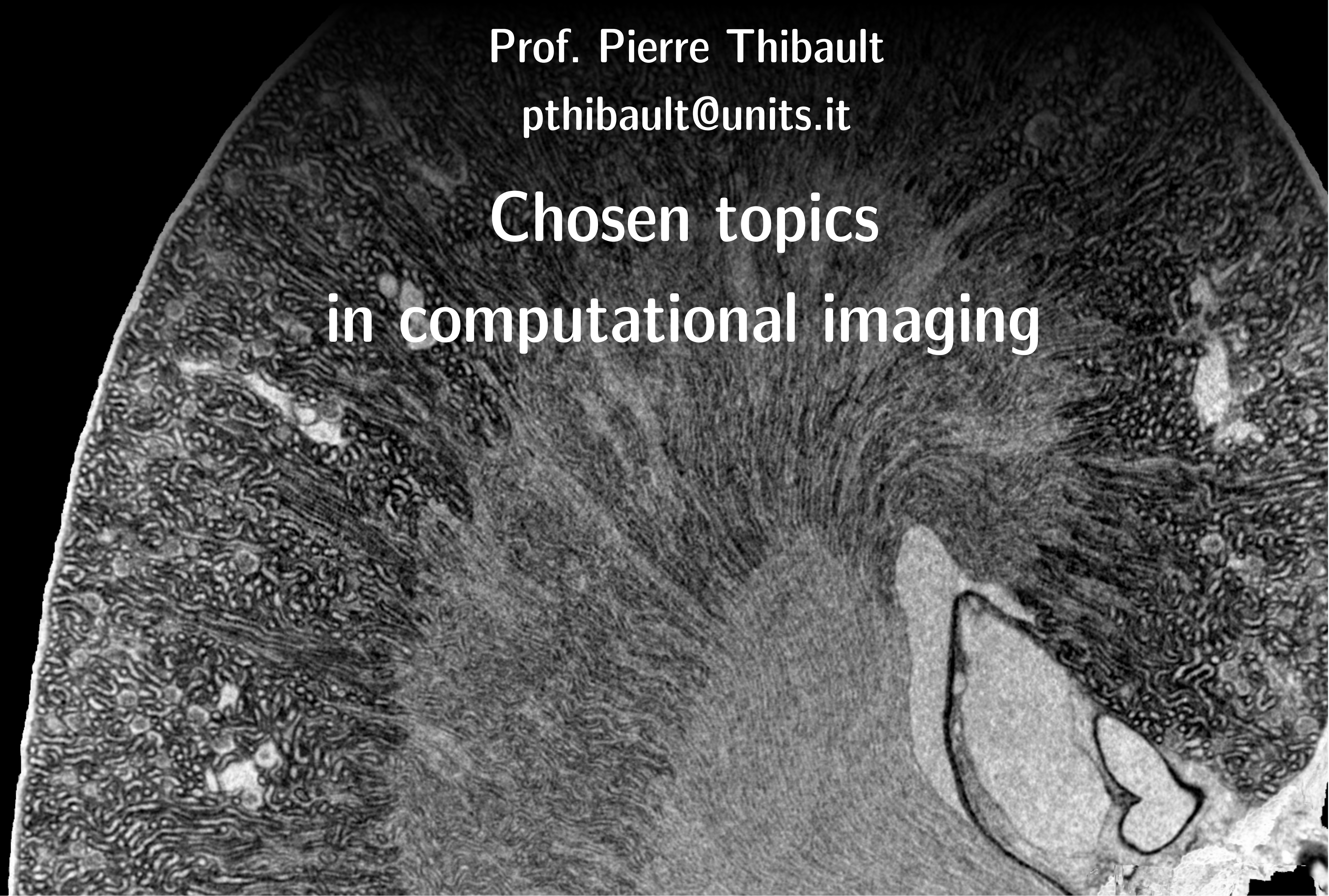
Image Processing for Physicists

Prof. Pierre Thibault

pthibault@units.it

Chosen topics in computational imaging



Overview

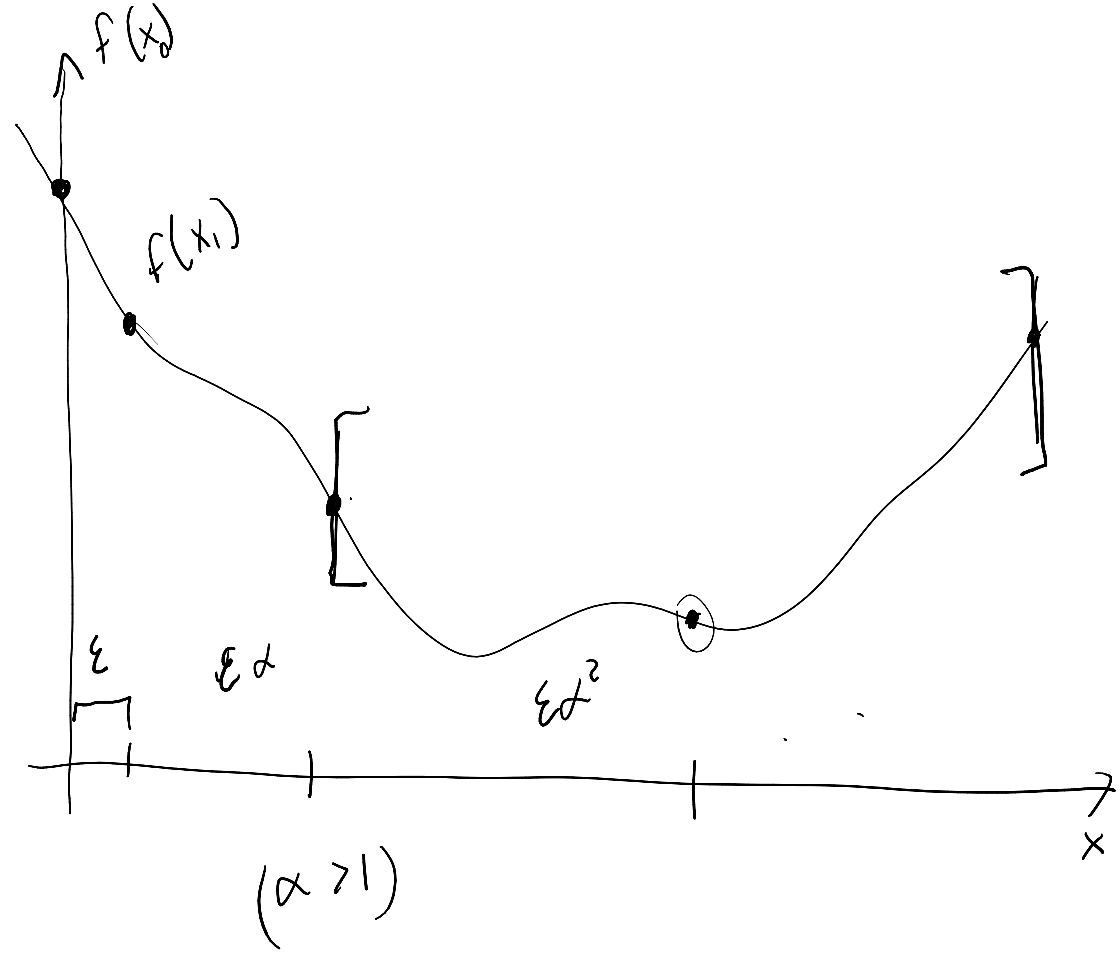
- Computational imaging
- Optimization
  - Characterization of linear systems
  - Standard optimization methods (in 1-D and N-D)
- Complex-valued variables: Wirtinger calculus
-

Computational imaging

1D optimization

- Bracketing

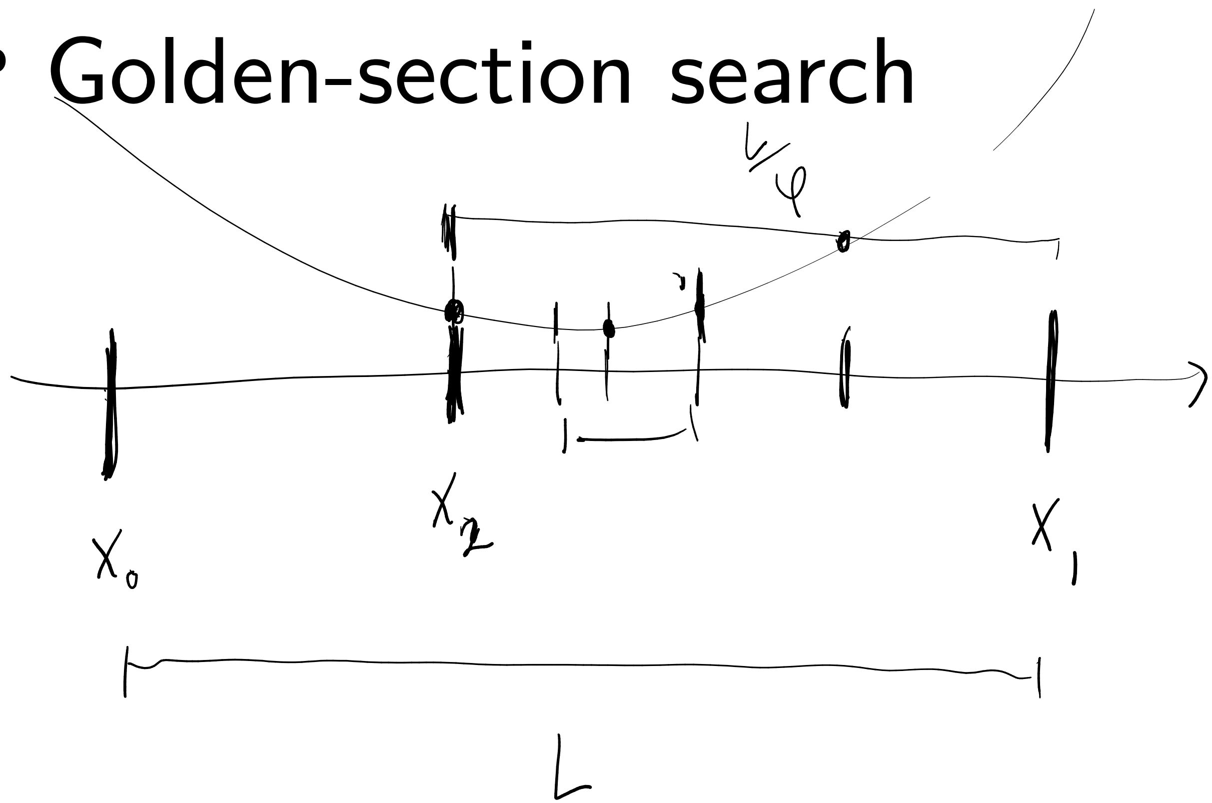
* commonly applied on $f(x)$ costly to evaluate, gradient unknown or also costly



1D optimization

- Golden-section search

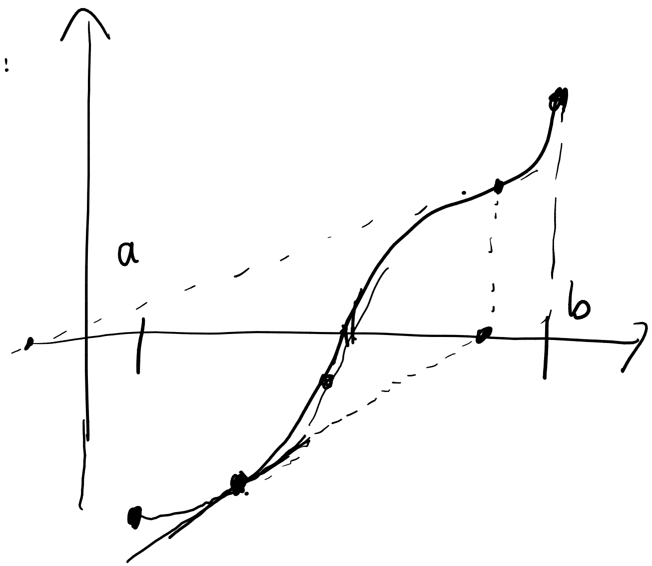
$$\varphi = \frac{\sqrt{5}+1}{2} \simeq 1.6181\ldots$$



1D optimization

- Newton's method ("Newton-Raphson")

root finding:



1D optimization: look for $f'(x) = 0$

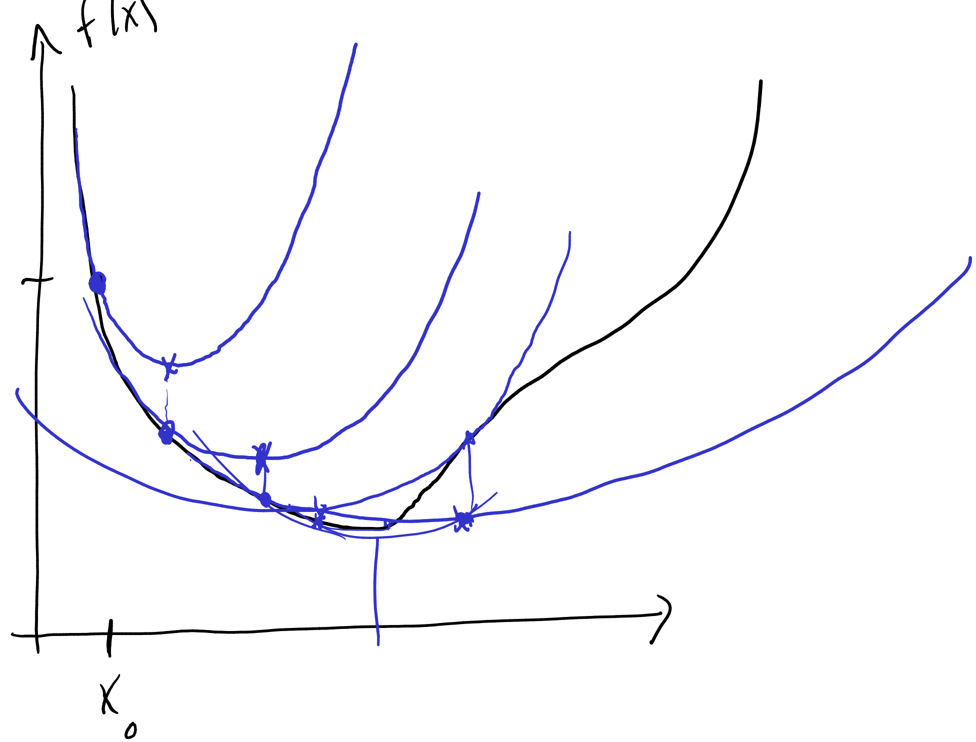
$\Longleftrightarrow$ root finding for $f'(x)$:

$\Rightarrow$ 2nd derivative of f is needed

$\Longleftrightarrow$ approximating f with a parabola

1D optimization

- Newton's method ("Newton-Raphson")



$$f(x) \approx f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2$$

$$f' = 0 \iff f'(x_0) + f''(x_0)(x-x_0) = 0$$

$$\Rightarrow x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

1D optimization

- Newton's method ("Newton-Raphson")

rule: iterate

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Brent method: combines golden section, Newton method

Optimization (multidimentional)

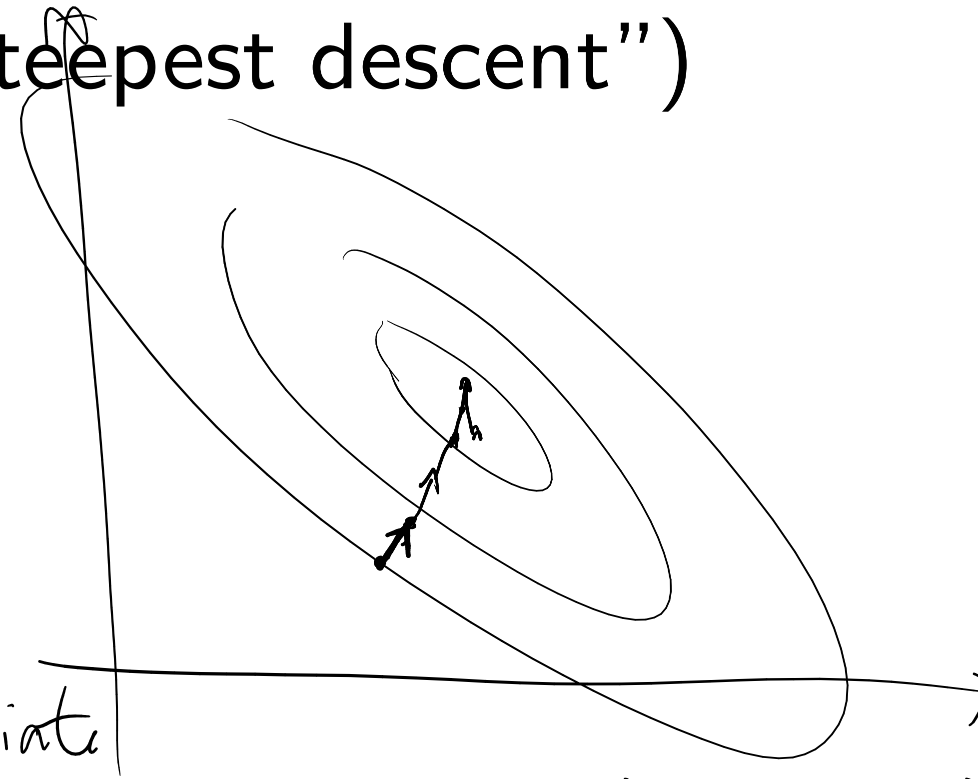
- Gradient descent ("steepest descent")

take a step in the direction

$-\nabla f$

* how far?

↳ what is the appropriate step size?

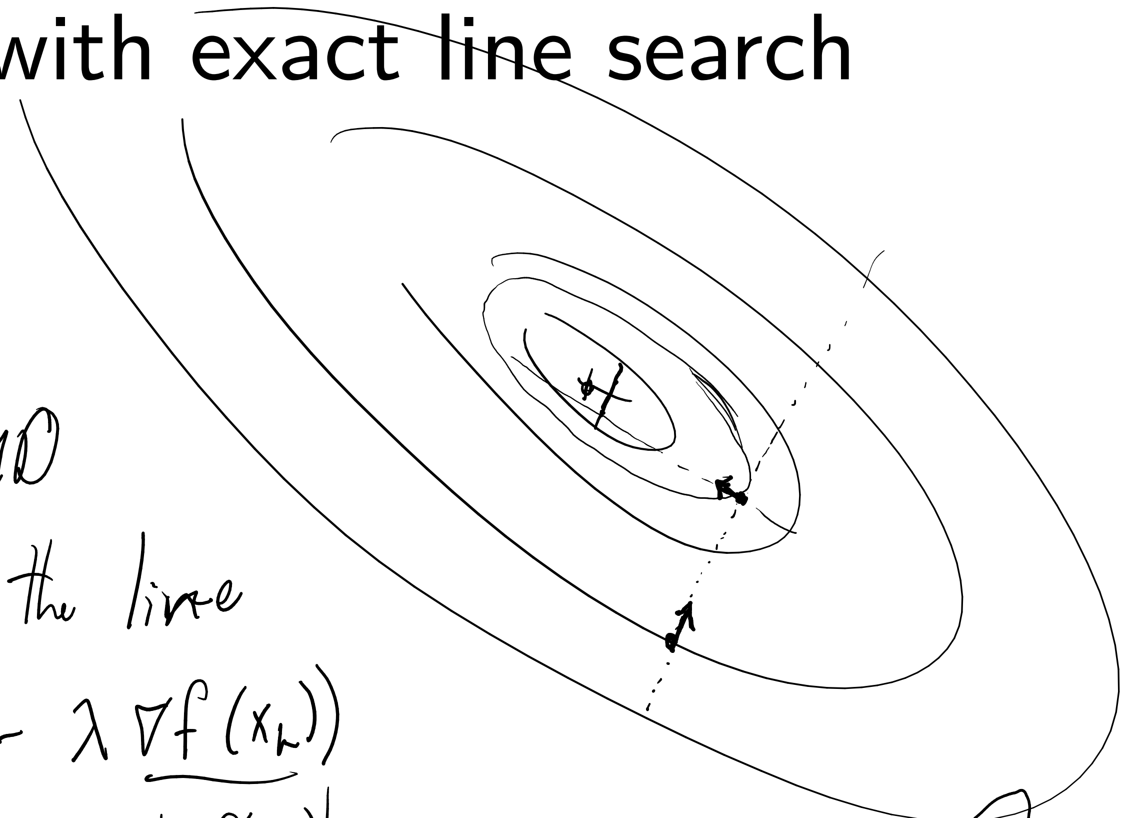


$$\vec{x}_{k+1} = \vec{x}_k - s_k \nabla f(\vec{x}_k)$$

"step size" ⟷ "learning rate"

Optimization

- Steepest descent with exact line search



* compute $\nabla f(\vec{x}_k)$

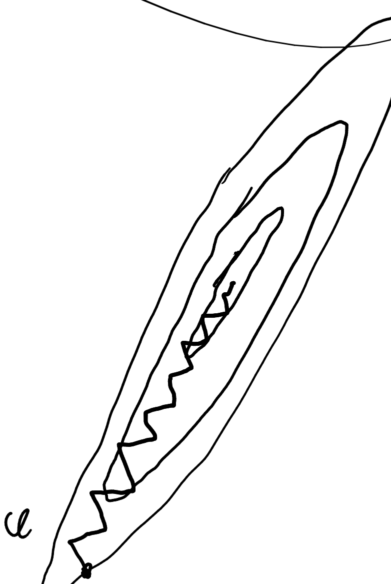
* carry out a full 1D minimization along the line

$$g(\lambda) = f\left(\vec{x}_k - \lambda \frac{\nabla f(x_k)}{|\nabla f(x_k)|}\right)$$

= "line search"

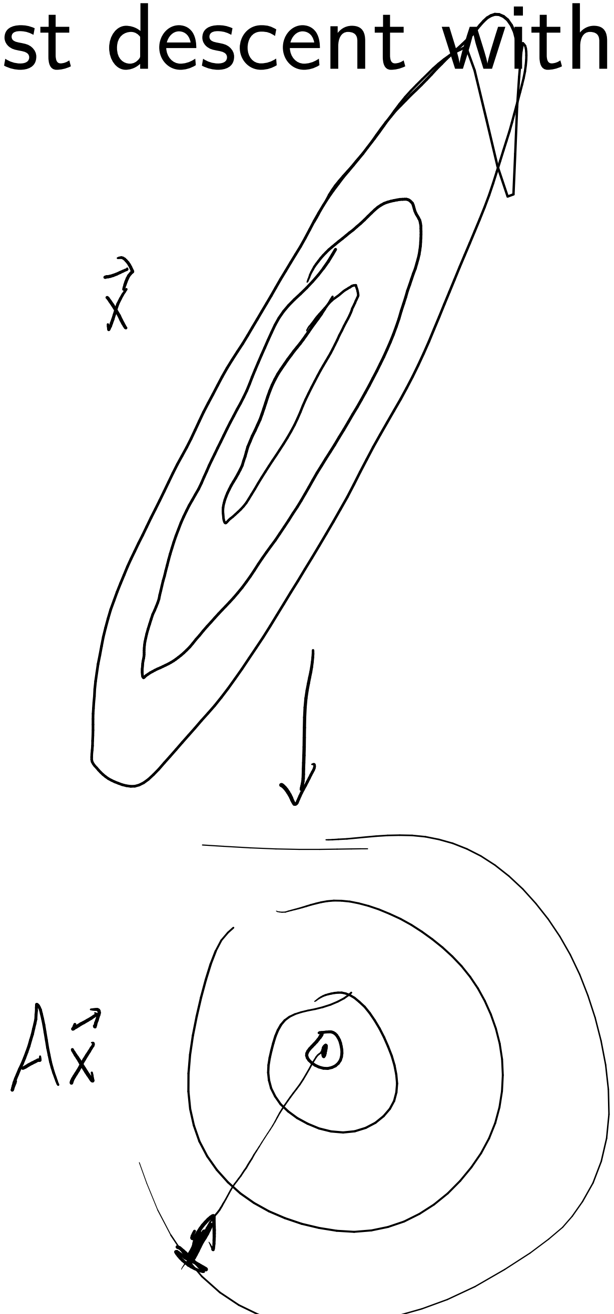
* iterate

* problem: often slow convergence



Optimization

- Steepest descent with exact line search



one possible solution:

use "preconditioner"

preconditioner: (linear) transform on $\vec{x}$ to make the function less elongated.

Optimization

- ~~Conjugate gradients~~ Newton's method

$$f(\vec{x}) \approx \underbrace{f(\vec{x}_0) + \nabla f(\vec{x}_0)\cdot(\vec{x}-\vec{x}_0) + \frac{1}{2}(\vec{x}-\vec{x}_0)^T H (\vec{x}-\vec{x}_0)}$$

Hessian matrix

minimum is at

$$\vec{x} = \vec{x}_0 - H^{-1}\cdot\nabla f$$

powerful but not very useful for large problems because Hessian is too big (or expensive to compute)

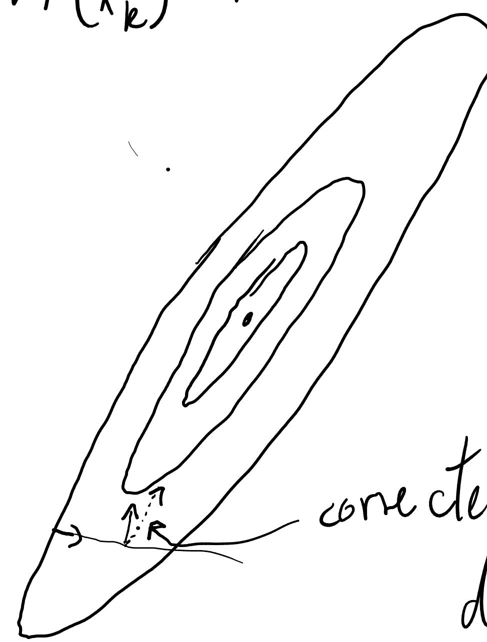
Optimization

- Conjugate gradients ← one of many "quasi-Newton" methods

* compute the gradient $\vec{g}_k = \nabla f(x_k)$

* compute correction factor

$$\beta_k = \frac{\vec{g}_k \cdot (\vec{g}_k - \vec{g}_{k-1})}{\vec{g}_{k-1} \cdot \vec{g}_{k-1}}$$



corrected search direction

* search direction:

$$\vec{h}_k = -(\vec{g}_k - \beta_k \vec{h}_{k-1})$$

previous search direction

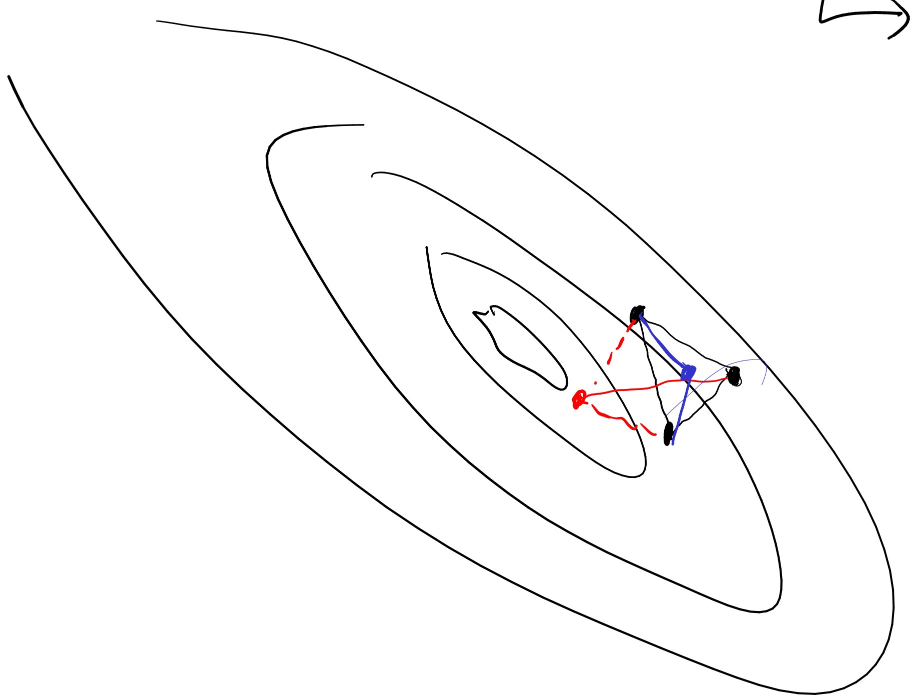
to avoid "undoing" minimization that has already been done.

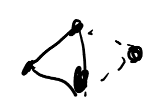
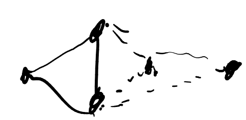
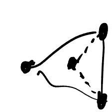
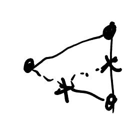
Optimization

- Nelder-Mead (downhill simplex method)

Used if gradient unknown or difficult to compute
(like a multidimensional version of golden section)

* uses a N+1 simplex in N dimensions
  ↳ Nd version of a triangle (2d)
  tetrahedron (3d)
  ⋮



* reflection 
* expansion 
* contraction 
* shrink 

Optimization

- Stochastic gradient descent

  * if cost function is large sum of terms (e.g. least squares with many data, or likelihood function)

  * idea: compute gradient using subset of the data

  * especially useful with very large and redundant datasets

Optimization

- Gauss-Newton method

Newton algorithm applied to non-linear least square cost function

$$f(\vec{\beta}) = \sum_i \left| M(\vec{\beta}; x_i) - y_i \right|^2$$

model

measured data

$$= \sum_i \left| r_i(\vec{\beta}) \right|^2$$

$$\frac{\partial f}{\partial \beta_j} = 2 \sum_i r_i(\vec{\beta}) \underbrace{\frac{\partial r_i}{\partial \beta_j}}_{J_{ij}} = 2 J^T r \qquad \frac{\partial^2 f}{\partial \beta_k \partial \beta_l} = \underbrace{2 \sum_i \frac{\partial r_i}{\partial \beta_k} \frac{\partial r_i}{\partial \beta_l}}_{(J^T J)_{\beta l}} + \underbrace{2 \sum_i r_i \frac{\partial^2 r_i}{\partial \beta_k \partial \beta_l}}_{\text{neglect this term}}$$

Newton: $\vec{\beta}^{(n+1)} = \vec{\beta}^{(n)} - (J^T J)^{-1} J^T r(\beta^{(n)})$

Optimization

- Levenberg–Marquardt algorithm

$$f(\vec{\beta}) \simeq f(\vec{\beta}_0) + \nabla f^T (\vec{\beta} - \vec{\beta}_0) + (\vec{\beta} - \vec{\beta}_0)^T H (\vec{\beta} - \vec{\beta}_0)$$

$$\downarrow$$

$$\nabla f^T + H(\vec{\beta} - \vec{\beta}_0) = 0$$

$$\cancel{2} J^T r + \cancel{2} \underbrace{J^T J}_{\downarrow \text{ replace with}} (\vec{\beta} - \vec{\beta}_0) = 0$$

$$(J^T J + \lambda \mathbb{1})(\vec{\beta} - \vec{\beta}_0)$$

$$\Rightarrow \vec{\beta} - \vec{\beta}_0 = -(J^T J + \lambda \mathbb{1})^{-1} J^T r$$

$$\vec{\beta}^{(n+1)} = \vec{\beta}^{(n)} - (J^T J + \lambda \mathbb{1})^{-1} J^T r$$

Optimization

- Levenberg–Marquardt algorithm

$$\text{if } \lambda \to \infty \qquad \vec{\beta}^{(n+1)} = \vec{\beta}^{(n)} - \frac{1}{2\lambda} \overbrace{2J^T r}^{\nabla f}$$

λ introduces a bias towards gradient descent

λ : Marquardt parameter - can be adjusted dynamically

Wirtinger derivatives

* Start with $f(z_1, z_2)$ where $z_1, z_2 \in \mathbb{C}$
* Introduce a change of variables:

$$z_1 = w_1 + w_2 \quad z_2 = w_1 - w_2 \qquad w_1, w_2 \in \mathbb{C}$$

$$F(w_1, w_2) := f\big(z_1(w_1, w_2), z_2(w_1, w_2)\big)$$

$$f(z_1, z_2) = F\left(\frac{z_1 + z_2}{2}, \frac{z_1 - z_2}{2}\right)$$

$$\frac{\partial f}{\partial z_1} = \frac{\partial F}{\partial w_1}\frac{\partial w_1}{\partial z_1} + \frac{\partial F}{\partial w_2}\frac{\partial w}{\partial z_1} = \frac{1}{2}\left(\frac{\partial F}{\partial w_1} + \frac{\partial F}{\partial w_2}\right)$$

$$\frac{\partial f}{\partial z_2} = \frac{1}{2}\left(\frac{\partial F}{\partial w_1} - \frac{\partial F}{\partial w_2}\right)$$

Wirtinger derivatives

Special case: $w_1 = x + i0$

$w_2 = 0 + iy$

$\Rightarrow z_1 = x + iy \qquad z_2 = x - iy$

$= z \qquad\qquad = z^*$

$$\boxed{\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial F}{\partial x} - i\frac{\partial F}{\partial y}\right) \qquad \frac{\partial f}{\partial z^*} = \frac{1}{2}\left(\frac{\partial F}{\partial x} + i\frac{\partial F}{\partial y}\right)}$$

special case: if $f \in \mathbb{R}$

then,

$$\left(\frac{\partial f}{\partial z}\right)^* = \frac{1}{2}\left(\frac{\partial F}{\partial x} + i\frac{\partial F}{\partial y}\right) = \frac{\partial f}{\partial z^*}$$

Wirtinger derivatives

* If f is a cost function, it is real (and probably non-negative)

then, $\frac{\partial f}{\partial z} = 0 \iff \frac{\partial f}{\partial z^*} = 0$ because $\left(\frac{\partial f}{\partial z}\right)^* = \frac{\partial f}{\partial z^*}$

$\Rightarrow$ to minimize a cost function with complex-valued arguments $(z_1, z_2, z_3, \ldots)$ we only need to set $\frac{\partial f}{\partial z_1} = \frac{\partial f}{\partial z_2} = \ldots = 0$ or $\frac{\partial f}{\partial z_1^*} = \frac{\partial f}{\partial z_2^*} = \ldots = 0$

(frequently used with Fourier space cost functions)

Singular Value Decomposition

Any matrix can be decomposed as

$$A = U \Sigma V^{\dagger}$$

where U and V are square unitary matrices and Σ has only non-negative diagonal entries.

$$
\begin{array}{cc}
 & n \\
m & \begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix}
\end{array}
=
\begin{array}{cc}
 & m \\
m & \begin{bmatrix} & & & \\ & & U & \\ & & & \end{bmatrix}
\end{array}
\begin{array}{c}
n \\
\begin{bmatrix} \sigma_1 & & 0 \\ & \sigma_2 & \\ & & \sigma_3 \ddots \\ & 0 & \end{bmatrix}
\end{array}
\begin{array}{cc}
\begin{bmatrix} & & \\ & V^{\dagger} & \\ & & \end{bmatrix} & n \\
n &
\end{array}
$$

Singular Value Decomposition

Relationship with eigen-decomposition:

$A^\dagger A$ is square. what are the eigen-vectors and eigenvalues of $A^\dagger A$?

$$A^\dagger A v = \lambda v$$

$$A^\dagger A W = \Lambda W$$

$$W = \begin{bmatrix} | & | & | & \\ w_1 & w_2 & w_3 & \cdots \\ | & | & | & \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & 0 \\ 0 & & \lambda_3 & \\ & & & \ddots \end{bmatrix}$$

$$W^\dagger A^\dagger A W = \Lambda$$

$$A^\dagger A = V \Sigma^T \underbrace{U^\dagger U}_{\mathbb{1}} \Sigma V^\dagger = V \Sigma^T \Sigma V^\dagger$$

Singular Value Decomposition

$$W^{\dagger}\Lambda W = W^{\dagger}A^{\dagger}AW = W^{\dagger}V\underbrace{\Sigma^{T}\Sigma}_{S}V^{\dagger}W$$

$$S = \begin{bmatrix} \sigma_1^2 & & & 0 \\ & \sigma_2^2 & & \\ & & \sigma_3^2 & \\ 0 & & & \ddots \end{bmatrix}$$

$$\Rightarrow V^{\dagger} = W \quad , \quad \Lambda = S$$

conclusion: * the eigenvalues of $A^T A$ are $\lambda_1 = \sigma_1^2$,

$\lambda_2 = \sigma_2^2$,

$\vdots$

* the colums of $V^{\dagger}$ are the eigenvectors of $A^T A$

$\Rightarrow$ convenient in some istances to compute σ_i and V

Singular Value Decomposition

relation with matrix rank

example $m > n$: represents overdetermined system

$$m \; \underbrace{\overset{n}{\begin{bmatrix} - r_1 - \\ - r_2 - \\ - r_3 - \\ \vdots \\ - r_N - \end{bmatrix}}}_{A} = \begin{bmatrix} \underbrace{\quad\quad}_{n} & \vdots & \underbrace{\quad U \quad}_{m-n \text{ columns are arbitrary}} \end{bmatrix} \begin{array}{c} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \\ & 0 & \end{bmatrix} \end{array} \begin{bmatrix} V^+ \end{bmatrix}$$

$$V^+ = \begin{bmatrix} | & | & \\ v_1 & v_2 & \cdots \\ | & | & \end{bmatrix}$$

columns: basis that "best" represent the rows of the matrix A

→ normally, the singular values are sorted in descending order,

Singular Value Decomposition

rank is at most min(m, n)

rank = number of non-zero singular values.

pseudo-inverse:

definition: $(A^\dagger A)^{-1} A^\dagger$

$(V\Sigma^T U^\dagger U \Sigma V^\dagger)^{-1} V \Sigma U^\dagger$

$= (VSV^\dagger)^{-1} V \Sigma U^\dagger$

maybe doesn't exist

$= V S^{-1} \underbrace{V^\dagger V}_{=1} \Sigma U^\dagger$

$= V S^{-1} \Sigma U^\dagger$

$$= V \begin{bmatrix} \frac{1}{\sigma_1} & & \\ & \frac{1}{\sigma_2} & \\ & & \ddots \end{bmatrix} U^{+}$$

* if $\sigma_i \to 0$, inversion becomes unstable

→ characterized by the ratio $\dfrac{\sigma_{max}}{\sigma_{min}}$

$\dfrac{\sigma_{max}}{\sigma_{min}}$ = conditioning number N

$N = 1$: well-conditioned, invertible ...

$N \to \infty$: ill-conditioned system

Dimensionality reduction

approximate Σ with fewer non-zero elements

(truncate the list of singular values, removing the smallest)