Optimization: introduction and common methods

Presenter: R. Lehe

Day 1

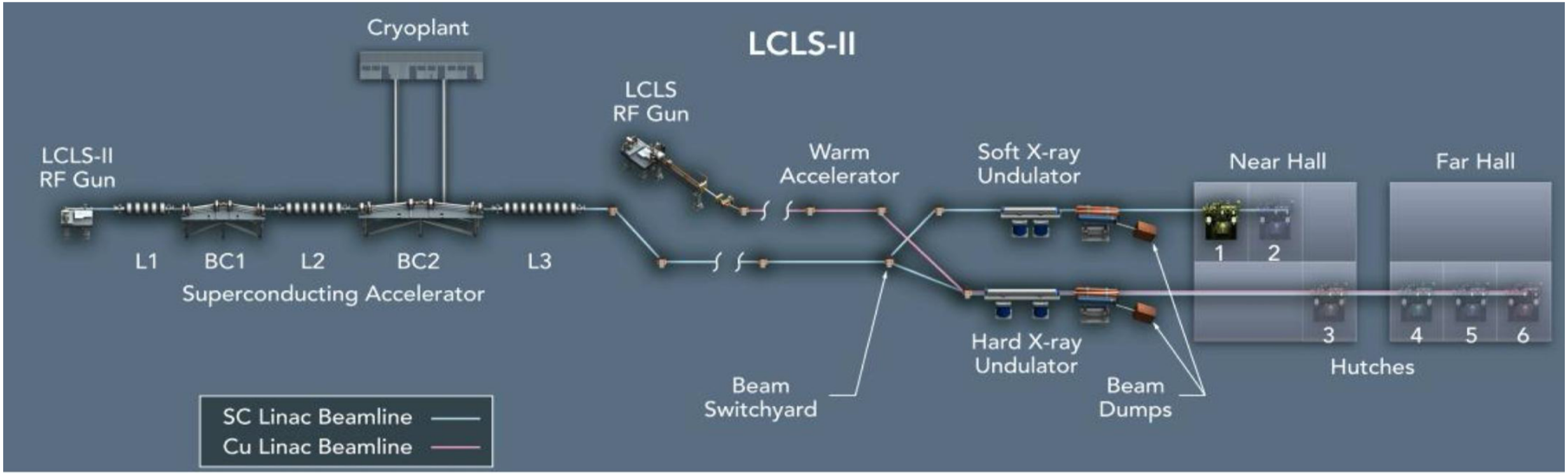
Outline

- Example and motivation for particle accelerators
- Optimization: general definition and naïve algorithms
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - Gradient-descent
  - Extremum Seeking
- Some general terms

Outline

- **Example and motivation for particle accelerators**
- Optimization: general definition and naïve algorithms
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - Gradient-descent
  - Extremum Seeking
- Some general terms

Example: Free-Electron lasers (e.g. European XFEL, LCLS II)



Example of objective:

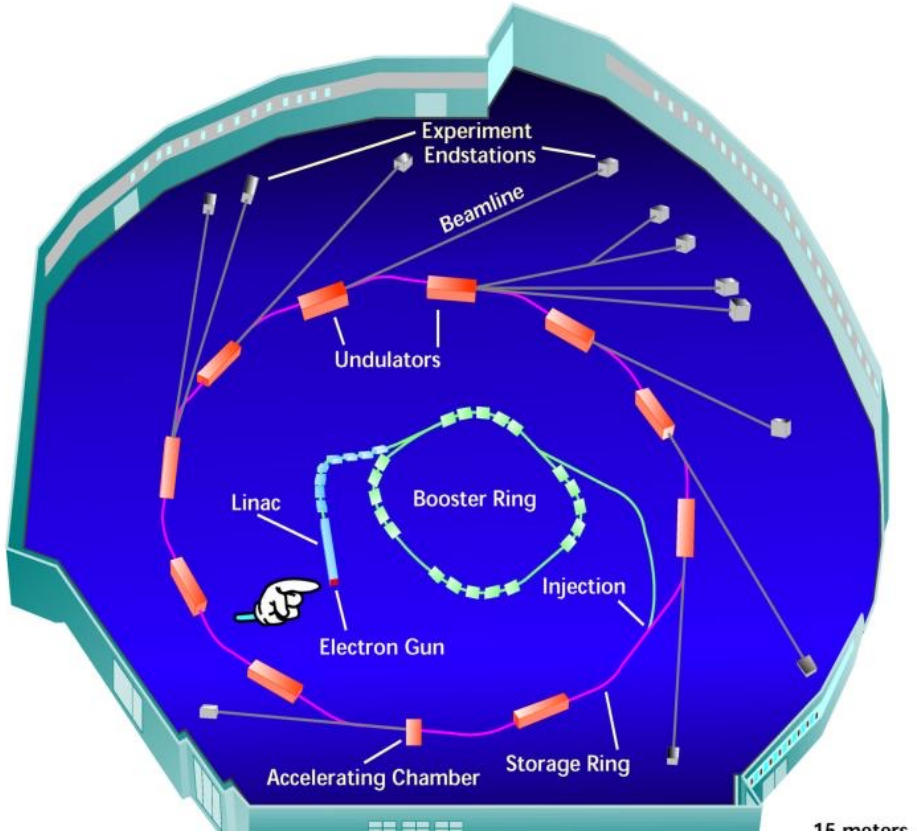
Maximize amount of X-ray photons, during operation

Example of tuning parameters:

- Strength of steerer magnets
- Strength of FODO quadrupoles
- RF parameters (phase and accelerating gradient)

Source: https://lcls.slac.stanford.edu/lcls-ii

Example: storage ring (e.g. ALS, SPEAR3)

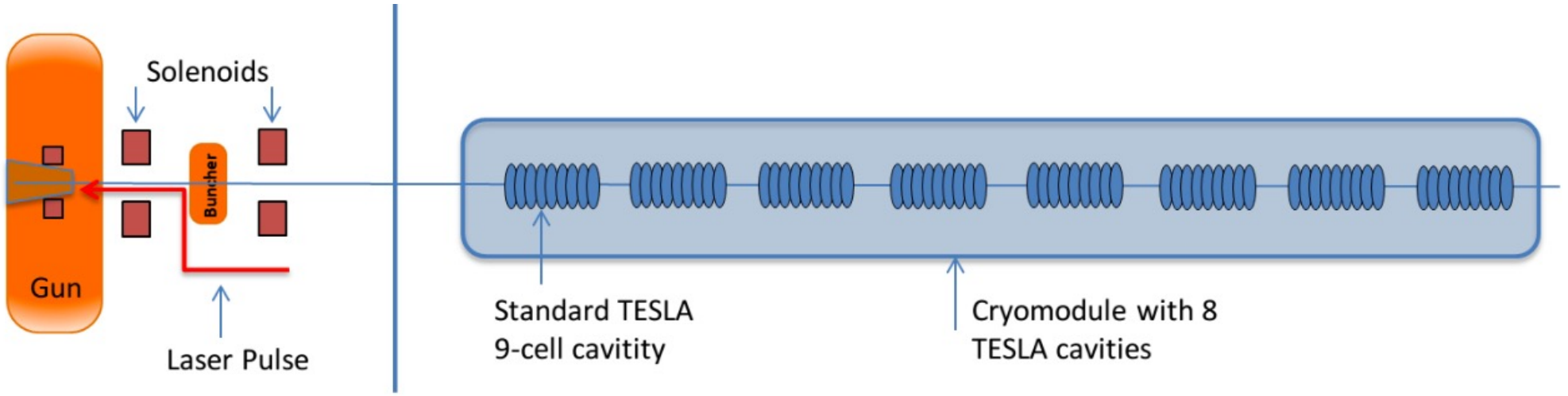


Example of objective:
Maximize injection efficiency

Example of tuning parameters:
Strength of sextupole magnets

Source: https://journals.aps.org/prab/pdf/10.1103/PhysRevSTAB.18.084001

Example: electron injector for LCLS-II



Example of objective:

Minimize bunch length and emittance, at the end of the injector

Example of tuning parameters:

- Duration and transverse size of laser pulse
- Magnetic field in solenoids
- Buncher field
- Accelerating gradient in RF cavities

Source: https://accelconf.web.cern.ch/IPAC2014/papers/wepro015.pdf

Optimization for particle accelerators: motivation

Design study, before building hardware:

- Aim: choose **best nominal parameters**, predict optimal performance
- Mainly based on **numerical simulations**
- Some unique features: evaluation in parallel

Online tuning of existing hardware:

- Aim: get **optimal performance during operation** ; maintain despite drifts
- Mainly based on **real-time measurements**
- Some unique features: noise, hysteresis (e.g. magnetic elements), drifts (e.g. temperature)

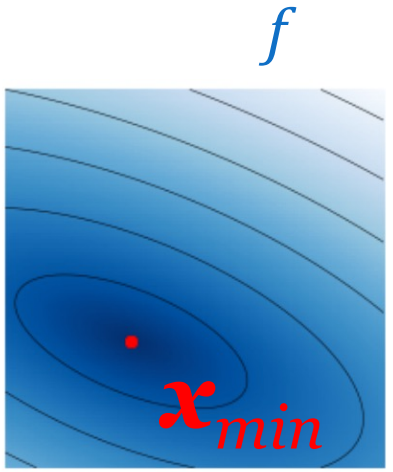
Outline

- Example and motivation for particle accelerators
- **Optimization: general definition and naïve algorithms**
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - Gradient-descent
  - Extremum Seeking
- Some general terms

Optimization: general definition and notation

Definition (minimization)

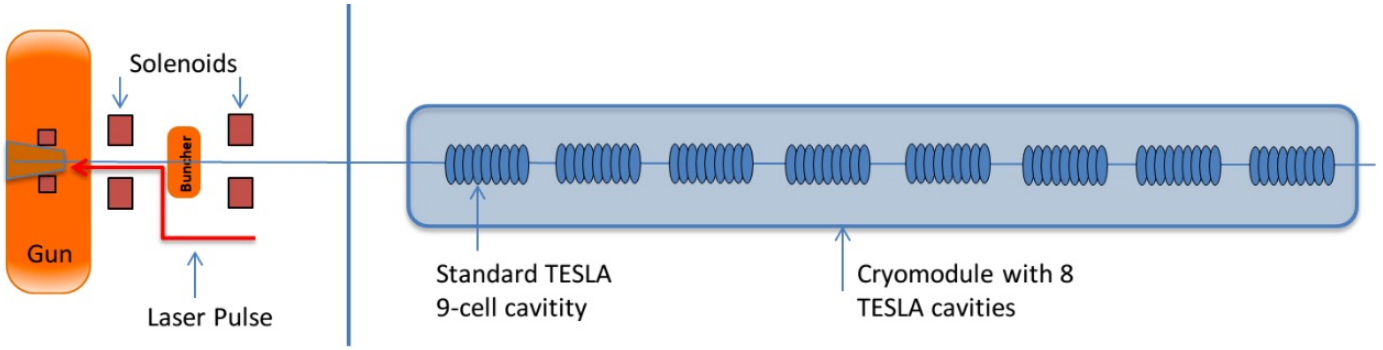
Find $\boldsymbol{x}_{min}$, such that $\quad \forall \boldsymbol{x} \in \Omega,\ f(\boldsymbol{x}_{min}) \leq f(\boldsymbol{x})$



$\boldsymbol{x}$: vector of input parameters ("knobs", "tuning parameters")
f: function to minimize ("objective function")
Ω: domain (limited by constraints on accelerator parameters)

Example: injector

Minimizing emittance by tuning solenoids and accelerating cavities



$$f = \epsilon_{\perp}$$

$$\boldsymbol{x} = \begin{pmatrix} B_{solenoid} \\ E_{cavity} \end{pmatrix}$$

Efficient optimization

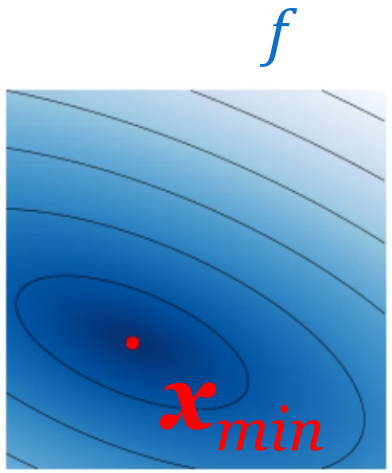
Aim:

Find $\boldsymbol{x}_{min}$ with **few** evaluations of f

Motivation: evaluations of f are usually costly

- **Design studies**:
  Evaluations of f require **computationally expensive** numerical simulations

- **Online tuning:**
  Evaluations of f **take time** on the machine
  Parameters of the machine may **drift** if it takes too long to find the minimum.



Minimization vs maximization

Minimization:

Find $\boldsymbol{x}_{min}$, such that $\quad \forall \boldsymbol{x} \in \Omega, \ f(\boldsymbol{x}_{min}) \leq f(\boldsymbol{x})$

Maximization:

Find $\boldsymbol{x}_{min}$, such that $\quad \forall \boldsymbol{x} \in \Omega, \ f(\boldsymbol{x}_{min}) \geq f(\boldsymbol{x})$

In order to **maximize** a function *f*, one can simply pass the function –*f* to a **minimization** algorithm.

In the rest of this course, we will focus on **minimization algorithms**.

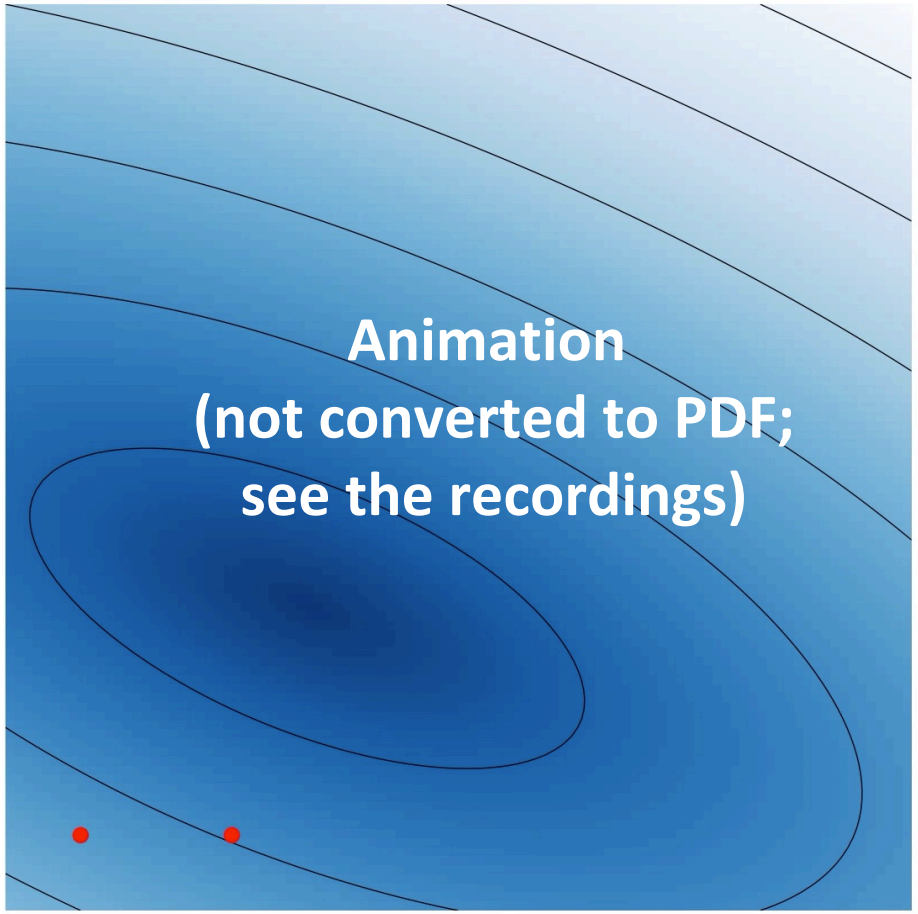
Naive algorithm: grid search

Algorithm:

Systematically evaluate *f* at points separated by **a fixed step** in each direction. At the end: find the best point among them.

Practical consideration:

- Takes a long time to even reach interesting regions.
- Scales badly with dimensionality!
- Does not use the information from **previous evaluations of *f*** to decide which point to evaluate next.



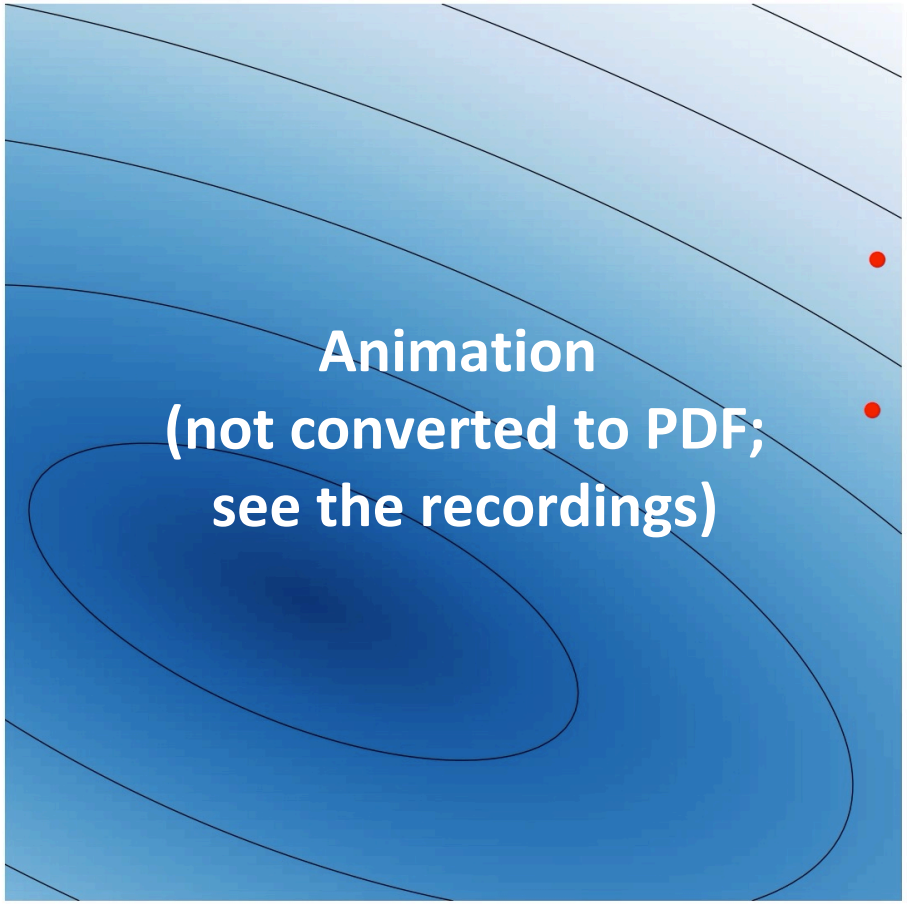
Naive algorithm: random search

Algorithm:

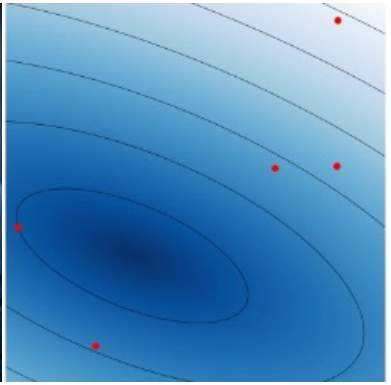
Evaluate *f* at **randomly chosen points.**
At the end: find the best point among them.

Practical consideration:

- May evaluate points that **are close to each other** and do not bring significantly more information
- Scales badly with dimensionality!
- Does not use the information from **previous evaluations of *f*** to decide which point to evaluate next.



Human intervention



Algorithm:
A human being chooses the points to evaluate

Practically consideration

- Humans sometimes accumulate unique experience/knowledge of a given accelerator
- But: slow reaction time
- Biases, bad at dealing with more than 1 or 2 dimensions (usually perform 1D search)

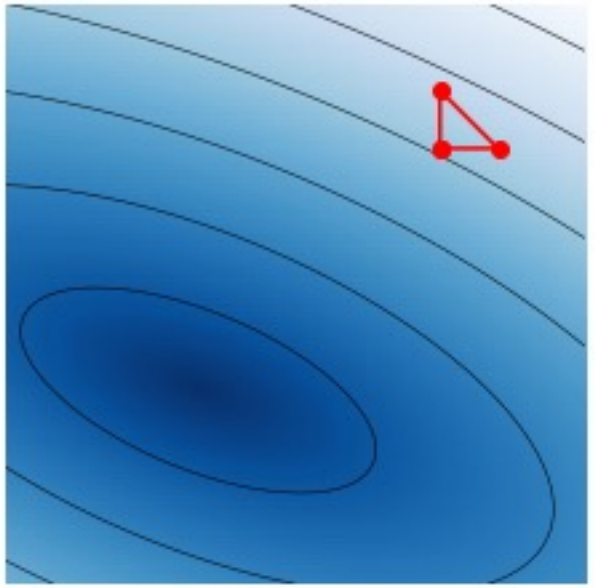
Outline

- Example and motivation for particle accelerators
- Optimization: general definition and naïve algorithms
- **Some common optimization algorithms**
  - **Nelder-Mead algorithm**
  - Gradient-descent
  - Extremum Seeking
- Some general terms

Nelder-Mead simplex: algorithm

- Choose N+1 **arbitrary initial points** (where N is the dimension of the input ***x*** of the objective function *f*)
  Evaluate *f* at these points.

2D example: 3 initial points



Note:

These points define a **"simplex".**

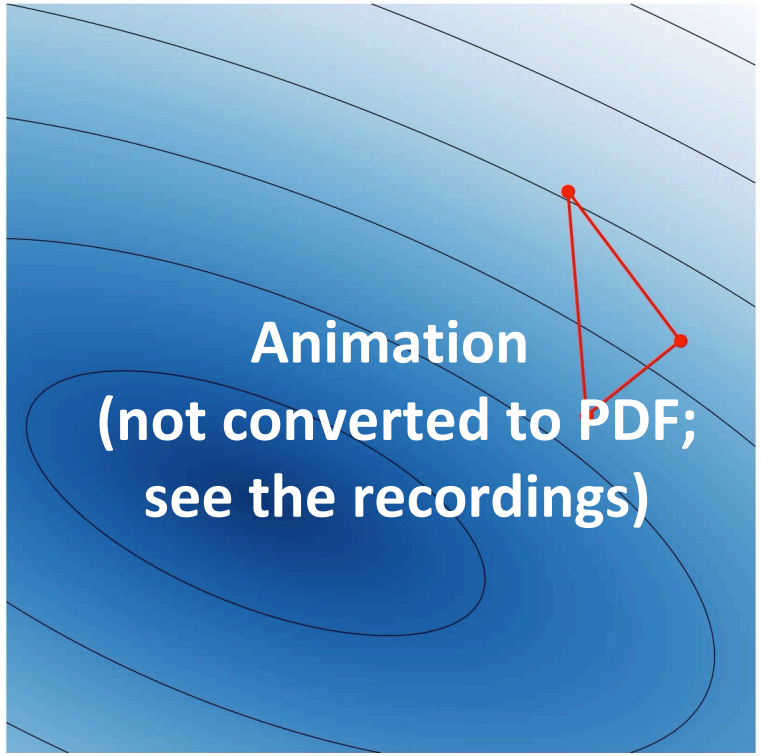
(The points are the **"vertices**" of the simplex.)

- In 2D (N=2), a simplex is a triangle.
- In 3D (N=3), a simplex is a terahedron.

Nelder-Mead simplex: algorithm

- Choose N+1 **arbitrary initial points** (where N is the dimension of the input ***x*** of the objective function *f*)
  Evaluate *f* at these points.

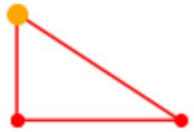
- Iteratively:
  - **Move** vertices according to a set of **basic rules** (see next slide)
  - **Evaluate** objective function *f* at the new vertices



- These rules effectively result in the simplex moving **towards the minimum**. The N+1 vertices allow to “feel” the direction in which to move (without calculating the gradient).

Nelder-Mead simplex: basic rules

Pick the **worse** point (i.e. the one with the highest value of f)

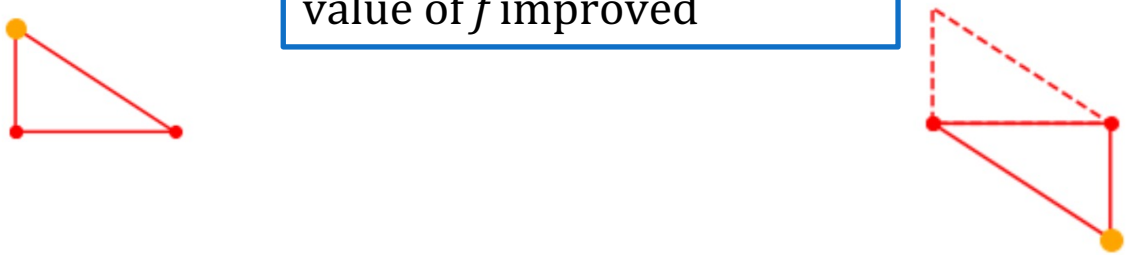


Exact algorithm with code: *Press et al., "Numerical Recipes"*

Nelder-Mead simplex: basic rules

Pick the **worst** point (i.e. the one with the highest value of f)

Try to perform a **reflection** through the barycenter of the other points ;
keep the new point if the value of f improved

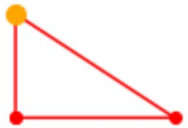


Heuristic: try to move **away** from the high values of f

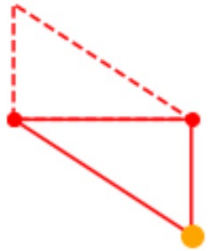
Exact algorithm with code: *Press et al., "Numerical Recipes"*

Nelder-Mead simplex: basic rules

Pick the **worst** point (i.e. the one with the highest value of *f*)

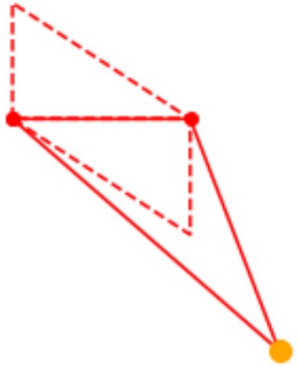


Try to perform a **reflection** through the barycenter of the other points ;
keep the new point if the value of *f* improved



Did it become the best point of the simplex?

Try to perform a **dilation** in the same direction ;
keep the new point if the value of *f* improved.

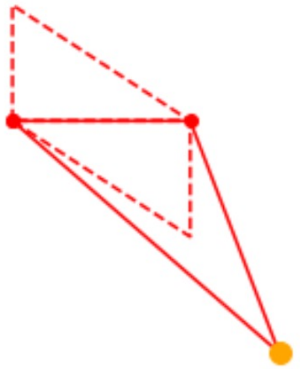


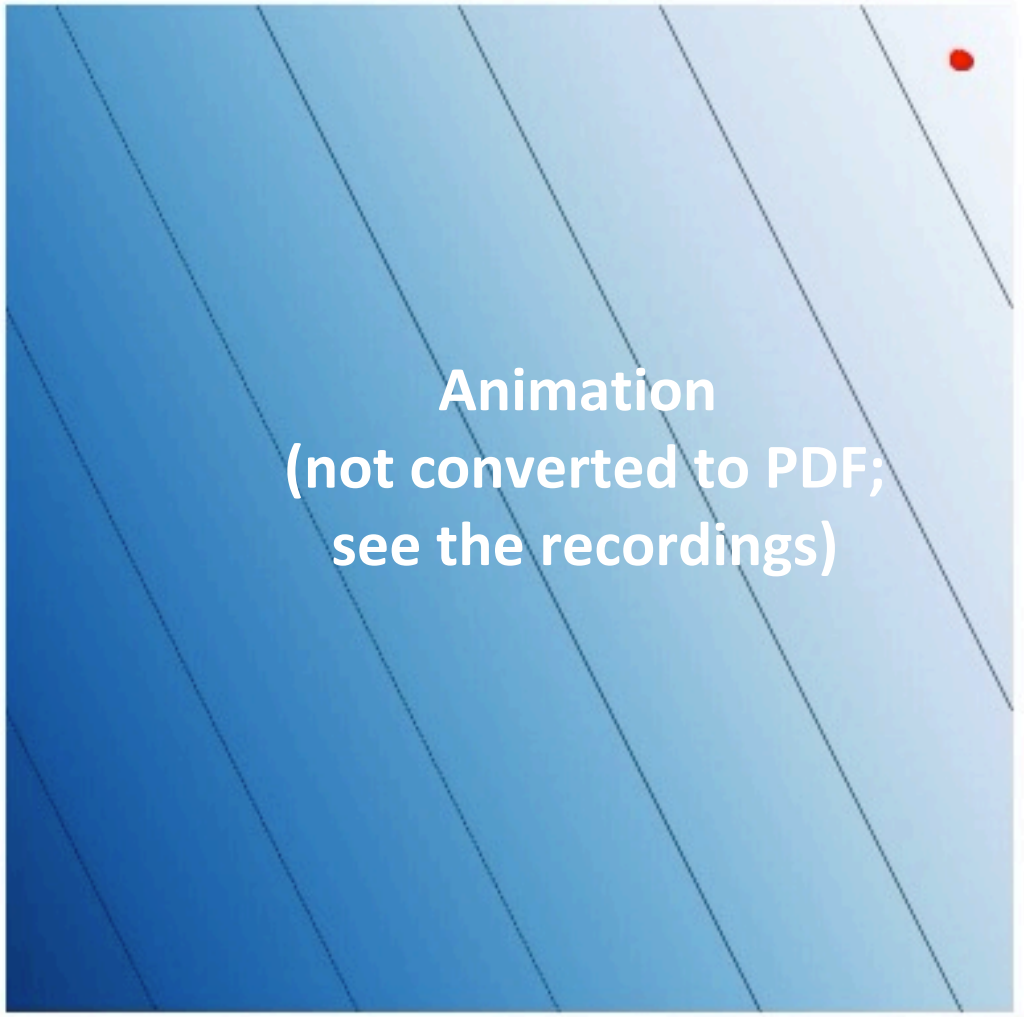
Heuristic: accelerate in the the direction in which *f* decreases

Exact algorithm with code: *Press et al., "Numerical Recipes"*

Nelder-Mead simplex: basic rules

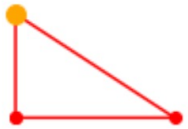
Simplex accelerating
in the direction of decreasing *f*:



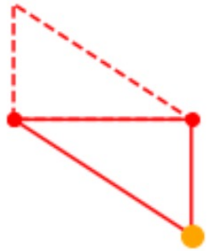


Nelder-Mead simplex: basic rules

Pick the **worst** point (i.e. the one with the highest value of *f*)

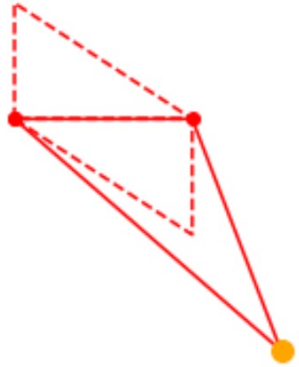


Try to perform a **reflection** through the barycenter of the other points ;
keep the new point if the value of *f* improved



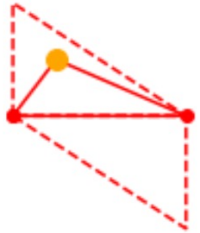
Did it become the best point of the simplex?

Try to perform a **dilation** in the same direction ;
keep the new point if the value of *f* improved.



Is it still the worse point of the simplex?

Try to perform a **contraction** towards the barycenter of the other points of the original simplex.

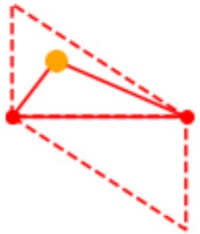


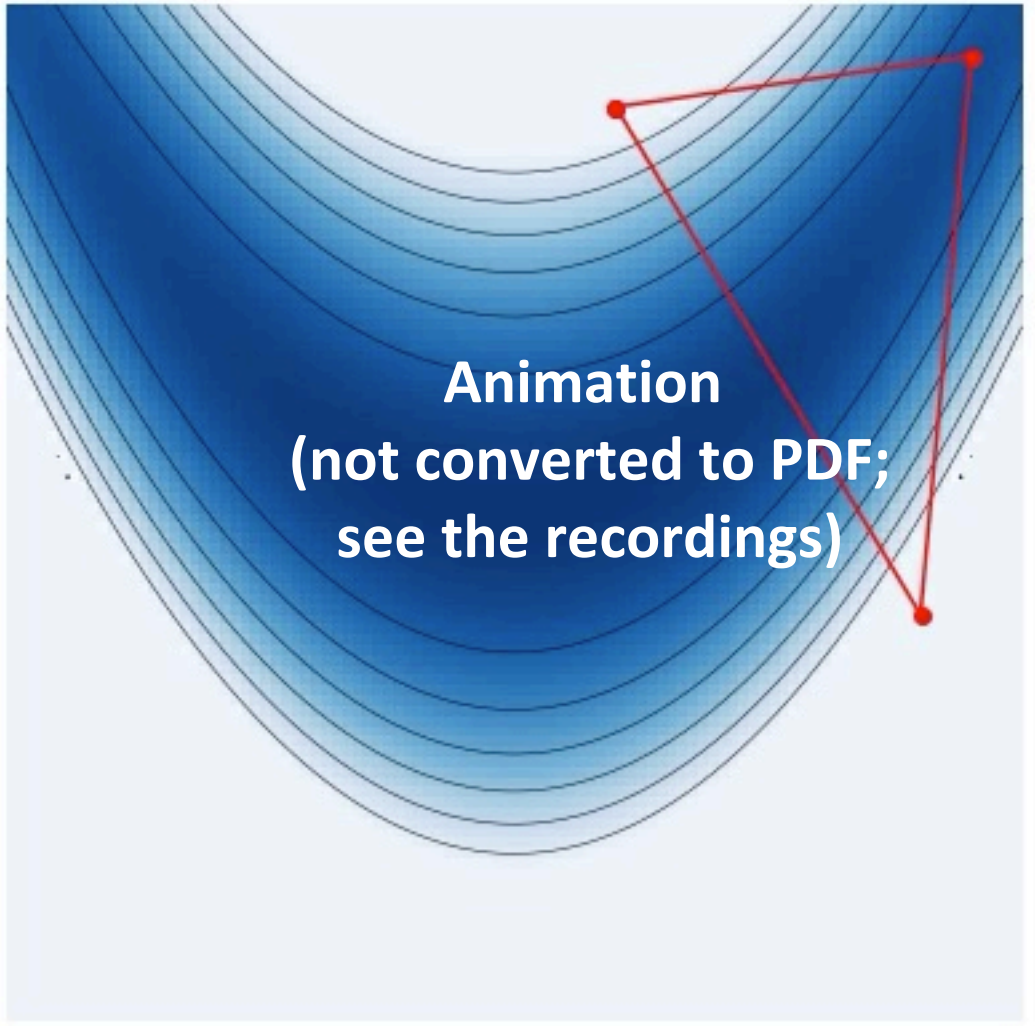
Heuristic: squeeze into narrow valleys of *f*

Exact algorithm with code: *Press et al., "Numerical Recipes"*

Nelder-Mead simplex: basic rules

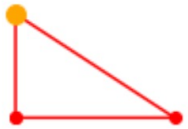
Squeeze into narrow valleys of *f*



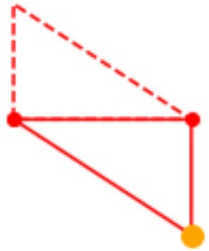


Nelder-Mead simplex: basic rules

Pick the **worst** point (i.e. the one with the highest value of *f*)

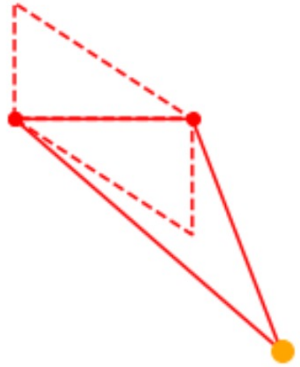


Try to perform a **reflection** through the barycenter of the other points ;
keep the new point if the value of *f* improved



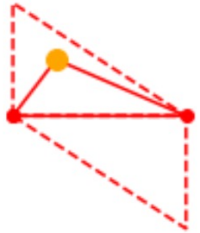
Did it become the best point of the simplex?

Try to perform a **dilation** in the same direction ;
keep the new point if the value of *f* improved.



Is it still the worse point of the simplex?

Try to perform a **contraction** towards the barycenter of the other points of the original simplex.

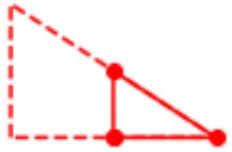


Improvement?

Keep the new point

No improvement?

Try to perform a **contraction** towards the best point of the simplex

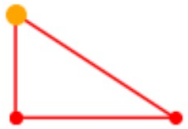


Heuristic: shrink into a trough of *f*

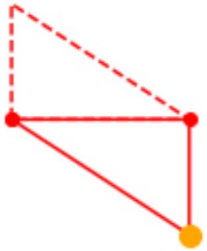
Exact algorithm with code: *Press et al., "Numerical Recipes"*

Nelder-Mead simplex: basic rules

Pick the **worst** point (i.e. the one with the highest value of *f*)

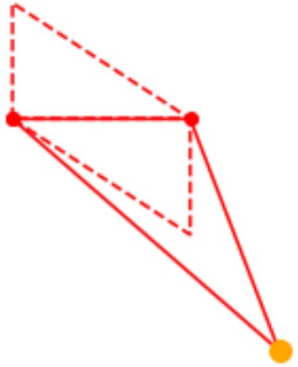


Try to perform a **reflection** through the barycenter of the other points ;
keep the new point if the value of *f* improved



Did it become the best point of the simplex?

Try to perform a **dilation** in the same direction ;
keep the new point if the value of *f* improved.



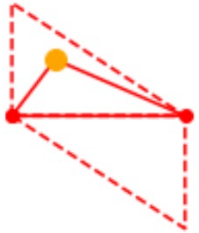
Is it still the worse point of the simplex?

Try to perform a **contraction** towards the barycenter of the other points of the original simplex.

No improvement?

Try to perform a **contraction** towards the best point of the simplex

Improvement?



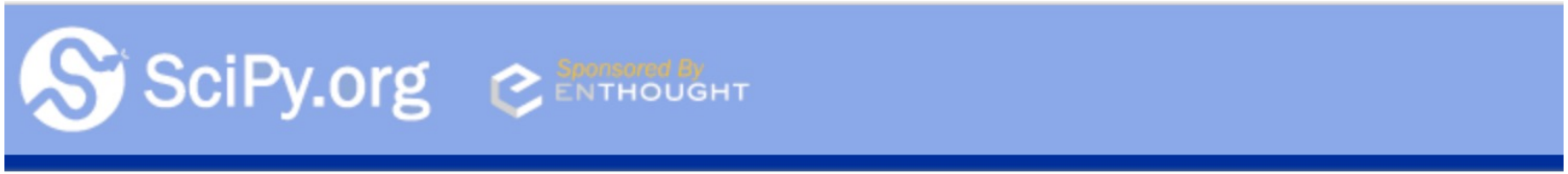
Keep the new point

Exact algorithm with code: *Press et al., "Numerical Recipes"*

+ Reiterate until convergence

How to use the Nelder-Mead simplex in Python

```
from scipy.optimize import fmin
```



Scipy.org | Docs | SciPy v1.2.3 Reference Guide | Optimization and Root Finding (**scipy.optimize**)

scipy.optimize.fmin

scipy.optimize.fmin(*func, x0, args=(), xtol=0.0001, ftol=0.0001, maxiter=None, maxfun=None, full_output=0, disp=1, retall=0, callback=None, initial_simplex=None*) [source]

Minimize a function using the downhill simplex algorithm.

This algorithm only uses function values, not derivatives or second derivatives.

Parameters: **func** : *callable func(x,*args)*

The objective function to be minimized.

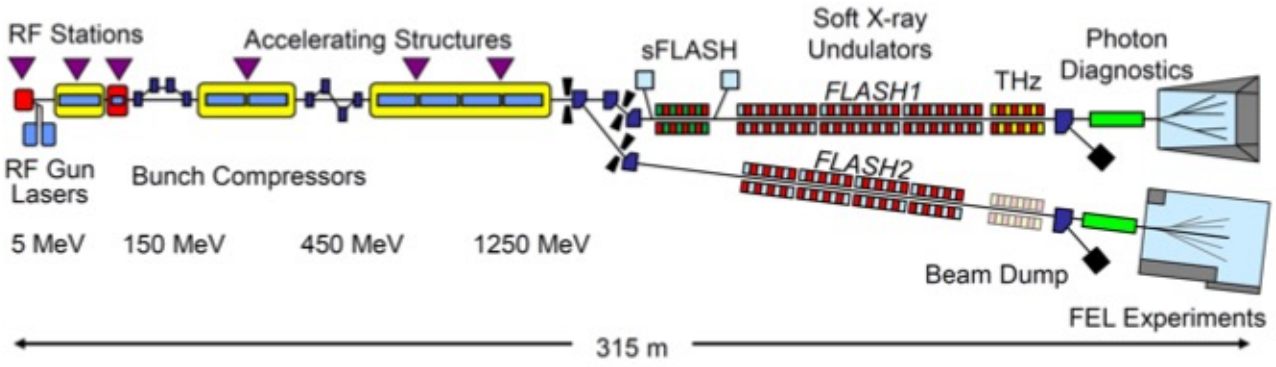
x0 : *ndarray*

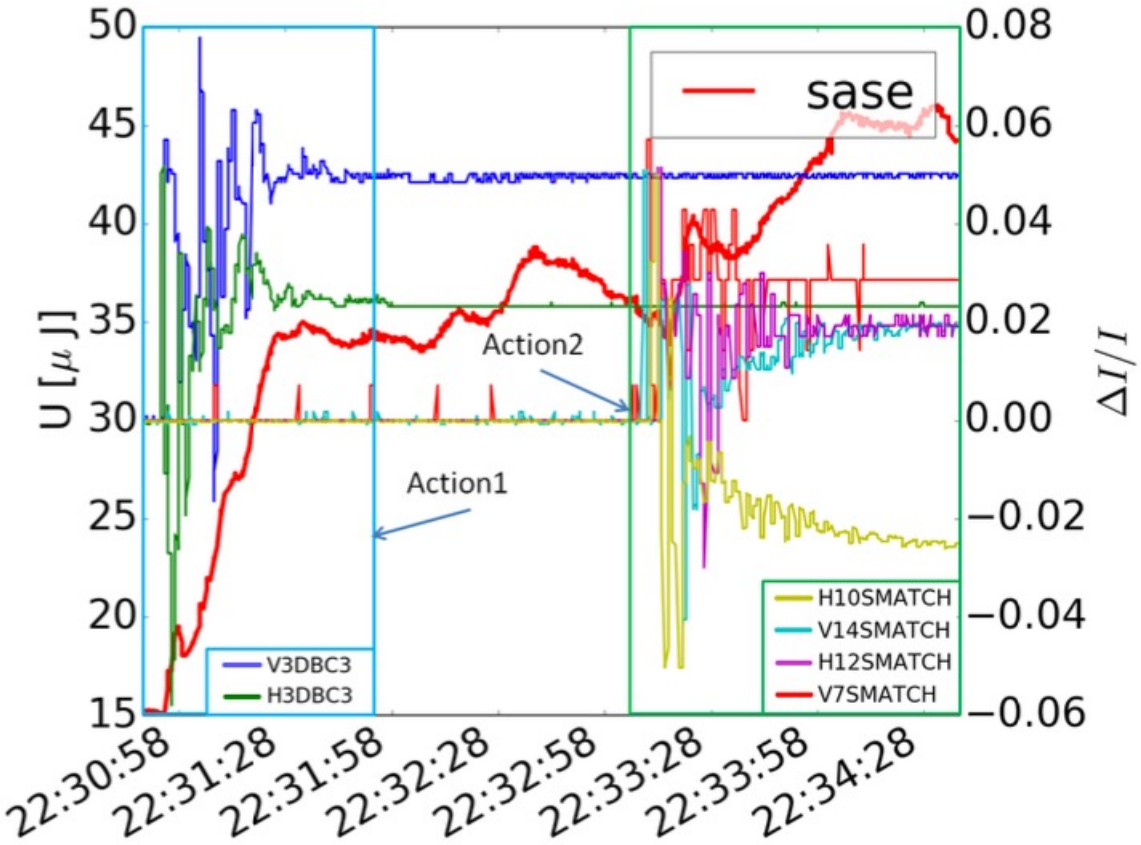
Initial guess.

Nelder-Mead simplex: example

Online optimization at the FLASH FEL (DESY):

Maximized FEL radiation (“sase” curve) with Nelder-Mead algorithm
by tuning two groups of beam optics elements (“Action 1” and ”Action 2”)





I. Agapov et al.,
“Automatic tuning of Free Electron Lasers” (2017)
https://arxiv.org/abs/1704.02335

Nelder-Mead simplex: practical considerations

- Relatively robust
- Extensively used for online tuning of accelerators
  Often considered as a **baseline method** in literature on optimization
- However, requires many evaluations of f compared to other methods
- Not very robust to noise
- No parallel evaluation (the algorithm is intrinsically sequential)

Outline

- Example and motivation for particle accelerators
- Optimization: general definition and naïve algorithms
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - **Gradient-descent**
  - Extremum Seeking
- Some general terms

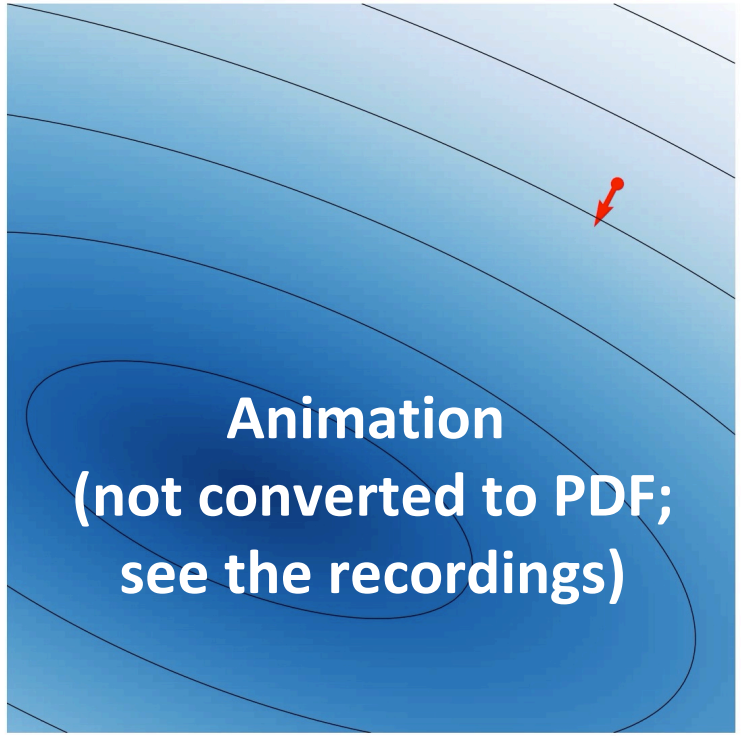
Gradient-descent: algorithm

- Calculate the **local gradient** of f
- Move in the **opposite** direction
  (i.e. towards the minimum)

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha \boldsymbol{\nabla} f(\boldsymbol{x}_n)$$

  α: "step size" (optimization)

- Iterate



Note: Gradient-descent is also **very common** in the context of **machine learning**.
In this case: f is the "loss function" (accuracy of the ML model), α is the "learning rate".
(See Wednesday's lecture)

Gradient-descent: how to choose the step size α

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha \boldsymbol{\nabla} f(\boldsymbol{x}_n)$$

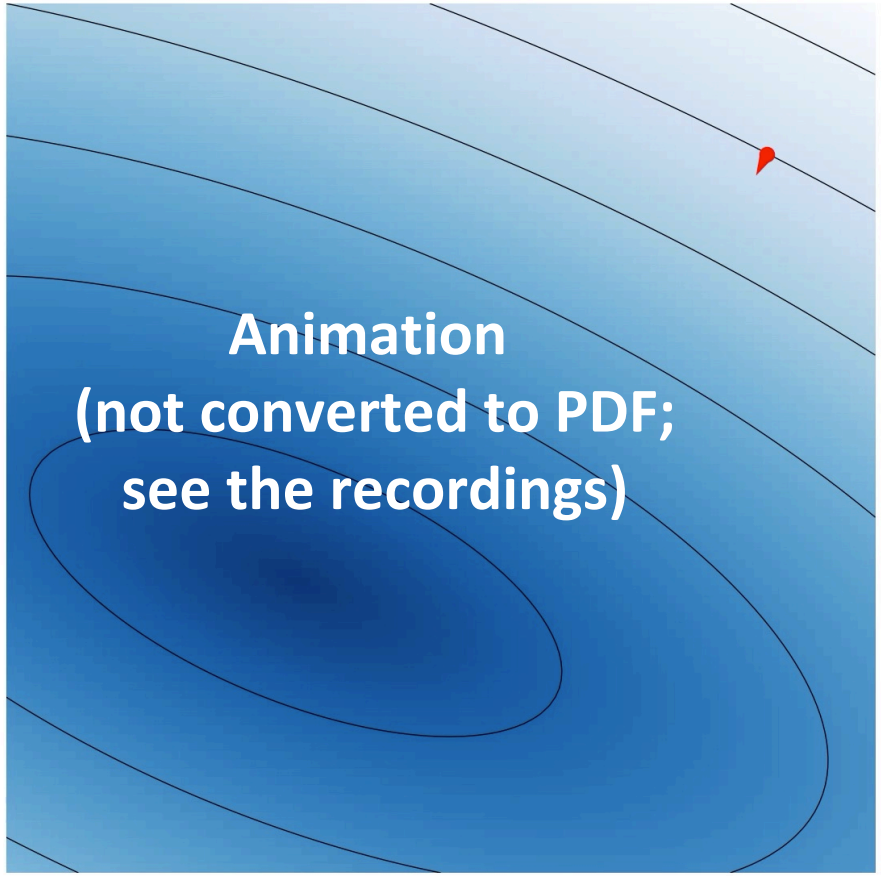
Trade-off:

- If α is too small: converges slowly (inefficient)
- If α is too large: may not converge

Common methods to choose α:

- Fixed, small value (e.g. $\alpha = 10^{-2}$)
- Adaptive: e.g. Adagrad, RMSProp algorithms
  (often used in ML context: see next week's lecture)

Step size too small:



Gradient-descent: how to choose the step size α

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha \boldsymbol{\nabla} f(\boldsymbol{x}_n)$$

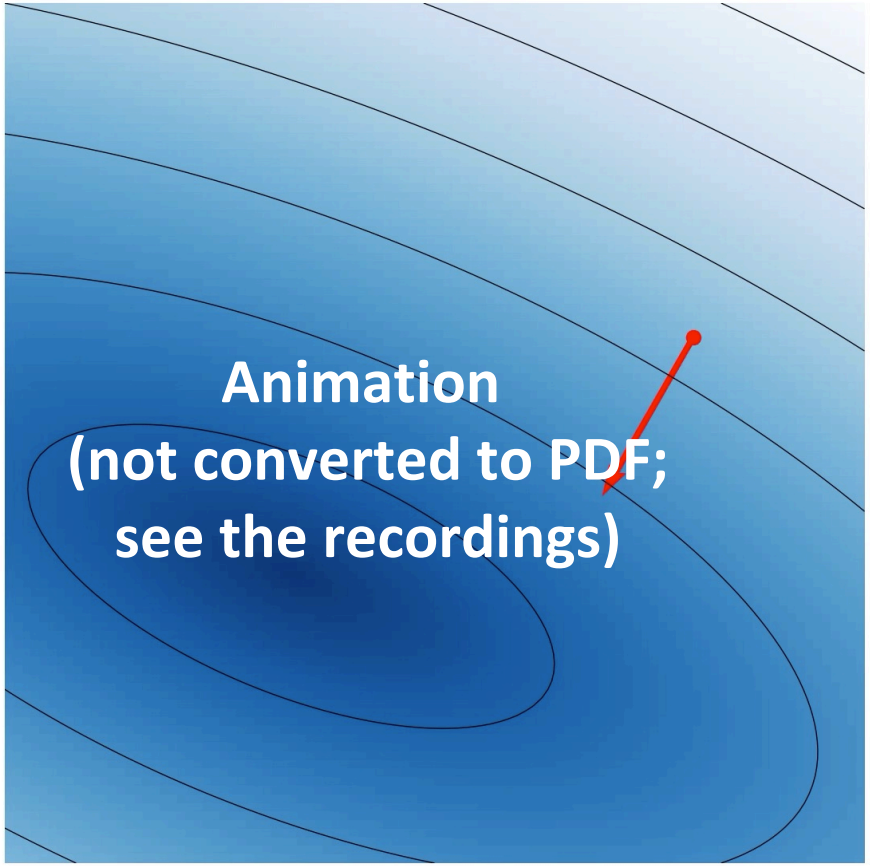
Trade-off:

- If α is too small: converges slowly (inefficient)
- If α is too large: may not converge

Common methods to choose α:

- Fixed, small value (e.g. $\alpha = 10^{-2}$)
- Adaptive: e.g. Adagrad, RMSProp algorithms
  (often used in ML context: see next week's lecture)

Step size too large:



Gradient-descent: how to calculate the gradient

Analytical calculation:

- Never possible if *f* is obtained from **real-time measurements**
- Sometimes possible when *f* is obtained from **numerical simulations** (some programming frameworks can automatically track the derivatives of every single mathematical operation in the simulation, e.g. `autograd`)
- Often possible when *f* is the loss function of an ML model

Numerical differentiation:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x_i + h) - f(x_i)}{h}$$

for **each** input parameter x_i

with *h* small

- Requires many (expensive) evaluations of *f*
- Sensitive to any noise in *f*

Numerical differentiation: sensitivity to noise

Assume evaluations of *f* are **noisy**: $f(\boldsymbol{x}) = \tilde{f}(\boldsymbol{x}) + \eta$

Noiseless part: always gives the same result, for a given ***x***

Stochastic part: value changes for each evaluation, with RMS σ_η

Numerical differentiation:

$$\frac{f(x_i + h) - f(x_i)}{h} = \frac{\tilde{f}(x_i + h) - \tilde{f}(x_i)}{h} + \left(\frac{\eta' - \eta}{h}\right)$$

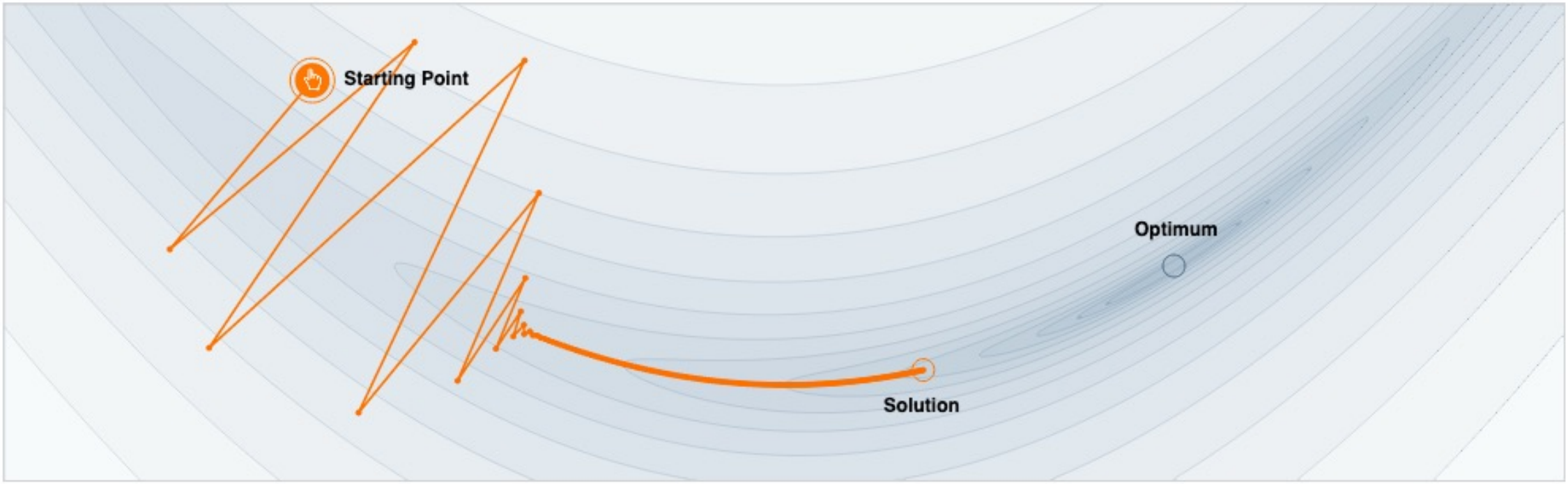
$$\approx \frac{\partial \tilde{f}}{\partial x_i} + \left(\frac{\eta' - \eta}{h}\right)$$

Stochastic term, with RMS $\frac{\sqrt{2}\sigma_\eta}{h}$

For small *h*, numerical differentiation **amplifies** the noise.

Gradient descent: the “valley problem”

If the objective function *f* presents a **long narrow valley**, gradient-descent converges very slowly.



Source: https://distill.pub/2017/momentum/

One possible solution: gradient descent with momentum

Gradient descent:

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \alpha \boldsymbol{\nabla} f(\boldsymbol{x}_n)$$

Gradient descent with momentum:

$$\boldsymbol{v}_{n+1} = \beta \boldsymbol{v}_n - \boldsymbol{\nabla} f(\boldsymbol{x}_n)$$

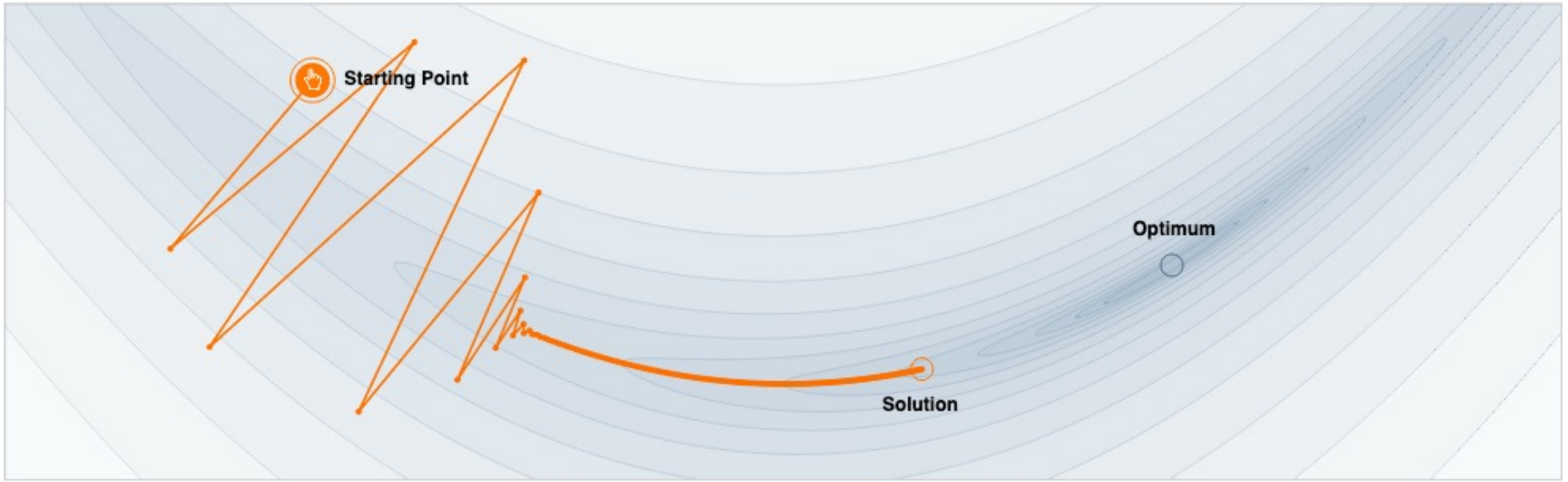
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha \boldsymbol{v}_{n+1}$$

$$0 \leq \beta < 1$$

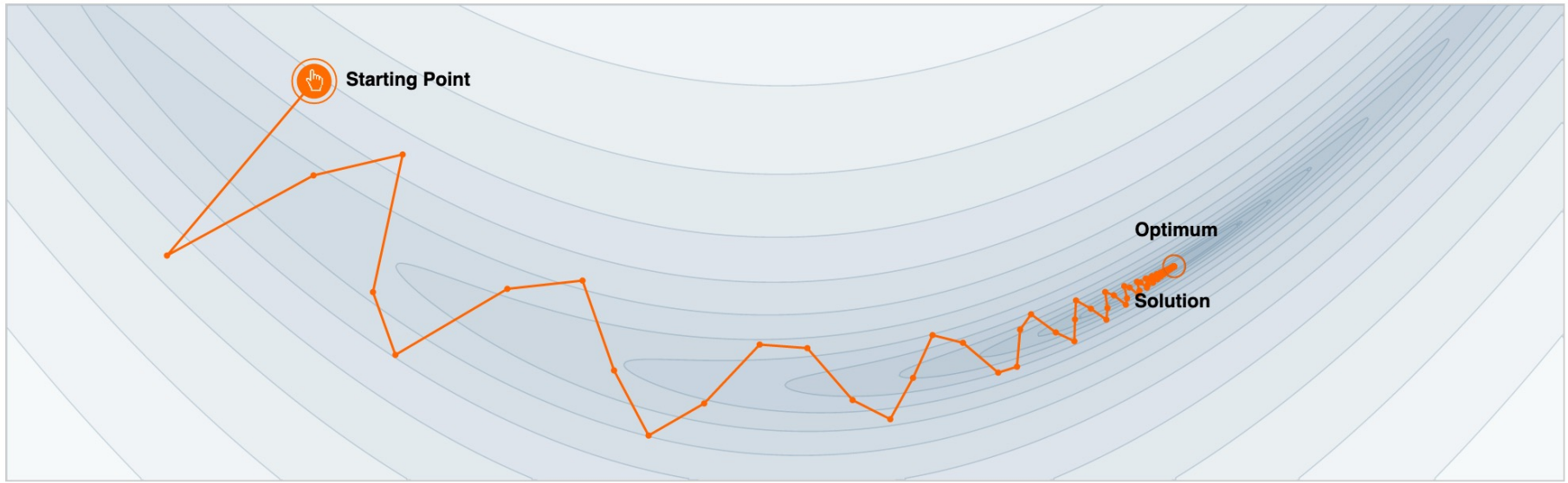
- For $\beta = 0$: gradient descent with momentum reduces to regular gradient descent
- But for β close to 1, $\boldsymbol{v}_n$ effectively **accumulates** $-\boldsymbol{\nabla} f$ over past iterations
- Similar to a point moving under a force $-\boldsymbol{\nabla} f$,
  with a friction coefficient proportional to $(1 - \beta)$

The “valley problem”

No momentum:
($\beta = 0$)



With momentum:
($\beta = 0.85$)



Source: https://distill.pub/2017/momentum/

Gradient descent: practical considerations

- Requires to carefully choose the step size ; issues with narrow valleys.
  (unless one uses gradient descent with momentum)
- Requires a reliable way to evaluate gradient (e.g. analytically)
- Relatively rarely used for optimization of particle accelerators,
  at least for the standard version of gradient descent
- Widely used within machine learning algorithm to optimize the loss function
- No parallel evaluation (the algorithm is intrinsically sequential)

Outline

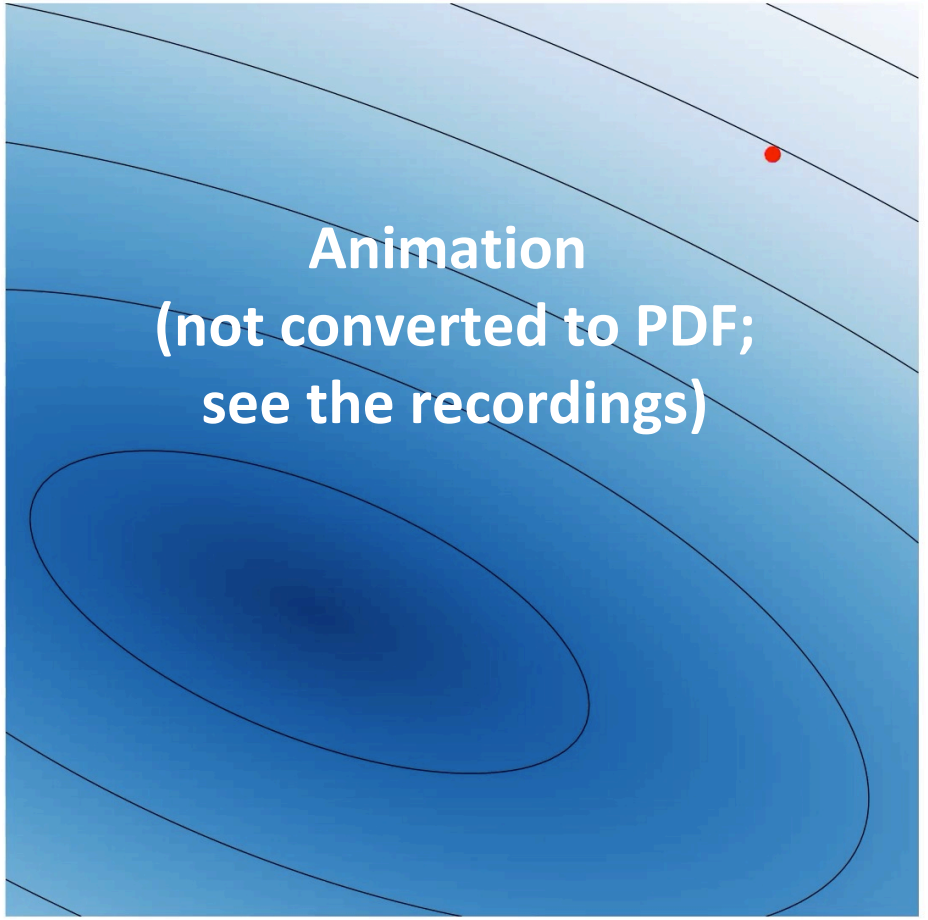
- Example and motivation for particle accelerators
- Optimization: general definition and naïve algorithms
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - Gradient-descent
  - **Extremum Seeking**
- Some general terms

Extremum seeking: introduction

- In **simplex** and **gradient descent** (with finite-difference derivative) the direction in which to move is inferred by **sampling neighboring points.**

- In **extremum seeking**, neighboring points are sampled by performing **small oscillations.**

- The aim here is not to be **efficient**, but rather to be **robust** for **real-time dynamic systems** (e.g. operating accelerators, in real-time, with drifts)

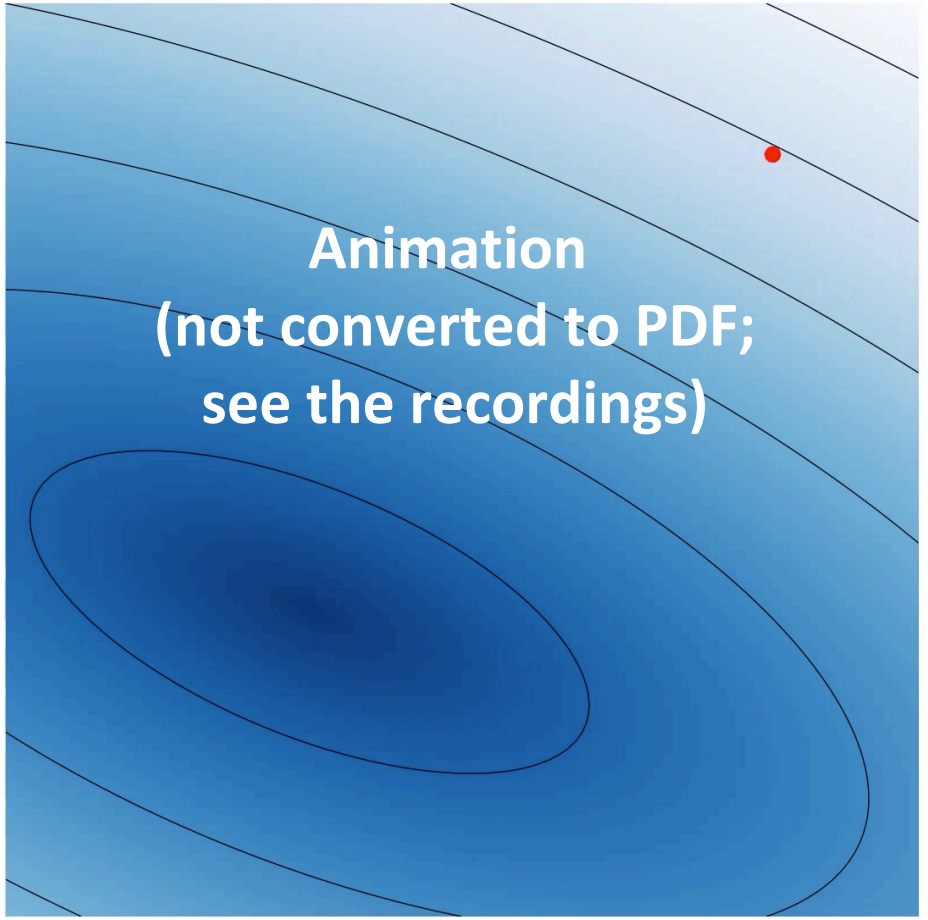


Extremum seeking: algorithm

At each step, the coordinates of the point are updated with:

$$x_{i,n+1} = x_{i,n} + \Delta t \sqrt{\alpha \omega_i} \cos(\omega_i n \Delta t + k f(\boldsymbol{x}_n))$$

- ω_i: real-time frequency of the oscillations (needs to be different for each coordinate for the method to work)
- Δt: real-time interval between evaluations
- α: controls the amplitude of the oscillations
- k: controls in which direction the average motion goes.



Extremum seeking: why does it work?

- The algorithm does not **explicitly** calculate the gradient (like gradient descent) or **explicitly** compare points (like simplex): how does it work?

$$x_{i,n+1} = x_{i,n} + \Delta t\sqrt{\alpha\omega_i}\cos(\omega_i n\Delta t + kf(\boldsymbol{x}_n))$$

- Note that the effective frequency of the oscillation is: $\omega_i + k\frac{\partial f}{\partial t}$

  If the point is at a phase where it is **already** moving towards a minimum, then $\frac{\partial f}{\partial t}<0$, and the point will **spend more time at this phase**.
  (similarities with ∇B drift for a charged particle gyrating in a non-uniform B field)

- Mathematically, it can be showed that the **average motion** satisfies

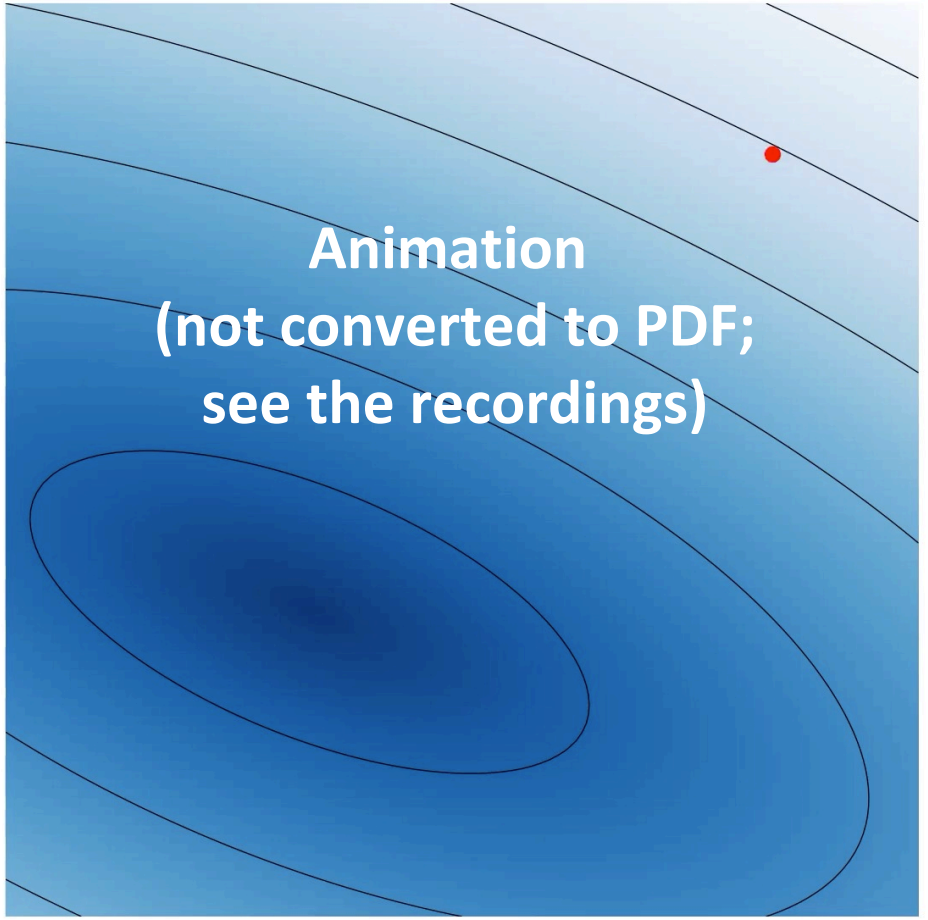
$$\frac{d\langle \boldsymbol{x}\rangle}{dt} = -\frac{k\alpha}{2}\boldsymbol{\nabla} f(\langle \boldsymbol{x}\rangle)$$

Extremum seeking: choosing parameters

$$x_{i,n+1} = x_{i,n} + \Delta t\sqrt{\alpha\omega_i}\cos(\omega_i n\Delta t + kf(\boldsymbol{x}_n))$$

$$\frac{d\langle \boldsymbol{x}\rangle}{dt} = -\frac{k\alpha}{2}\boldsymbol{\nabla} f(\langle \boldsymbol{x}\rangle)$$

- ω_i: needs to be fast compared to the drifting motion (again, needs to be different for each *i*)
- Δt: needs to be small compared to ω_i
- α: can be reduced as we get close to the minimum, in order to reduce the amplitude of the oscillation motion.

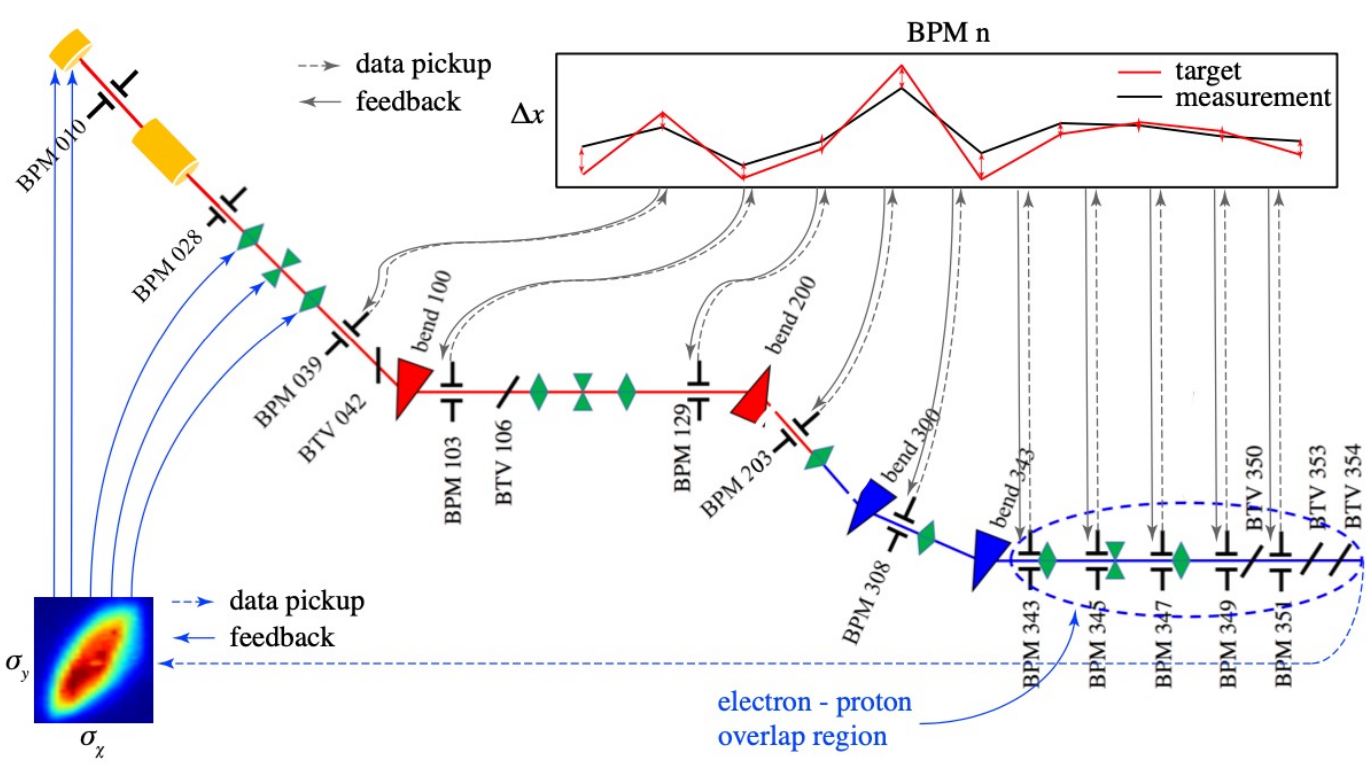


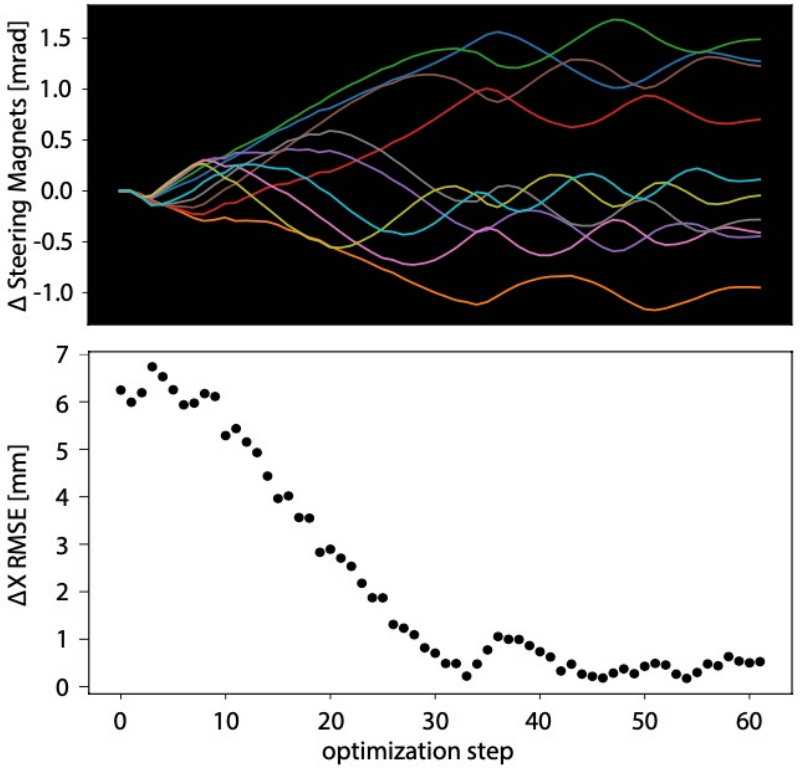
Extremum seeking: example at the AWAKE electron beam line

Aim: maintain beam on a target trajectory

Objective function (f): distance of beam centroid to the target trajectory, as measured by BPMs

Tuning parameters ($\boldsymbol{x}$): strength of 10 different steering magnets





Scheinker et al., "Online Multi-Objective Particle Accelerator Optimization of the AWAKE Electron Beam Line for Simultaneous Emittance and Orbit Control" (2020)
https://arxiv.org/abs/2003.11155v1

Outline

- Example and motivation for particle accelerators
- Optimization: general definition and naïve algorithms
- Some common optimization algorithms
  - Nelder-Mead algorithm
  - Gradient-descent
  - Extremum Seeking
- **Some general terms**

Optimization with constraints

Constraints directly on the input parameters:

e.g. minimize emittance by tuning steering magnets while ensuring that the **current that controls steering magnet** stays within a safe range.

Typical form: minimize $f(\boldsymbol{x})$ while ensuring $x_i \leq x_{max}$ for a given i and x_{max}

Easy to implement: simply restrict the domain Ω over which the optimization is performed.

Constraints that depend on the input parameters, but are difficult to predict and need to be measured/simulated:

e.g. minimize energy spread by tuning beam optics, while ensuring that the **beam loss** stays below a given threshold

Typical form: minimize $f(\boldsymbol{x})$ while ensuring $g(\boldsymbol{x}) \leq g_{max}$

More difficult to implement: need a to learng a model that can predict g and ensure that the optimization algorithm will not access unsafe parameters

Derivative-based vs. derivative-free optimization algorithm

Derivative-based algorithm	Derivative-free algorithm
The algorithm requires a way to evaluate the derivative of *f*.	The algorithm does not need to evaluate the derivative (only evaluates *f* itself).
Examples: • Gradient-descent	Examples: • Nelder-Mead • Extremum Seeking

Parallelizable vs. sequential optimization algorithm

Sequential algorithm

The point at which *f* is evaluated **depends** on the results of **all past evaluations.** Evaluations of *f* have to be carried out **sequentially.**

Examples:

- Nelder-Mead
- Gradient-descent
- (Extremum Seeking)

Parallelizable algorithm

Evaluations of *f* are (at least partially) **independent** and can be **carried out in parallel.**

Examples:

- Random search
- Grid search

Important for **simulation-based design studies:**
Parallel optimization algorithms allow independent simulations to be carried out on separate computational resources.

Local vs. global optimization algorithm

Local algorithm

Is likely to get “stuck” in **local minima.**

Examples:

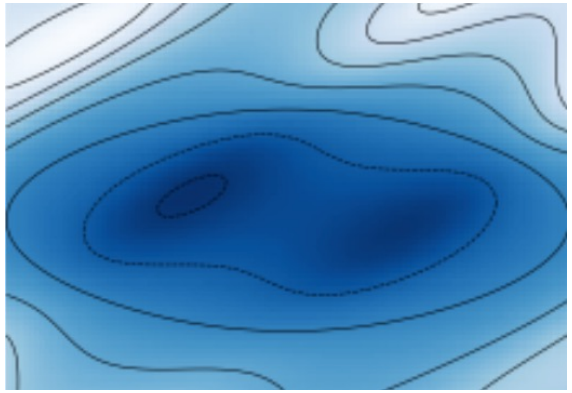
- Nelder-Mead
- Gradient-descent
- Extremum Seeking

Global algorithm

Attempts to find the global minimum, even in the presence of **local minima.**

Examples:

- Random search
- Grid search



Single-objective vs. Multi-objective optimization

Single-objective

Finds the minimum of a single **scalar** function.

Examples:

- Nelder-Mead
- Gradient-descent
- Extremum Seeking

Multi-objective

Simultaneously optimize **several** (potentially conflicting) functions ; find the optimal **trade-off**

See tomorrow's lecture

Thanks for your attention.

Feel free to ask questions!