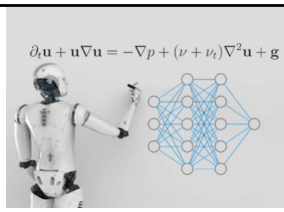


Hands on Machine Learning for Fluid Dynamics

7 – 11 February 2022

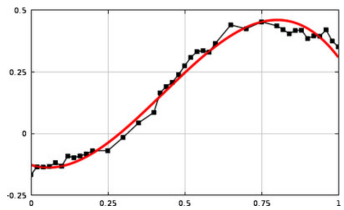
von KARMAN INSTITUTE FOR FLUID DYNAMICS

Lecture 5
A Review of Optimization Tools

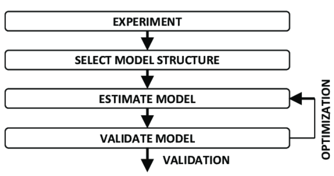
Pedro Afonso Marques
pedro.marques@vki.ac.be

Introduction

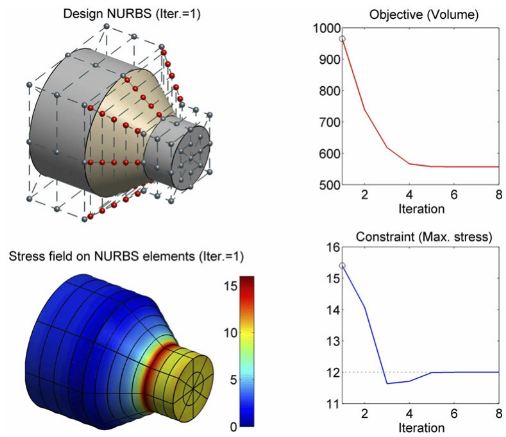
Optimization
"The process of finding the best possible solution to a problem. In mathematics, this often consists of maximizing or minimizing the value of a certain function, perhaps subject to given constraints." - Oxford dictionary



Curve fitting



System identification [1]

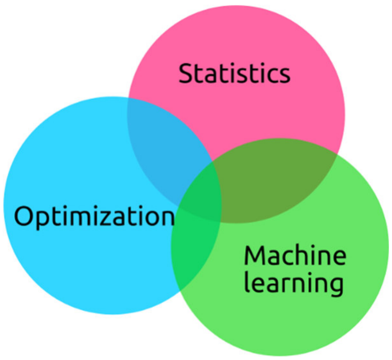


Shape optimization [2]

Economic models & resource allocation [3]

Introduction

Most Machine Learning algorithms can be framed as optimization problems…



Data: Training & Validation

$\mathbf{x}_*, \mathbf{x}_{**} \in \mathcal{X}$

$\mathbf{y}_*, \mathbf{y}_{**} \in \mathcal{Y}$

Hypothesis Set (Model):

$\mathbf{y} = \tilde{f}(\mathbf{x}, \mathbf{w}), \mathbf{w} \in \mathcal{W}$

Learning Algorithm
(Performance **Optimization**)

Final Hypothesis

$\tilde{f} \approx f$

Note: * denotes training data, ** denotes validation data

Table of Contents

Table of Contents

Formulating an optimization problem

Minimize:

- Objective function $f(\mathbf{x})$

Where:

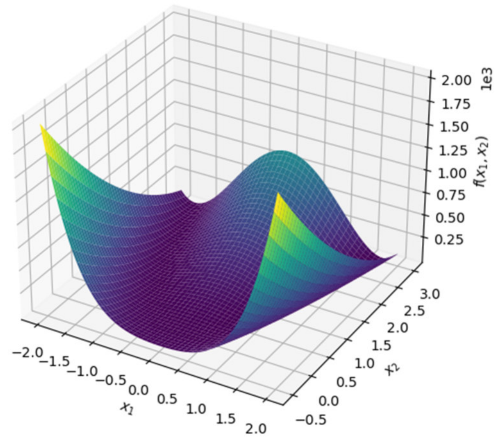
- Vector of variables $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}$

Subject to:

- Inequality constraints: $c_i(\mathbf{x}) \leq 0$ for $i = 1, ..., m$
- Equality constraints: $c_e(\mathbf{x}) = 0$ for $e = 1, ..., n$
- Variable range: $x_j \in [x_j^l, x_j^u]$ for $j = 1, ..., p$

Example: Rosenbrock function

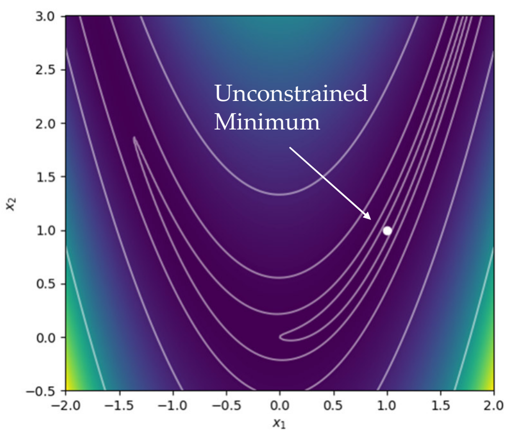
$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$



Formulating an optimization problem

Example: Rosenbrock function

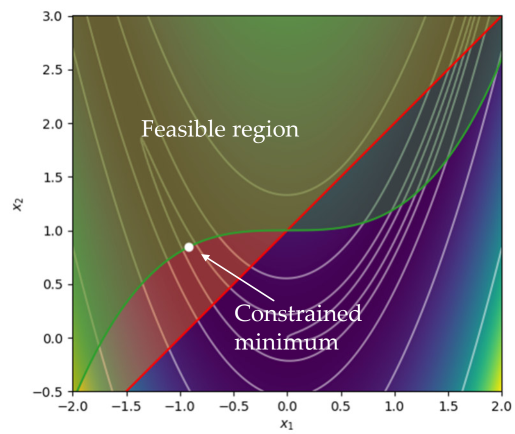
$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$



Unconstrained minimum: $f(1{,}1) = 0$

$c_1(x_1, x_2) = x_1 - x_2 + 1 \leq 0$

$c_2(x_1, x_2) = 0.2x_1^3 - x_2 + 1 \leq 0$



Constrained minimum: $f(-0.9{,}0.8) \approx 3.68$

Table of Contents

Constrained single-objective optimization

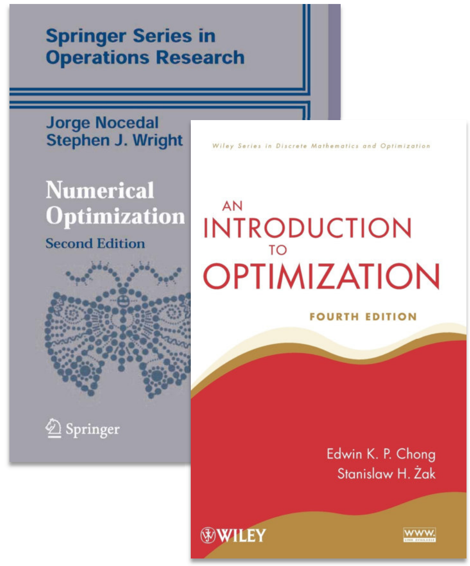
1. Handle constraints directly

2. External penalty method

3. Internal penalty method

Constrained single-objective optimization

1. Handle constraints directly
   - Equality: Lagrange multiplier method
   - Inequality: Linear/nonlinear/quadratic programming



2. External penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + R \cdot \sum_{i=1}^{m} \delta_i \cdot (c_i(\mathbf{x}))^2 \quad \text{where} \quad \delta_i = \begin{cases} 0 & , \quad c_i(\mathbf{x}) \leq 0 \\ 1 & , \quad c_i(\mathbf{x}) > 0 \end{cases}$$

3. Internal penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - R \cdot \sum_{i=1}^{m} \frac{1}{c_i(\mathbf{x})}$$

Constrained single-objective optimization

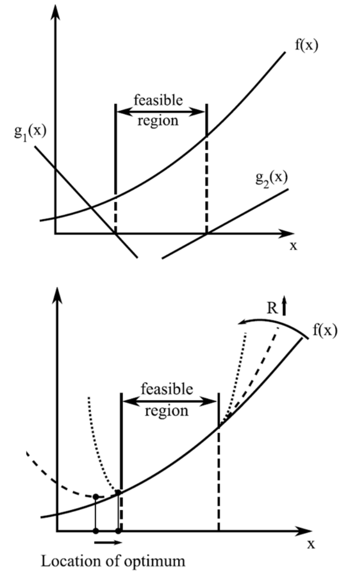
1. Handle constraints directly
   - Equality: Lagrange multiplier method
   - Inequality: Linear/nonlinear/quadratic programming

2. External penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + R \cdot \sum_{i=1}^{m} \delta_i \cdot (c_i(\mathbf{x}))^2 \quad \text{where} \quad \delta_i = \begin{cases} 0 & , \; c_i(\mathbf{x}) \leq 0 \\ 1 & , \; c_i(\mathbf{x}) > 0 \end{cases}$$

3. Internal penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - R \cdot \sum_{i=1}^{m} \frac{1}{c_i(\mathbf{x})}$$



Constrained single-objective optimization

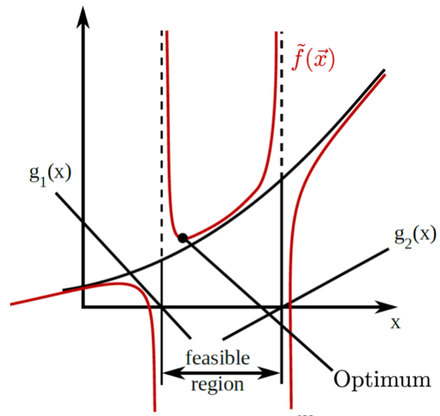
1. Handle constraints directly
   - Equality: Lagrange multiplier method
   - Inequality: Linear/nonlinear/quadratic programming

2. External penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + R \cdot \sum_{i=1}^{m} \delta_i \cdot (c_i(\mathbf{x}))^2 \quad \text{where} \quad \delta_i = \begin{cases} 0 & , \; c_i(\mathbf{x}) \le 0 \\ 1 & , \; c_i(\mathbf{x}) > 0 \end{cases}$$

3. Internal penalty method

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) - R \cdot \sum_{i=1}^{m} \frac{1}{c_i(\mathbf{x})}$$



$$\tilde{f}(\vec{x}) = f(\vec{x}) - R \sum_{j=1}^{m} \frac{1}{g_j(\vec{x})}$$

Table of Contents

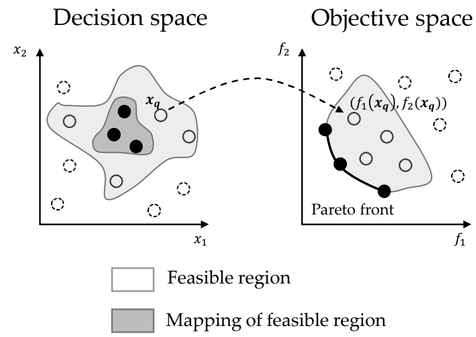
Multi-objective optimization

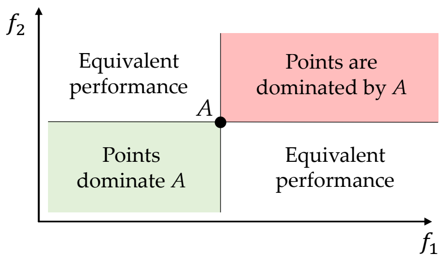
Minimize:

- Multiple objective functions $f_k(\mathbf{x})$ for $k = 1, \ldots, l$

Subject to:

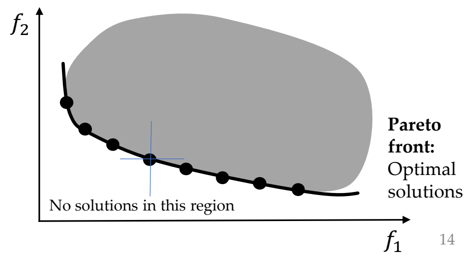
- Inequality constraints: $c_i(\mathbf{x}) \leq 0$ for $i = 1, \ldots, m$
- Equality constraints: $c_e(\mathbf{x}) = 0$ for $e = 1, \ldots, n$
- Variable range: $x_j \in [x_j^l, x_j^u]$ for $j = 1, \ldots, p$





A dominates:
$f_k(A) \leq f_1(\text{Red})$

A is dominated:
$f_k(\text{Green}) \leq f_k(A)$



Difference between objectives and constraints

Single-objective constrained optimization
E.g.: Optimize the volume $V(x, y, z)$ of a component, but make sure that the maximum stress $\sigma(x, y, z)$ is lower than a certain amount.

$$\begin{aligned} &\min \; V(\mathbf{x}) \\ &\text{s.t.} \;\; \sigma(\mathbf{x}) \leq C \\ &\qquad \mathbf{x} \in \mathbf{X} \end{aligned}$$

Multi-objective optimization
E.g.: Simultaneously optimize the volume $V(x, y, z)$ of a component, and minimize maximum stress $\sigma(x, y, z)$ it is subjected to.

$$\begin{aligned} &\min \;\; (V(\mathbf{x}), \sigma(\mathbf{x})) \\ &\text{s.t.} \;\; \mathbf{x} \in \mathbf{X} \end{aligned}$$

Multi to single-objective optimization

1. Weighted sum of each individual objective

2. Min-max formulation

3. Constrained minimum

Multi to single-objective optimization

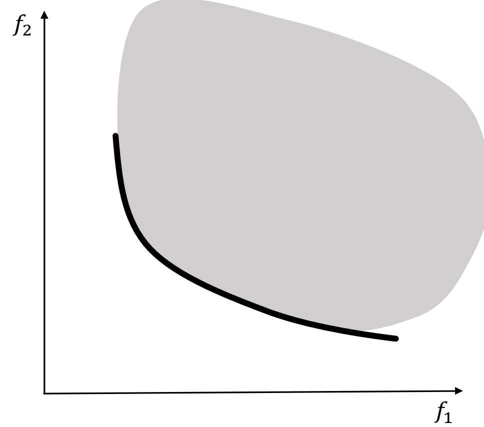
1. Weighted sum of each individual objective

$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum
   - Minimize f_1 assuming $f_2 = \text{cte}$
   - Minimize f_2 assuming $f_1 = \text{cte}$



$$F(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x})$$

Multi to single-objective optimization

1. Weighted sum of each individual objective

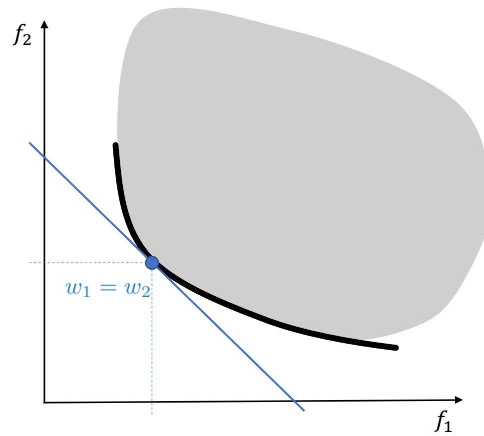
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming $f_2 = \text{cte}$
- Minimize f_2 assuming $f_1 = \text{cte}$



$$F(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x})$$

Multi to single-objective optimization

1. Weighted sum of each individual objective

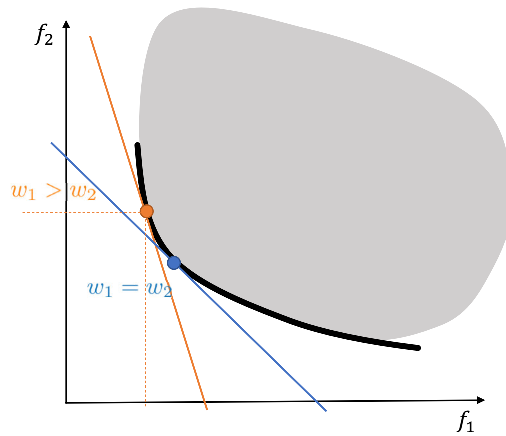
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming f_2 = cte
- Minimize f_2 assuming f_1 = cte



$$F(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x})$$

Multi to single-objective optimization

1. Weighted sum of each individual objective

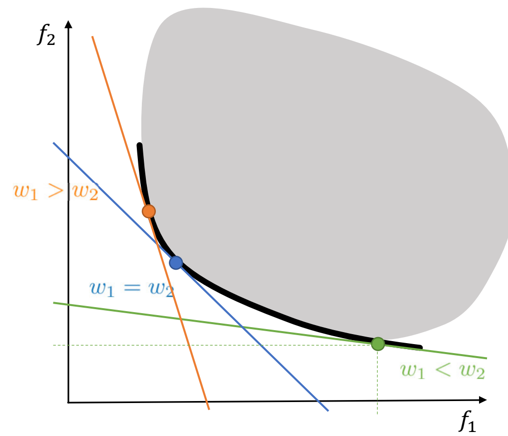
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming $f_2 = \text{cte}$
- Minimize f_2 assuming $f_1 = \text{cte}$



$$F(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x})$$

Multi to single-objective optimization

1. Weighted sum of each individual objective

$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming $f_2 = \text{cte}$
- Minimize f_2 assuming $f_1 = \text{cte}$

Multi to single-objective optimization

1. Weighted sum of each individual objective

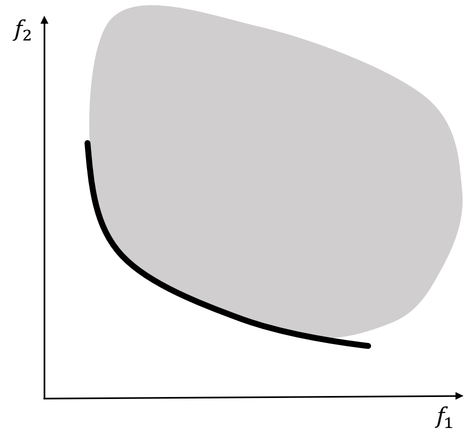
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming $f_2 = \text{cte}$
- Minimize f_2 assuming $f_1 = \text{cte}$



Multi to single-objective optimization

1. Weighted sum of each individual objective

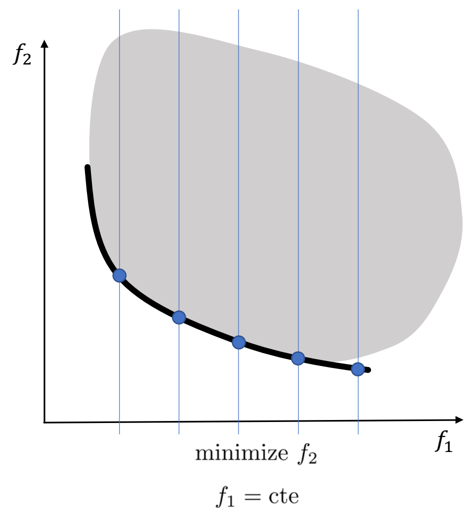
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming f_2 = cte
- Minimize f_2 assuming f_1 = cte



Multi to single-objective optimization

1. Weighted sum of each individual objective

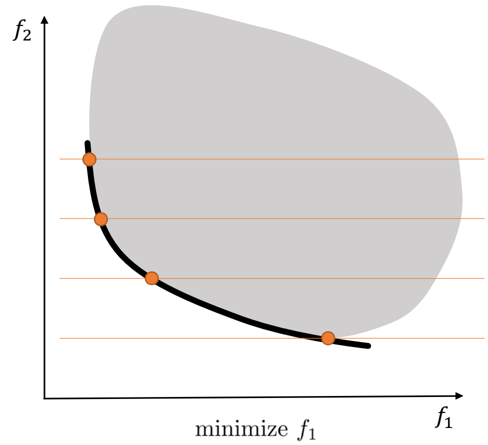
$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming f_2 = cte
- Minimize f_2 assuming f_1 = cte



minimize f_1

$f_2 = \text{cte}$

Multi to single-objective optimization

1. Weighted sum of each individual objective

$$F(\mathbf{x}) = \sum_{k=1}^{l} w_k f_k(\mathbf{x})$$

2. Min-max formulation

$$F(\mathbf{x}) = \max(\omega_1 f_1(\mathbf{x}), \omega_2 f_2(\mathbf{x}), ..., \omega_l f_l(\mathbf{x}))$$

3. Constrained minimum

- Minimize f_1 assuming f_2 = cte
- Minimize f_2 assuming f_1 = cte

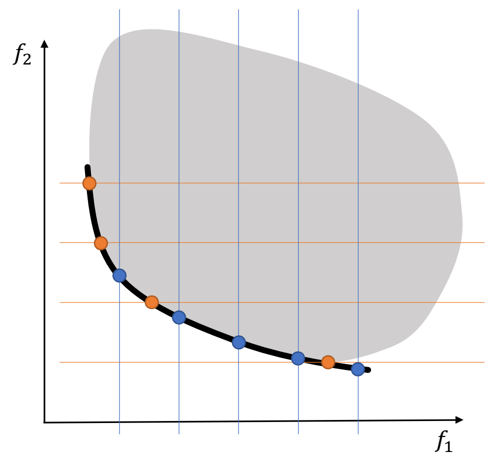
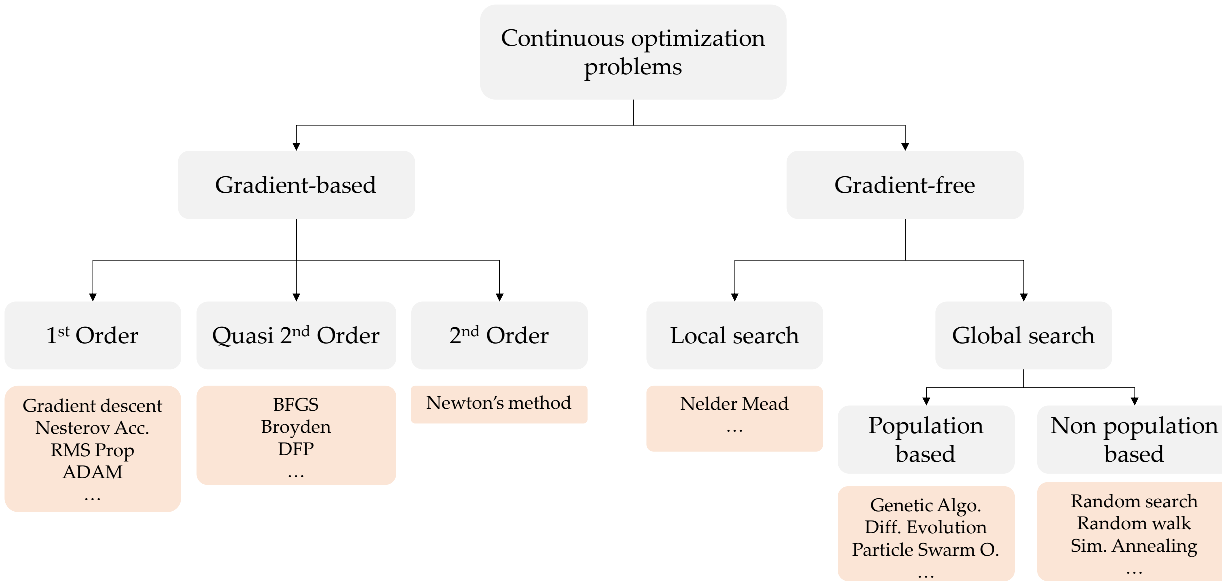


Table of Contents

Optimization methods



Optimization methods

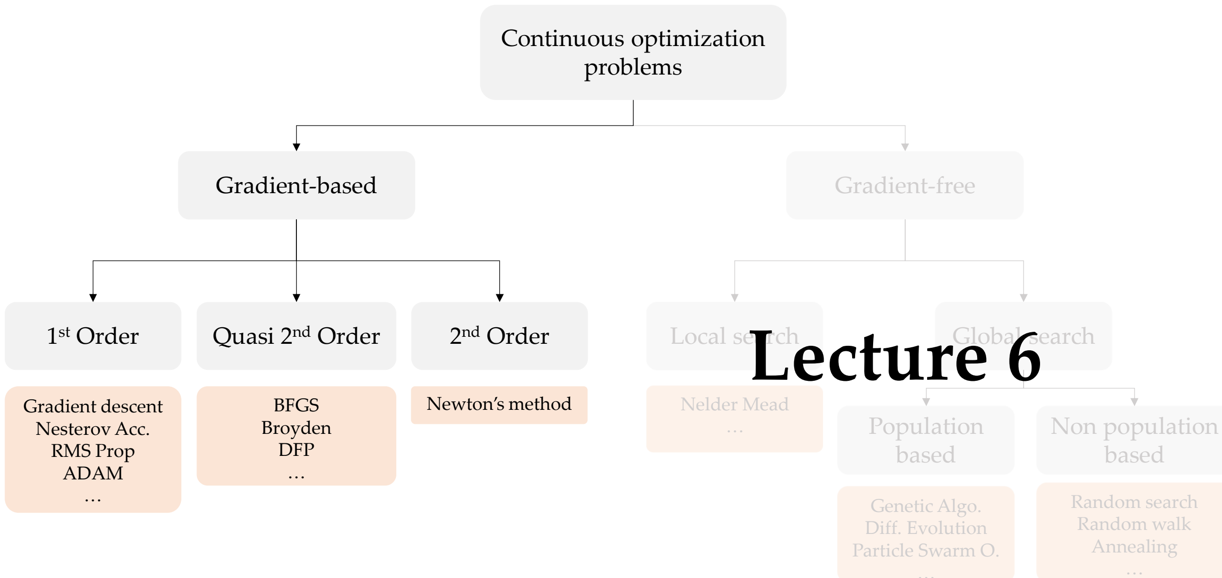


Table of Contents

Gradient-based methods

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} d^{(k)}$$

First order steepest descent

- Search direction is the steepest descent
- Cheap computational evaluation, only requires the gradient

Conjugate gradient methods

- Search direction is **not** the steepest descent
- Perform line search for optimal learning rate

High order gradient descent

- Compute or estimate the Hessian matrix
- Converge in fewer iterations at cost of computational load

Table of Contents

von KARMAN INSTITUTE FOR FLUID DYNAMICS

Gradient descent methods

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta(k)\nabla_w J(\mathbf{w}; \mathbf{X}, \mathbf{y})$$

ANN Optimizers	'Classic' gradient descent methods
Modify the learning rate: $\eta^{(k)}$	• Batch gradient descent • Mini-batch gradient descent • …

Table of Contents

Nonlinear Conjugate Gradient method

1. Define search direction

- In linear problems, we search in orthogonal directions.
- It can be shown that these models converge in **n** iterations, where **n** is the dimension of the search space!

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} d^{(k)}$$

$$d^{(k+1)} = \underbrace{-\nabla_w J\left(\mathbf{w}^{(k+1)}\right)}_{\text{Steepest descent direction}} + \underbrace{\beta^{(k+1)} d^{(k)}}_{\text{Previous search direction}}$$

Fletcher-Reeves:

$$\beta^{(k+1)} = \frac{\nabla_w J\left(\mathbf{w}^{(k+1)}\right)^T \nabla_w J\left(\mathbf{w}^{(k+1)}\right)}{\nabla_w J\left(\mathbf{w}^{(k)}\right)^T \nabla_w J\left(\mathbf{w}^{(k)}\right)}$$

Polak-Ribière:

$$\beta_{k+1} = \frac{\nabla_w J\left(\mathbf{w}^{(k+1)}\right)^T \left(\nabla_w J\left(\mathbf{w}^{(k+1)}\right) - \nabla_w J\left(\mathbf{w}^{(k)}\right)\right)}{\left\|\nabla_w J\left(\mathbf{w}^{(k)}\right)\right\|^2}$$

Nonlinear Conjugate Gradient method

2. Find optimal step length

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} d^{(k)}$$

- Minimize along search direction $\min_{\eta} J\left(\mathbf{w}^{(k)} + \eta d^{(k)}\right)$
- Iterative algorithm for 1D minimizer (e.g. cubic interpolation)
- Satisfy "ideal" step length conditions (e.g. Wolfe, Goldstein)

Sufficient decrease:

$$J\left(\mathbf{w}^{(k)} + \eta d^{(k)}\right) \leq J\left(\mathbf{w}^{(k)}\right) + c_1 \eta \nabla_w J\left(\mathbf{w}^{(k)}\right)^T d^{(k)}$$

Curvature condition:

$$\left|\nabla_w J\left(\mathbf{w}^{(k)} + \eta d^{(k)}\right)^T d_k\right| \leq c_2 \left|\nabla_w J\left(\mathbf{w}^{(k)}\right)^T d_k\right|$$

The exact minimizer $\mathbf{w}^*$ can be found in one iteration if f is a quadratic function

Rewrite the objective function as:

$$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T Q \mathbf{w} - b^T \mathbf{w}$$

Using the steepest descent algorithm:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} \nabla_w J\left(\mathbf{w}^{(k)}\right)$$

The exact minimizer is found:

$$\nabla_\eta J\left(\mathbf{w}^{(k)} - \eta^* \nabla_{\mathbf{w}} J\left(\mathbf{w}^{(k)}\right)\right) = 0$$

$$\eta^* = \frac{\nabla_w J\left(\mathbf{w}^{(k)}\right)^T \nabla_w J\left(\mathbf{w}^{(k)}\right)}{\nabla_w J\left(\mathbf{w}^{(k)}\right)^T Q \nabla_w J\left(\mathbf{w}^{(k)}\right)}$$

Table of Contents

Quasi-Newton methods

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta^{(k)} \underbrace{\left(B^{(k)}\right)}_{\approx \text{ Hessian}}^{-1} \nabla_w J(\mathbf{w})$$

Quadratic approximation of objective function:

$$m^{(k)}(d) = J\left(\mathbf{w}^{(k)}\right) + \nabla_w J\left(\mathbf{w}^{(k)}\right)^T d + \frac{1}{2} d^T B^{(k)} d$$

Search direction

$$d = -B^{-1} \nabla_w J(\mathbf{w})$$

- Gradient of the quadratic model must match $\nabla_w J(\boldsymbol{w})$ for $w^{(k)}$ and $w^{(k+1)}$:

$$B^{(k+1)} \left(\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\right) = \nabla J_w\left(\mathbf{w}^{(k+1)}\right) - \nabla J_w\left(\mathbf{w}^{(k)}\right)$$

- Can be solved if:

$$\left(\mathbf{w}^{(k+1)} - \mathbf{w}^{(k)}\right)^T \nabla J_w\left(\mathbf{w}^{(k+1)}\right) - \nabla J_w\left(\mathbf{w}^{(k)}\right) > 0 \longrightarrow$$

Line search for $\eta^{(k)}$ with Strong Wolfe conditions

Quasi-Newton methods

1. Compute search direction: $d^{(k)} = -H^{(k)} \nabla_w J(\mathbf{w})$
2. Line search with Strong Wolfe conditions: $\eta^{(k)}$
3. Compute next solution: $\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} d^{(k)}$
4. Compute $H^{(k+1)}$:

$$s^{(k)} = x^{(k+1)} - x^{(k)}$$

$$y^{(k)} = \nabla_w J\left(\mathbf{w}^{(k+1)}\right) - \nabla_w J\left(\mathbf{w}^{(k)}\right)$$

$$\rho^{(k)} = \left(y^{(k)} s^{(k)}\right)^{-1}$$

$$\textbf{(DFP)}\ H^{(k+1)} = H^{(k)} - \frac{H^{(k)} y^{(k)} \left(y^{(k)}\right)^T H^{(k)}}{\left(y^{(k)}\right)^T H^{(k)} y^{(k)}} + \frac{s^{(k)} \left(s^{(k)}\right)^T}{\left(y^{(k)}\right)^T s^{(k)}}$$

$$\textbf{(BFGS)}\ H^{(k+1)} = \left(I - \rho^{(k)} s^{(k)} \left(y^{(k)}\right)^T\right) H^{(k)} \left(I - \rho^{(k)} y^{(k)} \left(s^{(k)}\right)^T\right) + \rho^{(k)} s^{(k)} \left(s^{(k)}\right)^T$$

Initial estimate for the Hessian? Finite difference estimate | Identity matrix | …

Table of Contents

Gradient-based optimization: Minimize Rosenbrock function

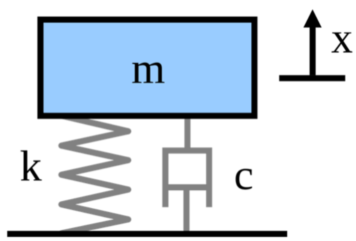
- Content here…

Table of Contents

Exercise: Mass-spring-damper system identification

Steepest descent
BFGS (compare with scipy)
Newton
Nelder Mead

Compare:
- Total elapsed time
- Cost-function vs iterations
- Gradient = efficiency



2nd order system
Step response

$$m\ddot{x} + b\dot{x} + kx = 0$$

Title

Content here…

Gradient-based methods

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} d^{(k)}$$

Machine learning
Heuristic learning rate.
Keep the learning rate,
Add momentum, rescaling,
(connection with Jan in lecture 11)
See **DDFM Lecture 10 Slide 29**

ANN Optimizers

Modify the learning rate:

$$\eta^{(k)}$$

Conjugate gradient methods

- Search direction is **not** the steepest descent
- Perform line search for optimal learning rate

High order gradient descent

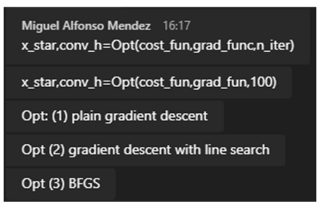
- Compute the Hessian (or approximate)
- Converge in fewer iterations…

State that the gradient is available through a function (for these cases assume that it's possible)

To do

Func(cost_function,grad_cost_function,n_iter)

Exercise:
- Gradient descent
- Quasi-Newton (BFGS)
- Newton
- Compare convergence rate (plot residual vs iterations)
- Compare time called by our routine vs scipy
- What if we don't give the function for the gradient
    - Takes even longer, because it has to evaluate it



Plot convergence rate
Compare evaluation time
Our evaluation vs scipy

Give a template with some missing parts, which can then be filled together with the people

Nonlinear Conjugate Gradient method

1. Define search direction

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} \vec{d}^{(k)}$$

2. Find optimal step length

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta^{(k)} \vec{d}^{(k)}$$

- Minimize in the search direction $\min_{\mathbf{w}} J\left(\mathbf{w}^{(k)} + \eta^{(k)} \vec{d}^{(k)}\right)$
- Iterative algorithm for 1D minimizer (e.g. cubic interpolation)
- Satisfy "ideal" step length conditions (e.g. Wolfe, Goldstein)

Sufficient decrease:

$$f(x_k + \alpha \vec{p}_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k$$

Curvature condition:

$$|\nabla f(x_k + \alpha \vec{p}_k)^T p_k| \leq c_2 |\nabla f_k^T p_k|$$

scipy.optimize.line_search

```
scipy.optimize.line_search(f, myfprime, xk, pk, gfk=None, old_fval=None, old_old_fval=None, args=(), c1=0.0001, c2=0.9, amax=None, extra_condition=None, maxiter=10)
```

[source]

Find alpha that satisfies strong Wolfe conditions.

The exact minimizer $\mathbf{w}^*$ can be found in one iteration if f is a quadratic function

Rewrite the objective function as:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T Q \mathbf{x} - b^T \mathbf{x}$$

Using the steepest descent algorithm:

$$x_{k+1} = x_k - \alpha_k \nabla f_k$$

The exact minimizer is found:

$$\nabla_\alpha f(x_k - \alpha^* \nabla f_k) = 0$$

$$\alpha^* = \frac{\nabla f_k^T \nabla f_k}{\nabla f_k^T Q \nabla f_k}$$

Linear conjugate gradient method

Minimize quadratic function: $\phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x}$

Residual defined as the gradient: $\nabla\phi(\mathbf{x}) = A\mathbf{x} - b = r(\mathbf{x})$

Search directions p_k: conjugate vector wrt A $\quad p_i^T A p_j = 0 \quad \forall \quad i \neq j$

Initial conditions: $\quad r_0 = A\mathbf{x_0} - b \qquad p_0 = r_0$

Minimization algorithm:

Exact minimizer for quadratic functions

- Ideal step length through line search $\quad \alpha_k = -\dfrac{r_k^T p_k}{p_k^T A p_k}$
- Move to the minimizer in direction p_k $\quad x_{k+1} = x_k + \alpha_k \nabla f_k$
- Calculate new residual r_{k+1} $\quad r_{k+1} = Ax_{k+1} - b$
- New direction p_{k+1} conjugate wrt A $\quad p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$

Linear combination of steepest descent $-r_{k+1}$ and previous search direction

$$\beta_{k+1} = \frac{r_{k+1}^T A p_k}{p_k^T A p_k}$$

Constant enforces that $p_{k-1}^T A p_k = 0$, so that they are conjugate directions

How to choose the conjugate direction set $\{p_0, p_1, \ldots, p_{n-1}\}$ for matrix A $[n \times n]$?

Eigenvectors $v_1, v_2, \ldots, v_n$ of A

- Mutually orthogonal (linearly independent)
  $\langle v_i, v_j \rangle = v_j^T v_i = 0 \quad \forall \quad i \neq j$
- Conjugate with respect to A
  $Av_i = \lambda v_i \Leftrightarrow v_j^T A v_i = \lambda v_j^T v_i \Rightarrow v_j^T A v_i = 0$
- Too expensive for large-scale applications

More efficient approaches are used (see left)

Linear conjugate gradient method

If A is a diagonal matrix $[n \times n]$, we converge to the minimizer of $\phi(\boldsymbol{x})$ in n iterations.

We need a diagonal matrix to guarantee convergence!

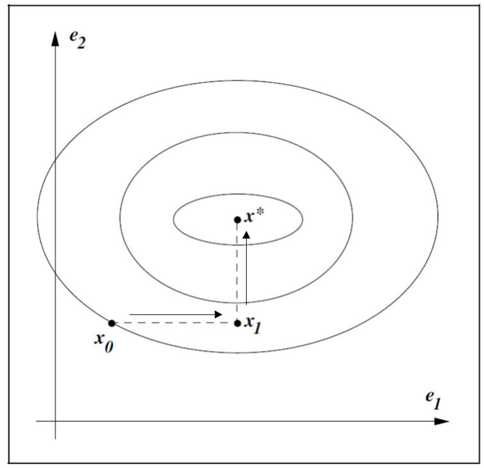
$$\hat{x} = S^{-1}x \quad \text{where} \quad S = [p_0 \;\; p_1 \;\; \dots \;\; p_{n-1}]$$

$$\hat{\phi}(\hat{\mathbf{x}}) = \frac{1}{2}\hat{\mathbf{x}}^T(S^T A S)\hat{\mathbf{x}} - (S^T b)^T \hat{\mathbf{x}}$$

Since $S^T AS$ is diagonal, we minimize $\hat{\phi}$ with n iterations in the $\hat{\boldsymbol{x}}$ space, and then compute the true minimum:

$$x = S\hat{x}$$

We can create a symmetric matrix during preconditioning.



Successive minimization for a general convex quadratic function using the linear conjugate gradient method

Optimization methods

Gradient-based (local)
- Line search methods
  - Deepest descent
  - Newton's method
  - Quasi-newton methods
- Trust region methods
  - Quasi-newton methods
  - Newton's method

Mention this as reference:
https://docs.scipy.org/doc/scipy/tutorial/optimize.html

Gradient-free (global)
- Random search (simplest approach of all)
- Random walk (kind of a line search but random)
- Nelder-Mead (polytopes)
- Evolutionary algorithms (discussed in Lecture 6/7)
- Particle Swarm Optimization (discussed in lecture 6)

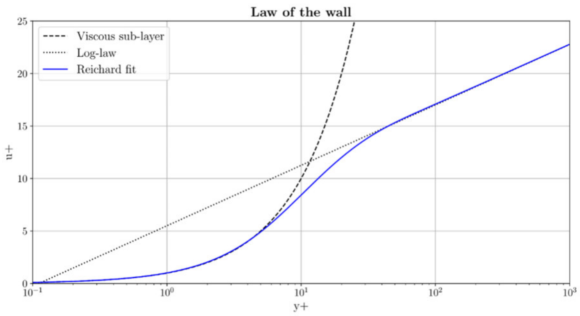
Hybrid
- Bayesian optimization (following lecture)

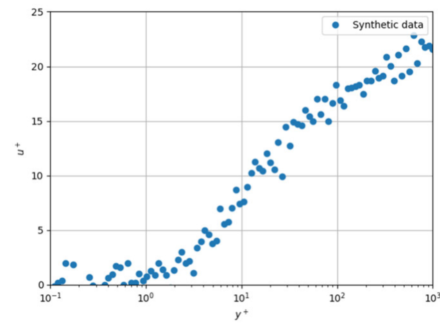
This lecture: mainly gradient based. Apply them to some problems.

Exercise: Fitting a turbulent boundary layer

Non-Linear Least Squares (only scipy)
Steepest descent
BFGS (compare with scipy)
Newton
Nelder Mead

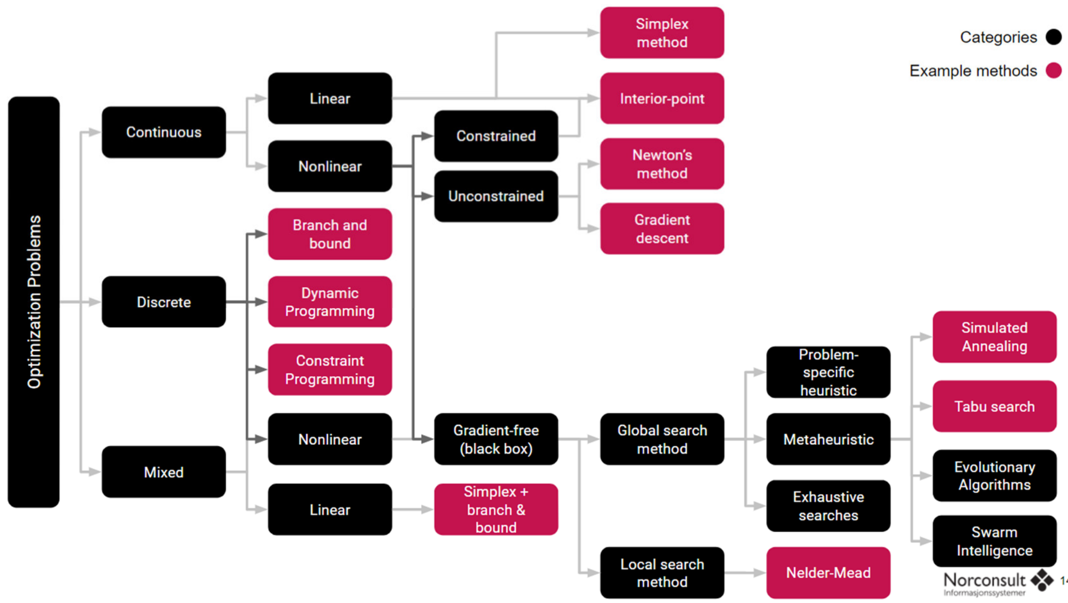
Compare:
- Total elapsed time
- Cost-function vs iterations
- Gradient = efficiency





$$u^+ = a_1 \log(1 + y^+ \kappa) + a_2 \left(1 - e^{-\frac{y^+}{11}} - \frac{y^+}{11} e^{-\frac{y^+}{3}}\right)$$

Optimization methods



Unused figures