Technischen Universität München

Summer Semester 2011/2012

3D COMPUTER VISION I

Non-linear Parameter Estimation

Slobodan Ilić

Content

- Applications
- Introduction. Cost functions and least squares.
- Linear least-squares. Line fitting.
- Non-linear methods
  - Gradient/steepest descent
  - Conjugate gradient
  - Gauss-Newton methods
  - Levenberg-Marquardt

Applications

Face modeling form video

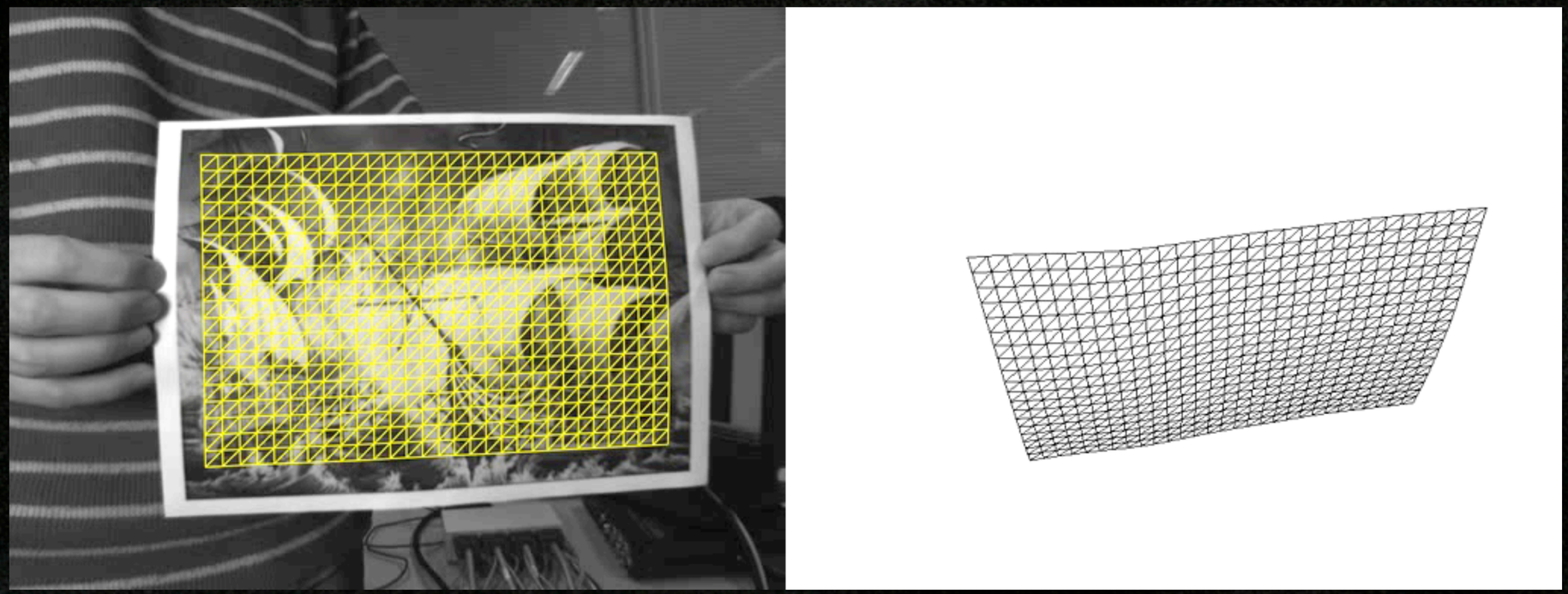
S. Ilic, M. Salzmann, P. Fua, Implicit Meshes for Effective Silhouette Handling, International Journal of Computer Vision, Vol. 72, Nr. 2, pp. 159 - 178, 2007.

Applications

Model based tracking of rigid objects

L. Vacchetti, V. Lepetit and P. Fua, Stable Real-Time 3D Tracking Using Online and Offline Information, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 26, Nr. 10, pp. 1391 - 1391, 2004.

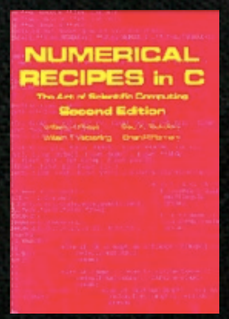
Applications
Deformable surface tracking



M. Salzmann, J.Pilet, S.Ilic, P.Fua, Surface Deformation Models for Non-Rigid 3--D Shape Recovery, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 29, Nr. 8, pp. 1481 - 1487, August 2007.

Text books

- Practical Methods of Optimization, R. Fletcher, Wiley, 1987.
  Note: Covers unconstrained and constrained optimization.
  Very clear and comprehensive.



- Numerical Recipes in C (or C++) : The Art of Scientific Computing, William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling, CUP 1992/2002.
  Note: Cook book with good chapters on optimization and data modeling with available implementations.

  Available on-line at http://www.nrbook.com/a/bookcpdf.php

- Convex Optimization, Stephen Boyd and Lieven Vandenberghe, CUP 2004,
  Note: Available on-line at: http://www.stanford.edu/~boyd/cvxbook/

Introduction
Problem definition

- Consider the objective or cost function:

$$f(\mathbf{x}) : \mathbb{R}^{\mathbf{n}} \rightarrow \mathbb{R}$$

- The goal is to find the values of parameters $\mathbf{x}$ which minimize this function:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{f}(\mathbf{x})$$

- subject to constraints:

$$c_i(\mathbf{x}) = \mathbf{0}, i = 1, \ldots, m_e$$

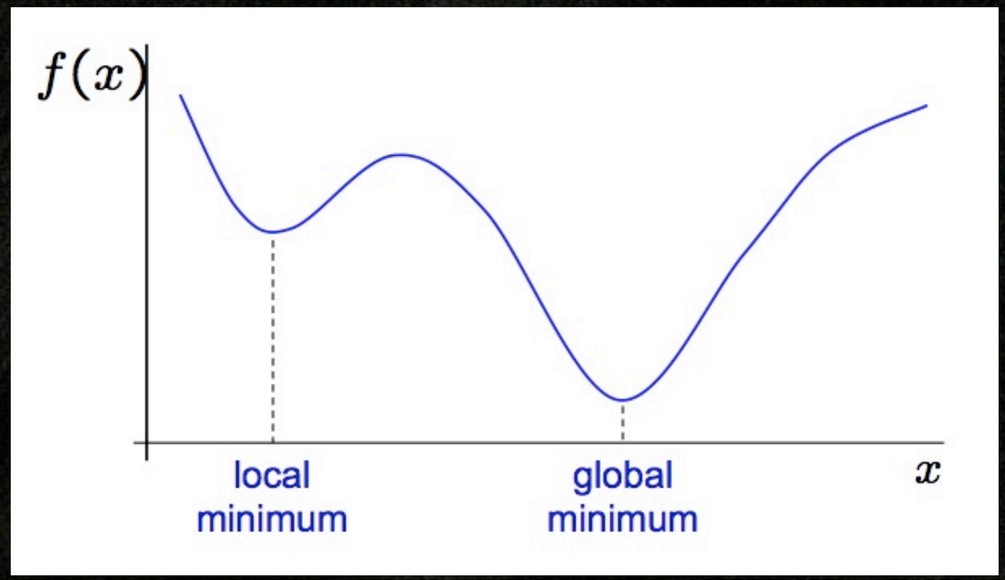
$$c_i(\mathbf{x}) \geq \mathbf{0}, i = m_e + 1, \ldots, m$$

- We do not impose constraints!
- We do unconstrained optimization.

Unconstrained optimization

function of one variable

$$\min_x f(x)$$



- down-hill search (gradient descent) algorithms can find local minima
- which of the minima is found depends on the starting point
- such minima often occur in real applications

Example: template matching in 2D images

- Input are two point sets:

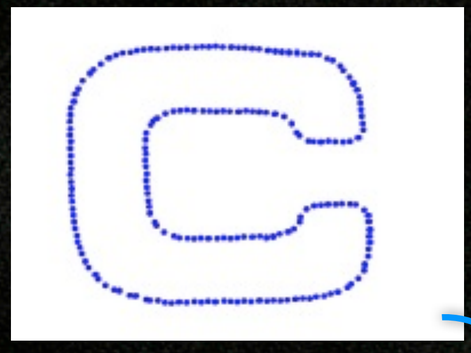
$\mathcal{M} = \{M_i\}$ model points

$\mathcal{D} = \{D_i\}$ data points

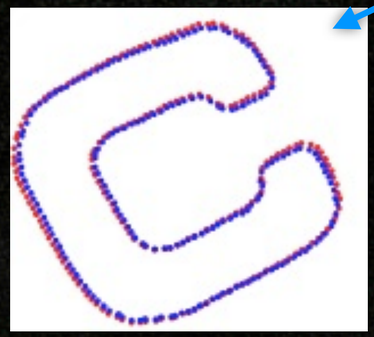
- Objective:

Determine the transformation $\mathcal{T}$ which minimizes the error between the model $\mathcal{M}$ and the data $\mathcal{D}$.

Model $\mathcal{M}$



Transformation $\mathcal{T}$



Data $\mathcal{D}$

Cost function

$$f(\theta, t_x, t_y) = \sum_j \min_i \underbrace{\|\mathbf{R}(\theta) M_i + \mathbf{t} - \mathbf{D}_j\|^2}$$

for each data point

find the closest model point

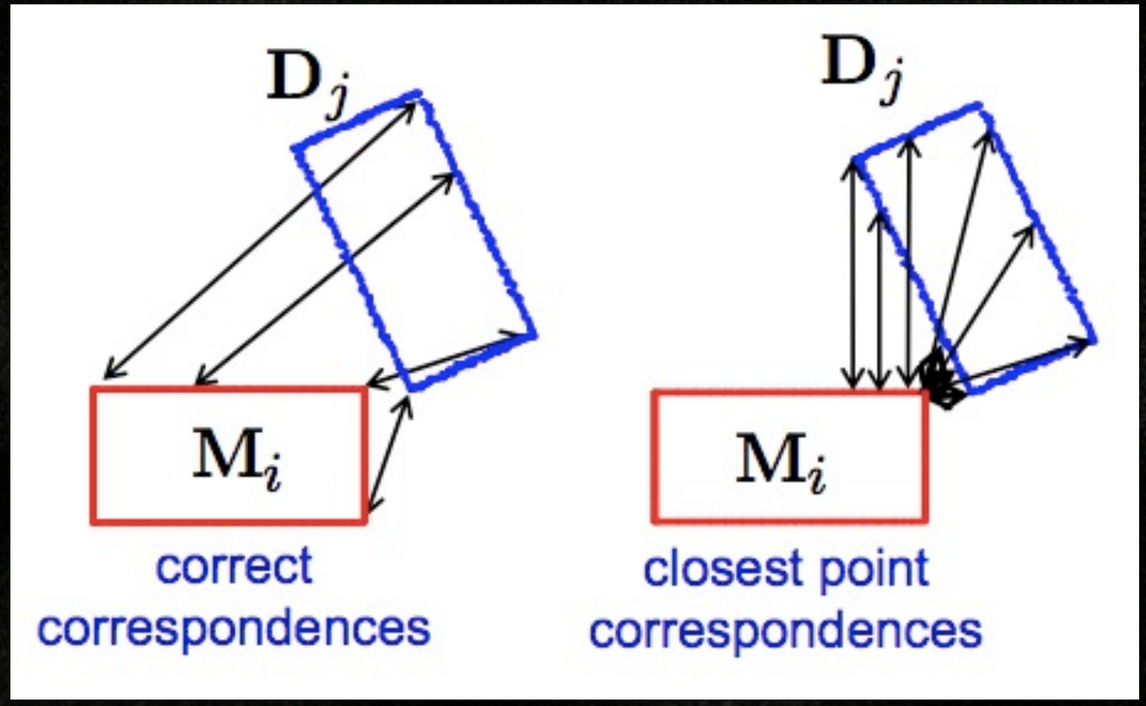
Model point: $M_i = (x_i, y_i)^T$

Transformation parameters:

rotation θ

translation $\mathbf{t} = (t_x, t_y)^T$

State vector: $\mathbf{s} = \{\theta, t_x, t_y\}$



Linear vs. non-linear cost functions

- Cost function linear wrt. its parameters:

$$f(t_x, t_y) = \sum_j \min_i \|M_i + \mathbf{t} - \mathbf{D}_j\|^2$$

- Cost function is non-linear in respect to its parameters:

$$f(\theta, t_x, t_y) = \sum_j \min_i \|\mathbf{R}(\theta) M_i + \mathbf{t} - \mathbf{D}_j\|^2$$

- Where the non-linearity comes from?

Cost function and least squares

- Cost function represents a sum of squared differences between the model and the data:

$$f(\mathbf{s}, D) = \sum_j f_j(\mathbf{s}, D_j)^2$$

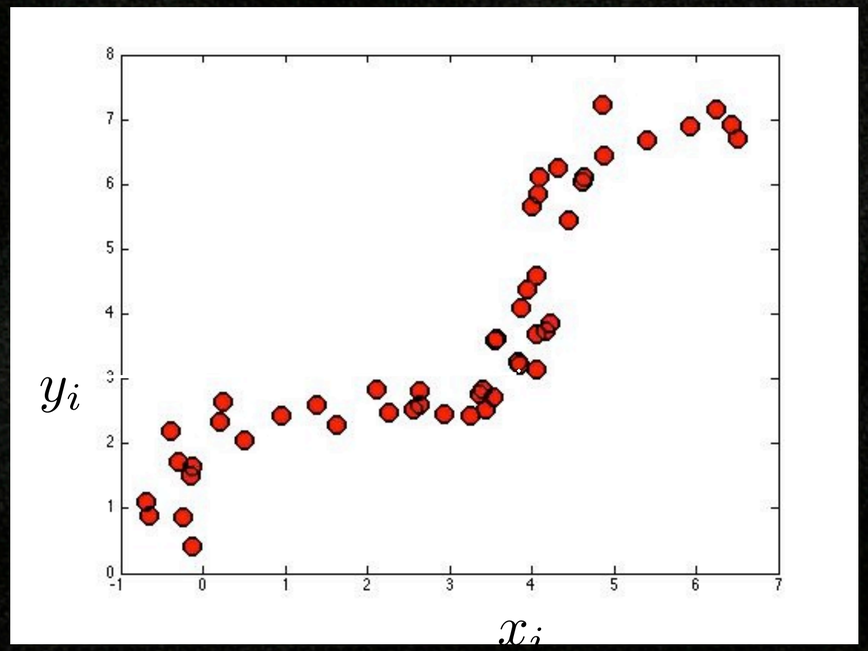
- can be represented as a system of equations:

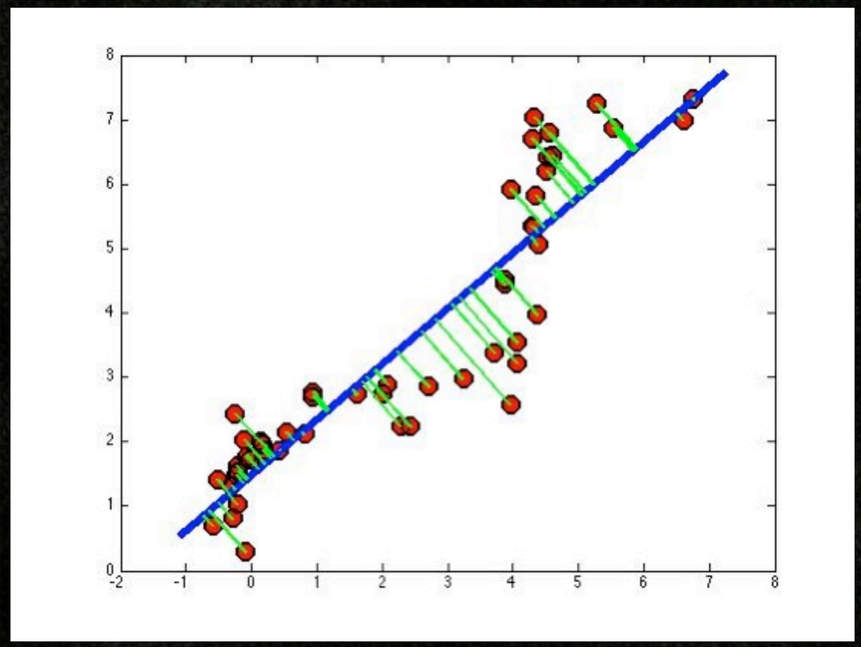
$$f_1(\mathbf{s}, D_1) = 0$$
$$f_2(\mathbf{s}, D_2) = 0$$
$$\vdots$$
$$f_N(\mathbf{s}, D_N) = 0$$

- Given system of equations $f(\mathbf{s}, D)$ has a solution $\mathbf{s}^*$ precisely when the above function has a minimal value:

$$\frac{\partial f}{\partial s_i} = \sum_j^N 2 f_j(s_1, s_2, ..., s_P, D) \frac{\partial f_i}{\partial s_i} = 0$$

Linear least squares





$$y_i = f(x_i, \mathbf{s})$$

$$y_i = ax_i + b$$

$$f(x_i, \mathbf{s}) = ax_i + b, \mathbf{s} = [a, b]^T$$

Linear least squares
Problem definition and notation

- Given the data pairs (x_i, y_i) :
  - x_i are independent data points (points at which the measurements are taken or observed)
  - y_i are dependent(measured or observed) data points
  - $f(x, \mathbf{s})$ is a function parameterized by $\mathbf{s}$, which we use to approximate the observations (x_i, y_i)
  - $\mathbf{s}$ is a state vector (parameter vector)
- The objective is to estimate the parameters of the function f by minimizing:

$$E(\mathbf{x}, \mathbf{s}) = \frac{1}{2} \sum_{i=1}^{N} (y_i - f(x_i, \mathbf{s}))^2$$

Linear least square (linear regression)

- Find the minimum of the energy(cost) function wrt. the parameters of the state vector s=[a,b]:

$$\mathbf{s}^* = \arg \min_{\mathbf{s}=[a,b]^T} \sum_i (y_i - f(x_i, \mathbf{s}))^2, \quad i = 1, N$$

- Minimum of this function minimizes the least square distance between the model and the data points.

$$\left.\begin{aligned} \frac{\partial E}{\partial a} &= -2 \sum_{j=1}^{N} (y_i - a x_i - b) x_i = 0 \\ \frac{\partial E}{\partial b} &= -2 \sum_{j=1}^{N} (y_i - a x_i - b) = 0 \end{aligned}\right\} \quad \begin{aligned} a \sum_{i=1}^{N} x_i^2 + b \sum_{i=1}^{N} x_i &= \sum_{i=1}^{N} y_i x_i \\ a \sum_{i=1}^{N} x_i + bN &= \sum_{i=1}^{N} y_i \end{aligned}$$

Derivation of the normal equation

- From the previous slide, the linear system in a matrix form is:

$$\underbrace{\begin{bmatrix} \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & N \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{\mathbf{s}} = \underbrace{\begin{bmatrix} \sum_{i=1}^{N} y_i x_i \\ \sum_{i=1}^{N} yi \end{bmatrix}}_{d}$$

- Minimizing the cost(energy) function is the same as solving the overdetermined system of linear equations(note that in case of line fitting above we do not solve over determined system of equations. However, we do this in case of homography estimation using DLT algorithm): $_N$

$$\arg\min_{a,b} \sum_{i=1} (y_i - ax_i - b) \Longrightarrow A\mathbf{s} = \mathbf{d}$$

$$A^T A\mathbf{s} = A^T \mathbf{d}$$

$$\mathbf{s} = (A^T A)^{-1} A^T \mathbf{d}$$

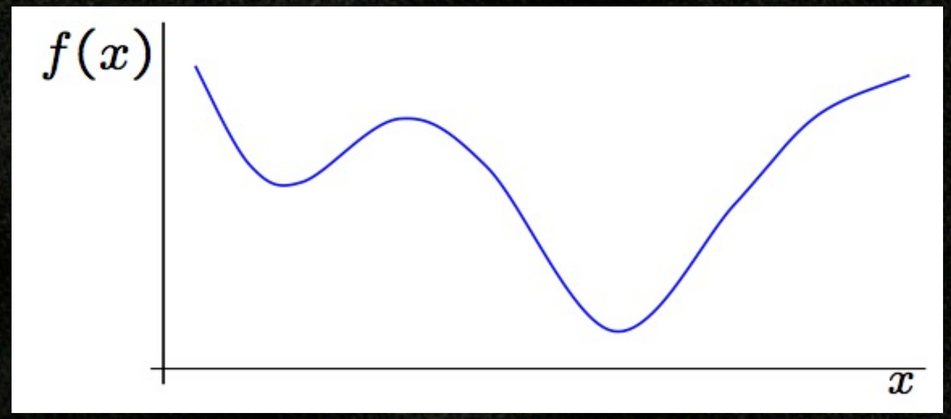
$$\mathbf{s} = A^{\dagger}\mathbf{d}, \quad A^{\dagger} = (A^T A)^{-1} A^T$$

- where $A^{\dagger}$ is Moore-Penrose pseudo inverse matrix, which has to be positive definite.

Iterative methods
Non-linear least squares

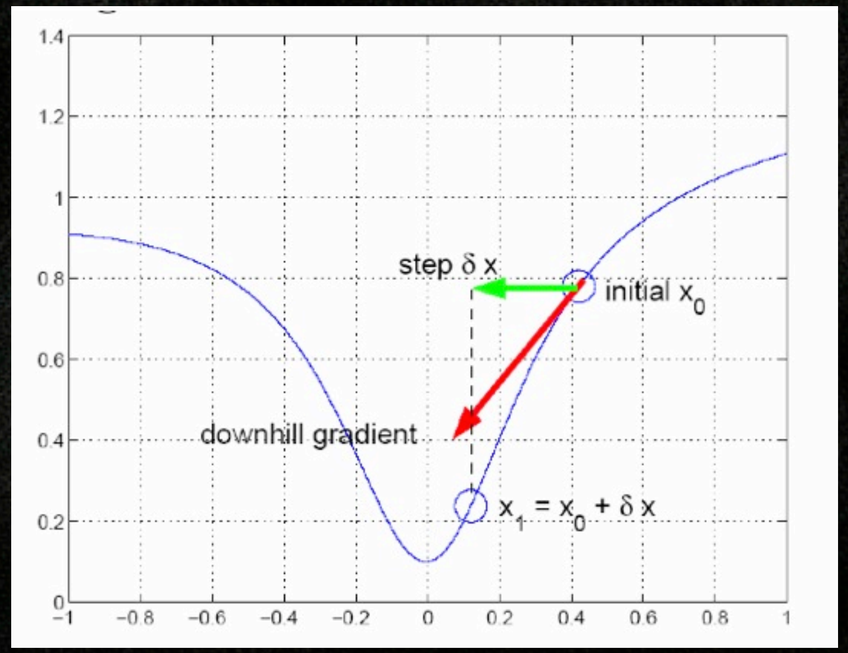
- We consider for the moment that we are not far from the global minimum.

$$\min_x f(x)$$



- We will consider three types of methods:
  - gradient descent methods
  - polynomial interpolation and
  - Newton methods

Gradient descent



$$x_1 = x_0 + \delta x$$

How do we determine the step size δx ?

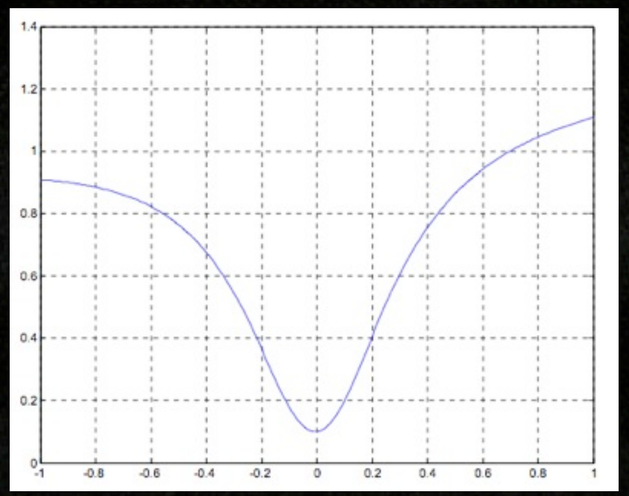
Given the initial solution move downhill in the direction opposite to the gradient, which cause the decrease of the function value at the next point:

$$x_1 = x_0 + \delta x = x_0 - \alpha \frac{df}{dx}$$

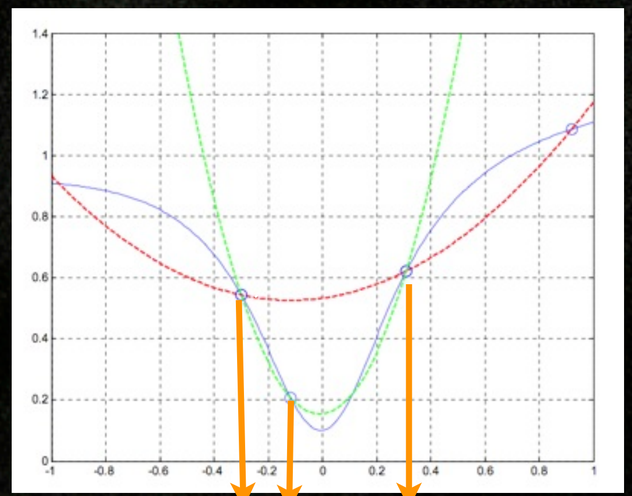
Polynomial interpolation

Algorithm

- Bracket the minimum between two given points
- Fit a quadratic or cubic polynomial which interpolates f(x) at some points in the interval.
- Jump to the (easily obtained) minimum of the polynomial.
- Throw away the worst point and repeat the process



$f(x) = 0.1 + 0.1x + x^2/(0.1 + x^2)$



Newton's methods

- Fit a quadratic approximation to f(x) using both gradient and curvature information at x.
- Expand f(x) locally using Taylor series:

$$f(x+\delta x) = f(x) + \delta x f^{'}(x) + \frac{\delta x^2}{2} f^{"}(x) + h.o.t$$

- Find the δx which minimizes this local quadratic approximation.

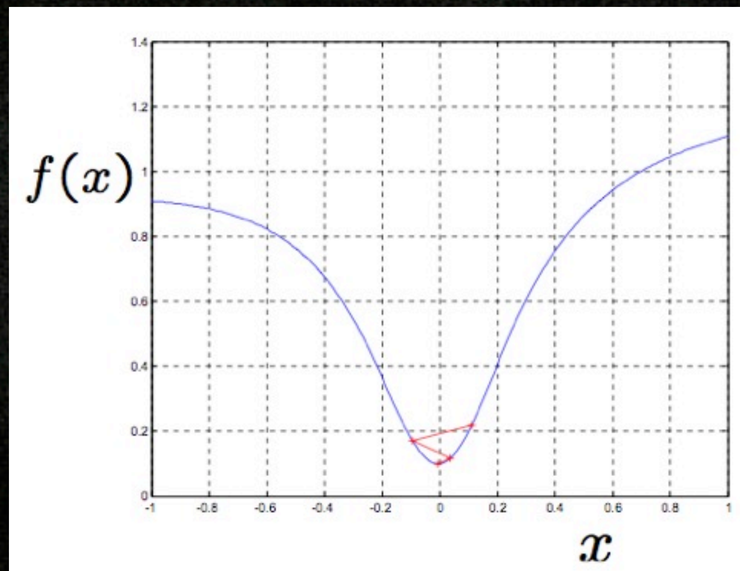
$$\delta x = -\frac{f^{'}(x)}{f^{"}(x)}$$

- Update x

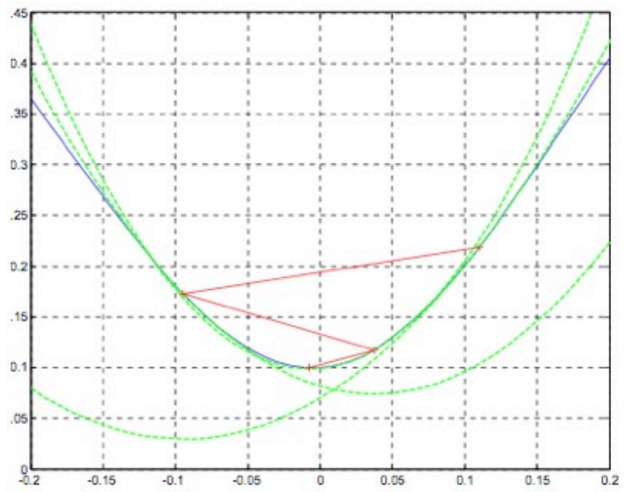
$$x_{n+1} = x_n - \frac{f^{'}(x)}{f^{"}(x)}$$

Newton's methods

Newton iterations



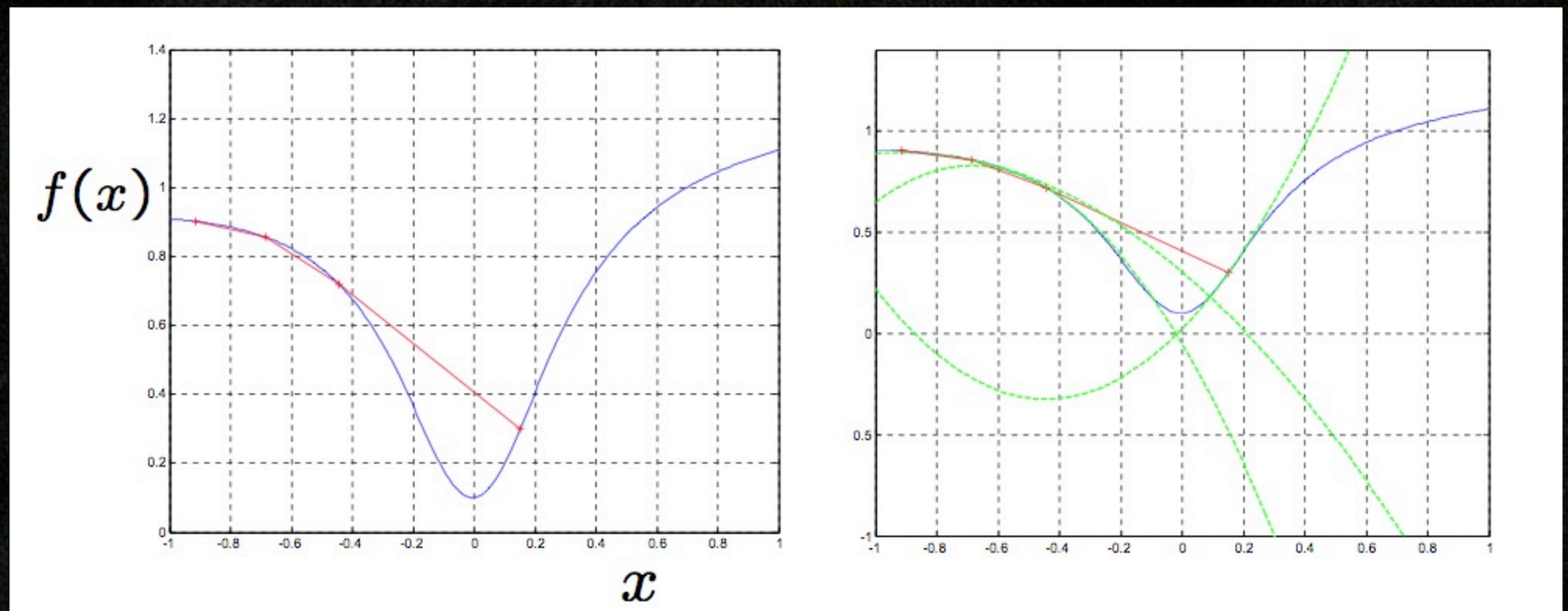
Quadratic approximation



- Avoids the need to bracket the root
- Quadratic convergence (decimal accuracy doubles at every iteration)

Newton's methods

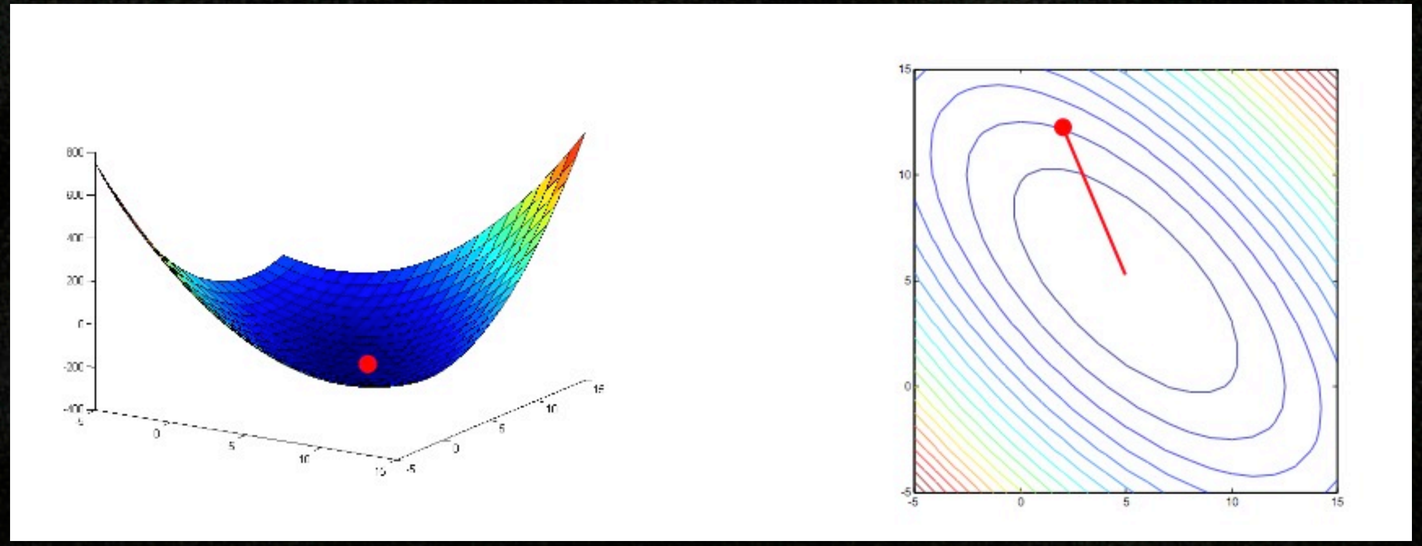
- Global convergence of the Newton's methods is poor
- Often fails if the starting point is far from the minimum



- in practice it must be used with the adaptive step length until the function decrees in not assured

Extension to N dimensions

- Problem size can vary from several to thousands of parameters
- The 2D example will be used in the examples for easier visualization:



An optimization algorithm

- Start with initial point x_0 and iterate:
  - compote a search direction p_k
  - compute a step length α_k, such that $f(x_k + \alpha_k p_k) < f(x_k)$
  - update $x_{k+1} = x_k + \alpha_k p_k$
  - check the convergence $\nabla f = 0$

Taylor expansion

- Function can be approximated locally around a give point x0 with its Taylor expansion series:

$$f(x_0 + x) \approx f(x_0) + \nabla f(x_0)^T x + \frac{1}{2} x^T H(x_0) x + h.o.t.$$

- where the gradient is:

$$\nabla f(x_0) = \left[\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_N} \right]$$

- and the Hessian H(x) is:

$$H(x_0) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_N} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}$$

- The Taylor expansion to second order is quadratic function:

$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^\mathbf{T}\mathbf{x} + \frac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^\mathbf{T}\mathbf{H}\mathbf{x}$$

Properties of quadratic functions

- Taylor expansion:

$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^{\mathbf{T}}\mathbf{x} + \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x}$$

- Expand about stationary point $x_0 = x^*$ in direction p

$$\begin{aligned} f(\mathbf{x}^* + \alpha\mathbf{p}) &= f(\mathbf{x}^*) + \mathbf{g}^T\alpha\mathbf{p} + \tfrac{1}{2}\alpha^2\mathbf{p}^T H\mathbf{p} \\ &= f(\mathbf{x}^*) + \tfrac{1}{2}\alpha^2\mathbf{p}^T H\mathbf{p} \end{aligned}$$

- since at the stationary point $\mathbf{g} = \nabla f|_{\mathbf{x}^*} = 0$
- At stationary point behavior is determined by H.

- H is a symmetric matrix and has orthogonal eigenvectors

$$H\mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad , \|\mathbf{u}_i\| = 1$$

$$f(\mathbf{x}^* + \alpha \mathbf{u}_i) = f(\mathbf{x}^*) + \tfrac{1}{2}\alpha^2 \mathbf{u}_i^T H \mathbf{u}_i$$

$$= f(\mathbf{x}^*) + \tfrac{1}{2}\alpha^2 \lambda_i$$

- As $|\alpha|$ is increasing than $f(\mathbf{x}^* + \alpha \mathbf{u}_i)$ is increasing, decreasing or is not changing according to whether λ_i is positive, negative or zero.

- Suppose x is 2D and $\|p\| = 1$, then:

$$\mathbf{p}\mathbf{u_1} = \cos\theta \ \ \textbf{and} \ \ \mathbf{p} = \cos\theta \mathbf{u_1} + \sin\theta \mathbf{u_2}$$

$$f(\mathbf{x}^* + \alpha \mathbf{p}) = f(\mathbf{x}) + \tfrac{1}{2}\alpha^2 (\cos\theta \mathbf{u}_1 + \sin\theta \mathbf{u}_2)^T H (\cos\theta \mathbf{u}_1 + \sin\theta \mathbf{u}_2)$$

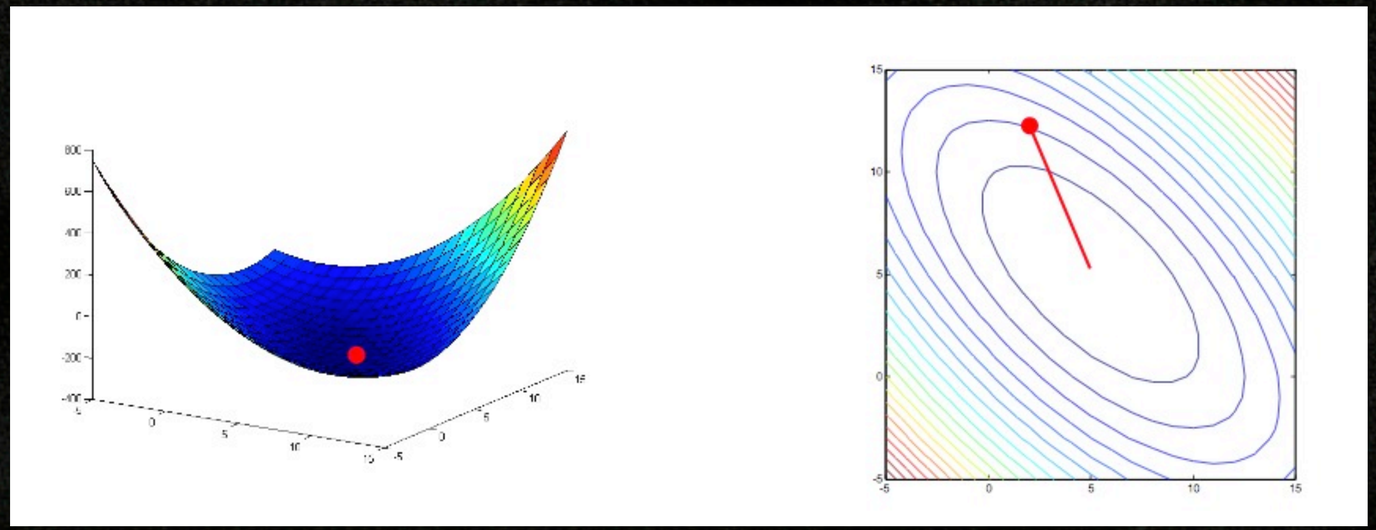
$$= f(\mathbf{x}^*) + \tfrac{1}{2}\alpha^2 (\cos^2\theta \lambda_1 + \sin^2\theta \lambda_2)$$

- If λ_1 and λ_2 are both positive then $\mathbf{x}^*$ is a minimum.

Examples of quadratic function
Case 1: Eigenvalues are both positive

$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^{\mathbf{T}}\mathbf{x} + \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x}$$

$$a = 0, \quad g = \begin{bmatrix} -50 \\ -50 \end{bmatrix}, \quad H = \begin{bmatrix} 6 & 4 \\ -4 & 6 \end{bmatrix}$$

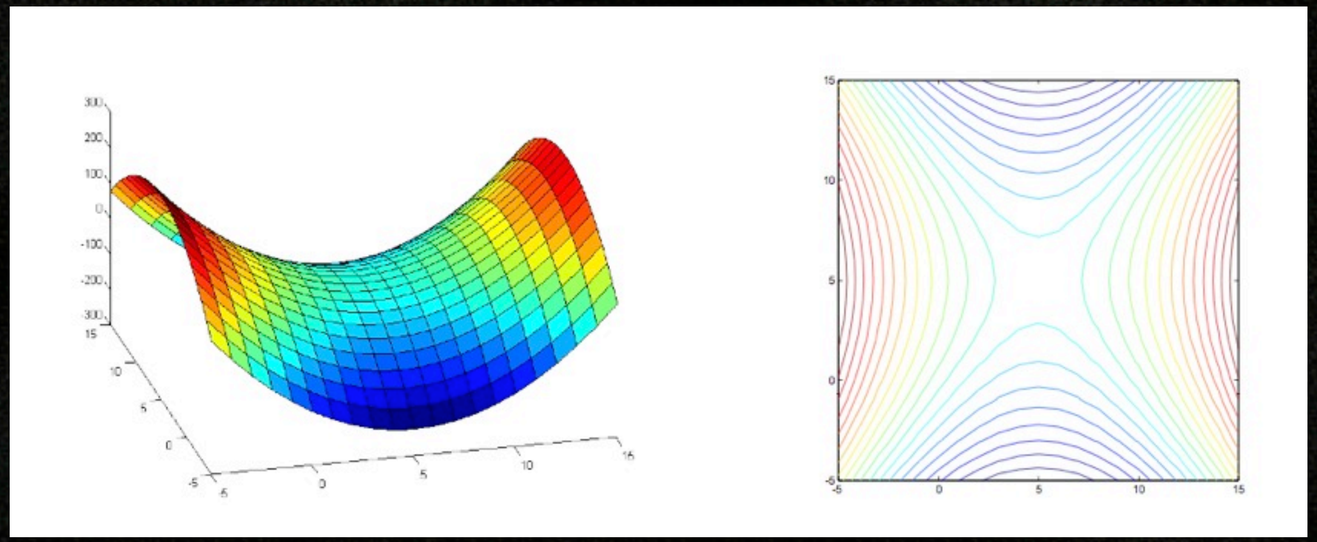


Both eigenvalues are positive, so it has unique minimum.

Case 2: Eigenvalues have different signs

$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^{\mathbf{T}}\mathbf{x} + \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x}$$

$$a = 0, \quad g = \begin{bmatrix} -30 \\ 20 \end{bmatrix}, \quad H = \begin{bmatrix} 6 & 0 \\ 0 & -4 \end{bmatrix}$$

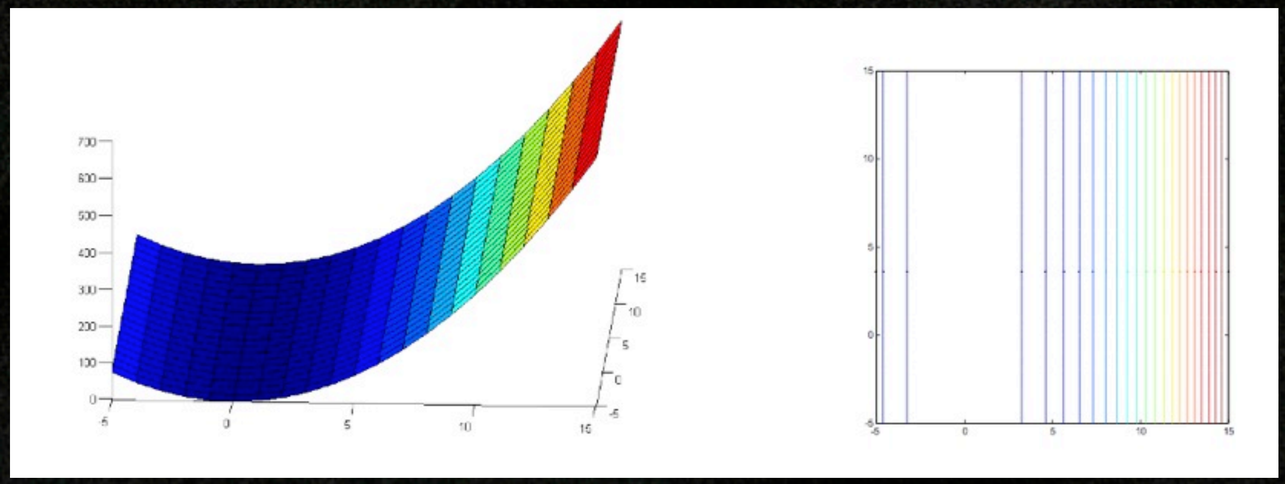


Saddle surface. There is extrema, but it is not a minimum.

Case 3: Eigenvalues are zero

$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^{\mathbf{T}}\mathbf{x} + \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x}$$

$$a = 0, \quad g = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad H = \begin{bmatrix} 6 & 0 \\ 0 & 0 \end{bmatrix}$$



The minimum is in the infinite valley, i.e the shape of the objective function is parabolic cylinder

Steepest-descent

- Basic principle is to minimize the N-dimensional function by a series of 1D line minimizations:

$$x_{n+1} = x_n + \alpha_n p_n$$

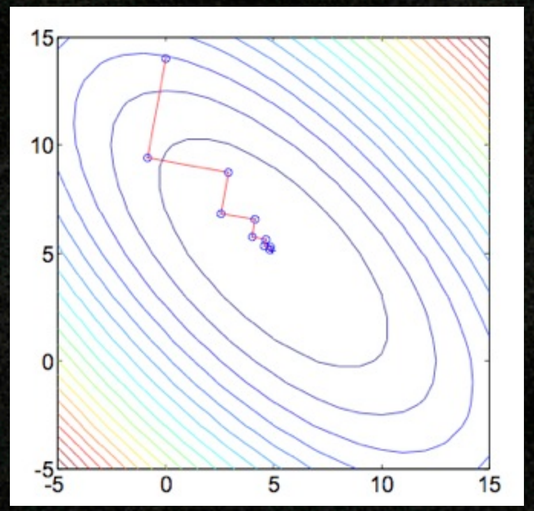
- The steepest descent method chooses pn to be parallel to the negative gradient: $p_n = -\nabla f(x_n)$

- Step-size α_n is chosen to minimize $f(x_n + \alpha_n p_n)$. For quadratic forms there is a closed form solution: $\alpha_n = \frac{p_n^T p_n}{p_n^T H p_n}$

Derivation of α_n : $f(x_n + \alpha_n p_n) = f(x_n) + \nabla f \alpha_n p_n + \frac{1}{2}\alpha_n^2 p_n^T H p_n$

$$\frac{\partial f}{\alpha_n} = \nabla f p_n + \alpha_n p_n^T H p_n = 0 \rightarrow -p_n^T p_n + \alpha_n p_n^T H p_n = 0$$

Steepest descent - Example



$$f(\mathbf{x}) = \mathbf{a} + \mathbf{g}^{\mathbf{T}}\mathbf{x} + \tfrac{\mathbf{1}}{\mathbf{2}}\mathbf{x}^{\mathbf{T}}\mathbf{H}\mathbf{x}$$

$$a = 0, \quad g = \begin{bmatrix} -50 \\ -50 \end{bmatrix}, \quad H = \begin{bmatrix} 6 & 4 \\ -4 & 6 \end{bmatrix}$$

- Gradients are perpendicular to the contour lines
- The gradients of two successful steps are orthogonal, and the
- Slowly converges to the minimum in a zig-zag manner

$$\frac{\partial f(x_{k+1})}{\partial \alpha_k} = \frac{\partial f}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial \alpha_k} = 0 \rightarrow \nabla f(x_{k+1}) \nabla f(x_k) = 0$$

Conjugate Gradient

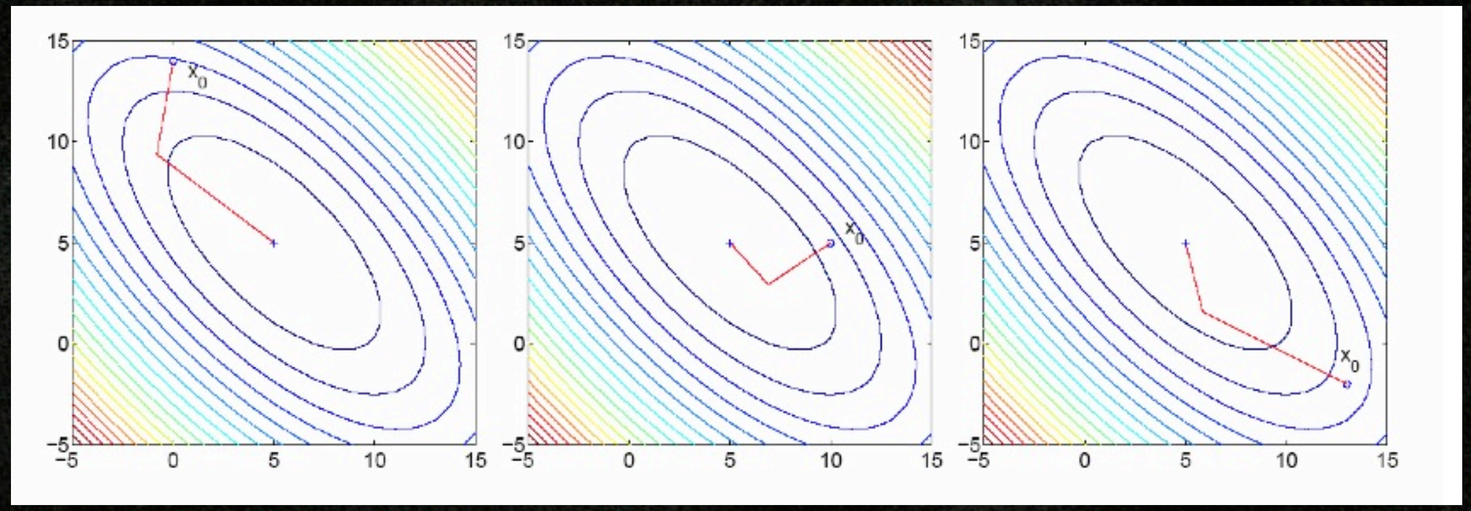
- The method of conjugate gradients chooses successive descent directions pn such that it is guaranteed to reach the minimum in a finite number of steps.
- Each pn is chosen to be conjugate to all previous search directions with respect to the Hessian H:

$$p_n^T H p_n = 0, 0 \leq j < n$$

- The resulting search directions are mutually linearly independent.
- Remarkably, pn can be chosen using only knowledge of pn, **∇**f(xn–1) and **∇**f(xn) (see Numerical Recipes)

$$p_n = \nabla f_n + \left(\frac{\nabla f_n^T \nabla f_n}{\nabla f_{n-1}^T \nabla f_{n-1}} \right) p_{n-1}$$

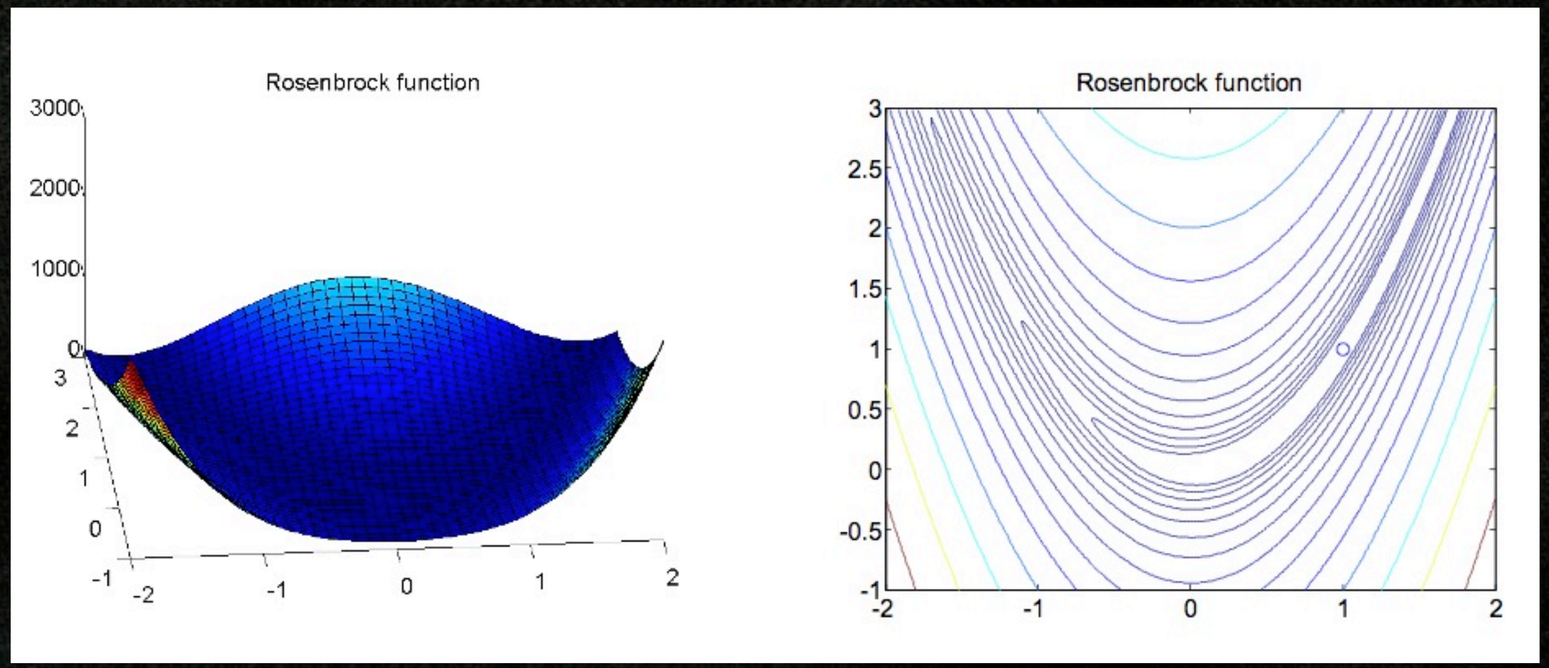
Conjugate gradient
Example



- An N-dimensional quadratic form can be minimized in at most N conjugate descent steps.
- 3 different starting points.
- Minimum is reached in exactly 2 steps.

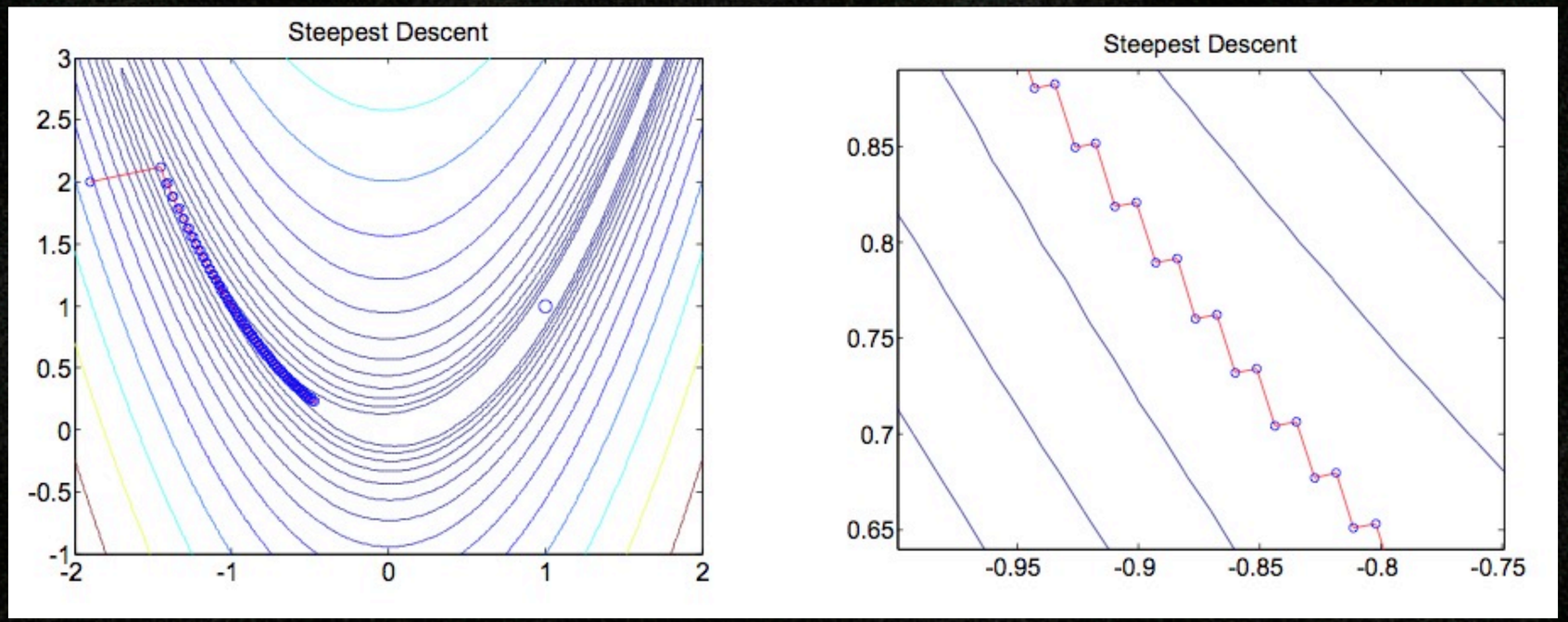
Rosenbrock's function

$$f(x, y) = 100(y - x^2)^2 + (1 - x)^2$$



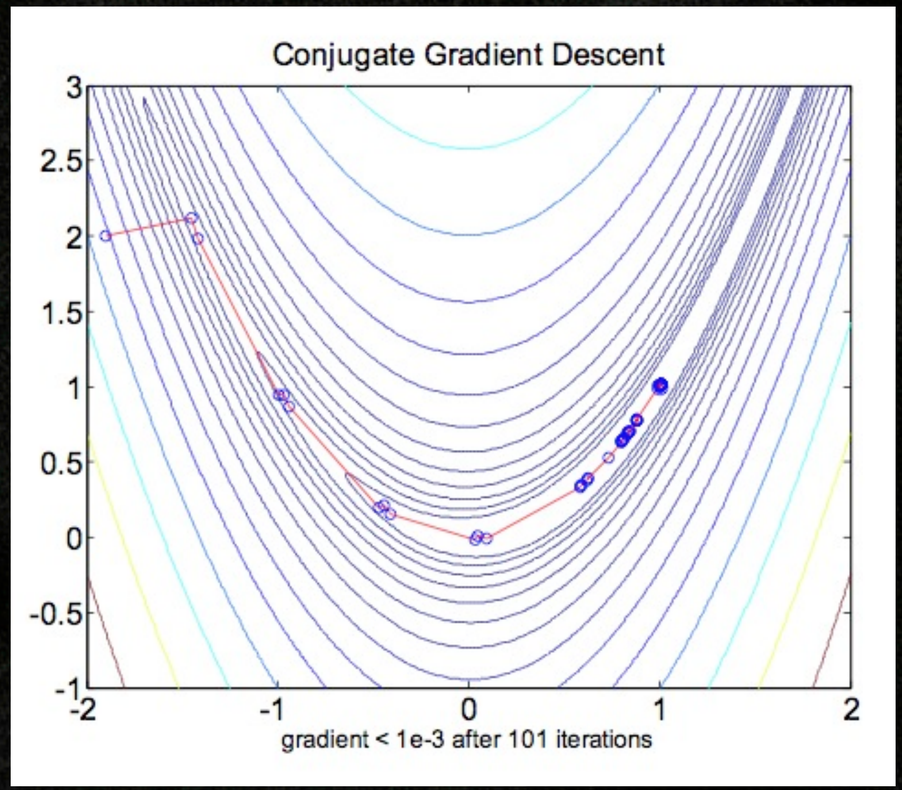
Minimum of this function is in (1,1)

Minimum of Roesnbrock's function with steepest descend



It does not converge after 100 iterations
It crawls very slowly down the valley

Minimum of Roesnbrock's function with conjugate gradients



It converged after 101 iterations
It is much faster then steepest descend

Newton methods

- Function can be approximated locally around a give point xn being a solution at n-th iteration with its Taylor expansion series:

$$f(x_n + \delta x) \approx f(\mathbf{x}_n) + g_n^T \delta\mathbf{x} + \tfrac{1}{2}\delta\mathbf{x}^T H \delta\mathbf{x} + h.o.t.$$

- where the gradient is: $\mathbf{g}_n = \nabla f(\mathbf{x}_n) = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_N}\right]$

- and the Hessian H(x) is:

$$H_n = H(\mathbf{x}_n) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_N} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}$$

- For minimum we require that $\nabla f(\mathbf{x}) = 0$

$$\nabla f(\mathbf{x}) = \mathbf{g}_n + H_n \delta\mathbf{x} = 0 \qquad \delta\mathbf{x} = -H_n^{-1}\mathbf{g}_n$$

- finally the Newton update step is:

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \mathbf{H_n^{-1}}\mathbf{g_n}$$

Newton methods

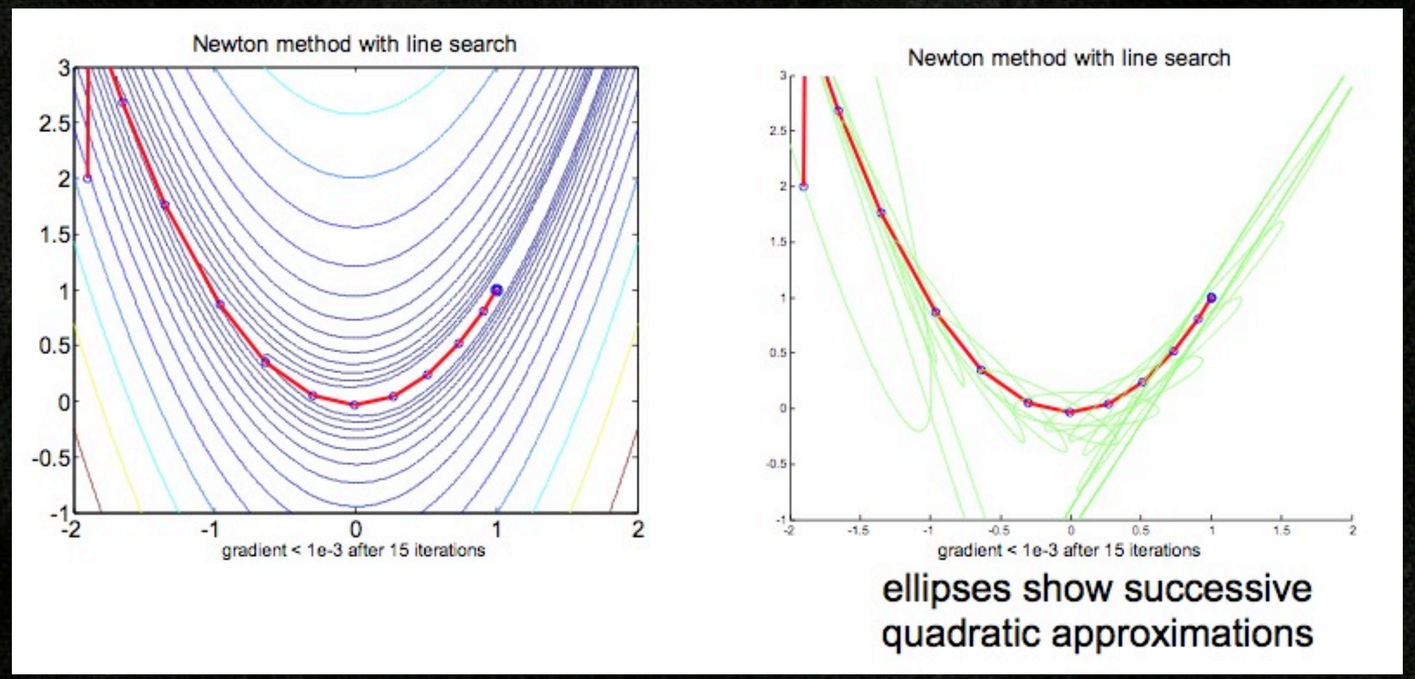
- If H is positive definite (all eigenvalues are greater then zero) we have solution at one iteration step:

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \mathbf{H_n^{-1}} \mathbf{g_n}$$

  - If f(x) is quadratic, then the solution is found in one step.
  - The method has quadratic convergence(as in the 1D case).
  - The solution $\delta \mathbf{x} = -H_n^{-1} \mathbf{g}_n$ is guaranteed to be a down hill direction.
  - Rather than jump straight to the minimum, it is better to perform a line minimization which ensures global convergence:

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \alpha_\mathbf{n} \mathbf{H_n^{-1}} \mathbf{g_n}$$

Newton method - Example



The algorithm converges in only 15 iterations compared to the 101 for conjugate gradients

However, the method requires computing the Hessian matrix at each iteration –this is not always feasible

Non-linear least squares

- It is very common in applications for a cost function f(x) to be the sum of a large number of squared residuals:

$$f(\mathbf{x}) = \sum_{i=1}^{M} r_i^2$$

- If each residual depends non-linearly on the parameters x then the minimization of f(x) is a non -linear least squares problem.
- Examples in computer vision are maximum likelihood estimators of image relations (such as homographies and fundamental matrices)

Non-linear least squares

$$f(\mathbf{x}) = \sum_{i=1}^{M} r_i^2$$

- For the non-linear least squares Jacobian matrix is:

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial r_M}{\partial x_1} & \cdots & \frac{\partial r_M}{\partial x_N} \end{pmatrix}$$

- Consider

$$\frac{\partial}{\partial x_k} \sum_{i=1} r_i^2 = \sum_i 2 r_i \frac{\partial r_i}{\partial x_k}$$

- Hence:

$$\nabla f(\mathbf{x}) = 2 J^T \mathbf{r}$$

Non-linear least squares

- For the Hessian we require:

$$\frac{\partial^2}{\partial x_l \partial x_k} \sum_i r_i^2 = 2 \frac{\partial}{\partial x_l} \sum_i r_i \frac{\partial r_i}{\partial x_k}$$

$$= 2 \sum_i \frac{\partial r_i}{\partial x_k} \frac{\partial r_i}{\partial x_l} + 2 \sum_i r_i \frac{\partial^2 r_i}{\partial x_k \partial x_l}$$

- Hence:

$$H(x) = 2 J^T J + 2 \sum_{i=1}^{M} r_i R_i$$

Non-linear least squares

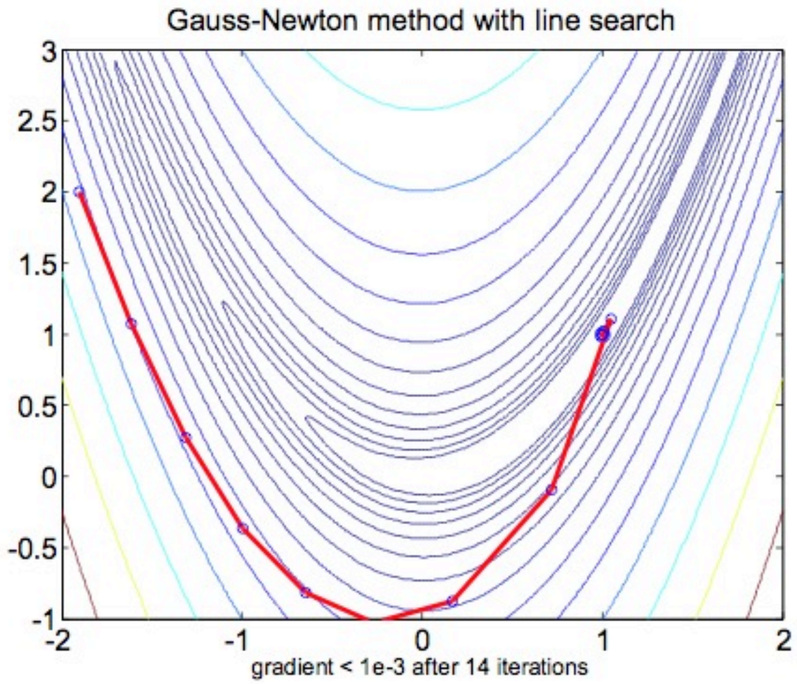
- Note that the second-order term in the Hessian H (x) is multiplied by the residuals ri .
- In most problems, the residuals will typically be small.
- Also, at the minimum, the residuals will typically be distributed with mean = 0.
- For these reasons, the second-order term is often ignored, giving the Gauss-Newton approximation to the Hessian :

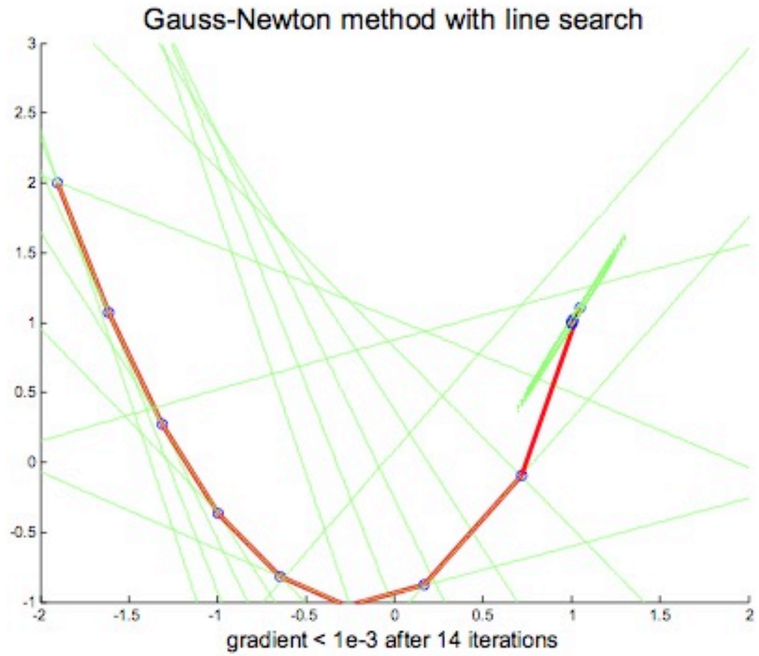
$$H(x) = 2J^T J$$

- Hence, explicit computation of the full Hessian can again be avoided.

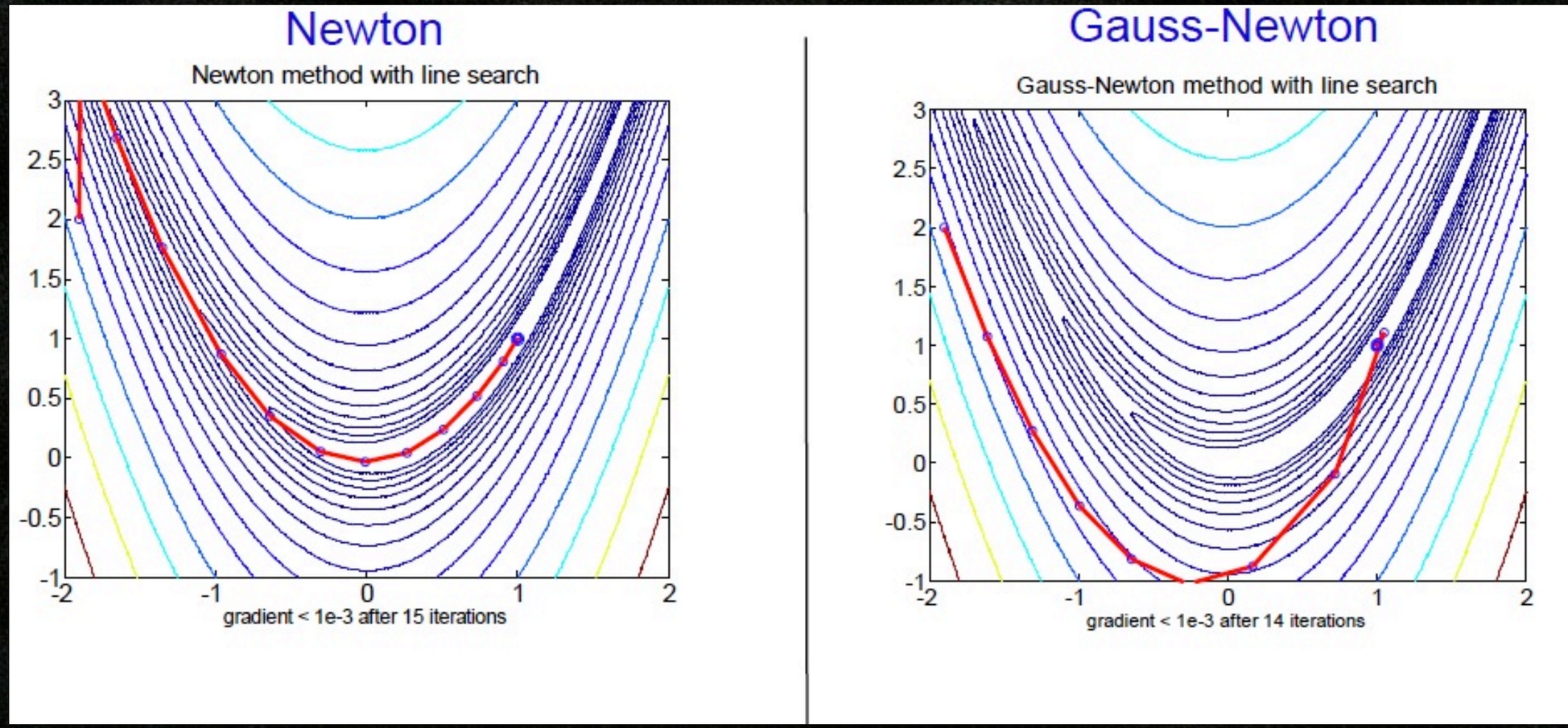
Gauss-Newton - Example

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha_n \mathrm{H}_n^{-1} \mathbf{g}_n \quad \text{with} \quad \mathrm{H}_n(\mathbf{x}) = 2\mathrm{J}_n^\top \mathrm{J}_n$$





Comparison



Requires computing Hessian
The exact solution is quadratic

Approximation of the Hessian
by a product of Jacobians
Requires only derivatives

Summary of the minimization methods

- Update

$$\mathbf{x_{n+1}} = \mathbf{x_n} + \delta\mathbf{x}$$

- Newton

$$H\delta\mathbf{x} = -\mathbf{g}$$

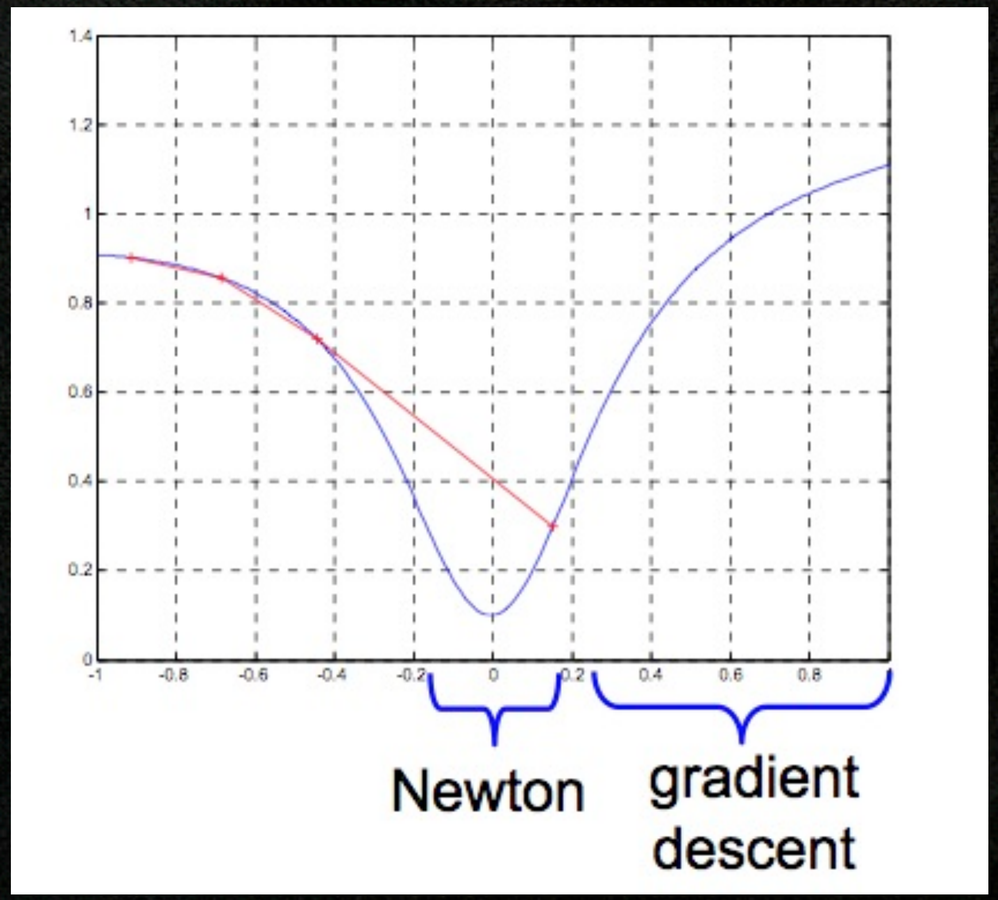
- Gauss-Newton

$$2J^T J\delta\mathbf{x} = -\mathbf{g}$$

- Gradient descent

$$\lambda\delta\mathbf{x} = -\mathbf{g}$$

Levenberg-Marquardt Algorithm



- Away from the minimum, in regions of negative curvature, the Gauss Newton approximation is not very good.
- In such regions, a simple steepest-descent step is probably the best plan.
- The Levenberg-Marquardt method is a mechanism for varying between steepest-descent and Gauss-Newton steps depending on how good the $J^T J$ approximation is locally.

Levenberg-Marquardt Algorithm

- The LM method uses modified Hessian

$$H(\mathbf{x}, \lambda) = 2J^T J + \lambda I$$

- When λ is small H approximates Gauss-Newton Hessian.
- When λ is large, H is close to the identity, causing steepest-descent step to be taken.

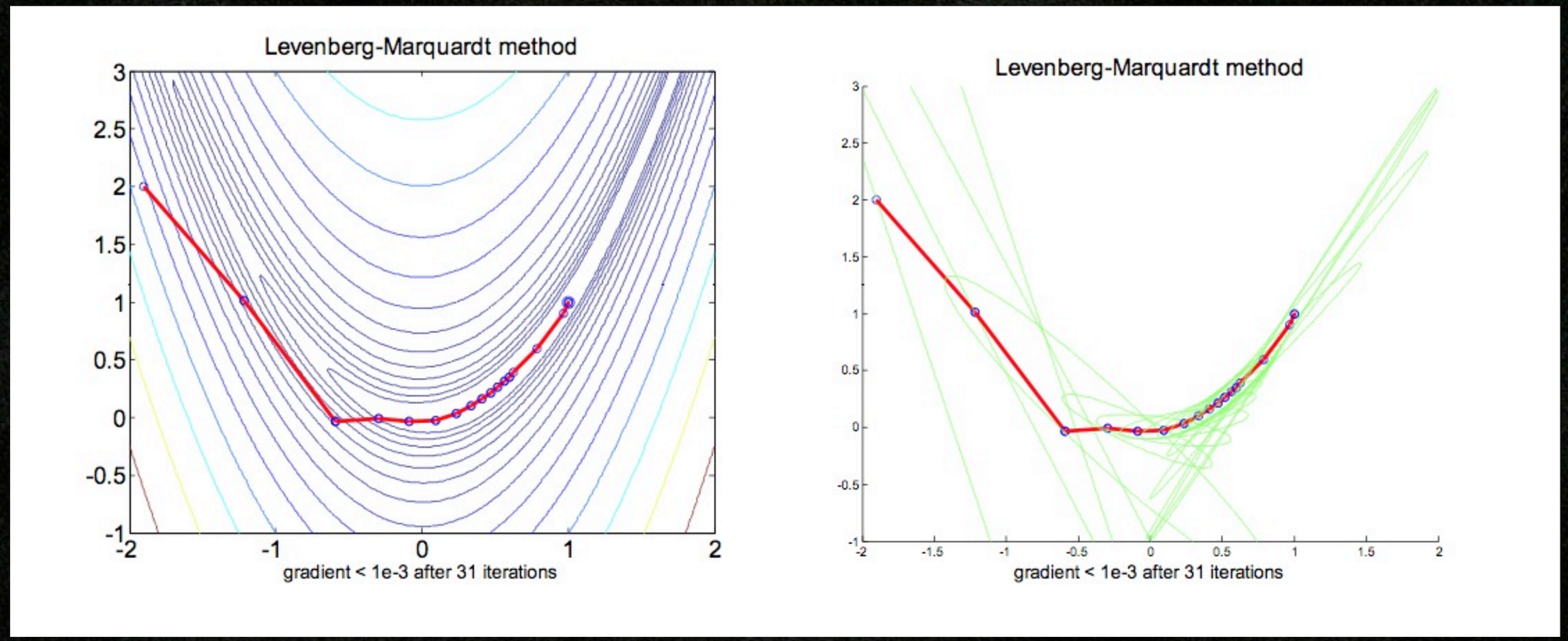
LM Algorithm outline

- Set $\lambda = 0.001$
- Solve $\delta\mathbf{x} = -H(\mathbf{x},\lambda)^{-1}\mathbf{g} \rightarrow H(\mathbf{x},\lambda)^T H(\mathbf{x},\lambda)\delta\mathbf{x} = -H(\mathbf{x},\lambda)^T\mathbf{g}$

  $H(\mathbf{x},\lambda) = 2J^T J + \lambda I$
- If $f(\mathbf{x}_n + \delta\mathbf{x}) > f(\mathbf{x}_n)$, increase $\lambda = 10 * \lambda$ and go to 2.
- Otherwise, decrease $\lambda = 0.1 * \lambda$, update parameters $\mathbf{x_{n+1}} = \mathbf{x_n} + \delta\mathbf{x}$ and jump to 2.

This algorithm does not require explicit line search.

LM Algorithm - Example



Minimization using LM algorithm with no line search took 31 iteration. What is more then Gauss-Newton, but no explicit line search is required and it converges more frequently.

Application - Bundle Adjustment

- Given n matching image points $\mathbf{x}_j$ over m views
- Find the cameras P_i and scene points $\mathbf{x}_j$

$$\arg\min_{P_i \mathbf{X}_j} \sum_{j \in points} \sum_{i \in views} d(\mathbf{x}_j^i, P_i \mathbf{X}_j)^2$$

Application - Structure From Motion

Towards Internet-scale
Multi-view Stereo

CVPR 2010

Yasutaka Furukawa[1] Brian Curless[2]

Steven M. Seitz[1,2] Richard Szeliski[3]

Google Inc.[1]
University of Washington[2]
Microsoft Research[3]