [4 \quad \pi \quad 0]^\top$$

$$\nabla x_5 = \frac{\partial x_5}{\partial x_1} \nabla x_1 = -\sin(x_1) e_1 = 0$$

$$\nabla x_6 = ?$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

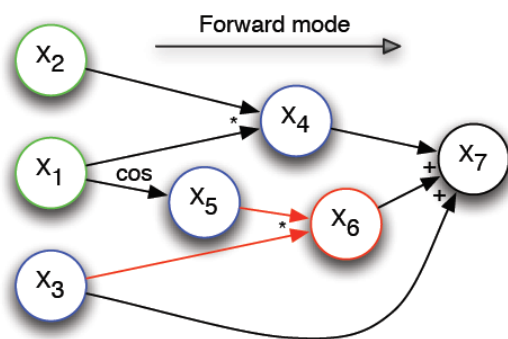
$$\nabla x_3 = e_3$$

$$\nabla x_4 = [4 \quad \pi \quad 0]^\top$$

$$\nabla x_5 = 0$$

$$\nabla x_6 = \frac{\partial x_6}{\partial x_3} \nabla x_3 + \frac{\partial x_6}{\partial x_5} \nabla x_5 = x_5 e_3 + x_3 0 = \cos(\pi) e_3 = -e_3$$

$$\nabla x_7 = ?$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\nabla x_1 = e_1$$

$$\nabla x_2 = e_2$$

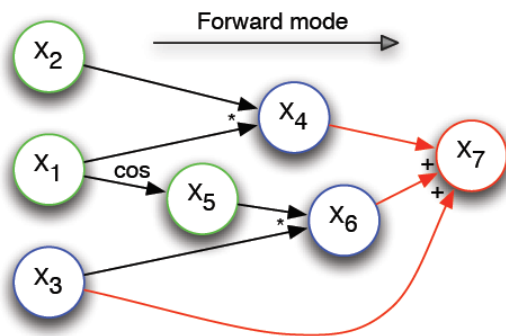
$$\nabla x_3 = e_3$$

$$\nabla x_4 = \begin{bmatrix} 4 & \pi & 0 \end{bmatrix}^\top$$

$$\nabla x_5 = 0$$

$$\nabla x_6 = -e_3$$

$$\nabla x_7 = \nabla f(x) = \frac{\partial x_7}{\partial x_4} \nabla x_4 + \frac{\partial x_7}{\partial x_6} \nabla x_6 + \frac{\partial x_7}{\partial x_3} \nabla x_3 = \begin{bmatrix} 4 \\ \pi \\ 0 \end{bmatrix} - e_3 + e_3 = \begin{bmatrix} 4 \\ \pi \\ 0 \end{bmatrix}$$

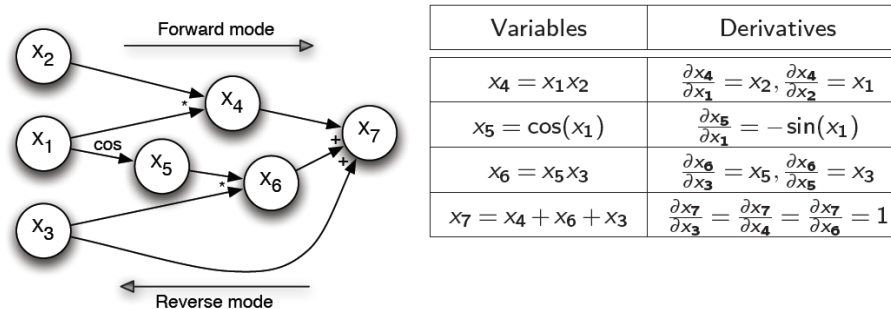


Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – forward and reverse

- Forward mode
 - Both x_i and ∇x_i are calculated by forward traversing computation graph

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$



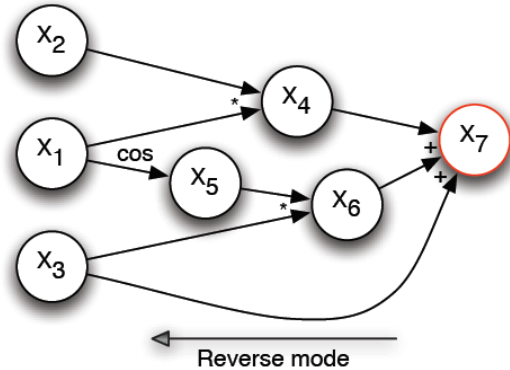
- Reverse mode
 - First, calculate x_i by traversing graph forward
 - Then, calculate derivatives by traversing graph backward

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$



$$\frac{\partial f}{\partial x_i} = \sum_{x_n \text{ child of } x_i} \frac{\partial f}{\partial x_n} \frac{\partial x_n}{\partial x_i}$$

Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

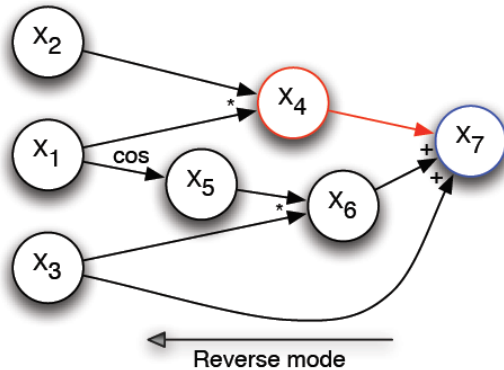
AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

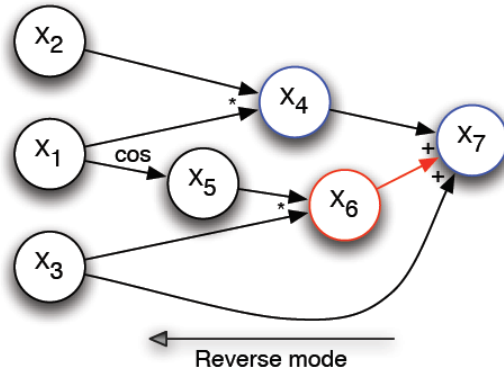
$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

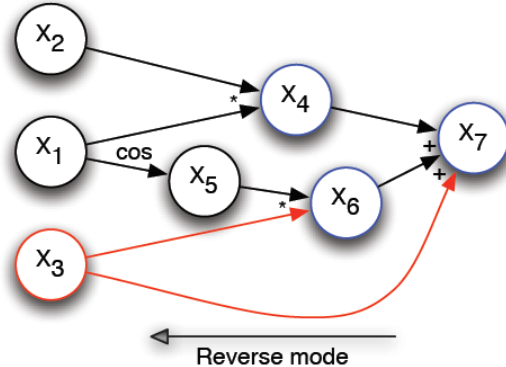
$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = 0$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

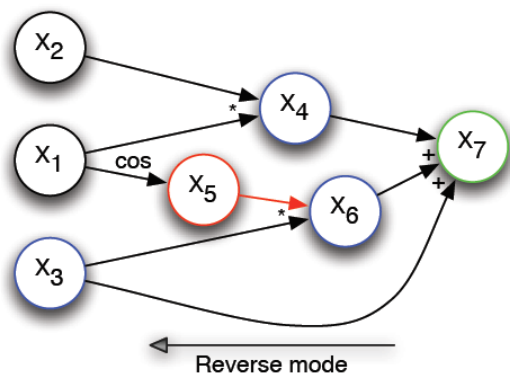
$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = 0$$

$$\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_5} = x_3 = 3$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

AD – reverse mode

$$f(x_1, x_2, x_3) = x_1 x_2 + x_3 \cos(x_1) + x_3$$

$$x = (\pi, 4, 3)^\top$$

$$\frac{\partial f}{\partial x_7} = \frac{\partial x_7}{\partial x_7} = 1$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_4} = 1$$

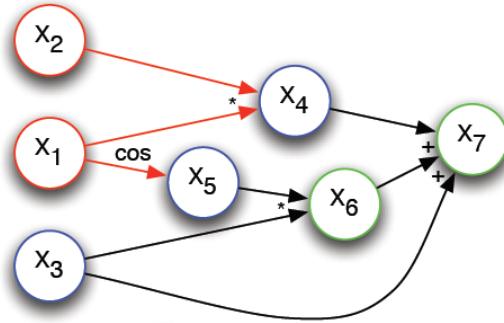
$$\frac{\partial f}{\partial x_6} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_6} = 1$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial x_7} \frac{\partial x_7}{\partial x_3} + \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_3} = 1 + x_5 = 1 + \cos(\pi) = \underline{0}$$

$$\frac{\partial f}{\partial x_5} = \frac{\partial f}{\partial x_6} \frac{\partial x_6}{\partial x_5} = x_3 = 3$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial x_4} \frac{\partial x_4}{\partial x_1} + \frac{\partial f}{\partial x_5} \frac{\partial x_5}{\partial x_1} = x_2 - 3 \sin(x_1) = 4 - 3 \sin(\pi) = \underline{4}$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial x_4} \frac{\partial x_4}{\partial x_2} = x_1 = \underline{\pi}$$



Variables	Derivatives
$x_4 = x_1 x_2$	$\frac{\partial x_4}{\partial x_1} = x_2, \frac{\partial x_4}{\partial x_2} = x_1$
$x_5 = \cos(x_1)$	$\frac{\partial x_5}{\partial x_1} = -\sin(x_1)$
$x_6 = x_5 x_3$	$\frac{\partial x_6}{\partial x_3} = x_5, \frac{\partial x_6}{\partial x_5} = x_3$
$x_7 = x_4 + x_6 + x_3$	$\frac{\partial x_7}{\partial x_3} = \frac{\partial x_7}{\partial x_4} = \frac{\partial x_7}{\partial x_6} = 1$

Navigation icons: back, forward, search, etc.

Example: optimization using CasADi

- CasADi (<https://casadi.org/>)
 - “CasADi is a symbolic framework for numeric optimization implementing automatic differentiation in forward and reverse modes on sparse matrix-valued computational graphs.”

$$\begin{aligned} \min_{x,y,z} \quad & x^2 + 100z^2 \\ \text{s.t.} \quad & z + (1-x)^2 - y = 0 \end{aligned}$$

Define variables

Define objective and constraints

Create solver object

Solve the opt problem



rosenbrock.m

```
import casadi.*

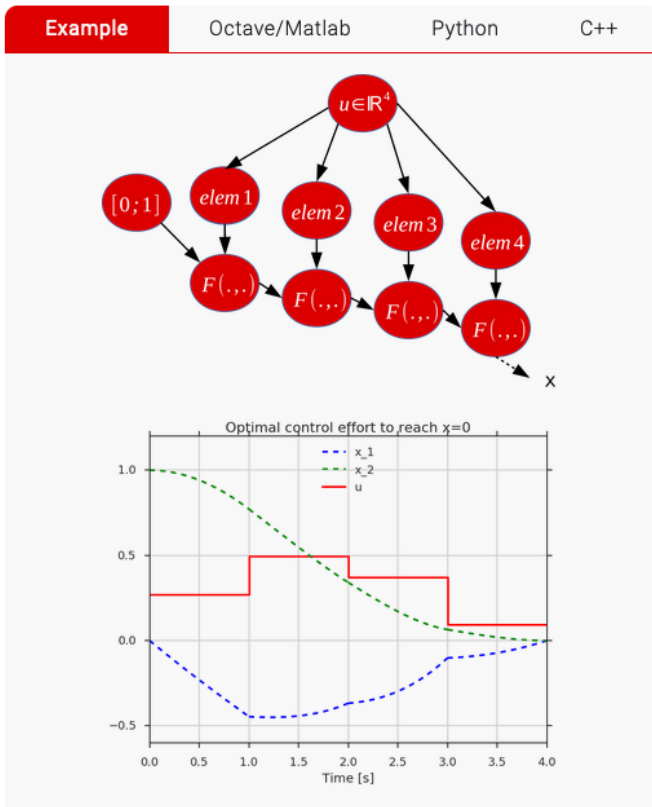
% Create NLP: Solve the Rosenbrock problem:
%   minimize    x^2 + 100*z^2
%   subject to  z + (1-x)^2 - y == 0
x = SX.sym('x');
y = SX.sym('y');
z = SX.sym('z');
v = [x;y;z];
f = x^2 + 100*z^2;
g = z + (1-x)^2 - y;
nlp = struct('x', v, 'f', f, 'g', g);

% Create IPOPT solver object
solver = nlpsol('solver', 'ipopt', nlp);

% Solve the NLP
res = solver('x0', [2.5 3.0 0.75], ... % solution guess
            'lbx', -inf, ...           % lower bound on x
            'ubx', inf, ...            % upper bound on x
            'lbz', 0, ...              % lower bound on z
            'ubz', 0);                % upper bound on z

% Print the solution
f_opt = full(res.f) % >> 0
x_opt = full(res.x) % >> [0; 1; 0]
lam_x_opt = full(res.lam_x) % >> [0; 0; 0]
lam_g_opt = full(res.lam_g) % >> 0
```

Example from CasADi

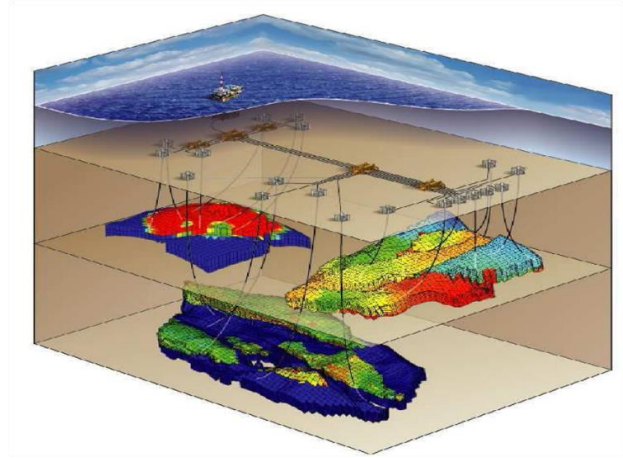
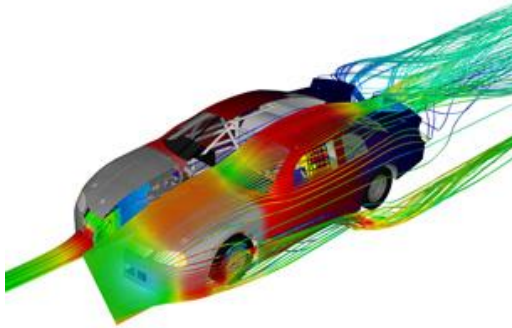


Example	Octave/Matlab	Python	C++
<pre> from casadi import * x = MX.sym('x',2); # Two states p = MX.sym('p'); # Free parameter # Expression for ODE right-hand side z = 1-x[1]**2; rhs = vertcat(z*x[0]-x[1]+2*tanh(p),x[0]) # ODE declaration with free parameter ode = {'x':x,'p':p,'ode':rhs} # Construct a Function that integrates over 1s F = integrator('F','cvodes',ode,{'tf':1}) # Control vector u = MX.sym('u',4,1) x = [0,1] # Initial state for k in range(4): # Integrate 1s forward in time: # call integrator symbolically res = F(x0=x,p=u[k]) x = res["xf"] # NLP declaration nlp = {'x':u,'f':dot(u,u),'g':x}; # Solve using IPOPT solver = nlpsol('solver','ipopt',nlp) res = solver(x0=0.2,lbg=0,ubg=0) plot(res["x"]) </pre>			



Derivative-free optimization

- If you have derivatives (gradients, possibly Hessian), use them!
 - “Always” more efficient than not using them!
- However, sometimes, obtaining derivatives is prohibitive
 - Typically: Objective function (and constraints) are calculated using “heavy” simulators
 - Often models from computational fluid dynamics (CFD)

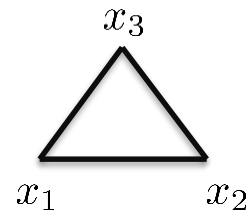


- This motivates “derivative-free optimization” methods”

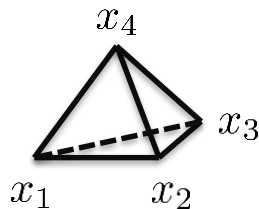
Derivative-free optimization (DFO)

- DFO use function values at a set of **sample points** to determine new iterates.
- Coarsely, two different classes of methods:
 1. “**Model-based**”: sample points -> approximate model -> search directions
 2. “**Metaheuristics**”: Often inspired by processes in nature, such as “genetic algorithm”, “simulated annealing”, “particle swarm optimization”, “wolf pack optimization”, ...
- Many of the metaheuristic methods claim to do “global optimization” and tackle “non-differentiable problems”, but this must be interpreted with care. Guarantees are seldom given.
- DFO generally works best if the number of optimization variables is relatively small
- Here: Nelder-Mead (old&simple, but fairly good)

Nelder-Mead Simplex method (Ch. 9.5)

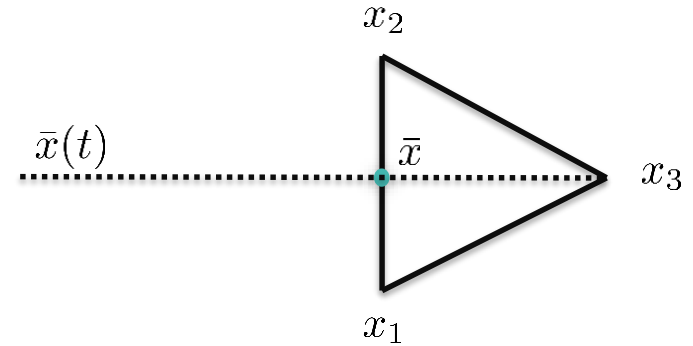


Simplex 2D

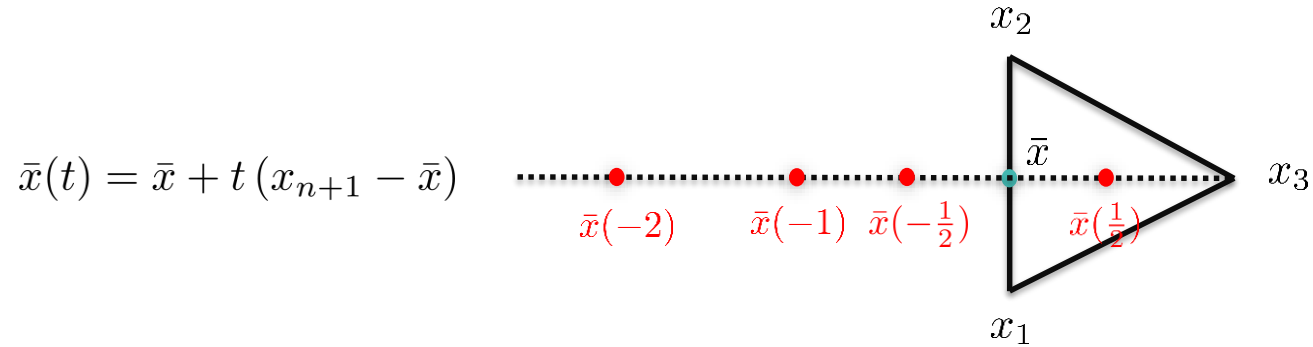


Simplex 3D

Nelder-Mead Simplex method (Ch. 9.5)



One iteration of Nelder-Mead

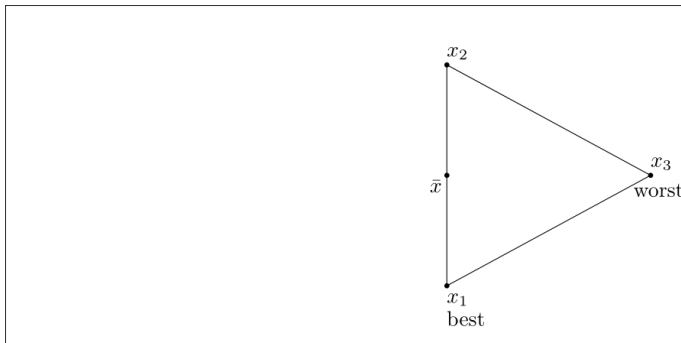


One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

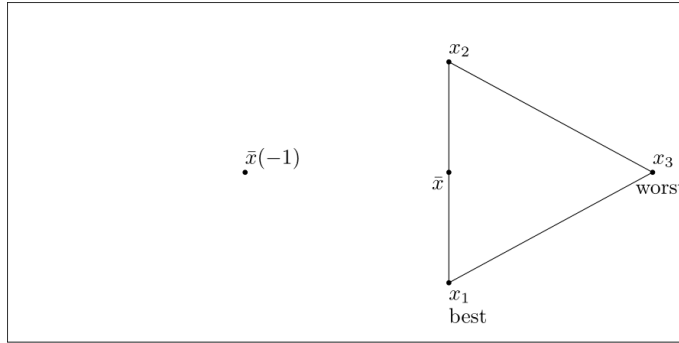
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Reflection

```

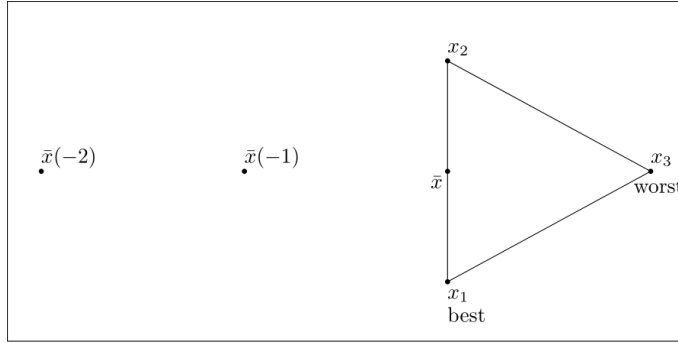
Compute  $\bar{x}(-1)$  and evaluate  $f_{-1} = f(\bar{x}(-1))$ 
if  $f(x_1) \leq f_{-1} < f(x_n)$   “ $\bar{x}(-1)$  is OK”
    replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} < f(x_1)$   “ $\bar{x}(-1)$  is great, try further”
    evaluate  $f_{-2} = f(\bar{x}(-2))$ 
    if  $f_{-2} < f_{-1}$ 
        replace  $x_{n+1}$  by  $\bar{x}(-2)$ , go to next iteration.
    else
        replace  $x_{n+1}$  by  $\bar{x}(-1)$ , go to next iteration.
else if  $f_{-1} \geq f(x_n)$   “ $\bar{x}(-1)$  is bad”
    if  $f(x_n) \leq f_{-1} < f(x_{n+1})$ 
        evaluate  $f_{-1/2} = f(\bar{x}(-1/2))$ 
        if  $f_{-1/2} \leq f_{-1}$ 
            replace  $x_{n+1}$  by  $\bar{x}(-1/2)$ , go to next iteration.
        else ( $f_{-1} > f(x_{n+1})$ )
            evaluate  $f_{1/2} = f(\bar{x}(1/2))$ 
            if  $f_{1/2} < f_{n+1}$ 
                replace  $x_{n+1}$  by  $\bar{x}(1/2)$ , go to next iteration.
            replace  $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$ ,  $i = 2, 3, \dots, n + 1$   “shrink”
    
```

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Expansion

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

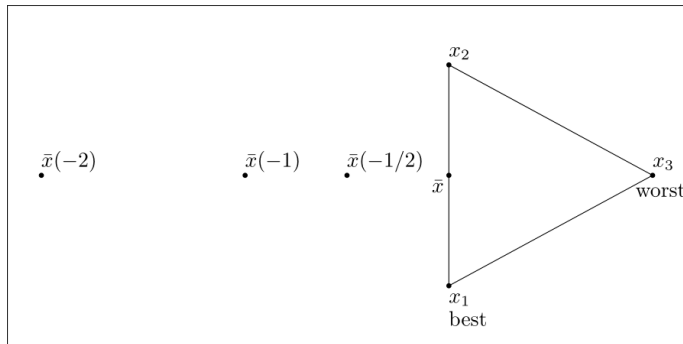
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Contraction (outside)

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

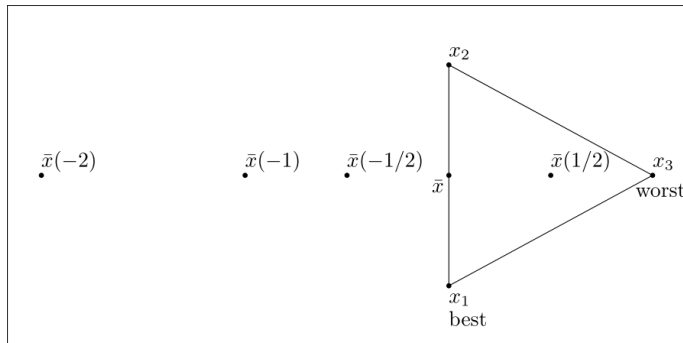
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Contraction (inside)

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

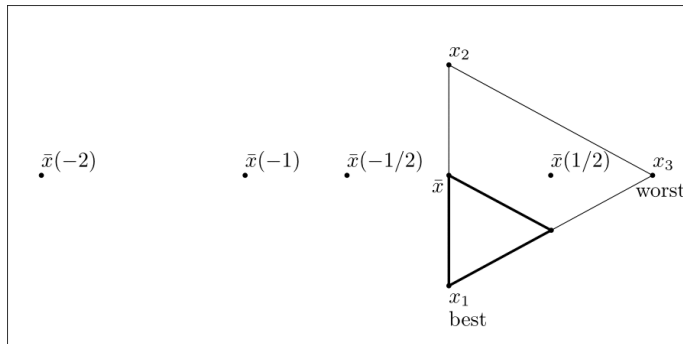
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

One iteration of Nelder-Mead

$n + 1$ vertices $\{x_1, x_2, \dots, x_{n+1}\}$ of *nonsingular* simplex, ordered such that $f(x_1) \leq f(x_2) \leq \dots \leq f(x_{n+1})$

Define centroid of n best points, $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$.

Define $\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x})$



Shrinkage

Compute $\bar{x}(-1)$ and evaluate $f_{-1} = f(\bar{x}(-1))$

if $f(x_1) \leq f_{-1} < f(x_n)$ “ $\bar{x}(-1)$ is OK”

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} < f(x_1)$ “ $\bar{x}(-1)$ is great, try further”

evaluate $f_{-2} = f(\bar{x}(-2))$

if $f_{-2} < f_{-1}$

replace x_{n+1} by $\bar{x}(-2)$, go to next iteration.

else

replace x_{n+1} by $\bar{x}(-1)$, go to next iteration.

else if $f_{-1} \geq f(x_n)$ “ $\bar{x}(-1)$ is bad”

if $f(x_n) \leq f_{-1} < f(x_{n+1})$

evaluate $f_{-1/2} = f(\bar{x}(-1/2))$

if $f_{-1/2} \leq f_{-1}$

replace x_{n+1} by $\bar{x}(-1/2)$, go to next iteration.

else ($f_{-1} > f(x_{n+1})$)

evaluate $f_{1/2} = f(\bar{x}(1/2))$

if $f_{1/2} < f_{n+1}$

replace x_{n+1} by $\bar{x}(1/2)$, go to next iteration.

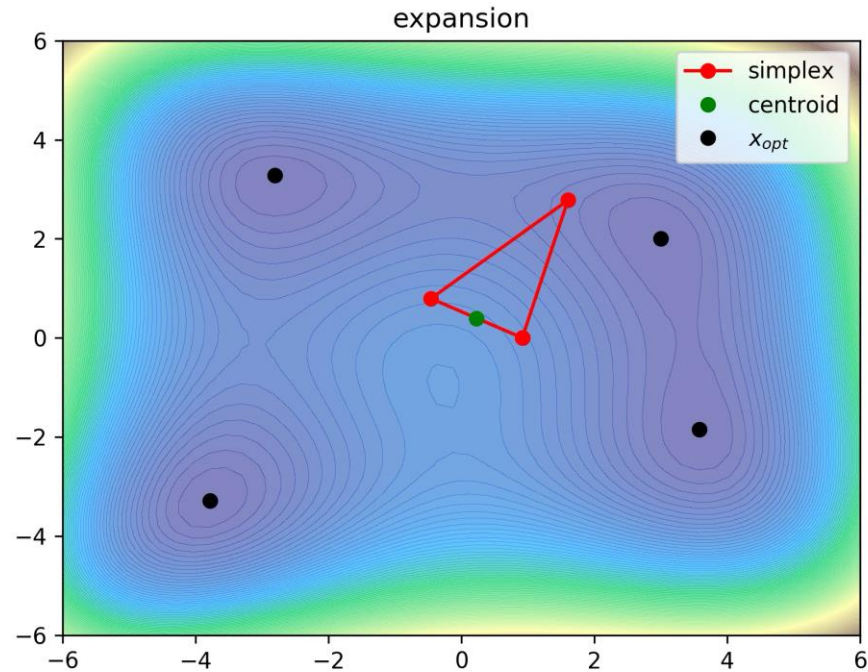
replace $x_i \leftarrow \frac{1}{2}(x_1 + x_i)$, $i = 2, 3, \dots, n + 1$ “shrink”

Termination and convergence

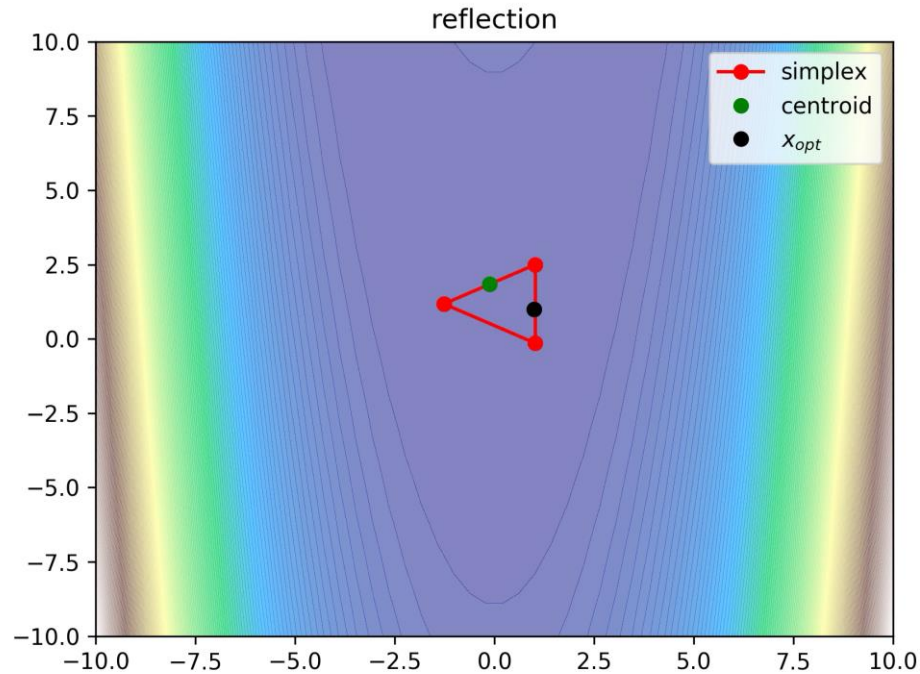
Termination: $|f(x_1) - f(x_{n+1})| \leq \text{tol}$

Convergence: The average value $\frac{1}{n+1} \sum_{i=1}^{n+1} f(x_i)$ decrease in most iterations

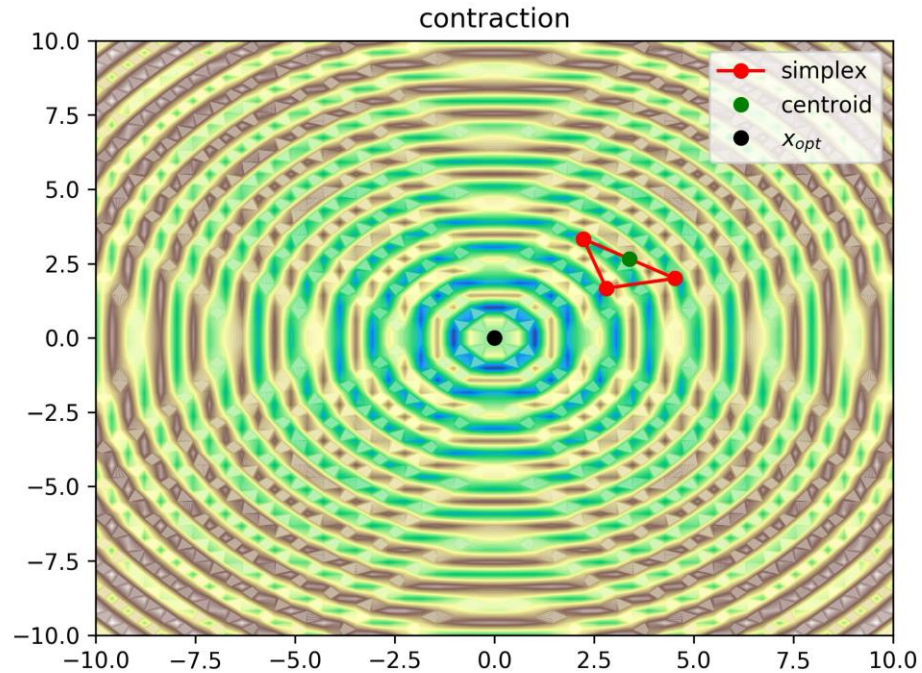
Examples: Nelder-Mead, Himmelblau



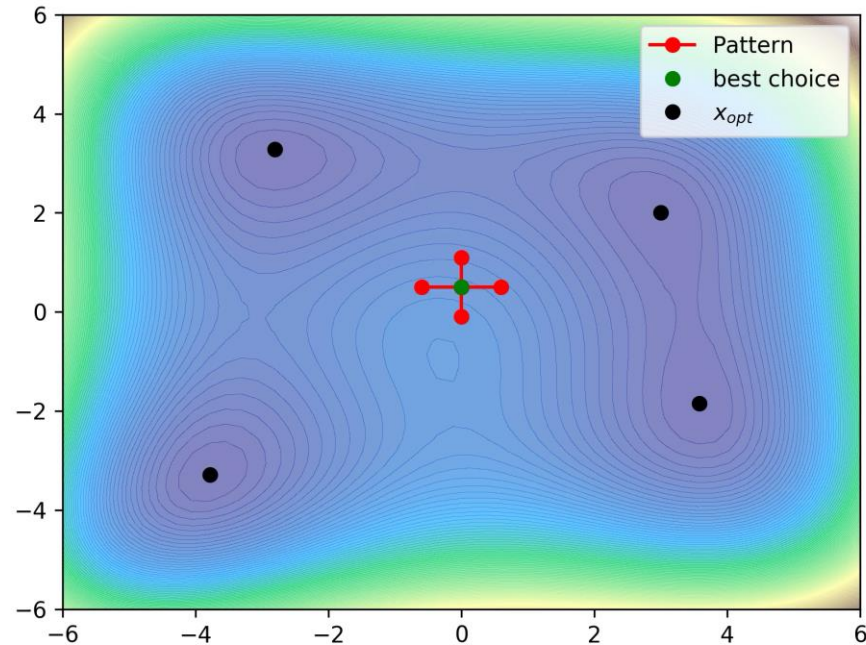
Example: Nelder-Mead, Rosenbrock



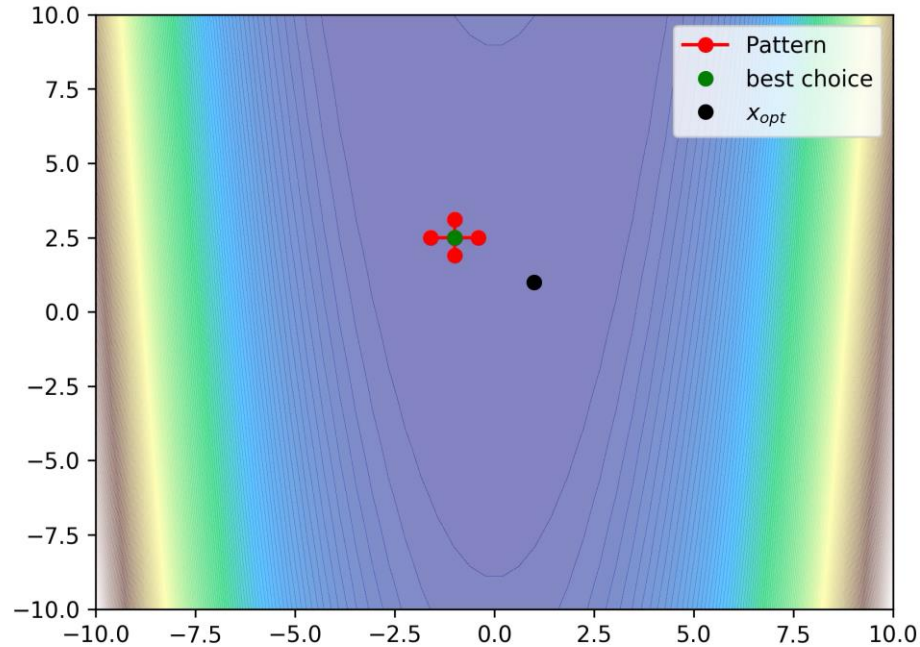
Example: Nelder-Mead, Salomon



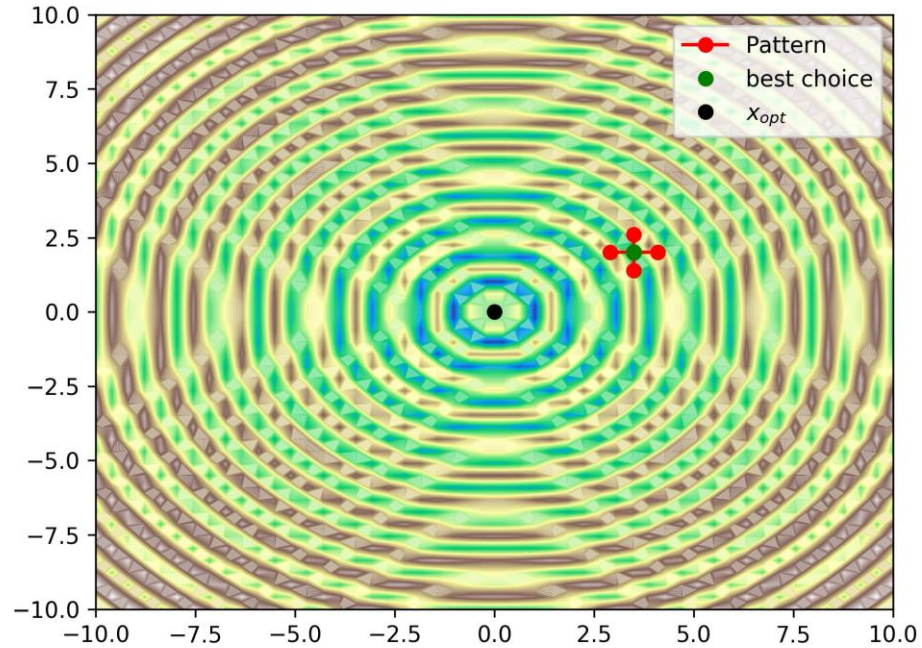
Example: Coordinate Descent, Himmelblau



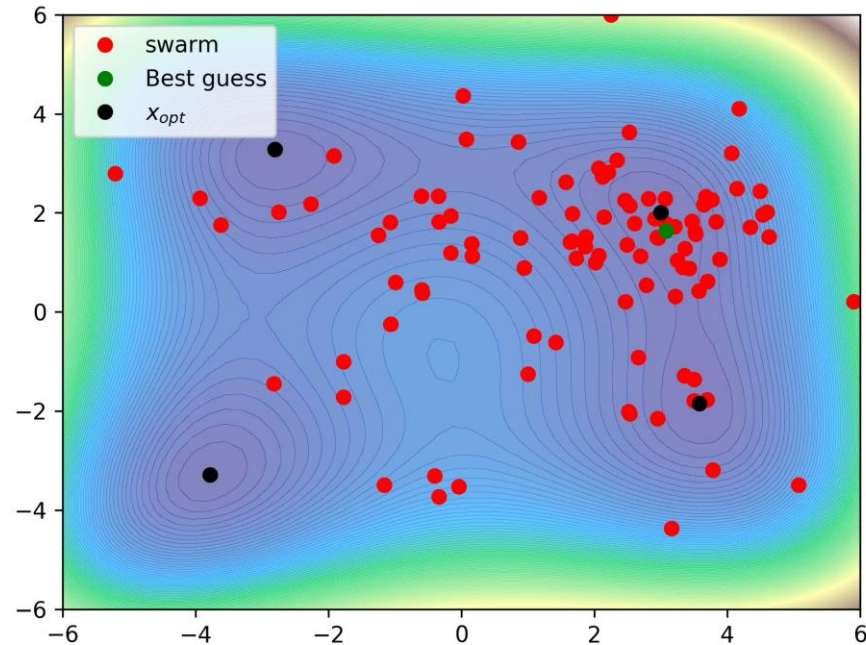
Example: Coordinate Descent, Rosenbrock



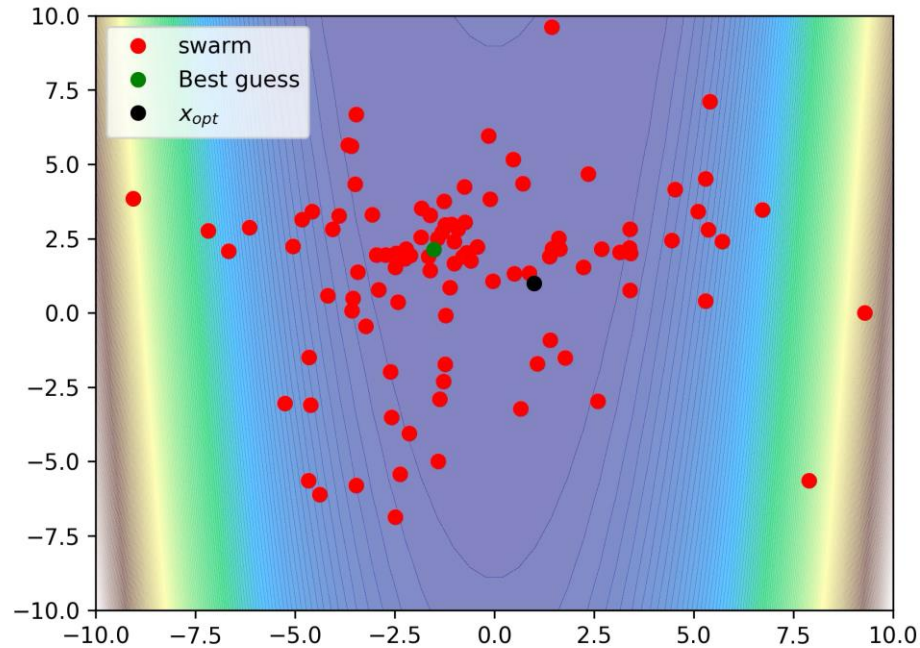
Example: Coordinate Descent, Salomon



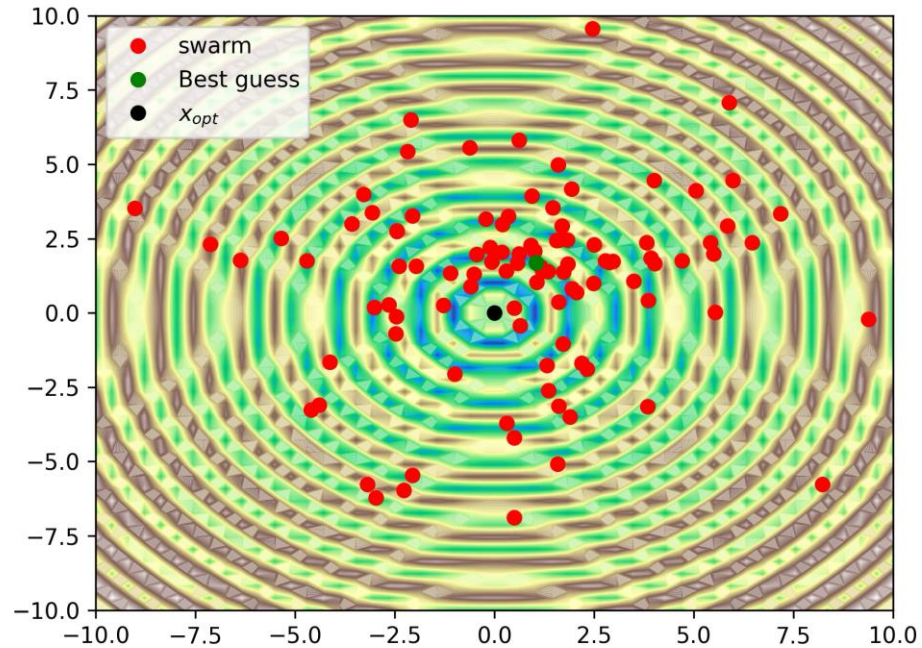
Example: Particle Swarm, Himmelblau



Example: Particle Swarm, Rosenbrock



Example: Particle Swarm, Salomon



Examples

