Optimization Methods

Categorization of Optimization Problems

- Continuous Optimization
- Discrete Optimization
- Combinatorial Optimization
- Variational Optimization

Common Optimization Concepts in Computer Vision

- Energy Minimization
- Graphs
- Markov Random Fields

Several general approaches to optimization are as follows:

- Analytical methods
- Graphical methods
- Experimental methods
- Numerical methods

Several branches of mathematical programming have evolved, as follows:

- Linear programming
- Integer programming
- Quadratic programming
- Nonlinear programming
- Dynamic programming

CONJUGATE-DIRECTION METHODS
6.1 Introduction
6.2 Conjugate Directions
6.3 Basic Conjugate-Directions Method
6.4 Conjugate-Gradient Method
6.5 Minimization of Nonquadratic Functions

QUASI-NEWTON METHODS
7.1 Introduction
7.2 The Basic Quasi-Newton Approach

MINIMAX METHODS
8.1 Introduction
8.2 Problem Formulation
8.3 Minimax Algorithms

APPLICATIONS OF UNCONSTRAINED OPTIMIZATION
9.1 Introduction
9.2 Point-Pattern Matching

FUNDAMENTALS OF CONSTRAINED OPTIMIZATION
10.1 Introduction
10.2 Constraints

LINEAR PROGRAMMING PART I: THE SIMPLEX METHOD
11.1 Introduction
11.2 General Properties
11.3 Simplex Method

LINEAR PROGRAMMING PART I: THE SIMPLEX METHOD
11.1 Introduction
11.2 General Properties
11.3 Simplex Method

QUADRATIC AND CONVEX PROGRAMMING
13.1 Introduction
13.2 Convex QP Problems with Equality Constraints
13.3 Active-Set Methods for Strictly Convex QP Problems

GENERAL NONLINEAR OPTIMIZATION PROBLEMS
15.1 Introduction
15.2 Sequential Quadratic Programming Methods

Problem specification

Suppose we have a cost function (or objective function)

$$f(\mathbf{x}) : \mathbb{R}^N \longrightarrow \mathbb{R}$$

Our aim is to find values of the parameters (decision variables) $\mathbf{x}$ that minimize this function

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

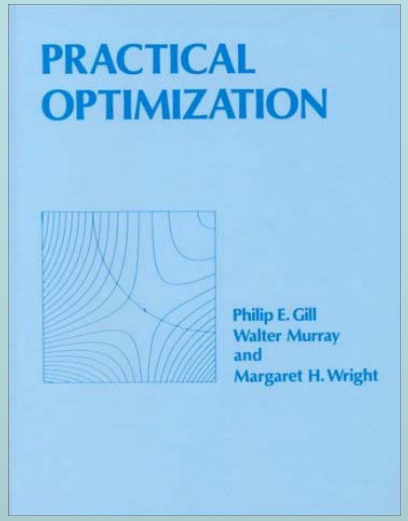
Subject to the following constraints:

- equality: $c_i(\mathbf{x}) = 0$
- nonequality: $c_j(\mathbf{x}) \geq 0$

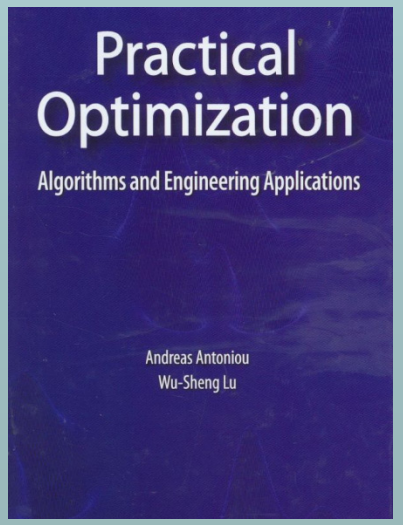
If we seek a maximum of $f(\mathbf{x})$ (profit function) it is equivalent to seeking a minimum of $-f(\mathbf{x})$

Books to read

- **Practical Optimization**
  - Philip E. Gill, Walter Murray, and Margaret H. Wright, Academic Press,
    1981



- **Practical Optimization: Algorithms and Engineering Applications**
  - Andreas Antoniou and Wu-Sheng Lu
    2007



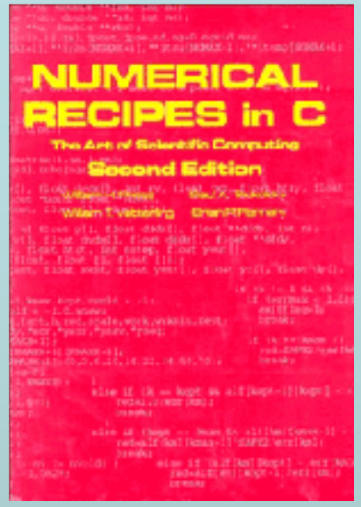
- Both cover unconstrained and constrained optimization. Very clear and comprehensive.

Further reading and web resources

- **Numerical Recipes in C (or C++) : The Art of Scientific Computing**
  - William H. Press, Brian P. Flannery, Saul A. Teukolsky, William T. Vetterling
  - Good chapter on optimization
  - Available on line at

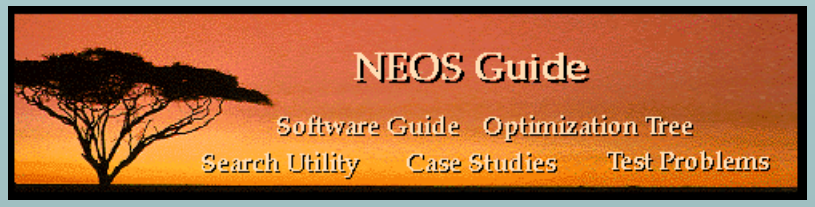
  (1992 ed.) www.nrbook.com/a/bookcpdf.php

  (2007 ed.) www.nrbook.com



- NEOS Guide

  www-fp.mcs.anl.gov/OTC/Guide/



- This powerpoint presentation

  www.utia.cas.cz

Introductory Items in OPTIMIZATION

- Category of Optimization methods
- Constrained vs Unconstrained
- Feasible Region

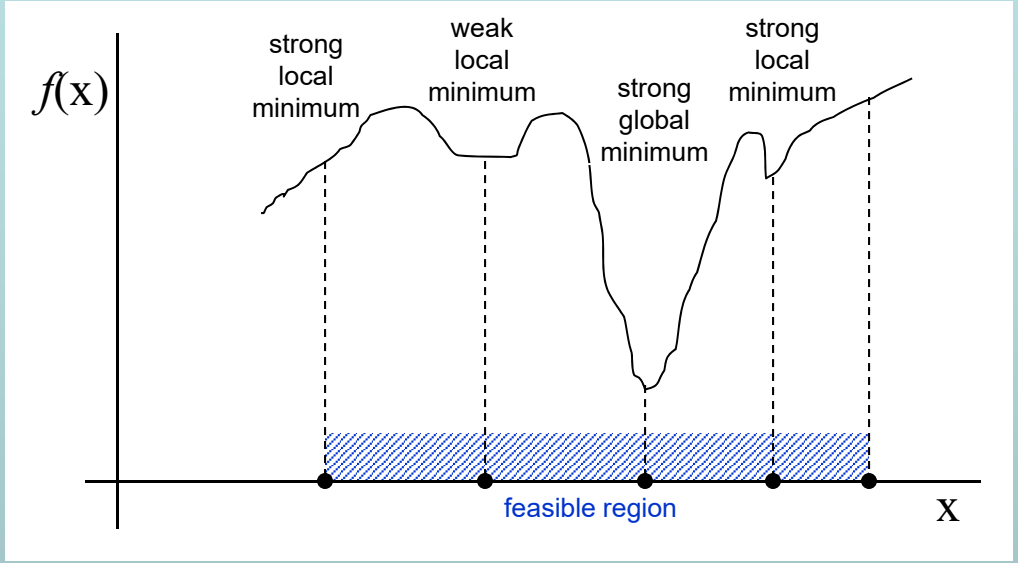
- Gradient and Taylor Series Expansion
- Necessary and Sufficient Conditions
- Saddle Point
- Convex/concave functions
- 1-D search – Dichotomous, Fibonacci – Golden Section, DSC;

- Steepest Descent; Newton; Gauss-Newton

- Conjugate Gradient;

- Quasi-Newton; Minimax;
- Lagrange Multiplier; Simplex; Prinal-Dual, Quadratic programming; Semi-definite;

Types of minima

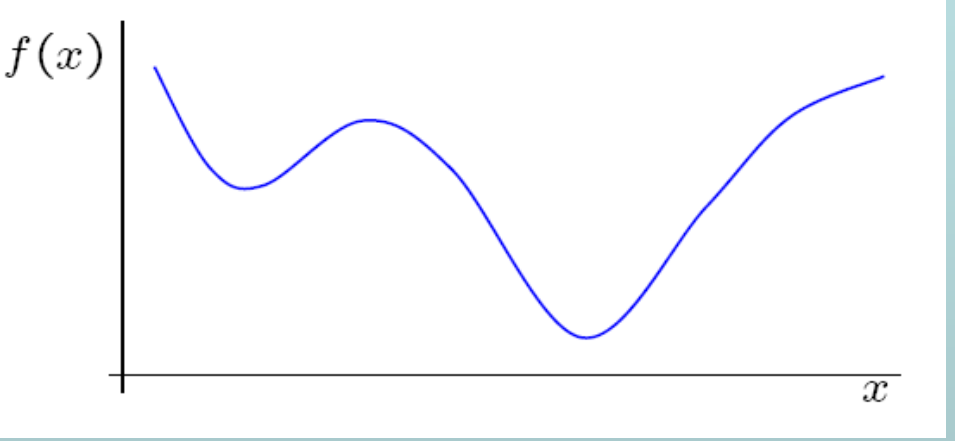


- which of the minima is found depends on the starting point
- such minima often occur in real applications

Unconstrained univariate optimization

Assume we can start close to the global minimum

$$\min_x f(x)$$

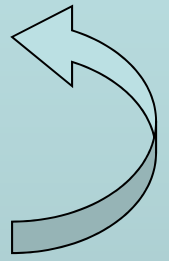


How to determine the minimum?

- Search methods (Dichotomous, Fibonacci, Golden-Section)
- Approximation methods
    1. Polynomial interpolation
    2. Newton method
- Combination of both (alg. of Davies, Swann, and Campey)

Search methods

- Start with the interval ("bracket") $[x_L, x_U]$ such that the minimum x* lies inside.
- Evaluate $f(x)$ at two point inside the bracket.
- Reduce the bracket.
- Repeat the process.



- Can be applied to any function and differentiability is not essential.

XL	XU	X1	Criteria
0	1	1/2	FA > FB

XL	XU	RANGE

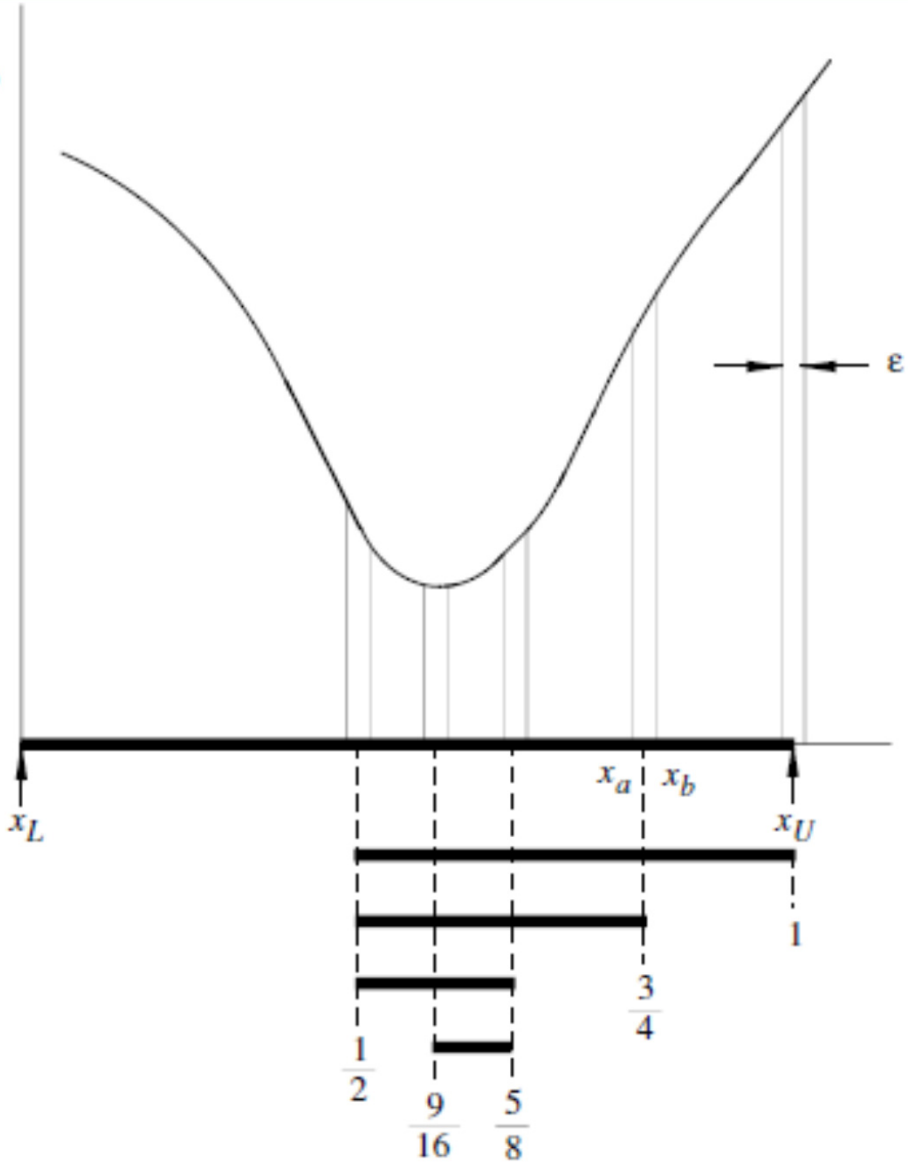
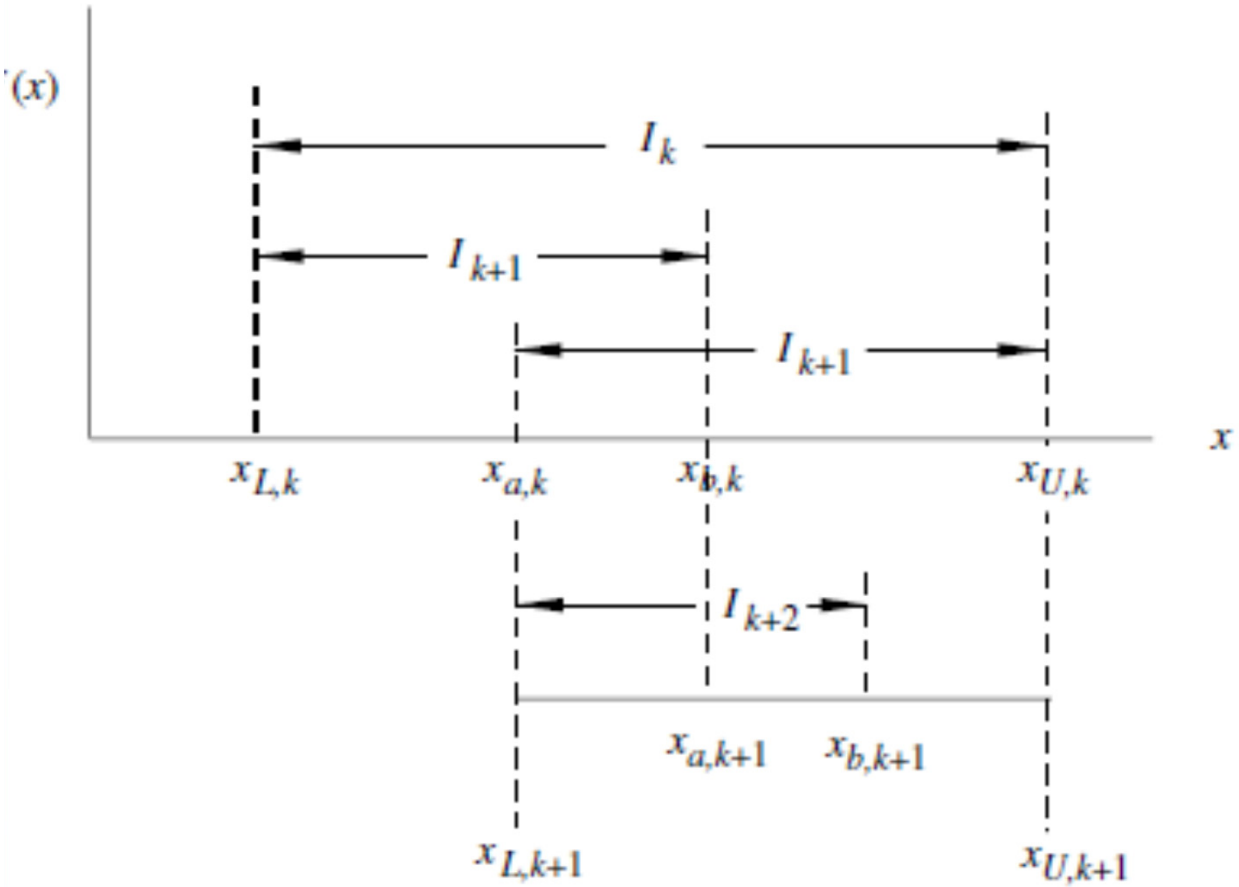


Figure 4.2. Construction for dichotomous search.



$$I_{k+2} = \frac{F_{n-k-1}}{F_{n-k}} I_{k+1}$$

If $f_{a,k} > f_{b,k}$, then x^* is in interval $[x_{a,k},\ x_{U,k}]$ and so the new bounds of x^* can be updated as

$$x_{L,k+1} = x_{a,k} \tag{4.7}$$

$$x_{U,k+1} = x_{U,k} \tag{4.8}$$

$$x_{a,k+1} = x_{b,k}$$

$$x_{b,k+1} = x_{L,k+1} + I_{k+2}$$

$$f_{a,k+1} = f_{b,k}$$

$$f_{b,k+1} = f(x_{b,k+1})$$

Algorithm 4.1 Fibonacci search

Step 1

Input $x_{L,1}$, $x_{U,1}$, and n.

Step 2

Compute $F_1, F_2, \ldots, F_n$ using Eq. (4.4).

Step 3

Assign $I_1 = x_{U,1} - x_{L,1}$ and compute

$$I_2 = \frac{F_{n-1}}{F_n} I_1 \quad \text{(see Eq. (4.6))}$$

$$x_{a,1} = x_{U,1} - I_2, \quad x_{b,1} = x_{L,1} + I_2$$

$$f_{a,1} = f(x_{a,1}), \quad f_{b,1} = f(x_{b,1})$$

Set $k = 1$.

Step 4

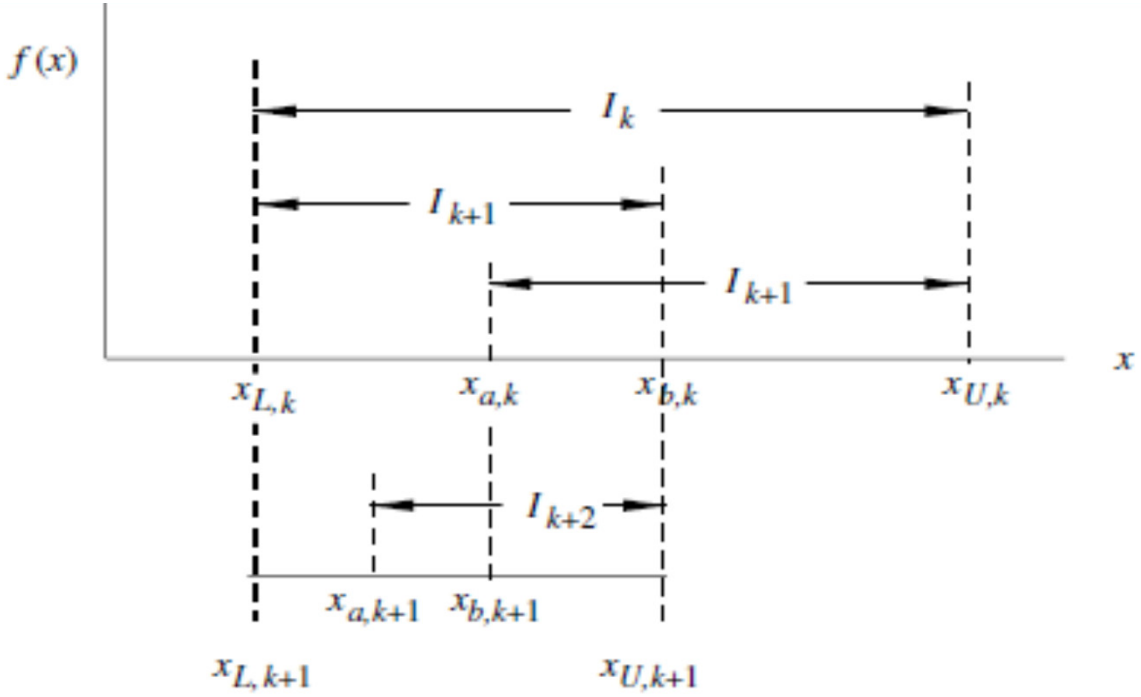
Compute I_{k+2} using Eq. (4.6).

If $f_{a,k} \geq f_{b,k}$, then update $x_{L,k+1}$, $x_{U,k+1}$, $x_{a,k+1}$, $x_{b,k+1}$, $f_{a,k+1}$, and $f_{b,k+1}$ using Eqs. (4.7) to (4.12). Otherwise, if $f_{a,k} < f_{b,k}$, update information using Eqs. (4.13) to (4.18).

Step 5

If $k = n - 2$ or $x_{a,k+1} > x_{b,k+1}$, output $x^* = x_{a,k+1}$ and $f^* = f(x^*)$, and stop. Otherwise, set $k = k + 1$ and repeat from Step 4.

The condition $x_{a,k+1} > x_{b,k+1}$ implies that $x_{a,k+1} \approx x_{b,k+1}$ within



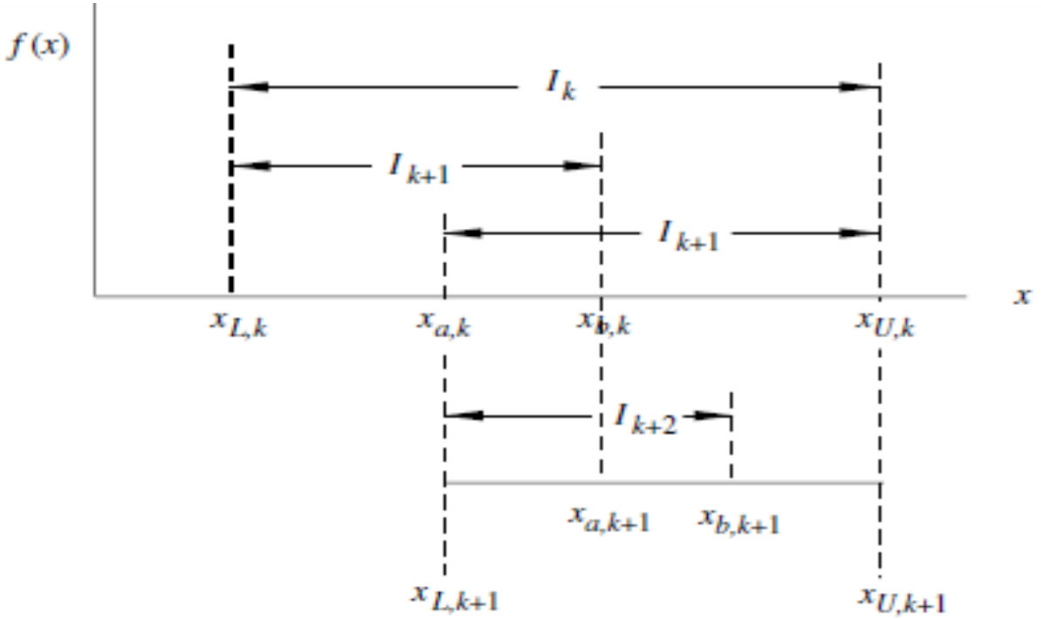
$$I_{k+2} = \frac{F_{n-k-1}}{F_{n-k}} I_{k+1}$$

ure 4.6. Assignments in kth iteration of the fibonacci search if $f_{a,k} < f_{b,k}$

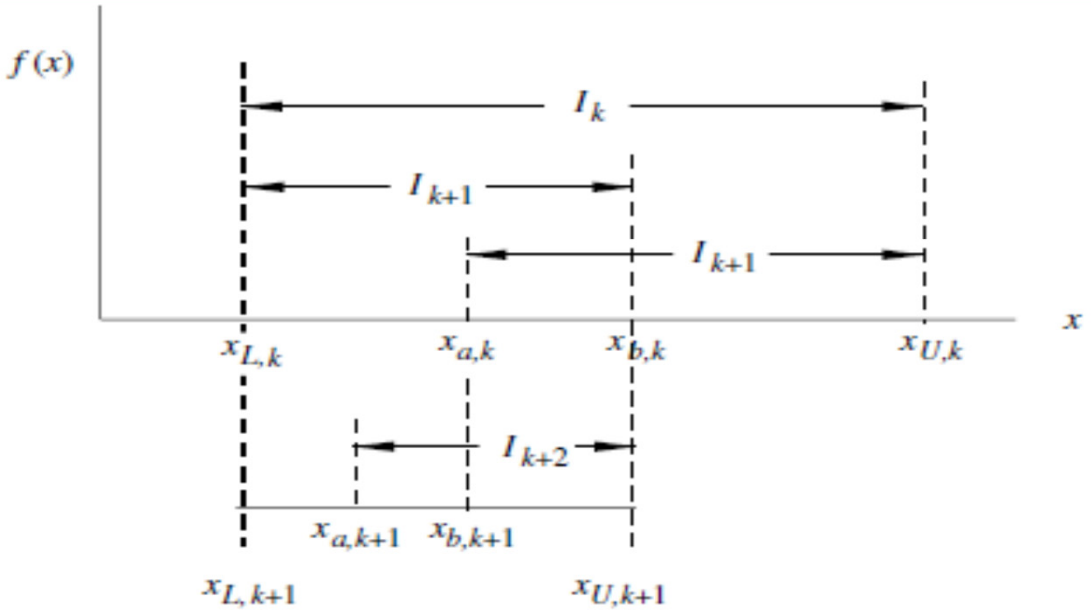
if $f_{a,k} < f_{b,k}$, then x^* is in interval $[x_{L,k},\ x_{b,k}]$.

$$\begin{aligned} x_{L,k+1} &= x_{L,k} \\ x_{U,k+1} &= x_{b,k} \\ x_{a,k+1} &= x_{U,k+1} - I_{k+2} \\ x_{b,k+1} &= x_{a,k} \\ f_{b,k+1} &= f_{a,k} \end{aligned}$$

$$f_{a,k+1} = f(x_{a,k+1})$$



gure 4.5. Assignments in kth iteration of the Fibonacci search if $f_{a,k} > f_{b,k}$.



ure 4.6. Assignments in kth iteration of the fibonacci search if $f_{a,k} < f_{b,k}$

Algorithm 4.2 Golden-section search

Step 1

Input $x_{L,1}$, $x_{U,1}$, and ε.

Step 2

Assign $I_1 = x_{U,1} - x_{L,1}$, $K = 1.618034$ and compute

$$I_2 = I_1/K$$
$$x_{a,1} = x_{U,1} - I_2, \quad x_{b,1} = x_{L,1} + I_2$$
$$f_{a,1} = f(x_{a,1}), \quad f_{b,1} = f(x_{b,1})$$

Set $k = 1$.

Step 3

Compute

$$I_{k+2} = I_{k+1}/K$$

If $f_{a,k} \geq f_{b,k}$, then update $x_{L,k+1}$, $x_{U,k+1}$, $x_{a,k+1}$, $x_{b,k+1}$, $f_{a,k+1}$, and $f_{b,k+1}$ using Eqs. (4.7) to (4.12). Otherwise, if $f_{a,k} < f_{b,k}$, update information using Eqs. (4.13) to (4.18).

Step 4

If $I_k < \varepsilon$ or $x_{a,k+1} > x_{b,k+1}$, then do:

If $f_{a,k+1} > f_{b,k+1}$, compute

$$x^* = \tfrac{1}{2}(x_{b,k+1} + x_{U,k+1})$$

If $f_{a,k+1} = f_{b,k+1}$, compute

$$x^* = \tfrac{1}{2}(x_{a,k+1} + x_{b,k+1})$$

If $f_{a,k+1} < f_{b,k+1}$, compute

$$x^* = \tfrac{1}{2}(x_{L,k+1} + x_{a,k+1})$$

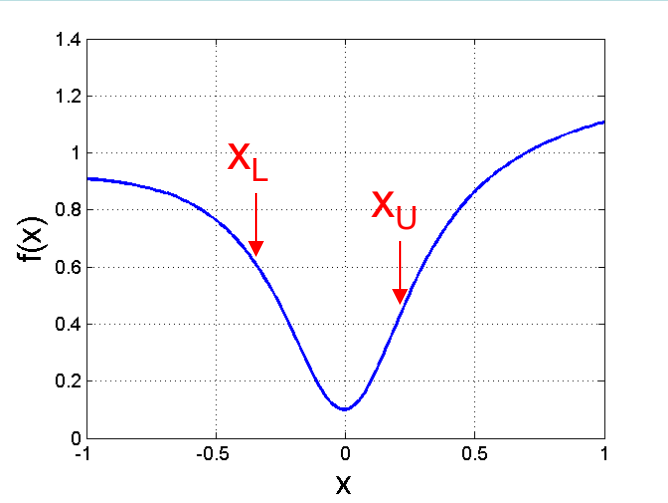
Compute $f^* = f(x^*)$.

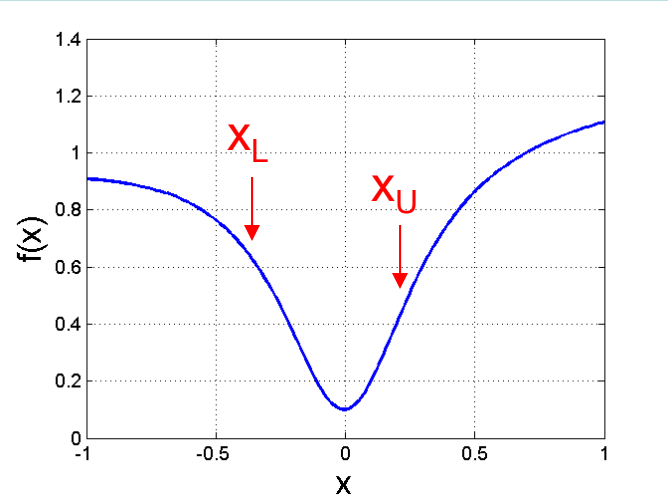
Output x^* and f^*, and stop.

Step 5

Set $k = k + 1$ and repeat from Step 3.

Search methods





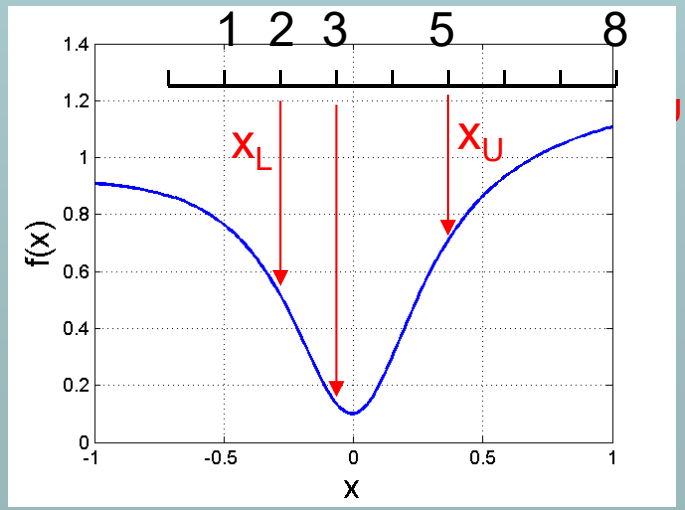
Dichotomous

Fibonacci:

1 1 2 3 5 8 …

I_{k+5} I_{k+4} I_{k+3} I_{k+2} I_{k+1} I_k

$$I_k = I_{k+1} + I_{k+2}$$



Golden-Section Search divides intervals by $K = 1.6180$

$$\frac{I_k}{I_{k+1}} = K$$

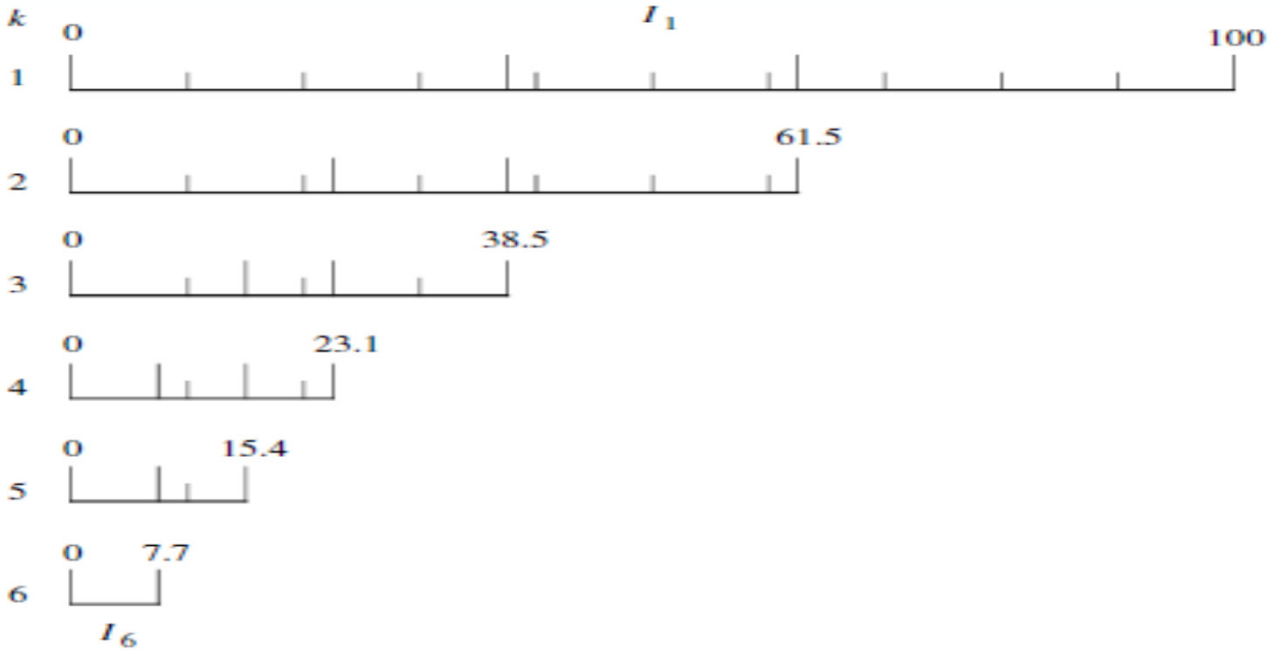


Figure 4.4. Fibonacci search for $n = 6$.

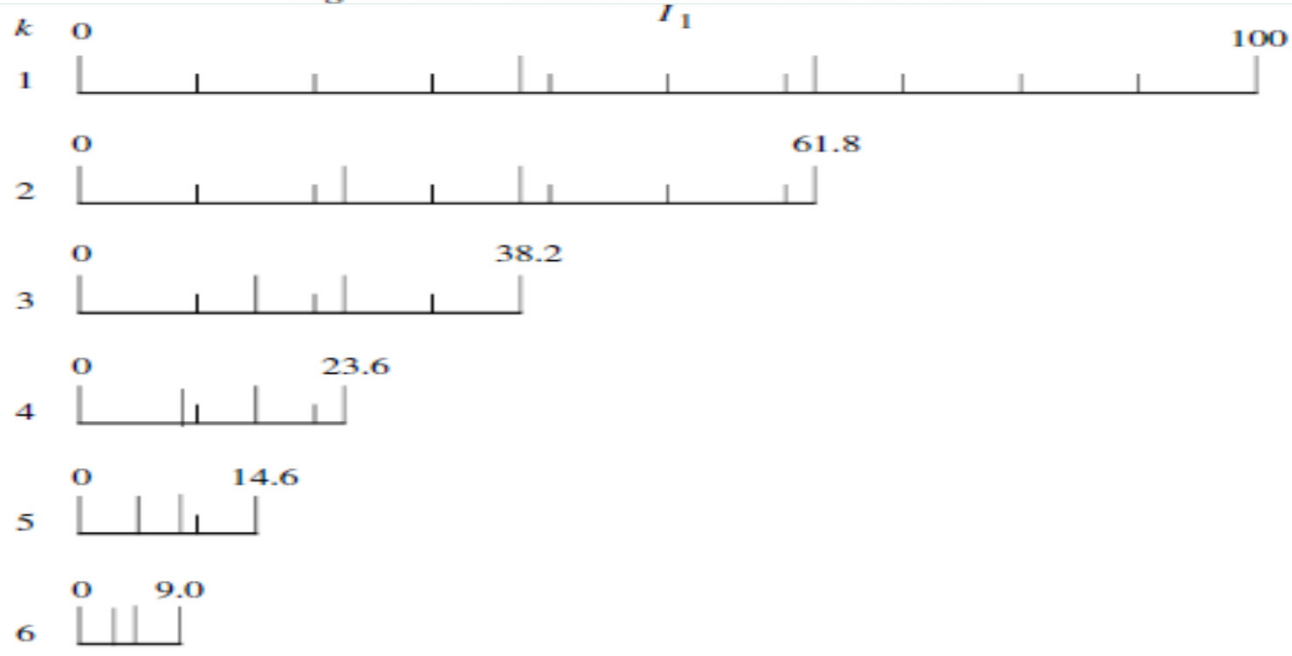


Figure 4.8. Golden section search.

1-D search

$$I_k = (\tfrac{1}{2})^k I_0$$

$$I_n = \frac{I_1}{F_n}$$

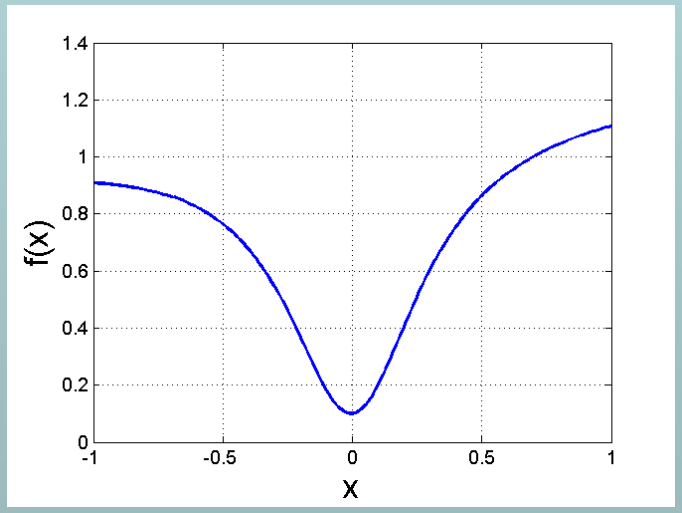
$$\frac{I_k}{I_{k+1}} = \frac{I_{k+1}}{I_{k+2}} = \frac{I_{k+2}}{I_{k+3}} = \cdots = K$$

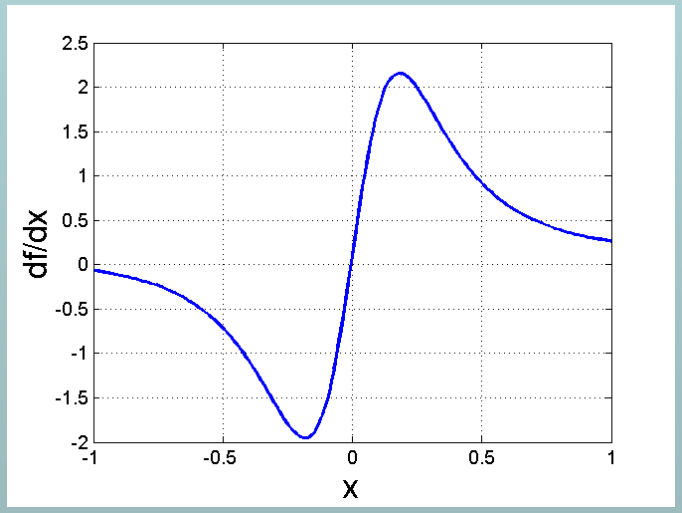
$$K = \frac{1 \pm \sqrt{5}}{2}$$

1D function

As an example consider the function

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





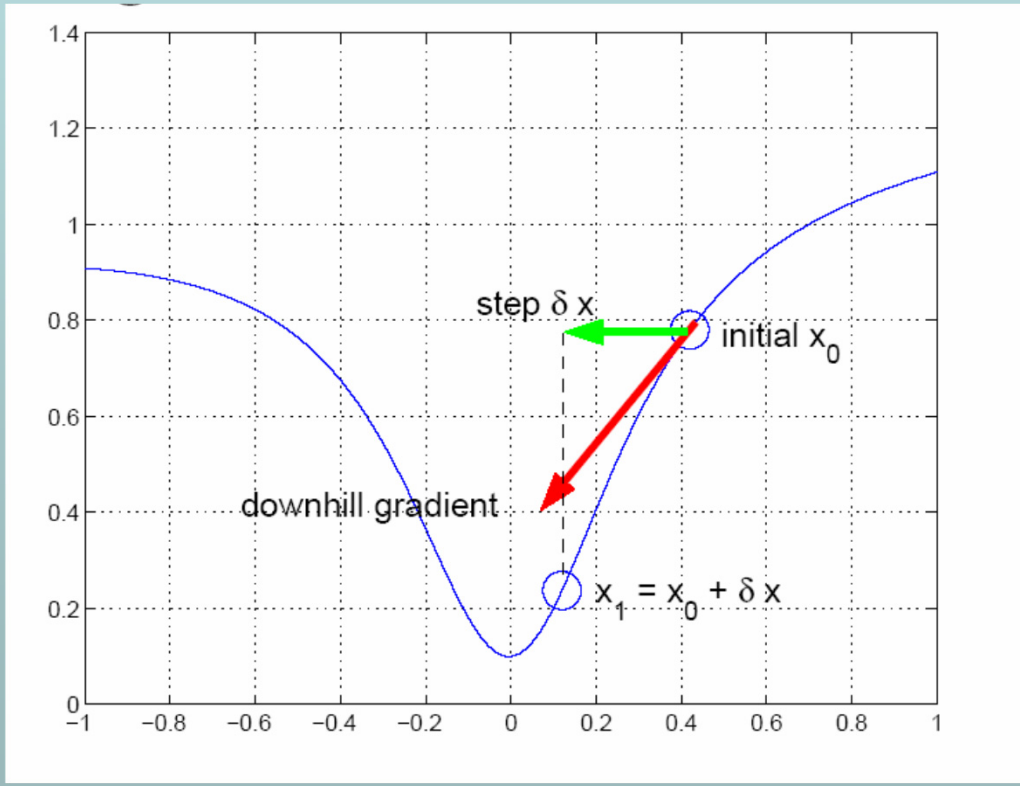
(assume we do not know the actual function expression from now on)

Gradient descent

Given a starting location, x_0, examine df/dx

and move in the *downhill* direction

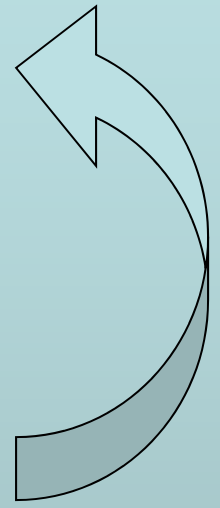
to generate a new estimate, $x_1 = x_0 + \delta x$



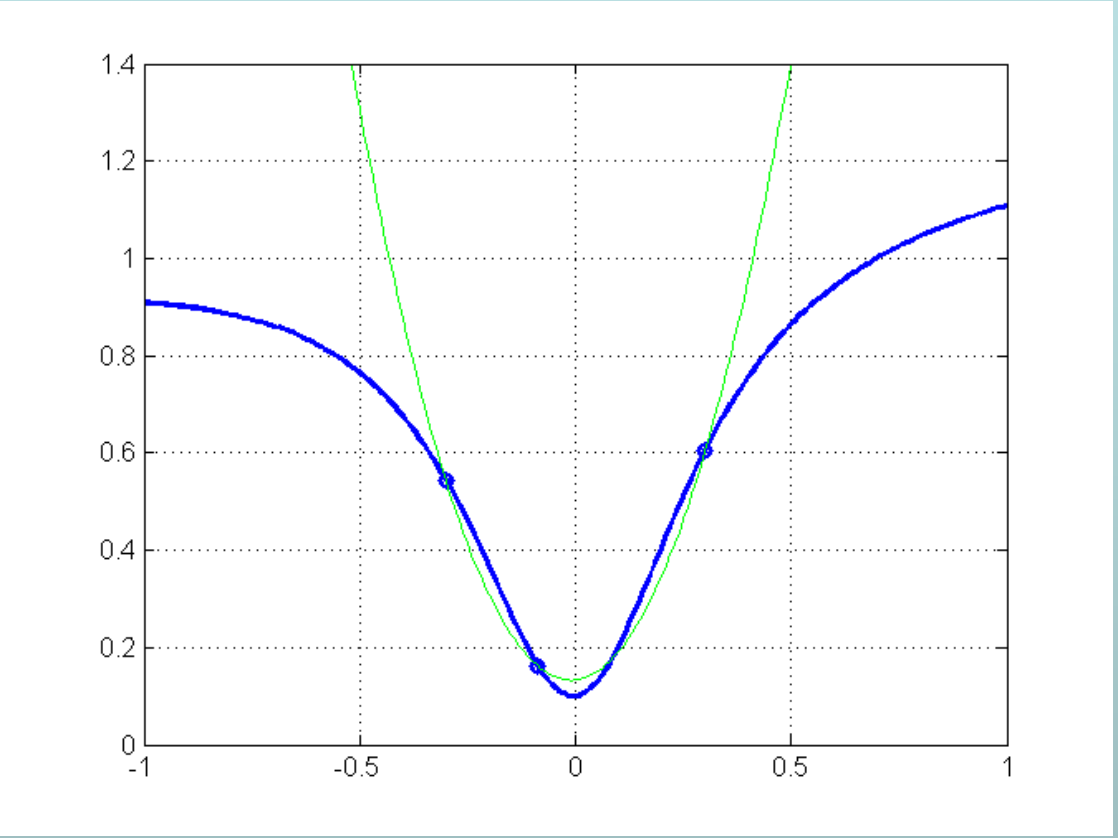
How to determine the step size δx?

Polynomial interpolation

- Bracket the minimum.
- 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.



Polynomial interpolation



- Quadratic interpolation using 3 points, 2 iterations
- Other methods to interpolate?
  - 2 points and one gradient
  - Cubic interpolation

Newton method

Fit a quadratic approximation to $f(x)$ using both gradient and curvature information at x.

- Expand $f(x)$ locally using a Taylor series.

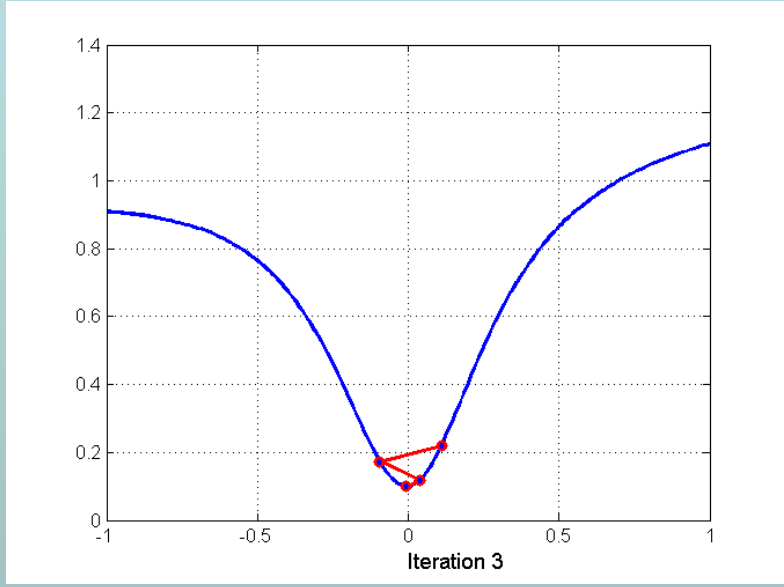
$$f(x+\delta x) = f(x) + f'(x)\delta x + \frac{1}{2}f''(x)\delta x^2 + o(\delta x^2)$$

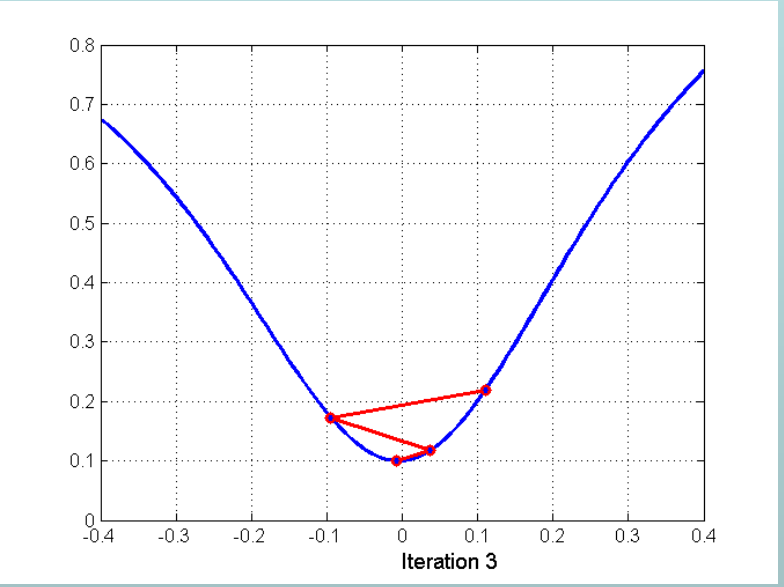
- Find the δx which minimizes this local quadratic approximation.

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

- Update x. $x_{n+1} = x_n - \delta x = x_n - \dfrac{f'(x)}{f''(x)}$

Newton method

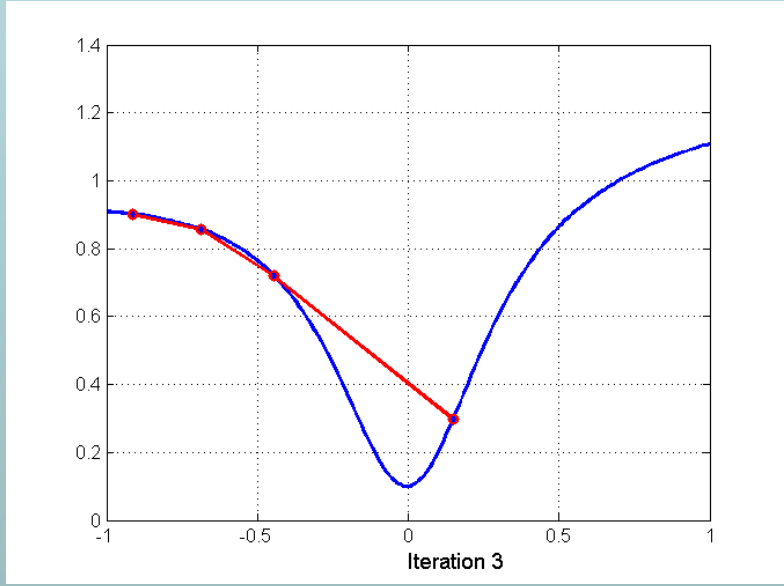


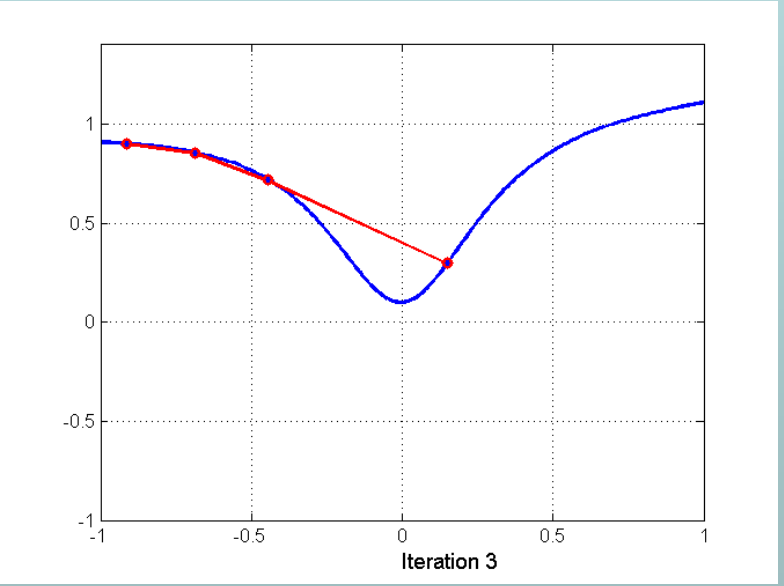


- avoids the need to bracket the root
- quadratic convergence (decimal accuracy doubles at every iteration)

Newton method

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

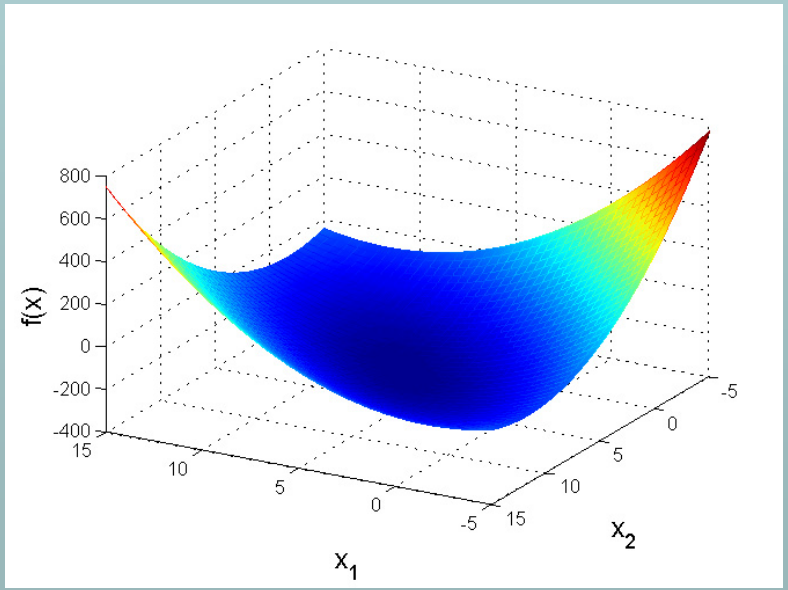


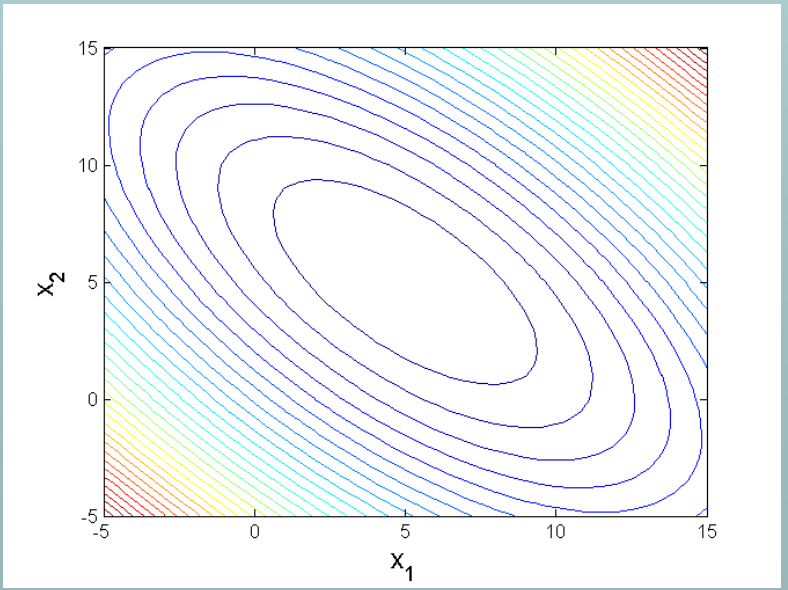


- in practice, must be used with a globalization strategy which reduces the step length until function decrease is assured

Extension to N (multivariate) dimensions

- How big N can be?
  - problem sizes can vary from a handful of parameters to many thousands
- We will consider examples for N=2, so that cost function surfaces can be visualized.

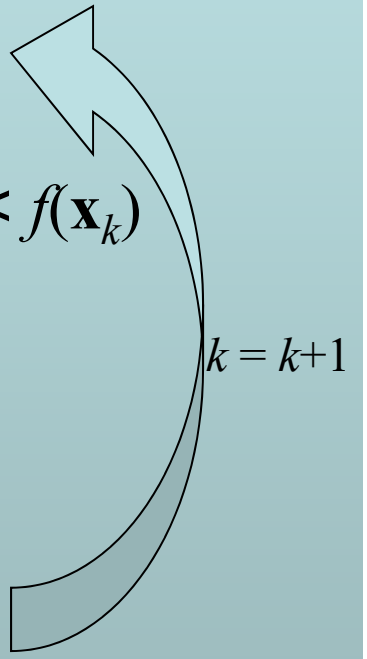




An Optimization Algorithm

- Start at $\mathbf{x}_0$, $k = 0$.

1. Compute a search direction $\mathbf{p}_k$
2. Compute a step length α_k, such that $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) < f(\mathbf{x}_k)$
3. Update $\mathbf{x}_k = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
4. Check for convergence (stopping criteria)
   e.g. $df/d\mathbf{x} = \mathbf{0}$



Reduces optimization in N dimensions to a series of (1D) line minimizations

Taylor expansion

A function may be approximated locally by its Taylor series expansion about a point $\mathbf{x}^*$

$$f(\mathbf{x}^* + \mathbf{x}) \approx f(\mathbf{x}^*) + \nabla f^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \mathbf{H} \mathbf{x}$$

where the gradient $\nabla f(\mathbf{x}^*)$ is the vector

$$\nabla f(\mathbf{x}^*) = \left[\frac{\partial f}{x_1} \cdots \frac{\partial f}{x_N} \right]^T$$

and the Hessian $\mathbf{H}(\mathbf{x}^*)$ is the symmetric matrix

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

Summary of Eqns. Studies so far

$$\mathrm{g}(\mathrm{x}) = \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n}\right]^T = \nabla f(\mathrm{x})$$

$$\nabla = \left[\frac{\partial}{\partial x_1} \quad \frac{\partial}{\partial x_2} \quad \cdots \quad \frac{\partial}{\partial x_n}\right]^T$$

the *Hessian*[1] of $f(\mathrm{x})$ is defined as

$$\mathrm{H}(\mathrm{x}) = \nabla \mathrm{g}^T = \nabla\{\nabla^T f(\mathrm{x})\}$$

$$\mathrm{H}(\mathrm{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

$$f(\mathrm{x}+\delta) = f(\mathrm{x}) + \mathrm{g}(\mathrm{x})^T\delta + \tfrac{1}{2}\delta^T \mathrm{H}(\mathrm{x})\delta + o(\|\delta\|^2)$$

$$f(\mathrm{x}+\delta) \approx f(\mathrm{x}) + \mathrm{g}(\mathrm{x})^T\delta$$

$$f(\mathrm{x}+\delta) \approx f(\mathrm{x}) + \mathrm{g}(\mathrm{x})^T\delta + \tfrac{1}{2}\delta^T \mathrm{H}(\mathrm{x})\delta$$

$$f(\mathrm{x}+\delta) = f(\mathrm{x}) + \mathrm{g}(\mathrm{x})^T\delta + \tfrac{1}{2}\delta^T \mathrm{H}(\mathrm{x}+\alpha\delta)\delta$$

Theorem 2.1 *First-order necessary conditions for a minimum*

(a) If $f(\mathrm{x}) \in C^1$ *and* x^* *is a local minimizer, then*

$$\mathrm{g}(\mathrm{x}^*)^T \mathrm{d} \geq 0$$

for every feasible direction d *at* x^*.

(b) If x^* *is located in the interior of* $\mathcal{R}$ *then*

$$\mathrm{g}(\mathrm{x}^*) = 0$$

Theorem 2.2 ***Second-order necessary conditions for a minimum***

(a) If $f(\mathbf{x}) \in C^2$ *and* $\mathbf{x}^*$ *is a local minimizer, then for every feasible direction* $\mathbf{d}$ *at* $\mathbf{x}^*$

(i) $\mathbf{g}(\mathbf{x}^*)^T \mathbf{d} \geq 0$

(ii) If $\mathbf{g}(\mathbf{x}^*)^T \mathbf{d} = 0$, *then* $\mathbf{d}^T \mathbf{H}(\mathbf{x}^*)\mathbf{d} \geq 0$

(b) If $\mathbf{x}^*$ *is a local minimizer in the interior of* $\mathcal{R}$, *then*

(i) $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$

(ii) $\mathbf{d}^T \mathbf{H}(\mathbf{x})^* \mathbf{d} \geq 0$ *for all* $\mathbf{d} \neq \mathbf{0}$

Theorem 2.4 ***Second-order sufficient conditions for a minimum*** *If* $f(\mathbf{x}) \in C^2$ *and* $\mathbf{x}^*$ *is located in the interior of* $\mathcal{R}$, *then the conditions*

(a) $\mathbf{g}(\mathbf{x}^*) = \mathbf{0}$

(b) $\mathbf{H}(\mathbf{x}^*)$ *is positive definite*

are sufficient for $\mathbf{x}^*$ *to be a strong local minimizer.*

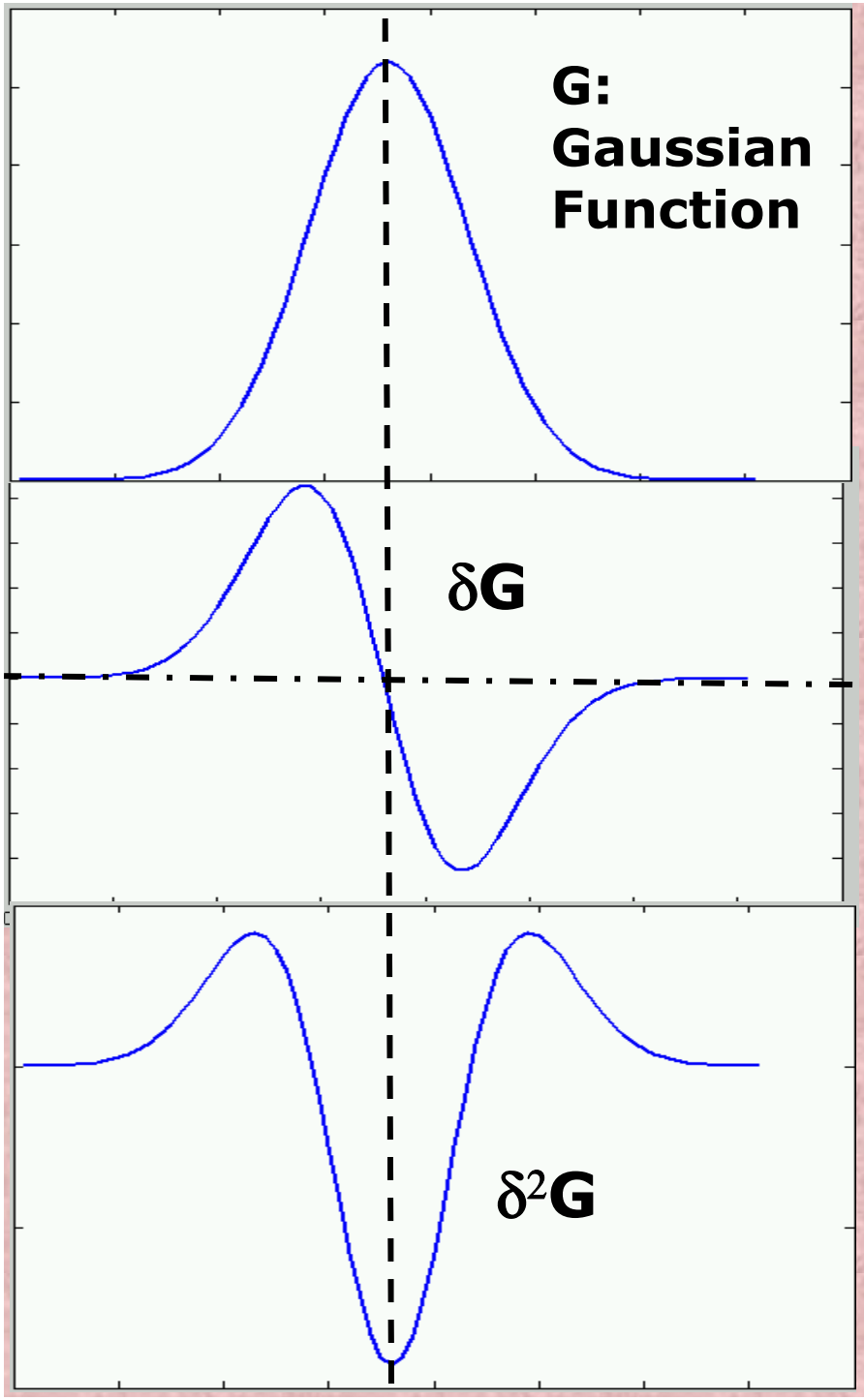
Definition 2.6 A point $\bar{\mathbf{x}} \in \mathcal{R}$, where $\mathcal{R}$ is the feasible region, is said to be a *saddle point* if

(a) $\mathbf{g}(\bar{\mathbf{x}}) = \mathbf{0}$

(b) point $\bar{\mathbf{x}}$ is neither a maximizer nor a minimizer.

Stationary points can be located and classified as follows:

1. Find the points $\mathbf{x}_i$ at which $\mathbf{g}(\mathbf{x}_i) = \mathbf{0}$.
2. Obtain the Hessian $\mathbf{H}(\mathbf{x}_i)$.
3. Determine the character of $\mathbf{H}(\mathbf{x}_i)$ for each point $\mathbf{x}_i$.



$$\text{Take}: G(x) = \frac{-1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

$$\nabla G(x) =$$

$$\frac{x}{\sqrt{2\pi}\sigma_2^3} \exp\left(-\frac{x^2}{2\sigma_2^2}\right)$$

$$\nabla^2 G(x) =$$

$$g(x^*) = \nabla G(x^*) = 0$$

$$g(x^*)^T d \geq 0 \qquad g(x^*) = 0$$

$$d^T H(x^*) d \geq 0$$

Quadratic functions

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

- The vector $\mathbf{g}$ and the Hessian $\mathbf{H}$ are constant.
- Second order approximation of any function by the Taylor expansion is a quadratic function.

We will assume only quadratic functions for a while.

Necessary conditions for a minimum

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

Expand $f(\mathbf{x})$ about a stationary point $\mathbf{x}^*$ in direction $\mathbf{p}$

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

$$= f(\mathbf{x}^*) + \frac{1}{2}\alpha^2\mathbf{p}^T\mathbf{H}\mathbf{p}$$

since at a stationary point $\mathbf{g}(\mathbf{x}^*) = 0$

At a stationary point the behavior is determined by $\mathbf{H}$

- $\mathbf{H}$ is a symmetric matrix, and so has orthogonal eigenvectors

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

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

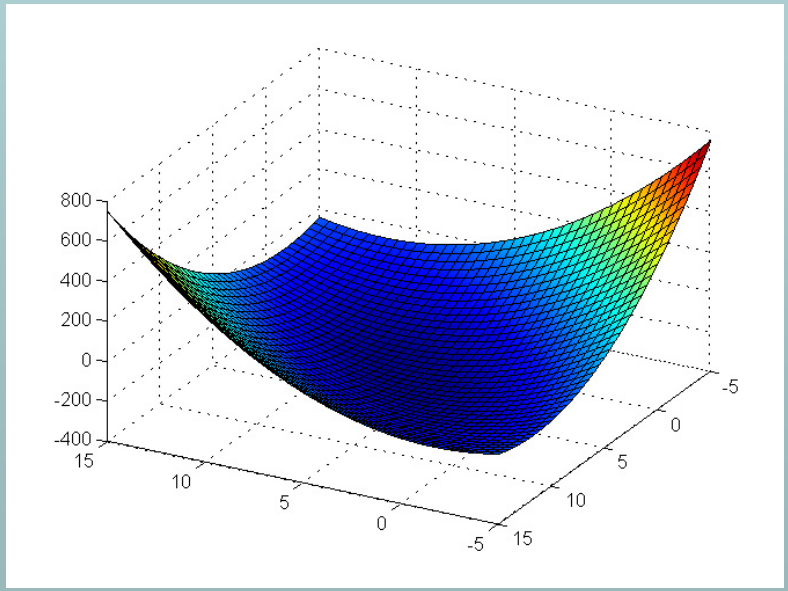
- As $|\alpha|$ increases, $f(\mathbf{x}^* + \alpha \mathbf{u}_i)$ increases, decreases or is unchanging according to whether λ_i is positive, negative or zero

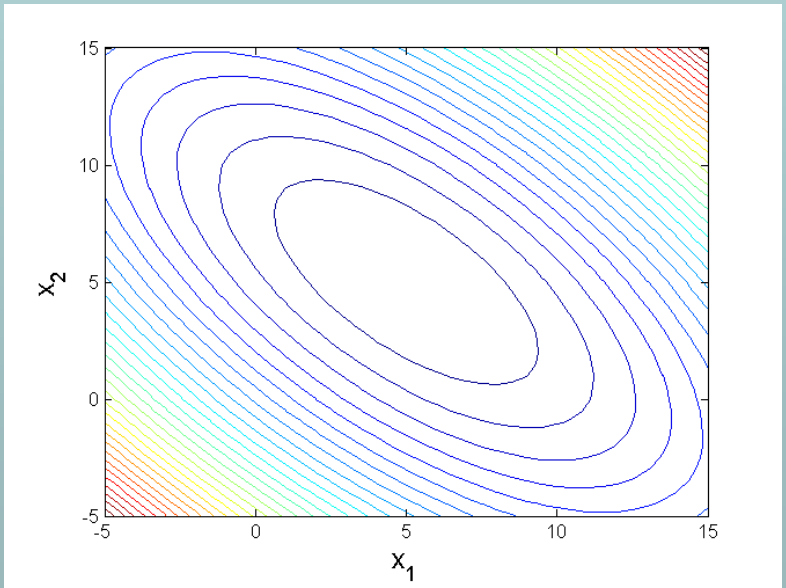
Examples of quadratic functions

Case 1: both eigenvalues positive

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

with $a = 0\,, \qquad \mathbf{g} = \begin{bmatrix} -50 \\ -50 \end{bmatrix}\,, \qquad \mathbf{H} = \begin{bmatrix} 6 & 4 \\ 4 & 6 \end{bmatrix}$ positive definite





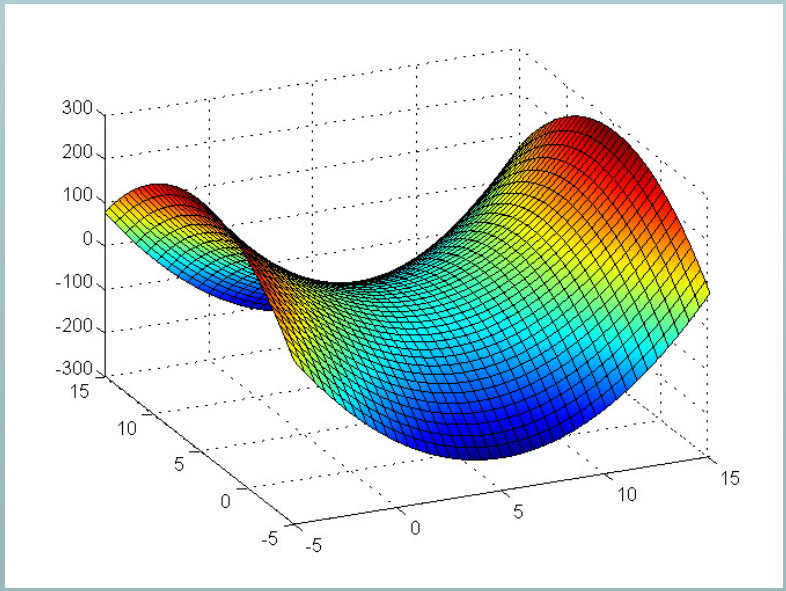
minimum

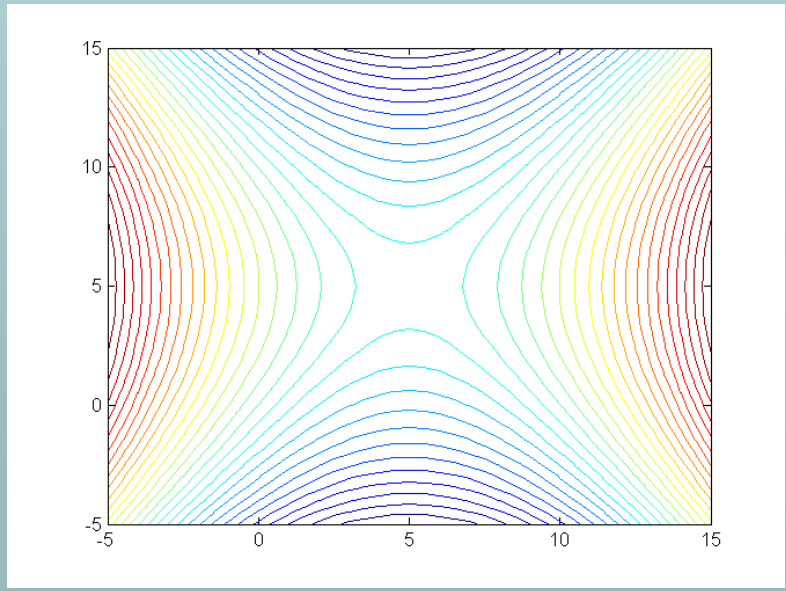
Examples of quadratic functions

Case 2: eigenvalues have different sign

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

with $a = 0\,, \qquad \mathbf{g} = \begin{bmatrix} -30 \\ 20 \end{bmatrix}\,, \qquad \mathbf{H} = \begin{bmatrix} 6 & 0 \\ 0 & -4 \end{bmatrix}$ indefinite





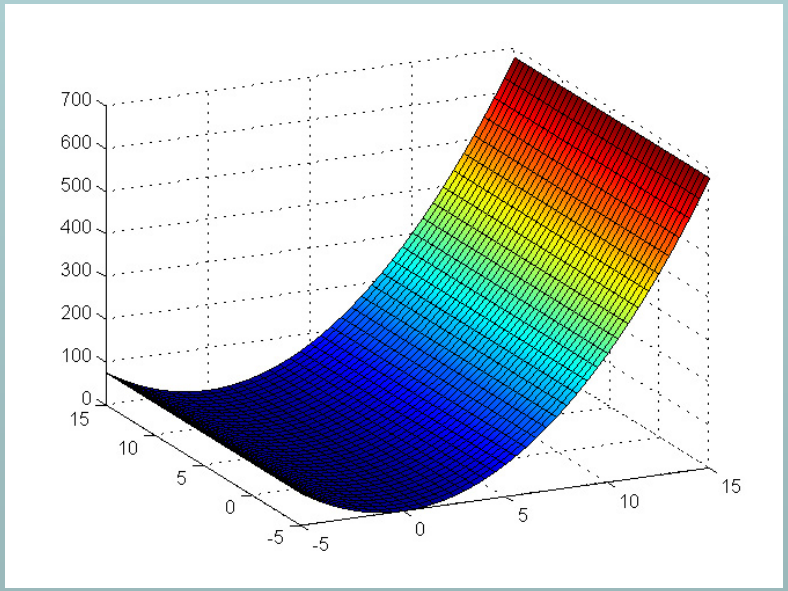
saddle point

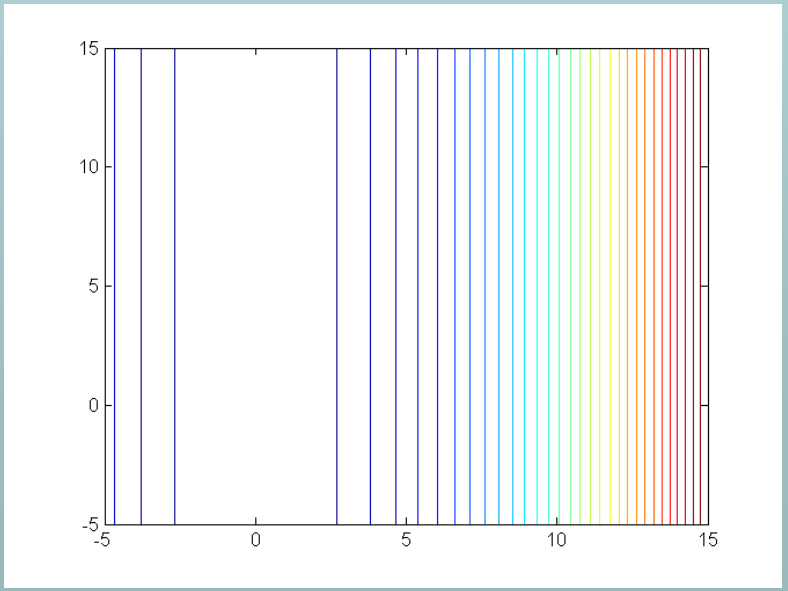
Examples of quadratic functions

Case 3: one eigenvalues is zero

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

with $a = 0\,, \qquad \mathbf{g} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\,, \qquad \mathbf{H} = \begin{bmatrix} 6 & 0 \\ 0 & 0 \end{bmatrix}$ positive semidefinite





parabolic cylinder

Optimization for quadratic functions

Assume that $\mathbf{H}$ is positive definite

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

$$\nabla f(\mathbf{x}) = \mathbf{g} + \mathbf{H}\mathbf{x}$$

There is a unique minimum at

$$\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{g}$$

If N is large, it is not feasible to perform this inversion directly.

Steepest descent

$$F + \Delta F = f(\mathbf{x} + \delta) \approx f(\mathbf{x}) + \mathbf{g}^T \delta + \tfrac{1}{2}\delta^T \mathbf{H}\delta$$

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

- The steepest descent method chooses $\mathbf{p}_k$ to be parallel to the gradient

$$\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$$

- Step-size α_k is chosen to minimize $f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$.
  For quadratic forms there is a closed form solution:

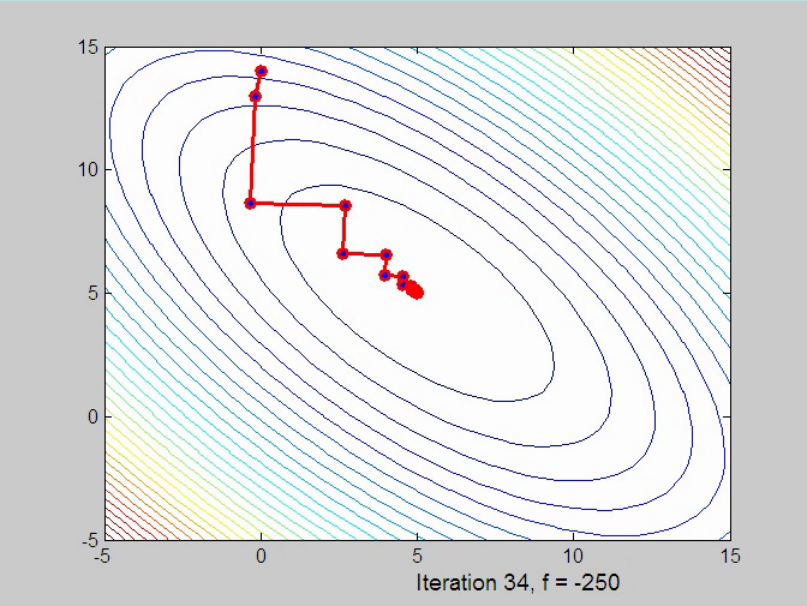
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{H}_k \mathbf{g}_k} \mathbf{g}_k$$

$$\alpha_k = \frac{\mathbf{p}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{H} \mathbf{p}_k}$$

Prove it!

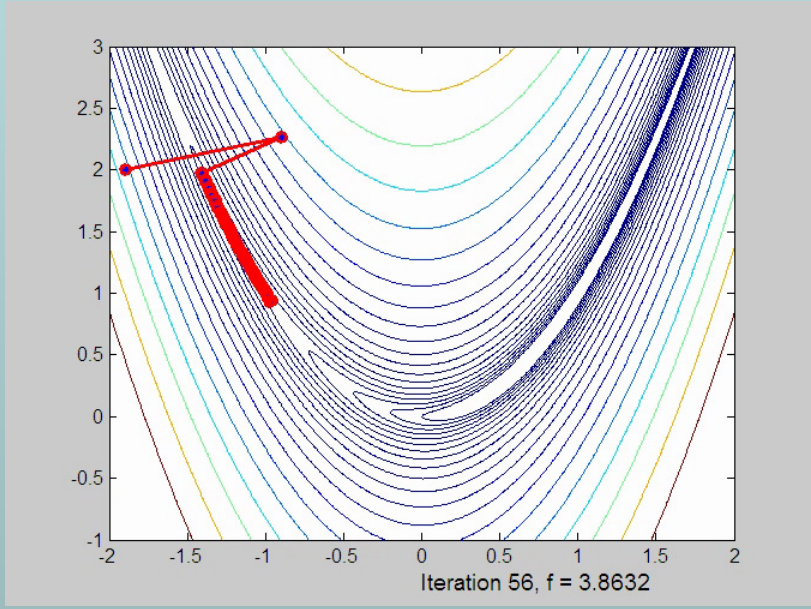
Steepest descent

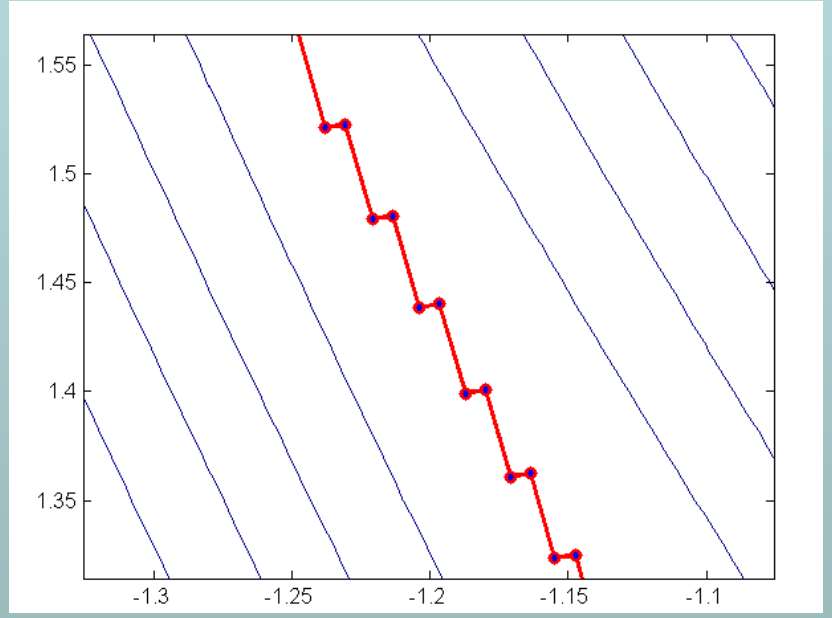


- The gradient is everywhere perpendicular to the contour lines.
- After each line minimization the new gradient is always *orthogonal* to the previous step direction (true of any line minimization).
- Consequently, the iterates tend to zig-zag down the valley in a very inefficient manner

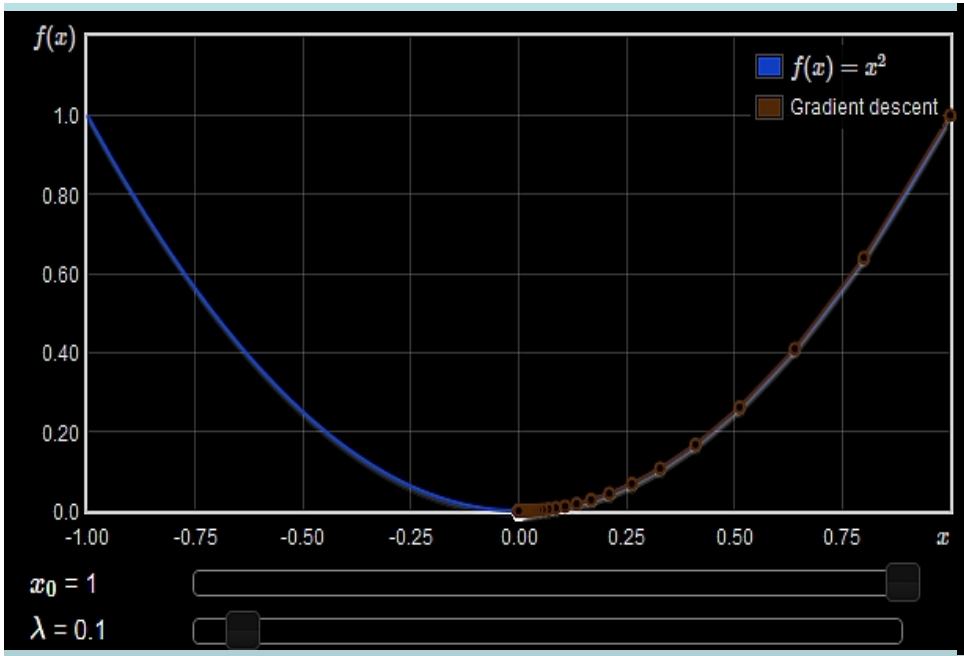
Steepest descent

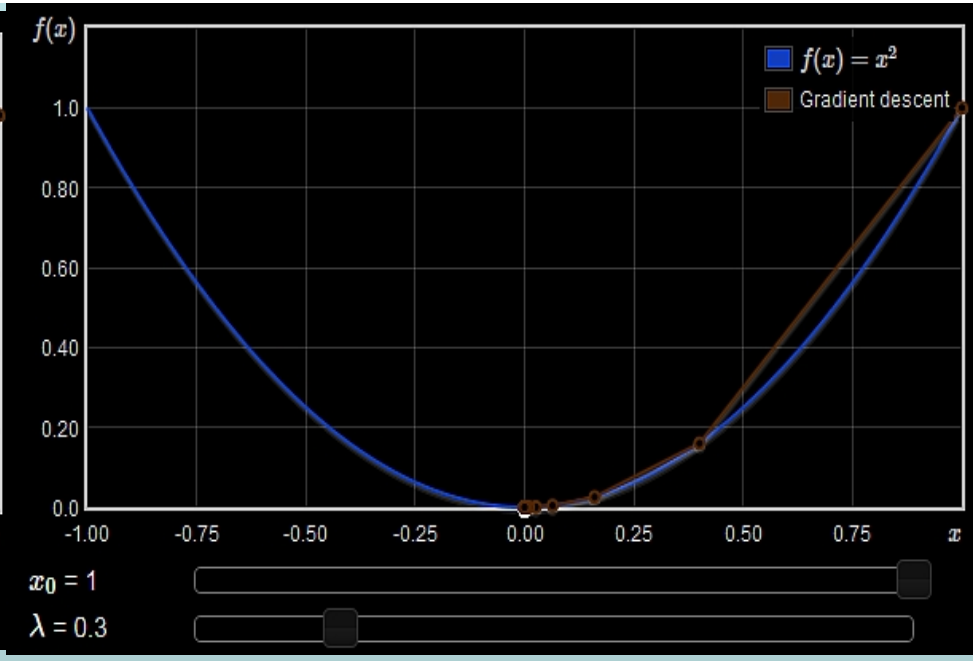
- The 1D line minimization must be performed using one of the earlier methods (usually cubic polynomial interpolation)

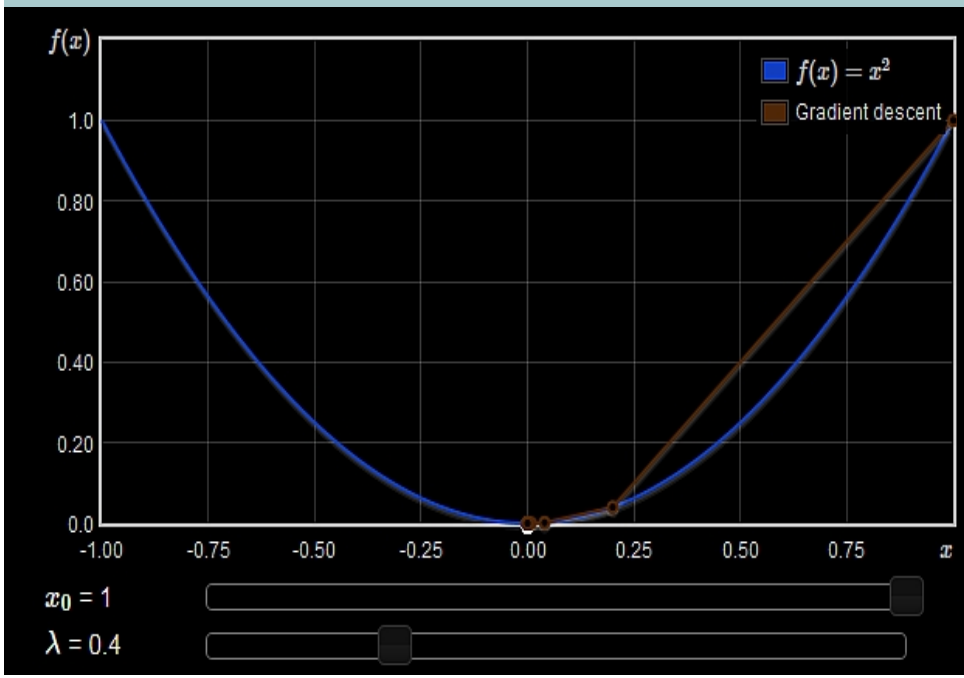


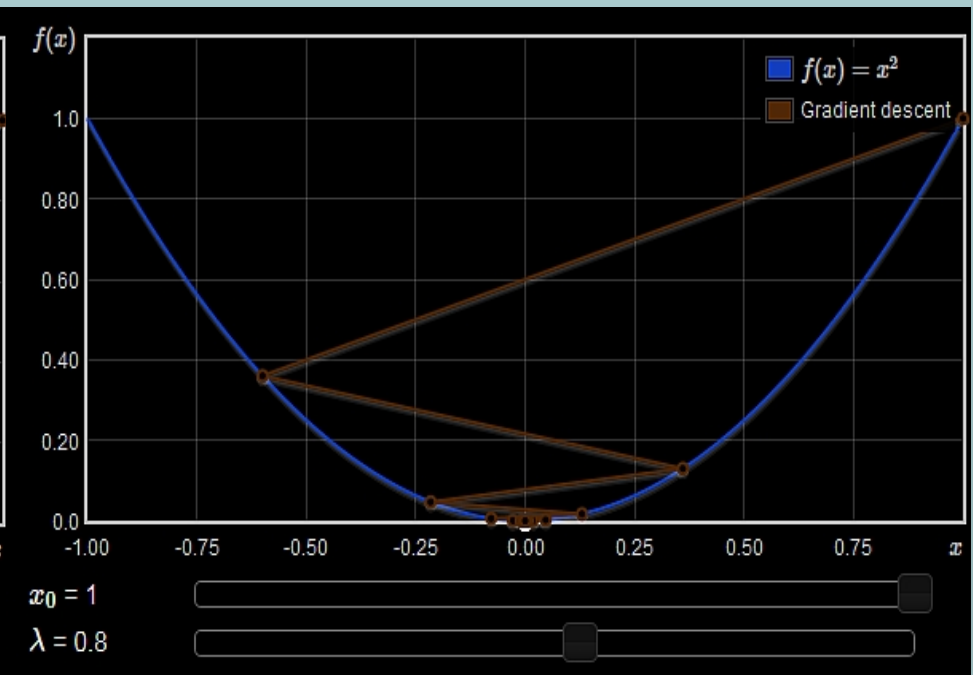


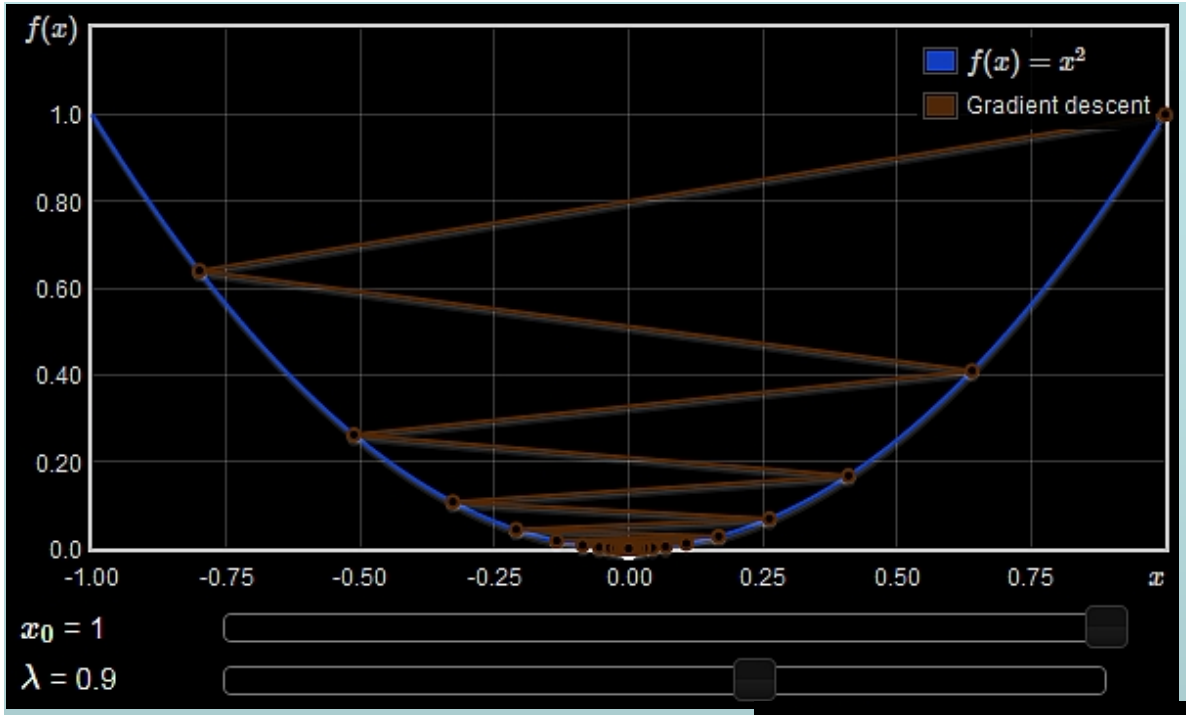
- The zig-zag behaviour is clear in the zoomed view
- The algorithm crawls down the valley

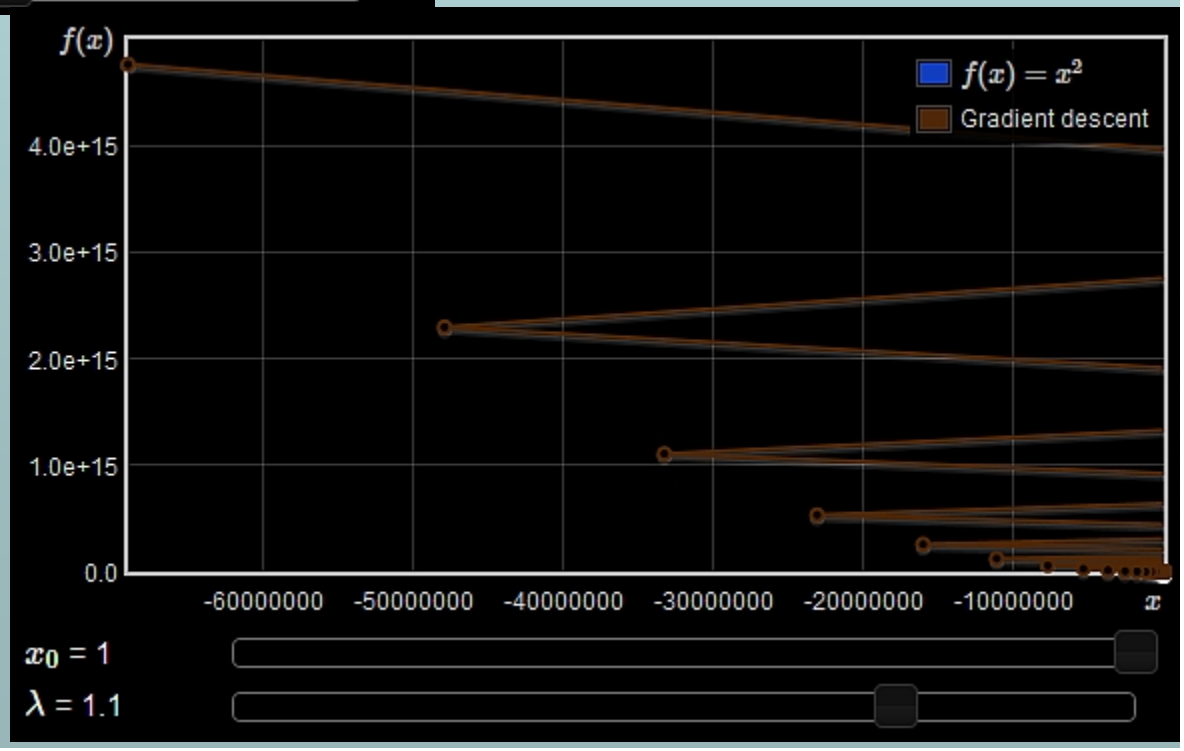

$f(x)$
$f(x) = x^2$
Gradient descent
1.0
0.80
0.60
0.40
0.20
0.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
x
x_0 = 1
λ = 0.1


$f(x)$
$f(x) = x^2$
Gradient descent
1.0
0.80
0.60
0.40
0.20
0.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
x
x_0 = 1
λ = 0.3

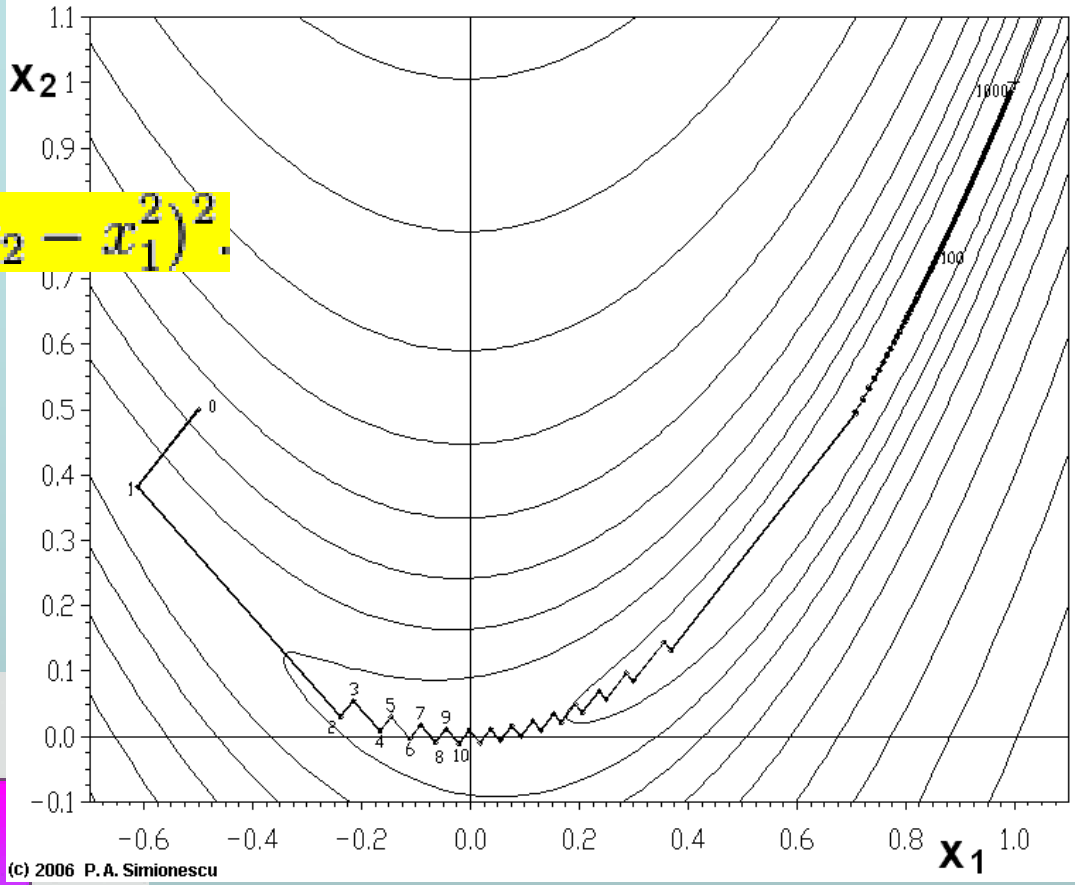

$f(x)$
$f(x) = x^2$
Gradient descent
1.0
0.80
0.60
0.40
0.20
0.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
x
x_0 = 1
λ = 0.4

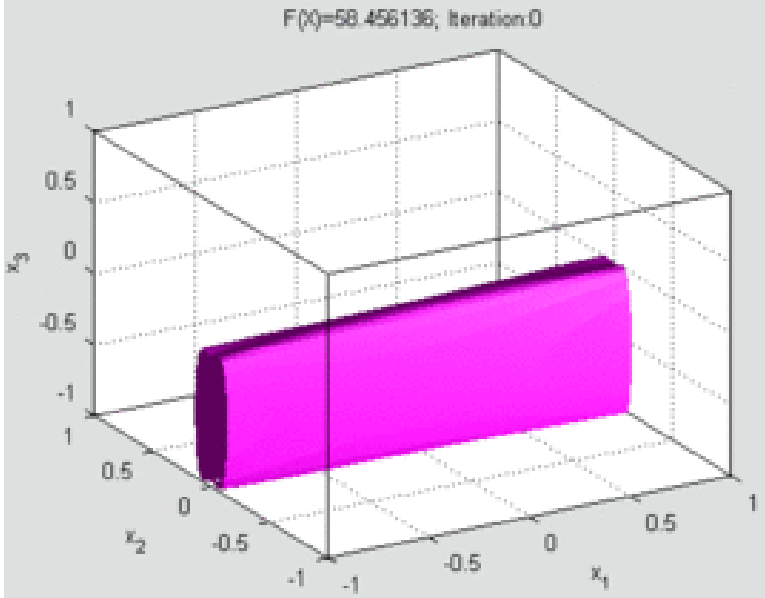

$f(x)$
$f(x) = x^2$
Gradient descent
1.0
0.80
0.60
0.40
0.20
0.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
x
x_0 = 1
λ = 0.8

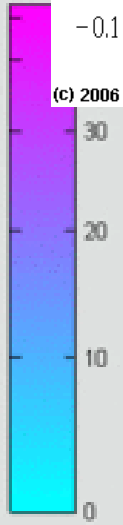

$f(x)$
$f(x) = x^2$
Gradient descent
1.0
0.80
0.60
0.40
0.20
0.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
x
x_0 = 1
λ = 0.9


$f(x)$
$f(x) = x^2$
Gradient descent
4.0e+15
3.0e+15
2.0e+15
1.0e+15
0.0
-60000000
-50000000
-40000000
-30000000
-20000000
-10000000
x
x_0 = 1
λ = 1.1

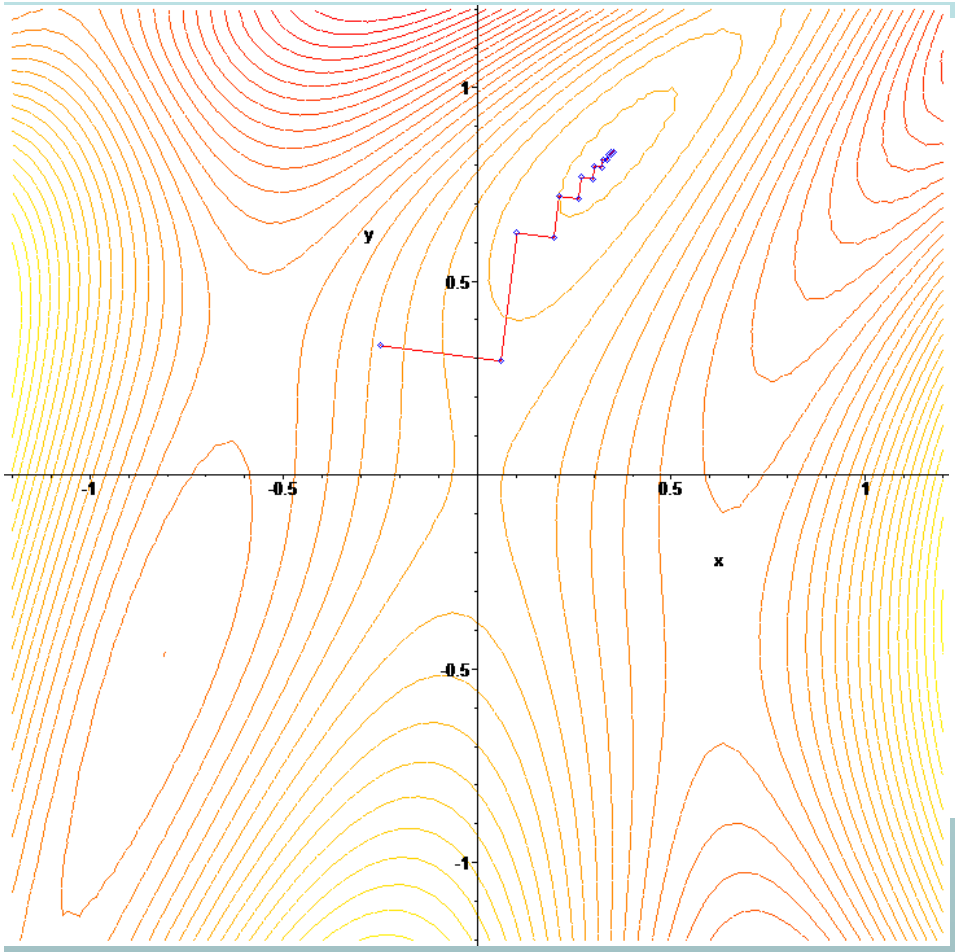
$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2.$$

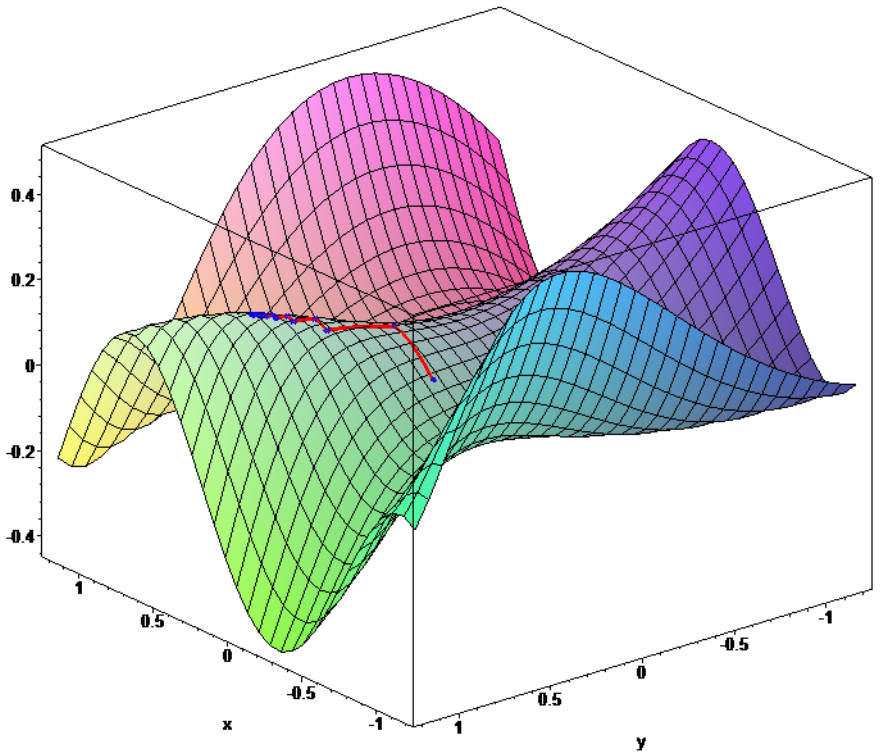






$$= \frac{1}{2}G^T(\mathbf{x})G(\mathbf{x}) = \frac{1}{2}\left(\left(3x_1 - \cos(x_2x_3) - \frac{3}{2}\right)^2 + \left(4x_1^2 - 625x_2^2 + 2x_2 - 1\right)^2 + \left(\exp(-x_1x_2) + 20x_3 + \frac{1}{3}(10\pi - 3)\right)^2\right)$$





$$F(x,y) = \sin\left(\frac{1}{2}x^2 - \frac{1}{4}y^2 + 3\right)\cos(2x + 1 - e^y)$$

Newton method

Expand $f(\mathbf{x})$ by its Taylor series about the point $\mathbf{x}_k$

$$f(\mathbf{x}_k + \delta\mathbf{x}) \approx f(\mathbf{x}_k) + \mathbf{g}_k^T \delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^T \mathbf{H}_k \delta\mathbf{x}$$

where the gradient is the vector

$$\mathbf{g}_k = \nabla f(\mathbf{x}_k) = \left[\frac{\partial f}{x_1} \cdots \frac{\partial f}{x_N}\right]^T$$

and the Hessian is the symmetric matrix

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \boldsymbol{\delta}_k = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{d}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k$$

Newton method

For a minimum we require that $\nabla f(\mathbf{x}) = 0$, and so

$$\nabla f(\mathbf{x}) = \mathbf{g}_k + \mathbf{H}_k \delta\mathbf{x} = 0$$

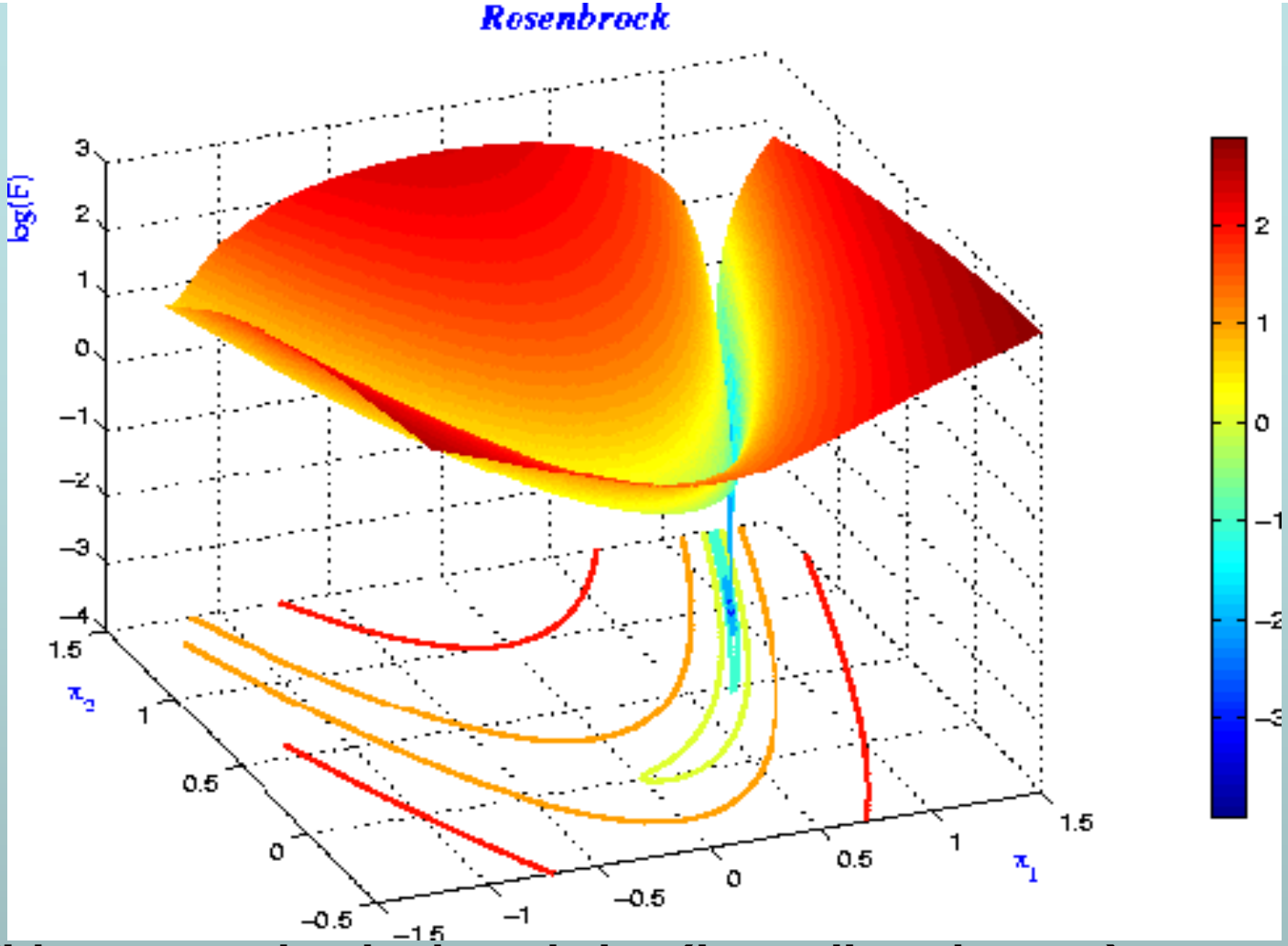
with solution $\delta\mathbf{x} = -\mathbf{H}_k^{-1}\mathbf{g}_k$. This gives the iterative update

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1}\mathbf{g}_k$$

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{H}_k^{-1}\mathbf{g}_k$$

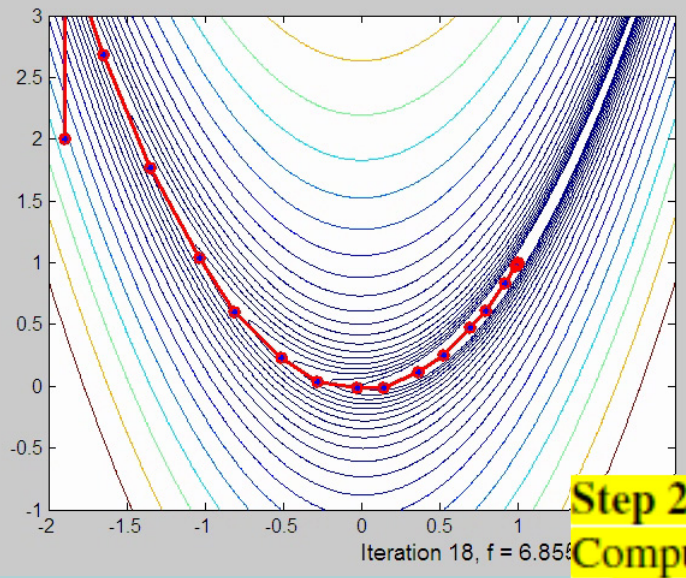
- If $\mathbf{H}=\mathbf{I}$ then this reduces to steepest descent.



Without second order knowledge (ie gradient descent), you can miss the narrow valley entirely with fixed sized steps in the direction of the gradient (i.e. fixed steps are too large). Even if you go into the valley, you would spend a great deal of time zig zagging back and forth the steep walls because the gradient at those walls would simply direct the descent to each side of the valley.

Second order information (ie Hessian) allows you to take into account the curvature and take steps sized inverse to the 'steepness' (very steep -> small steps, very flat -> large steps).

Newton method - example



Step 2
Compute $\mathbf{g}_k$ and $\mathbf{H}_k$.
If $\mathbf{H}_k$ is not positive definite, force it to become positive definite.
Step 3
Compute $\mathbf{H}_k^{-1}$ and $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$.
Step 4
Find α_k, the value of α that minimizes $f(\mathbf{x}_k + \alpha\mathbf{d}_k)$, using a line search.
Step 5
Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$.
Compute $f_{k+1} = f(\mathbf{x}_{k+1})$.

- The algorithm converges in only 18 iterations compared to the 98 for conjugate gradients.
- However, the method requires computing the Hessian matrix at each iteration – this is not always feasible

Gauss - Newton method

$$\mathbf{f} = [f_1(\mathbf{x})\ f_2(\mathbf{x})\ \cdots\ f_m(\mathbf{x})]^T$$

$$F = \sum_{p=1}^{m} f_p(\mathbf{x})^2 = \mathbf{f}^T\mathbf{f}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\mathbf{g}_F = 2\mathbf{J}^T\mathbf{f}$$

$$\mathbf{H}_F \approx 2\mathbf{J}^T\mathbf{J}$$

Step 2
Compute $f_{pk} = f_p(\mathbf{x}_k)$ for $p = 1, 2, \ldots, m$ and F_k.
Step 3
Compute $\mathbf{J}_k$, $\mathbf{g}_k = 2\mathbf{J}_k^T\mathbf{f}_k$, and $\mathbf{H}_k = 2\mathbf{J}_k^T\mathbf{J}_k$.
Step 4
Compute $\mathbf{L}_k$ and $\hat{\mathbf{D}}_k$ using Algorithm 5.4.
Compute $\mathbf{y}_k = -\mathbf{L}_k\mathbf{g}_k$ and $\mathbf{d}_k = \mathbf{L}_k^T\hat{\mathbf{D}}_k^{-1}\mathbf{y}_k$.
Step 5
Find α_k, the value of α that minimizes $F(\mathbf{x}_k + \alpha\mathbf{d}_k)$.
Step 6
Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{d}_k$.
Compute $f_{p(k+1)}$ for $p = 1, 2, \ldots, m$ and F_{k+1}.

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha_k(2\mathbf{J}^T\mathbf{J})^{-1}(2\mathbf{J}^T\mathbf{f}) \\ &= \mathbf{x}_k - \alpha_k(\mathbf{J}^T\mathbf{J})^{-1}(\mathbf{J}^T\mathbf{f})\end{aligned}$$

Method	Alpha (α) Calculation	Intermediate Updates	Convergence Condition
Steepest-Descent Method (Method 1)	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	If $\|\|\alpha_k d_k\|\| < \in$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$
Steepest-Descent Method (Method 2)	Without Using Line Search $\alpha_k \approx \frac{g_k^T g_k}{g_k^T H_k g_k}$	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	--" "--
Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -H_k^{-1} g_k$ $f_{k+1} = f(x_{k+1})$	--" "--

Method	Alpha (α) Calculation	Intermediate Updates	Convergence Condition
Steepest-Descent Method (Method 1)	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	If $\|\|\alpha_k d_k\|\| < \epsilon$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$
Steepest-Descent Method (Method 2)	Without Using Line Search $\alpha_k \approx \frac{g_k^T g_k}{g_k^T H_k g_k}$	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	--" "--
Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -H_k^{-1} g_k$ $f_{k+1} = f(x_{k+1})$	--" "--
Gauss-Newton Method	Find α_k, the value of α that minimizes F$(x_k + \alpha d_k)$, using line search $F = \sum_{p=1}^{m} f_p(x)^2 = f^T f$ $f = [f_1(x)\ f_2(x) \dots f_m(x)]^T$	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -H_k^{-1} g_k$ $g_F = 2J^T f$ $H \approx 2J^T J = \boldsymbol{L^{-1} D (L^T)^{-1}}$ $x_{k+1} = x_k - \alpha_k (J^T J)^{-1} (J^T f)$ $x_{k+1} = x_k - \alpha_k \boldsymbol{L^T D L} g_k$ $f_{p(k+1)} = f_p(x_k)$ $F_{(k+1)} = F(x_k)$	If $\|F_{k+1} - F_k\| < \epsilon$ then $x^* = x_{k+1}$, F$(x^*) = F_{k+1}$ Else $k = k + 1$

Conjugate gradient

- Each $\mathbf{p}_k$ is chosen to be conjugate to all previous search directions with respect to the Hessian $\mathbf{H}$:

$$\mathbf{p}_i^T \mathbf{H} \mathbf{p}_j = 0\,, \qquad i \neq j$$

- The resulting search directions are mutually linearly independent.

Prove it!

- *Remarkably*, $\mathbf{p}_k$ can be chosen using only knowledge of $\mathbf{p}_{k-1}$, $\nabla f(\mathbf{x}_{k-1})$, and $\nabla f(\mathbf{x}_k)$

$$\mathbf{p}_k = \nabla f_k + \left(\frac{\nabla f_k^\top \nabla f_k}{\nabla f_{k-1}^\top \nabla f_{k-1}} \right) \mathbf{p}_{k-1}$$

Conjugate-gradient algorithm

Step 3

Input $\mathbf{H}_k$, i.e., the Hessian at $\mathbf{x}_k$.

Compute

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{d}_k^T \mathbf{H}_k \mathbf{d}_k}$$

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ and calculate $f_{k+1} = f(\mathbf{x}_{k+1})$.

Step 4

If $\|\alpha_k \mathbf{d}_k\| < \varepsilon$, output $\mathbf{x}^* = \mathbf{x}_{k+1}$ and $f(\mathbf{x}^*) = f_{k+1}$, and stop.

Step 5

Compute $\mathbf{g}_{k+1}$.

Compute

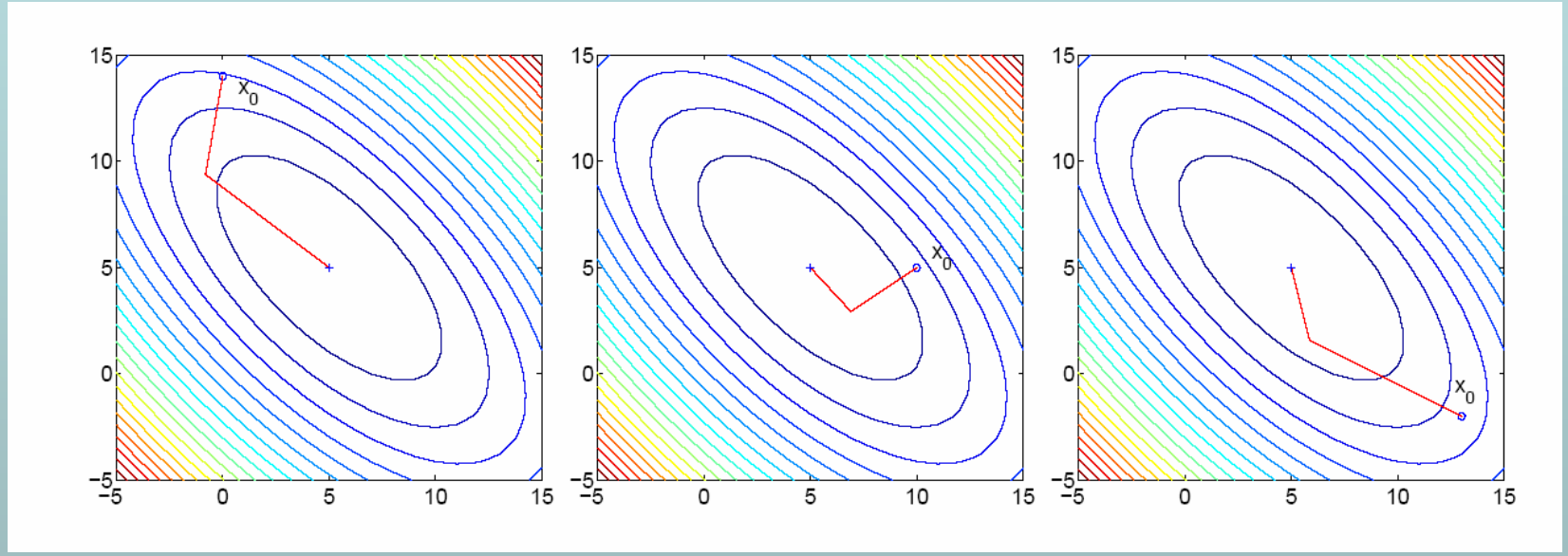
$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}$$

Generate new direction

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$$

Conjugate gradient

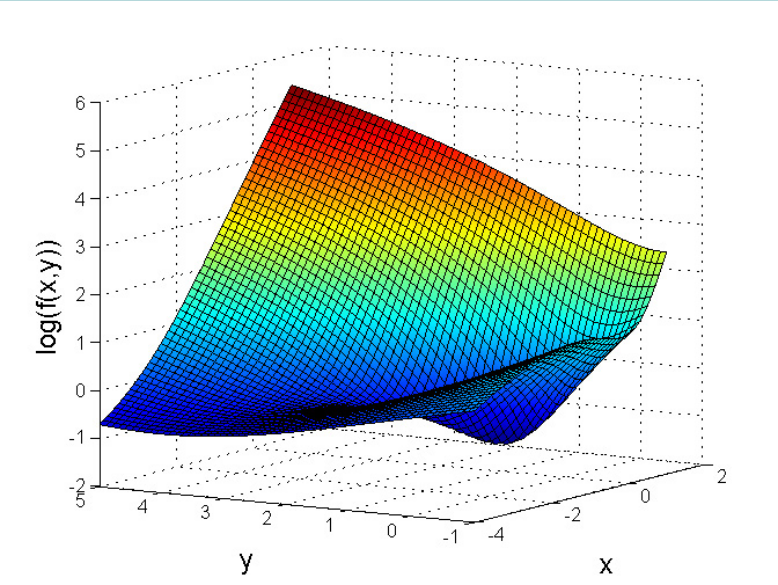
- An N-dimensional quadratic form can be minimized in at most N conjugate descent steps.

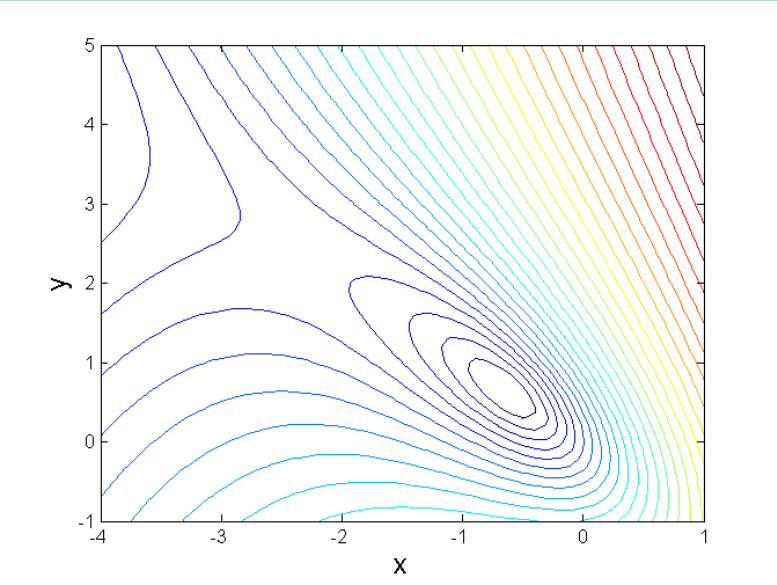


- 3 different starting points.
- Minimum is reached in exactly 2 steps.

Optimization for General functions

$$f(x,y) = \exp(x)(4x^2 + 2y^2 + 4xy + 2x + 1)$$

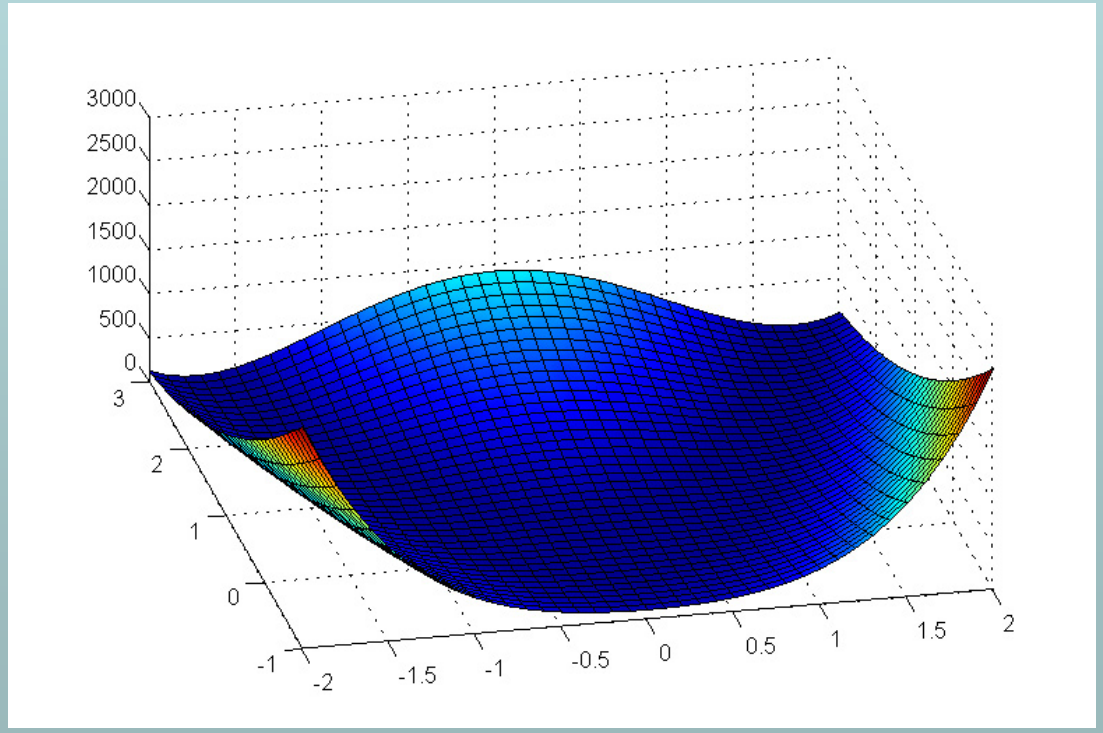


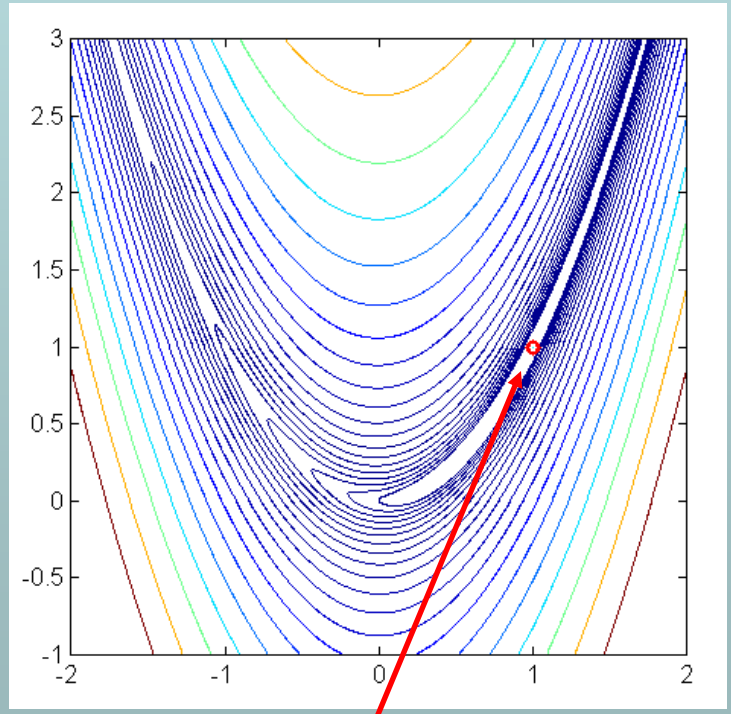


Apply methods developed using quadratic Taylor series expansion

Rosenbrock's function

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

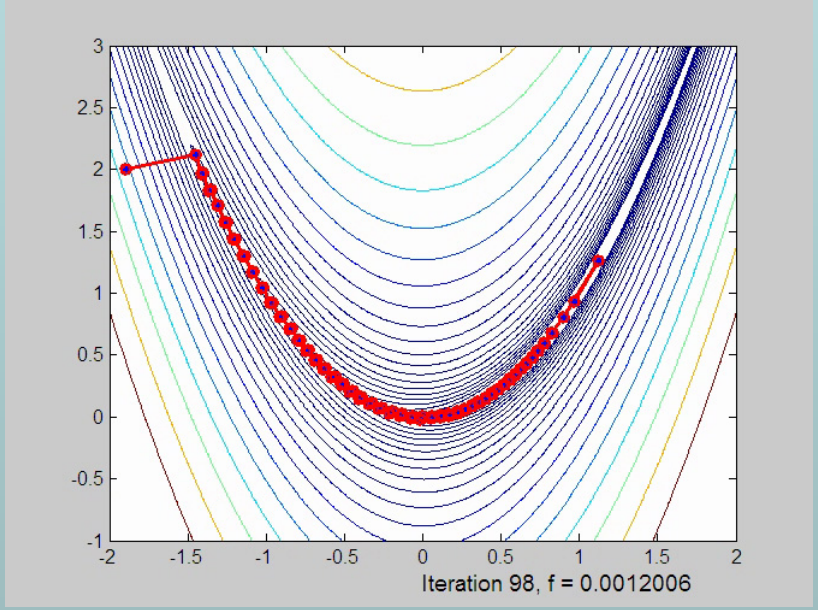




Minimum at [1, 1]

Conjugate gradient

- Again, an explicit line minimization must be used at every step



- The algorithm converges in 98 iterations
- Far superior to steepest descent

Quasi-Newton methods

- If the problem size is large and the Hessian matrix is dense then it may be infeasible/inconvenient to compute it directly.
- Quasi-Newton methods avoid this problem by keeping a "rolling estimate" of H(x), updated at each iteration using new gradient information.
- Common schemes are due to Broyden, Goldfarb, Fletcher and Shanno (BFGS), and also Davidson, Fletcher and Powell (DFP).
- The idea is based on the fact that for quadratic functions holds

$$\mathbf{g}_{k+1} - \mathbf{g}_k = \mathbf{H}(\mathbf{x}_{k+1} - \mathbf{x}_k)$$

and by accumulating $\mathbf{g}_k$'s and $\mathbf{x}_k$'s we can calculate $\mathbf{H}$.

Quasi-Newton BFGS method

- Set $\mathbf{H}_0 = \mathbf{I}$.
- Update according to

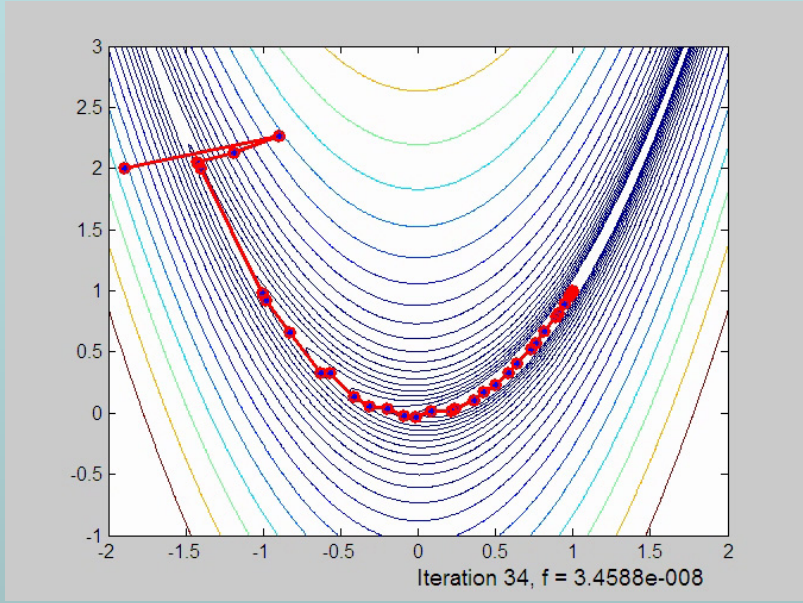
$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\gamma_k \gamma_k^T}{\gamma_k^T \delta_k} - \frac{\mathbf{H}_k \gamma_k \gamma_k^T \mathbf{H}_k}{\delta_k^T \mathbf{H}_k \delta_k}$$

where

$$\gamma_k = \mathbf{g}_{k+1} - \mathbf{g}_k \qquad \delta_k = \mathbf{x}_{k+1} - \mathbf{x}_k$$

- The matrix inverse can also be computed in this way.
- Directions δ_k's form a conjugate set.
- $\mathbf{H}_{k+1}$ is positive definite if $\mathbf{H}_k$ is positive definite.
- The estimate $\mathbf{H}_k$ is used to form a local quadratic approximation as before

BFGS example



- The method converges in 34 iterations, compared to 18 for the full-Newton method

Method	(α) Calculation	Intermediate Updates	Conv. Condition
Conjugate Gradient	Without Using Line Search $\alpha_k = \frac{g_k^T g_k}{d_k^T H_k d_k}$	$d_0 = -g_0;\quad x_{k+1} = x_k + \alpha_k d_k$ $\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k};\ d_{k+1} = -g_{k+1} + \beta_k d_k$ $f_{k+1} = f(x_{k+1})$	If $\lvert\lvert \alpha_k d_k \rvert\rvert < \in$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k+1$
Fletcher-Reeves	Line search	$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}; d_{k+1} = -g_{k+1} + \beta_k d_k$ $f_{k+1} = f(x_{k+1}); x_{k+1} = x_k + \alpha_k d_k$	-,,-
Powell	Line search	$X_k = X_{k-1} + \alpha_k d_k;\ d_k = [0\ 0\ \ldots 0\ x_k 0\ \ldots 0]^T$ $f_{k+1} = f(X_{k+1});$ $d_{k(n+1)} = x_{kn} - x_{k0}$	- ,,-
Quasi-Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \delta_k;\quad \delta_k = \alpha_k d_k;$ $d_k = -S_k g_k;\quad x_{k+1} = x_k - \alpha_k S_k g_k$ $Compute\ g_{k+1}\ /* = g_k + H\delta_k\ */$ $S_0 = I_n$ $S_{k+1} = S_k + \frac{(\delta_k - S_k\gamma_k)(\delta_k - S_k\gamma_k)^T}{\gamma_k^T(\delta_k - S_k\gamma_k)}$ $\gamma_k = g_{k+1} - g_k$	If $\lvert\lvert \delta_k \rvert\rvert < \in$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k+1$

Method	Alpha (α) Calculation	Intermediate Updates	Convergence Condition
Coordinate Descent	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = [0\ 0\ \ldots 0\ d_k 0\ \ldots 0]^T$ $f_{k+1} = f(x_{k+1})$	If $\lvert\lvert \alpha_k d_k \rvert\rvert < \epsilon$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Elseif k==n, then $x_1 = x_{k+1}, k = 1$ Else $k = k + 1$
Conjugate Gradient	Without Using Line Search $\alpha_k = \frac{g_k^T g_k}{d_k^T H_k d_k}$	$d_0 = -g_0$ $x_{k+1} = x_k + \alpha_k d_k$ $\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k}$ $d_{k+1} = -g_{k+1} + \beta_k d_k$ $f_{k+1} = f(x_{k+1})$	If $\lvert\lvert \alpha_k d_k \rvert\rvert < \epsilon$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$
Quasi-Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \delta_k$ $\delta_k = \alpha_k d_k;\ \ d_k = -S_k g_k$ $x_{k+1} = x_k - \alpha_k S_k g_k$ $Compute\ g_{k+1}\ \ /* = g_k + H\delta_k\ */$ $S_0 = I_n$ $S_{k+1} = S_k + \frac{(\delta_k - S_k\gamma_k)(\delta_k - S_k\gamma_k)^T}{\gamma_k^T(\delta_k - S_k\gamma_k)}$ $\gamma_k = g_{k+1} - g_k$	If $\lvert\lvert \delta_k \rvert\rvert < \epsilon$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$

is used to update the approximate Hessian B_{k+1}, or directly its inverse $H_{k+1} = B_{k+1}^{-1}$ using the Sherman-Morrison formula.

- A key property of the BFGS and DFP updates is that if B_k is positive definite and α_k is chosen to satisfy the Wolfe conditions then B_{k+1} is also positive definite.

The most popular update formulas are:

Method	$B_{k+1} =$	$H_{k+1} = B_{k+1}^{-1} =$
DFP	$\left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right) B_k \left(I - \frac{\Delta x_k y_k^T}{y_k^T \Delta x_k}\right) + \frac{y_k y_k^T}{y_k^T \Delta x_k}$	$H_k + \frac{\Delta x_k \Delta x_k^T}{y_k^T \Delta x_k} - \frac{H_k y_k y_k^T H_k^T}{y_k^T H_k y_k}$
BFGS	$B_k + \frac{y_k y_k^T}{y_k^T \Delta x_k} - \frac{B_k \Delta x_k (B_k \Delta x_k)^T}{\Delta x_k^T B_k \Delta x_k}$	$\left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right)^T H_k \left(I - \frac{y_k \Delta x_k^T}{y_k^T \Delta x_k}\right) + \frac{\Delta x_k \Delta x_k^T}{y_k^T \Delta x_k}$
Broyden	$B_k + \frac{y_k - B_k \Delta x_k}{\Delta x_k^T \Delta x_k} \Delta x_k^T$	$H_k + \frac{(\Delta x_k - H_k y_k) \Delta x_k^T H_k}{\Delta x_k^T H_k y_k}$
Broyden family	$(1 - \varphi_k) B_{k+1}^{BFGS} + \varphi_k B_{k+1}^{DFP}, \qquad \varphi \in [0, 1]$	
SR1	$B_k + \frac{(y_k - B_k \Delta x_k)(y_k - B_k \Delta x_k)^T}{(y_k - B_k \Delta x_k)^T \Delta x_k}$	$H_k + \frac{(\Delta x_k - H_k y_k)(\Delta x_k - H_k y_k)^T}{(\Delta x_k - H_k y_k)^T y_k}$

Non-linear least squares

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

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

- If each residual depends non-linearly on the parameters $\mathbf{x}$ then the minimization of $f(\mathbf{x})$ is a non-linear least squares problem.

Non-linear least squares

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

- The M × N Jacobian of the vector of residuals r is defined as

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

- Consider

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

- Hence

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

Non-linear least squares

- For the Hessian holds

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

$$\mathbf{H}(\mathbf{x}) \approx \quad 2\mathbf{J}^T\mathbf{J}$$

Gauss-Newton approximation

- Note that the second-order term in the Hessian is multiplied by the residuals r_i.
- 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.
- Hence, explicit computation of the full Hessian can again be avoided.

Gauss-Newton example

- The minimization of the Rosenbrock function

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

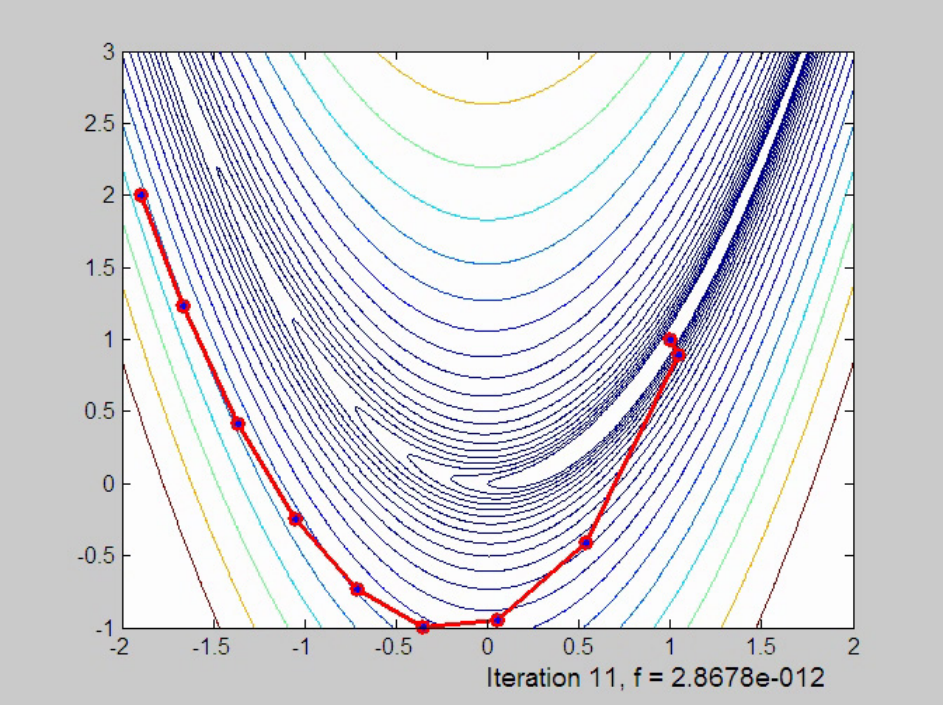
- can be written as a least-squares problem with residual vector

$$\mathbf{r} = \begin{bmatrix} 10(y - x^2) \\ (1 - x) \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial r_1}{\partial x} & \frac{\partial r_1}{\partial y} \\ \frac{\partial r_2}{\partial x} & \frac{\partial r_2}{\partial y} \end{bmatrix} = \begin{bmatrix} -20x & 10 \\ -1 & 0 \end{bmatrix}$$

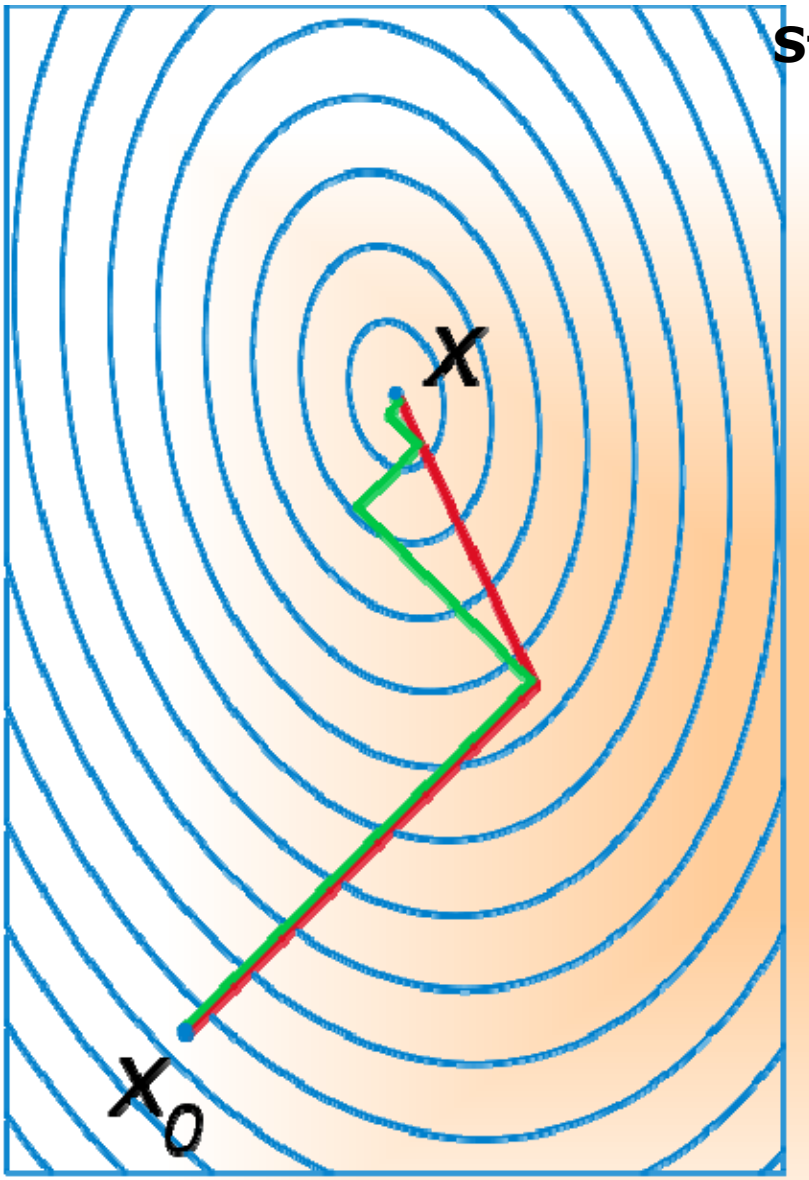
Gauss-Newton example

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{H}_k^{-1} \mathbf{g}_k \qquad \mathbf{H}_k = 2\mathbf{J}_k^T \mathbf{J}$$

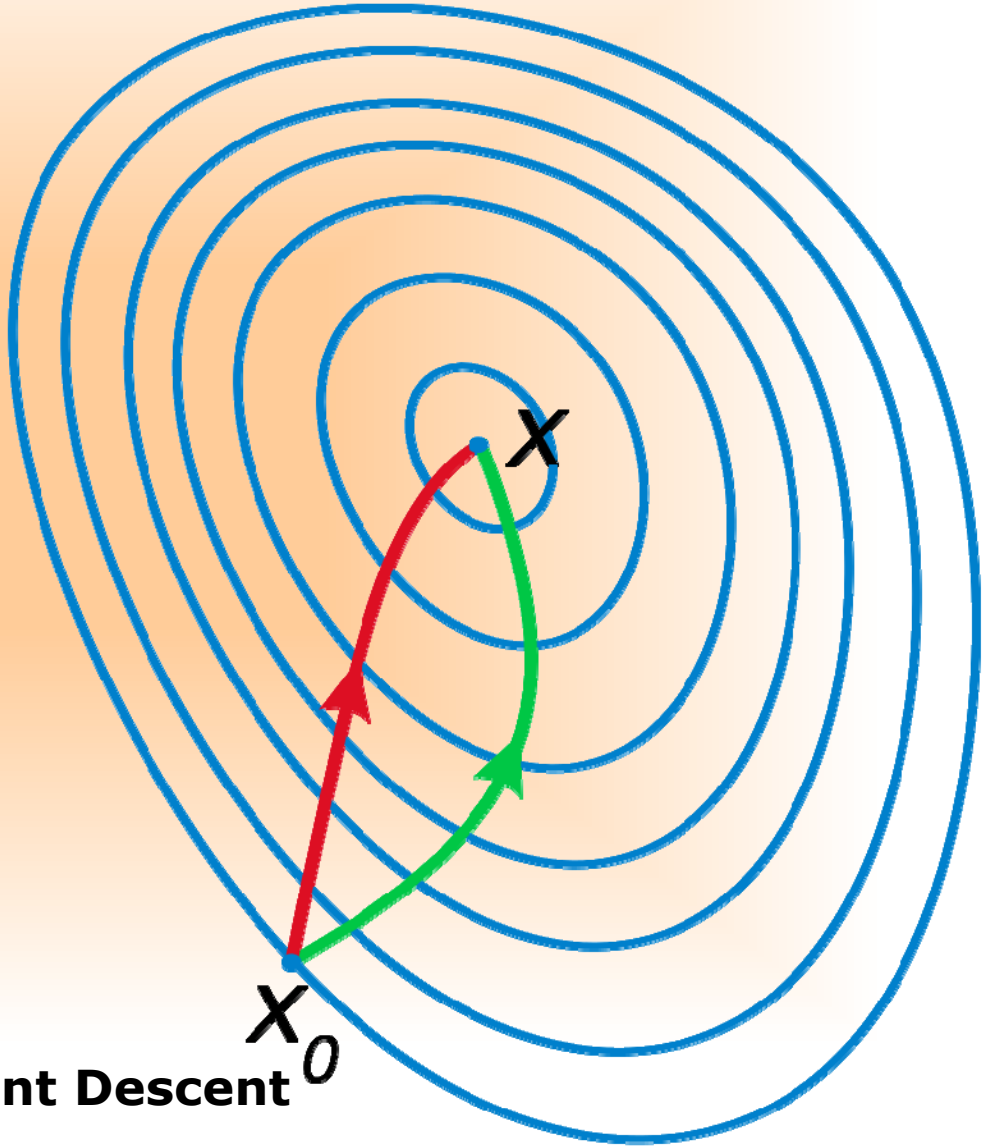


- minimization with the Gauss-Newton approximation with line search takes only 11 iterations

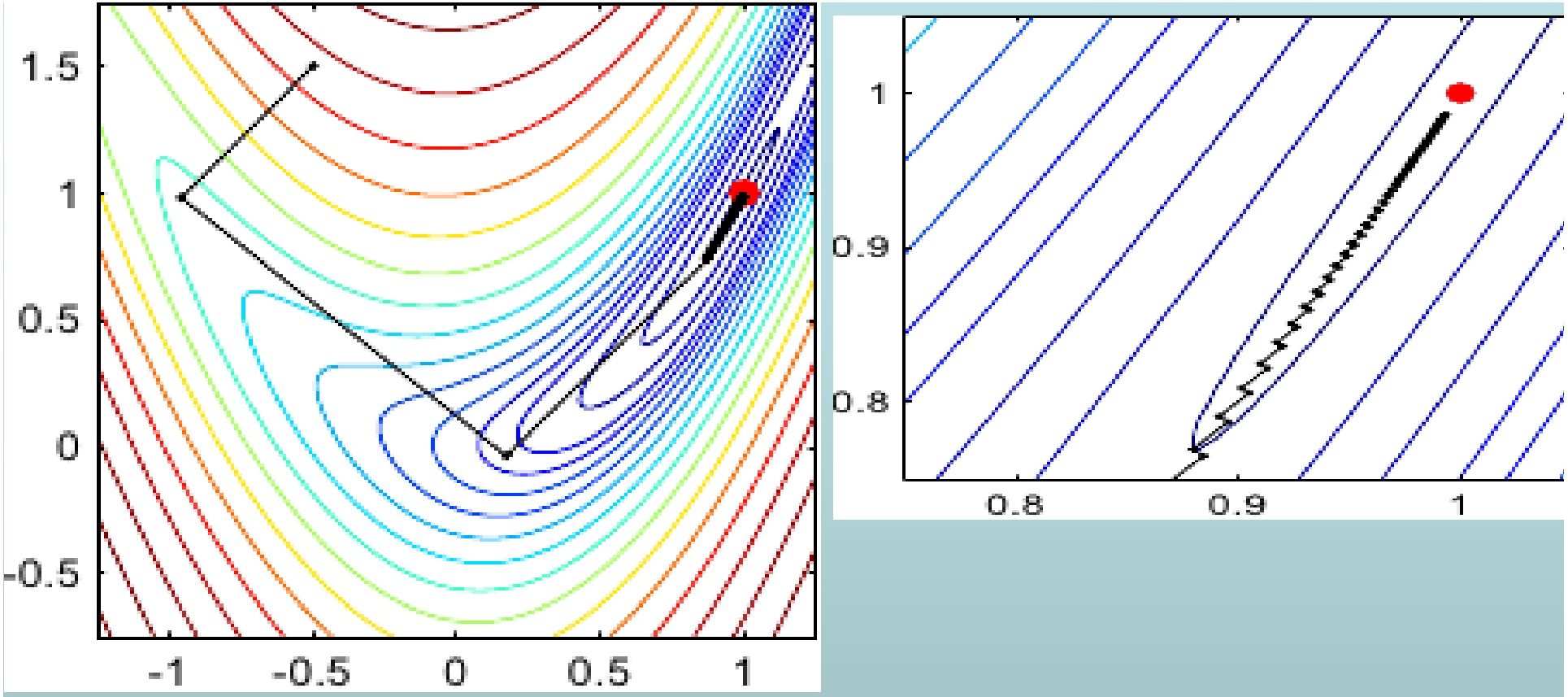
Method	Alpha (α) Calculation	Intermediate Updates	Convergence Condition
Steepest-Descent Method (Method 1)	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	If $\|\|\alpha_k d_k\|\| < \epsilon$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$
Steepest-Descent Method (Method 2)	Without Using Line Search $\alpha_k \approx \frac{g_k^T g_k}{g_k^T H_k g_k}$	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -g_k$ $f_{k+1} = f(x_{k+1})$	--" "--
Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -H_k^{-1} g_k$ $f_{k+1} = f(x_{k+1})$	--" "--
Gauss-Newton Method	Find α_k, the value of α that minimizes F$(x_k + \alpha d_k)$, using line search $F = \sum_{p=1}^{m} f_p(x)^2 = f^T f$ $f = [f_1(x)\ f_2(x) \dots f_m(x)]^T$	$x_{k+1} = x_k + \alpha_k d_k$ $d_k = -H_k^{-1} g_k$ $g_F = 2J^T f$ $H \approx 2J^T J = \boldsymbol{L^{-1} D (L^T)^{-1}}$ $x_{k+1} = x_k - \alpha_k (J^T J)^{-1} (J^T f)$ $x_{k+1} = x_k - \alpha_k \boldsymbol{L^T D L} g_k$ $f_{p(k+1)} = f_p(x_k)$ $F_{(k+1)} = F(x_k)$	If $\|F_{k+1} - F_k\| < \epsilon$ then $x^* = x_{k+1}$, F$(x^*) = F_{k+1}$ Else $k = k + 1$



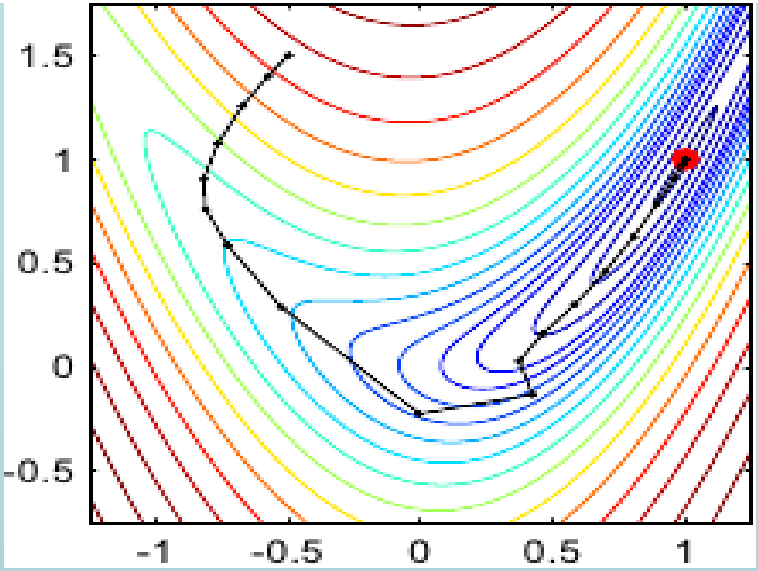
Steepest Descent vs Conjugate Gradient



Newton vs Gradient Descent



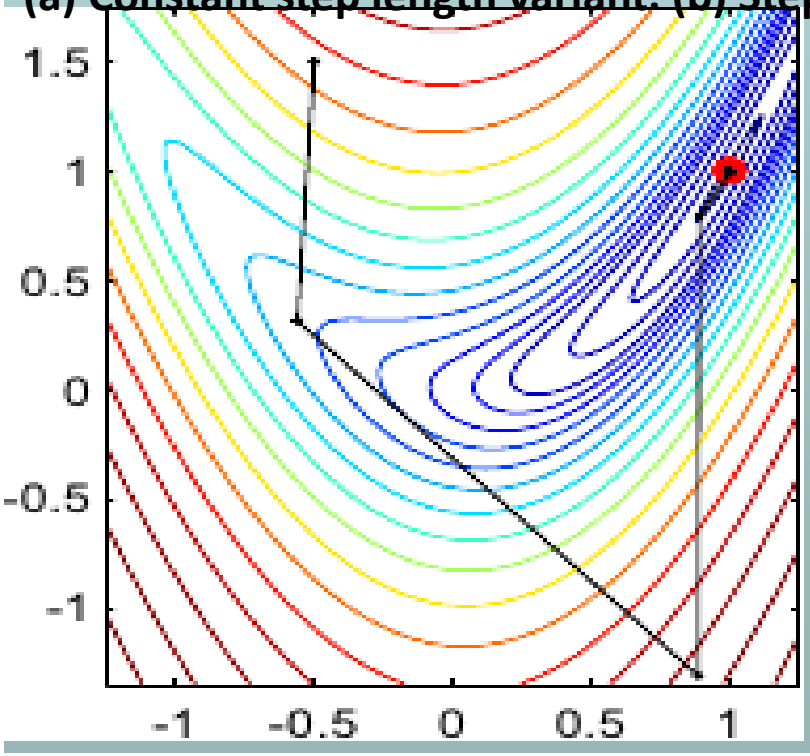
The successive steps taken by the method of the gradient descent for optimizing the Rosenbrock functions

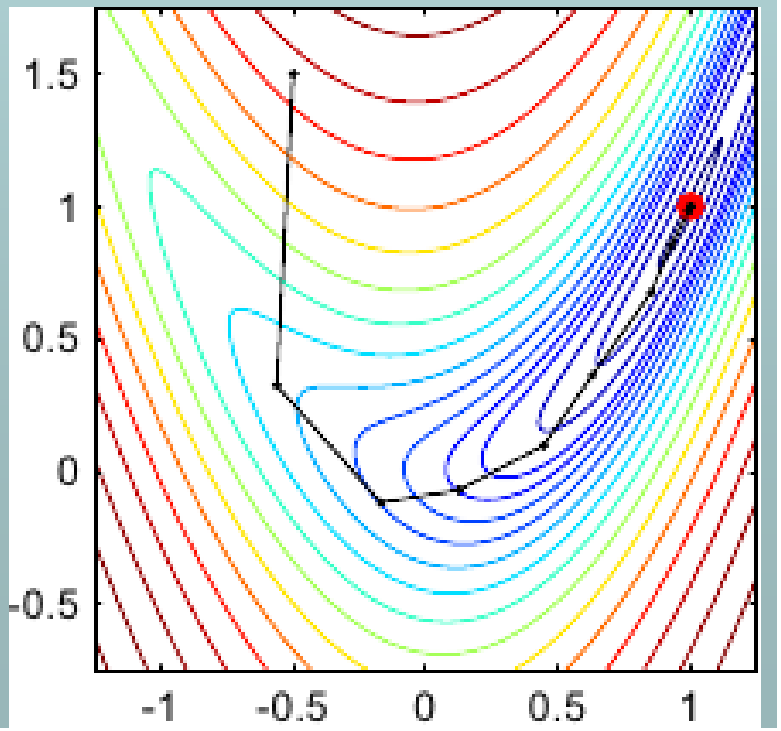


Levenberg-Marquardt algorithm for optimizing the Rosenbrock function

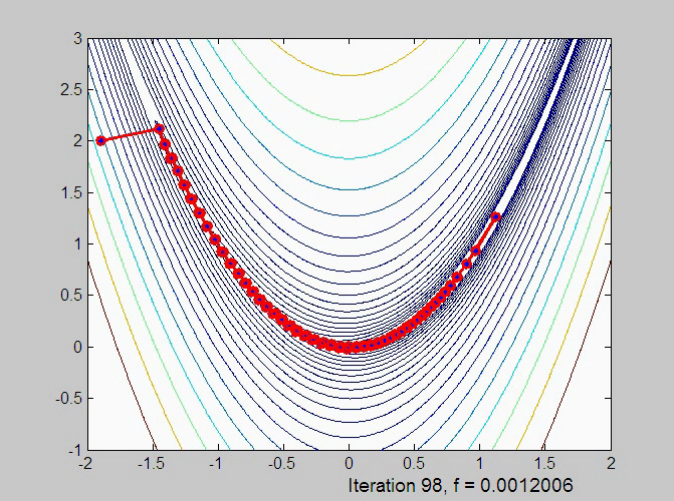
Newton's method for optimizing the Rosenbrock functions.

(a) Constant step length variant. (b) Step length determined with an optimal line search strategy

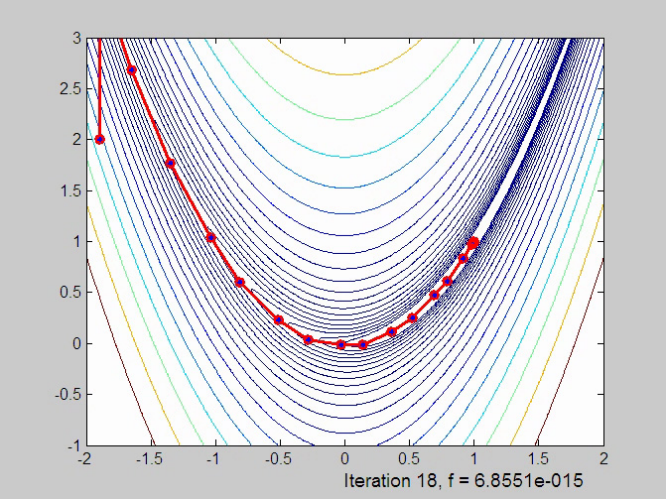




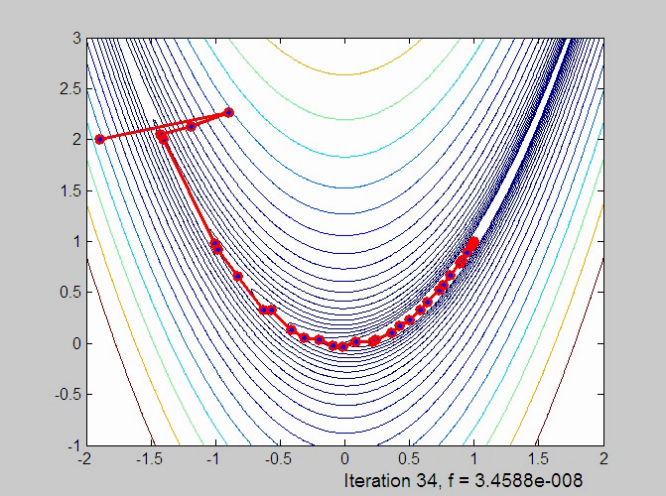
Comparison



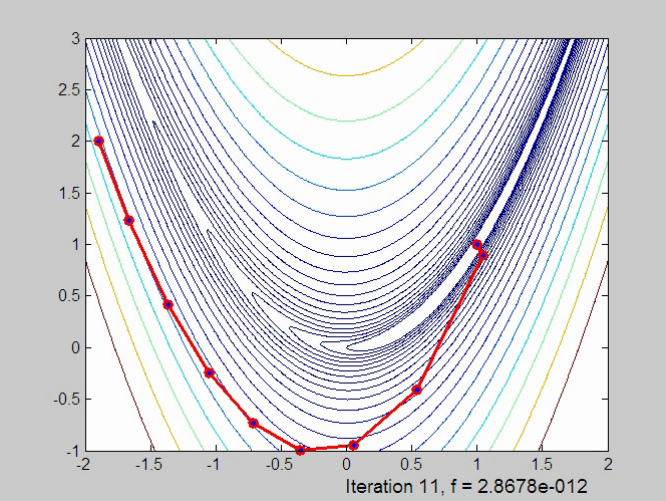
CG



Newton

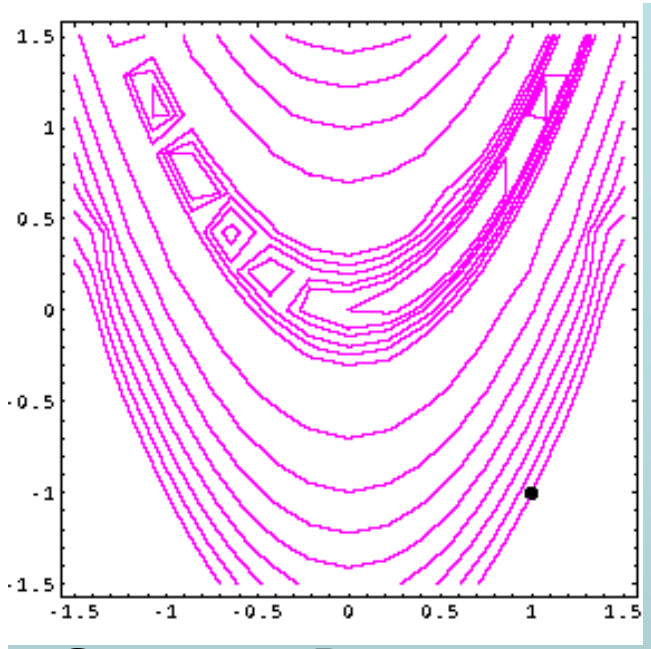


Quasi-Newton

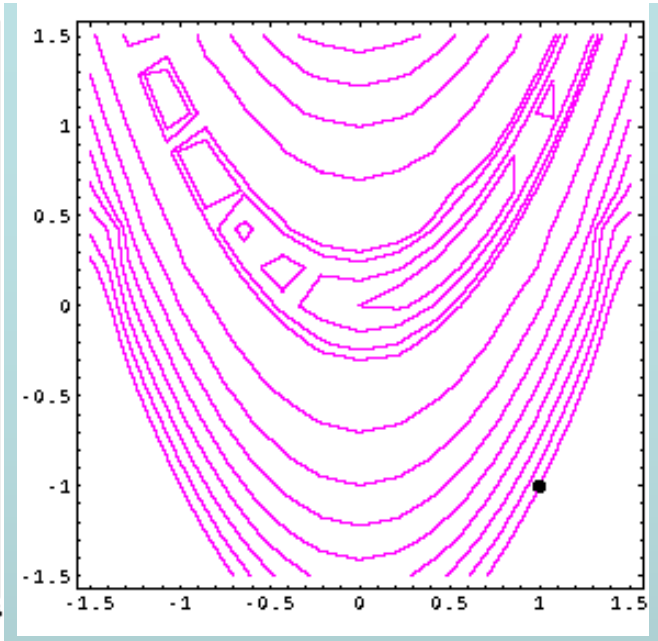


Gauss-Newton

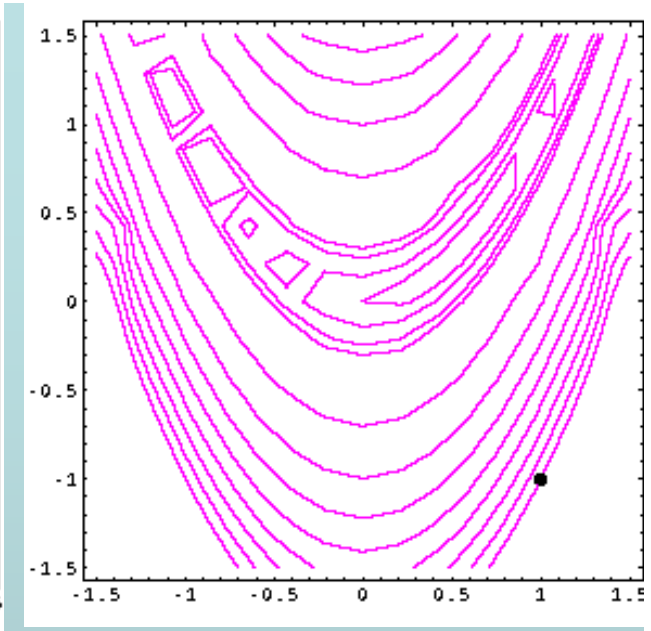
Method	(α) Calculation	Intermediate Updates	Conv. Condition
Conjugate Gradient	Without Using Line Search $$\alpha_k = \frac{g_k^T g_k}{d_k^T H_k d_k}$$	$d_0 = -g_0;\quad x_{k+1} = x_k + \alpha_k d_k$ $$\beta_k = \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k};$$ $$d_{k+1} = -g_{k+1} + \beta_k d_k$$ $$f_{k+1} = f(x_{k+1})$$	If $\lvert\lvert \alpha_k d_k \rvert\rvert < \in$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$
Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$$x_{k+1} = x_k + \alpha_k d_k$$ $$d_k = -H_k^{-1} g_k$$ $$f_{k+1} = f(x_{k+1})$$	--" "--
Gauss-Newton Method	Use line search, for α_k $$F = \sum_{p=1}^{m} f_p(x)^2 = f^T f$$ $f = [f_1(x)\ f_2(x) \dots f_m(x)]^T$	$x_{k+1} = x_k + \alpha_k d_k;\ d_k = -H_k^{-1} g_k$ $g_F = 2J^T f; H \approx 2J^T J = \boldsymbol{L^{-1} D (L^T)^{-1}}$ $$x_{k+1} = x_k - \alpha_k (J^T J)^{-1} (J^T f)$$ $$x_{k+1} = x_k - \alpha_k \boldsymbol{L^T D L} g_k$$ $$f_{p(k+1)} = f_p(x_k);\ F_{(k+1)} = F(x_k)$$	If $\lvert F_{k+1} - F_k \rvert < \in$ then $x^* = x_{k+1}$, $F(x^*) = F_{k+1}$ Else $k = k + 1$
Quasi-Newton Method	Find α_k, the value of α that minimizes $f(x_k + \alpha d_k)$, using line search	$x_{k+1} = x_k + \delta_k;\quad \delta_k = \alpha_k d_k;$ $d_k = -S_k g_k;\ x_{k+1} = x_k - \alpha_k S_k g_k$ $$g_{k+1} = g_k + H\delta_k;\ S_0 = I_n$$ S_{k+1} $$= S_k + \frac{(\delta_k - S_k \gamma_k)(\delta_k - S_k \gamma_k)^T}{\gamma_k^T (\delta_k - S_k \gamma_k)}$$ $\gamma_k = g_{k+1} - g_k$	If $\lvert\lvert \delta_k \rvert\rvert < \in$, then $x^* = x_{k+1}$, $f(x^*) = f_{k+1}$ Else $k = k + 1$



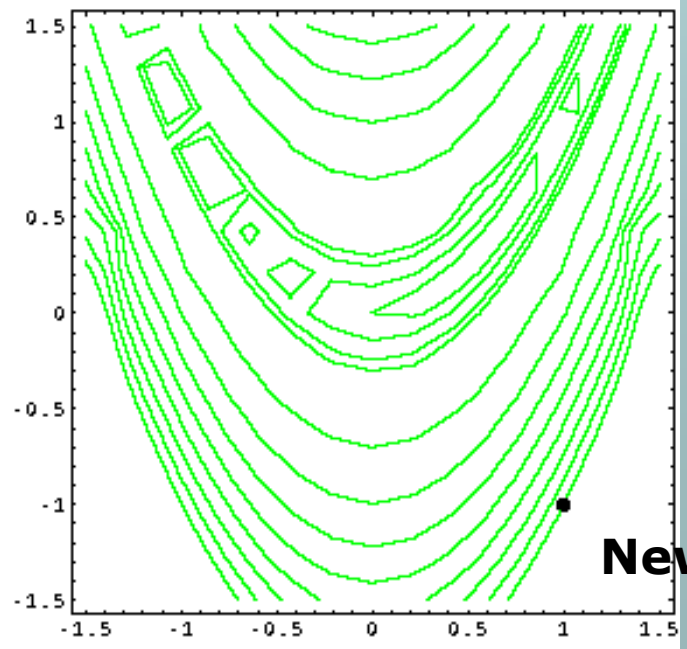
Steepest Descent



Fletcher Reeve's Conjugate Gradient

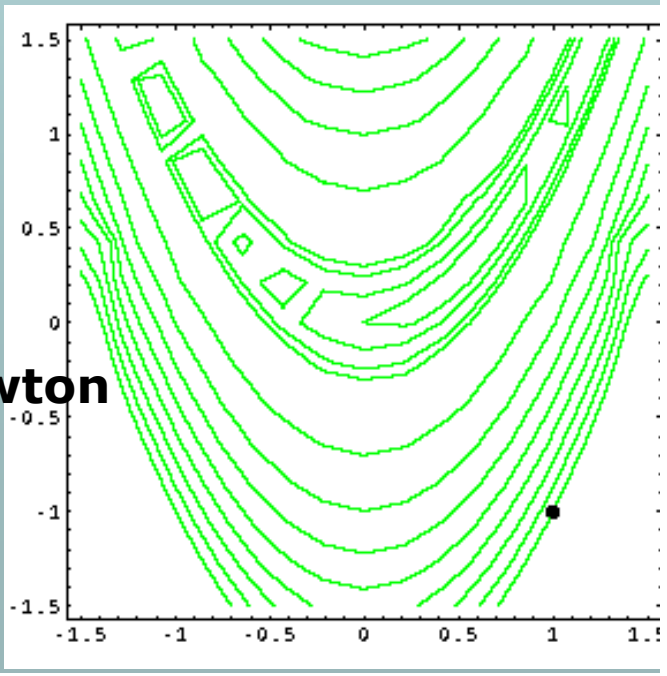


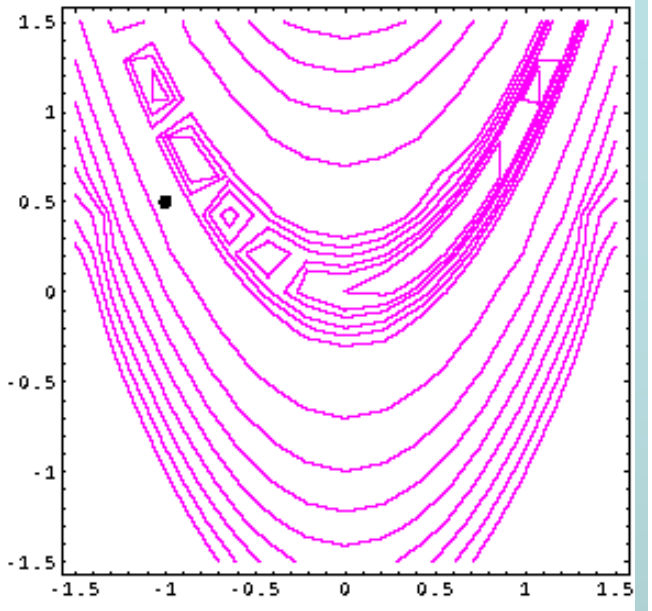
Powels's Conjugate Gradient



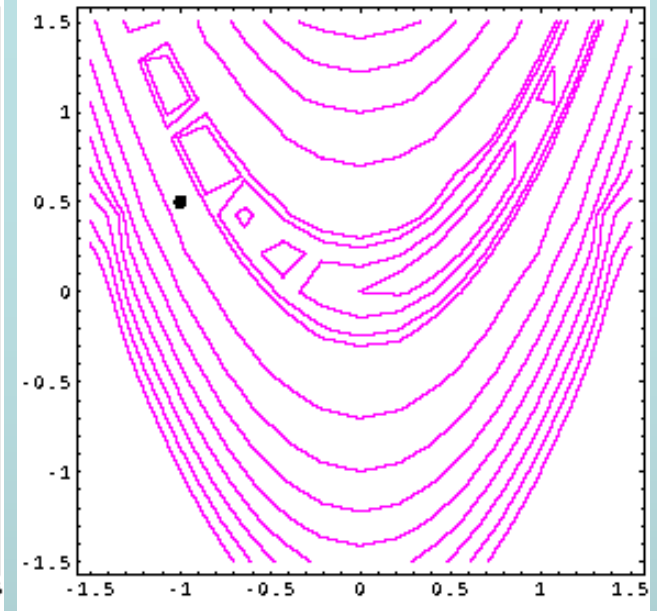
Newton's

Quasi-Newton

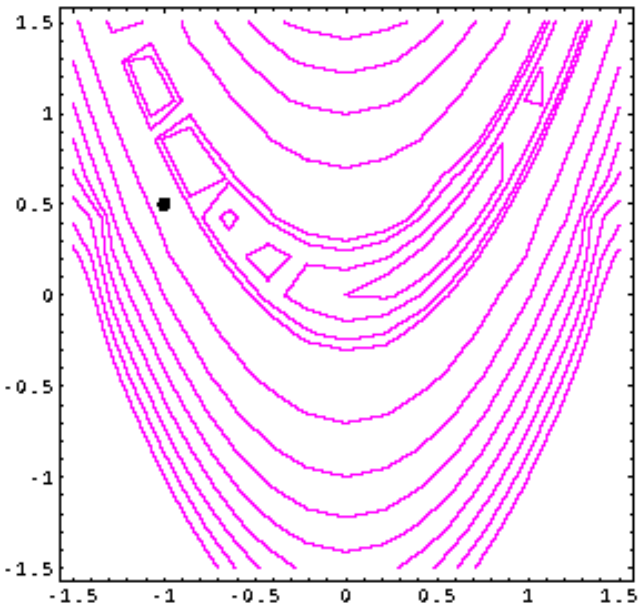




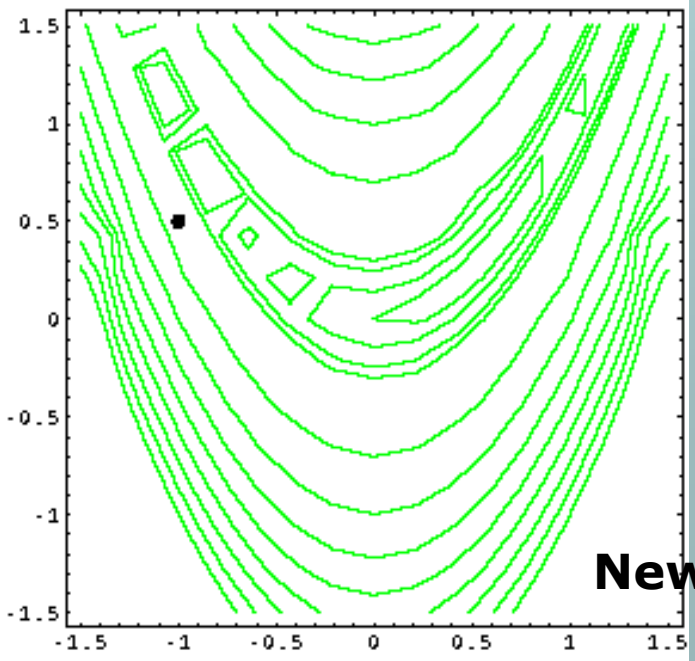
Steepest Descent



Fletcher Reeve's Conjugate Gradient

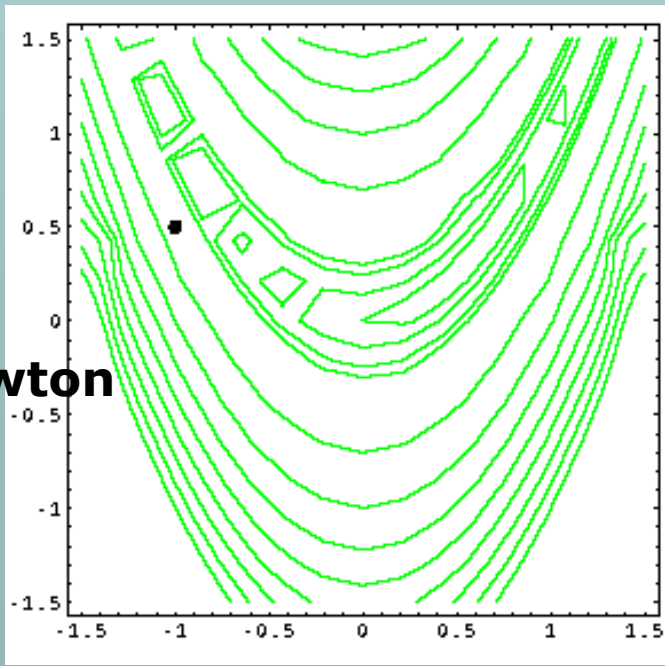


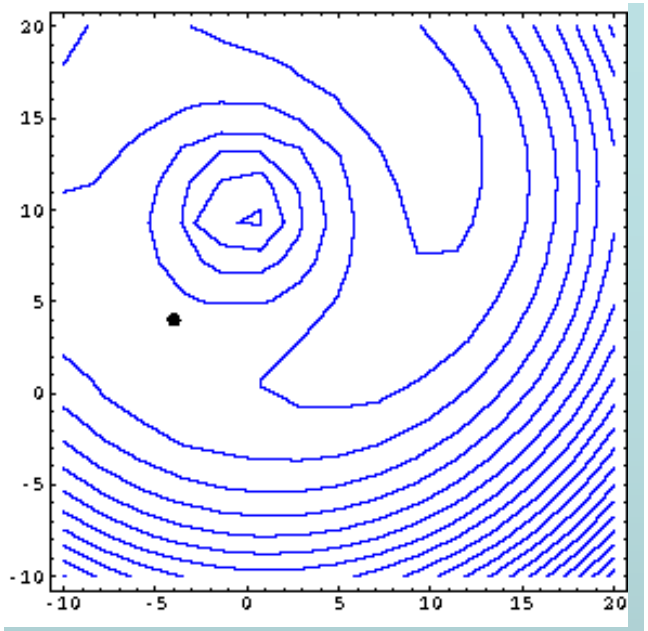
Powels's Conjugate Gradient



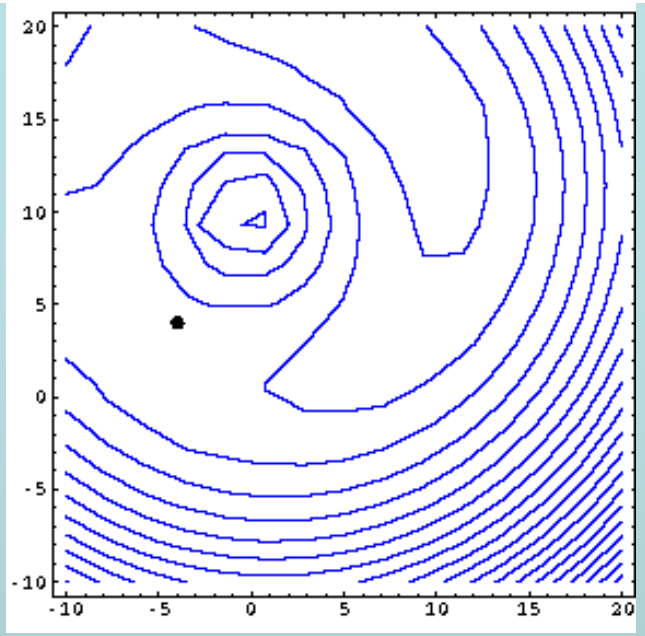
Newton's

Quasi-Newton

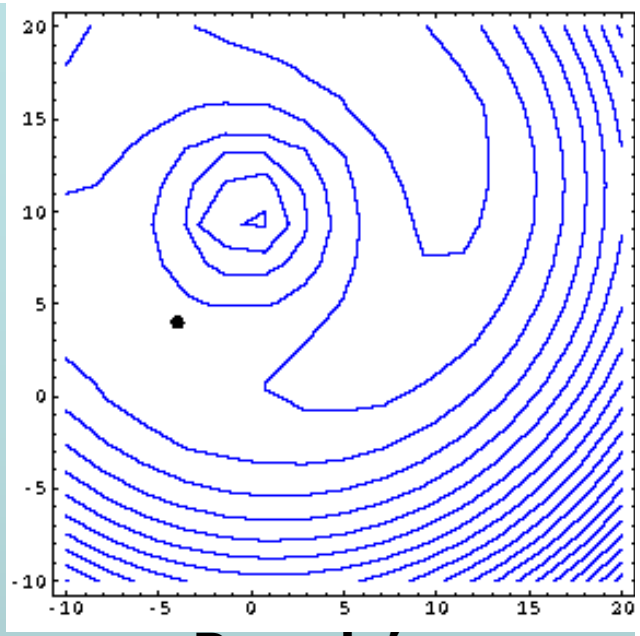




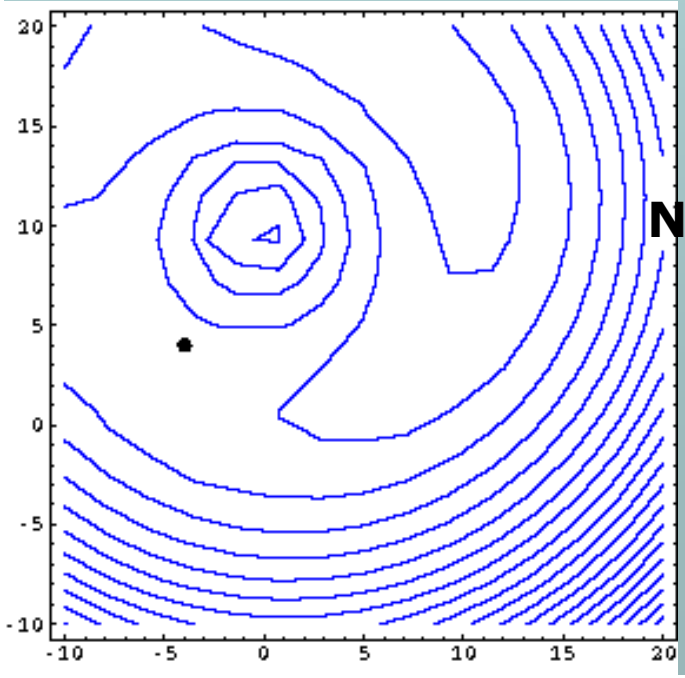
Steepest Descent



Fletcher Reeve's Conjugate Gradient

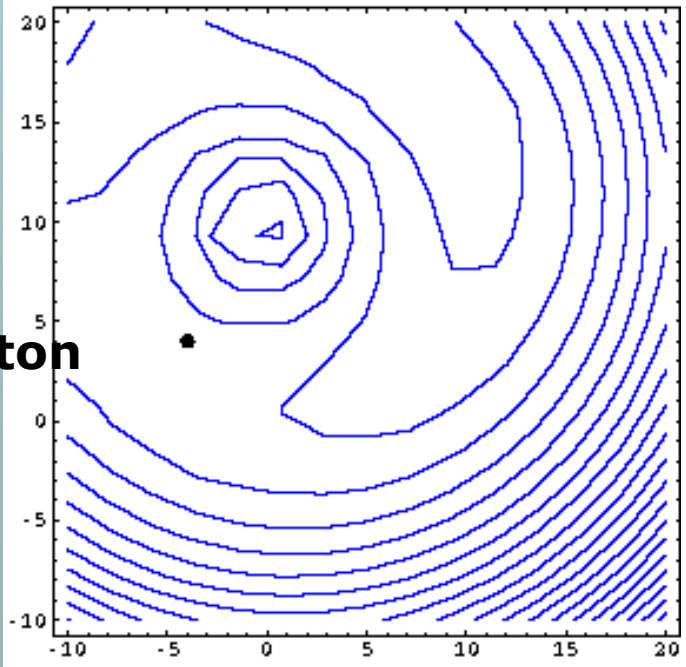


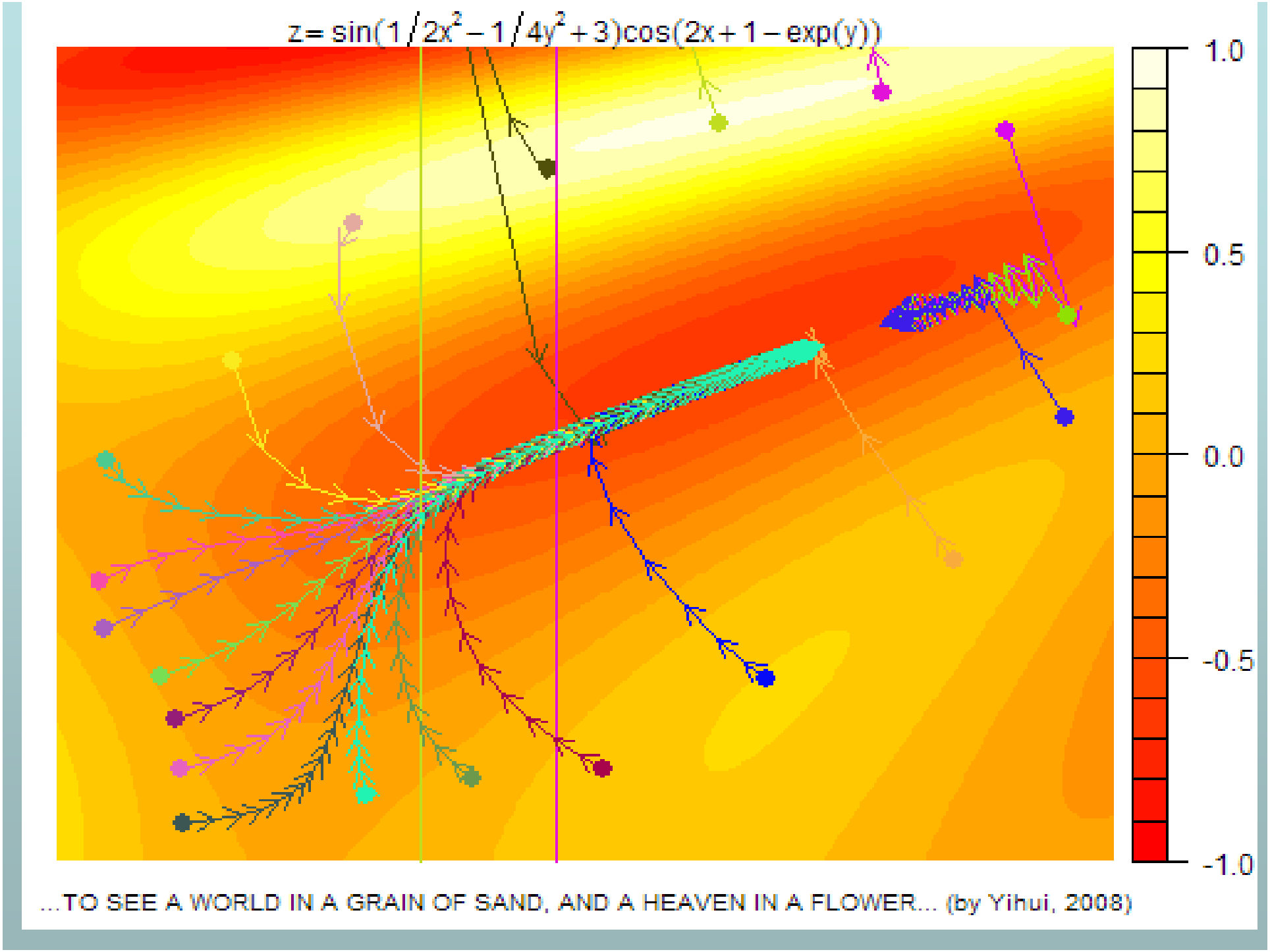
Powels's Conjugate Gradient



Newton's

Quasi-Newton




z= sin(1/2x^2 − 1/4y^2 + 3)cos(2x + 1 − exp(y))
1.0
0.5
0.0
-0.5
-1.0
...TO SEE A WORLD IN A GRAIN OF SAND, AND A HEAVEN IN A FLOWER... (by Yihui, 2008)

2. Momentum and Nesterov's Accelerated Gradient

Ilya Sutskever[1] ILYASU@GOOGLE.COM
James Martens JMARTENS@CS.TORONTO.EDU
George Dahl GDAHL@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU

The momentum method (Polyak, 1964), w
to as classical momentum (CM), is a tech
celerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, classical momentum is given by:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t) \quad (1)$$
$$\theta_{t+1} = \theta_t + v_{t+1} \quad (2)$$

where $\varepsilon > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\nabla f(\theta_t)$ is the gradient at θ_t.

Since directions d of low-curvature have, by definition, slower local change in their rate of reduction (i.e., $d^\top \nabla f$), they will tend to persist across iterations and be amplified by CM. Second-order methods also amplify steps in low-curvature directions, but instead of accumulating changes they reweight the update along each eigen-direction of the curvature matrix by the inverse of the associated curvature. And just as second-

n. While NAG is not typically thought of as a
momentum, it indeed turns out to be closely re-
o classical momentum, differing only in the pre-
cise update of the velocity vector v, the significance of which we will discuss in the next sub-section. Specifically, as shown in the appendix, the NAG update may be rewritten as:

$$v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t) \quad (3)$$
$$\theta_{t+1} = \theta_t + v_{t+1} \quad (4)$$

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

Algorithm 1 The Hessian-free optimization method

1: **for** $n = 1, 2, \ldots$ **do**
2: $g_n \leftarrow \nabla f(\theta_n)$
3: compute/adjust λ by some method
4: define the function $B_n(d) = \mathbf{H}(\theta_n)d + \lambda d$
5: $p_n \leftarrow \text{CG-Minimize}(B_n, -g_n)$
6: $\theta_{n+1} \leftarrow \theta_n + p_n$
7: **end for**

In the standard Newton's method, $q_\theta(p)$ is optimized by computing the $N \times N$ matrix B and then solving the system $\mathrm{B}p = -\nabla f(\theta)$. This is prohibitively expensive when N is large, as it is with even modestly sized neural networks. Instead, HF optimizes $q_\theta(p)$ by exploiting two simple ideas. The first is that for an N-dimensional vector d, $\mathrm{H}d$ can be easily computed using finite differences at the cost of a single extra gradient evaluation via the identity:

$$\mathrm{H}d = \lim_{\epsilon \to 0} \frac{\nabla f(\theta + \epsilon d) - \nabla f(\theta)}{\epsilon}$$

Deep learning via Hessian-free optimization

James Martens

JMARTENS@CS.TORONTO.EDU

University of Toronto, Ontario, M5S 1A1, Canada

proceedings of the 27 th International Conference on Machine Learning, , Haifa, Israel, 2010.

On Optimization Methods for Deep Learning

Quoc V. Le quocle@cs.stanford.edu; Jiquan Ngiam jngiam@cs.stanford.edu; Adam Coates acoates@cs.stanford.edu; Abhik Lahiri alahiri@cs.stanford.edu; Bobby Prochnow prochnow@cs.stanford.edu; Andrew Y. Ng; **ICML'11**

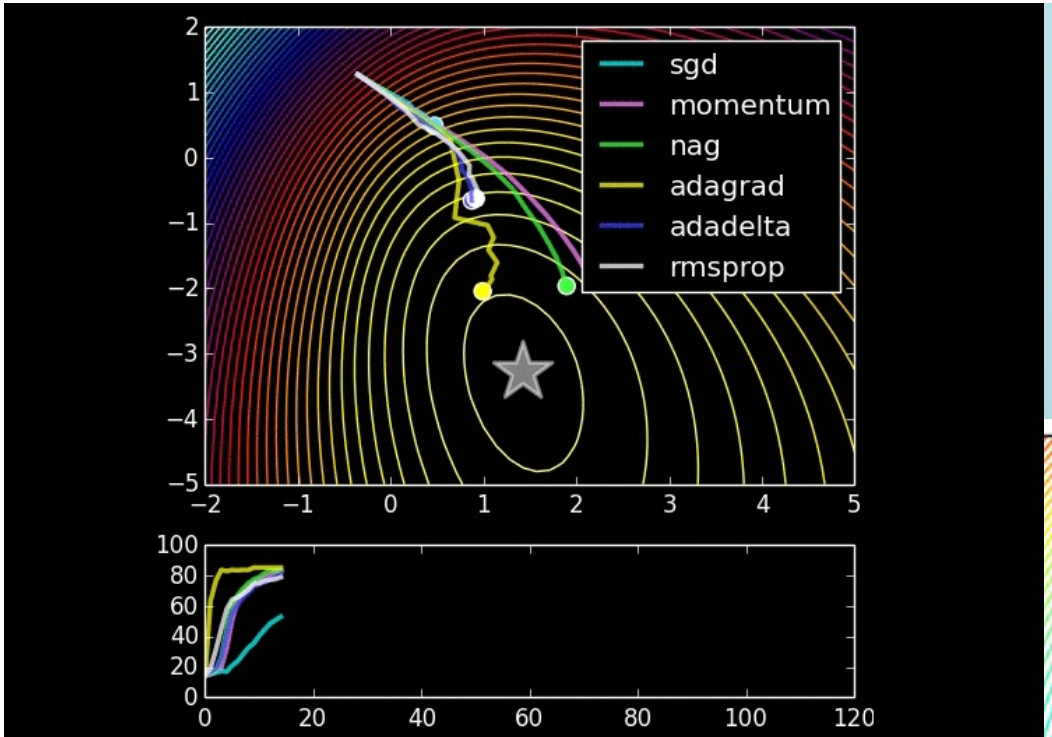
Successful unconstrained optimization methods include Newton-Raphson's method, BFGS methods, Conjugate Gradient methods and Stochastic Gradient Descent methods.

These methods are usually associated with a line search method to ensure that the algorithms consistently improve the objective function. When it comes to large scale machine learning, the favorite optimization method is usually SGDs. Recent work on SGDs focuses on adaptive strategies for the learning rate for improving SGD convergence by approximating second-order information.

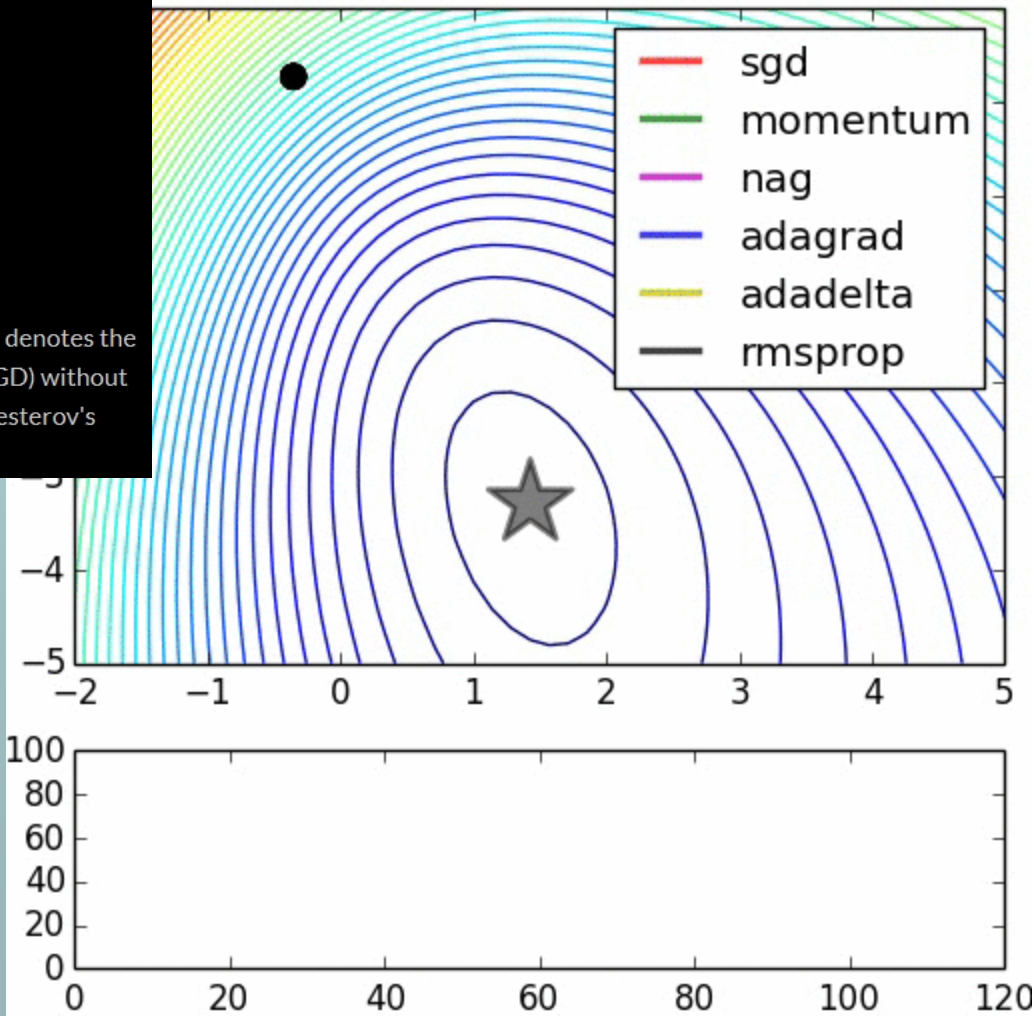
In practice, plain SGDs with constant learning rates or learning rates of the form (α/β+t) are still popular thanks to their ease of implementation. These simple methods are even more common in deep learning because the optimization problems are nonconvex and the convergence properties of complex methods no longer hold.

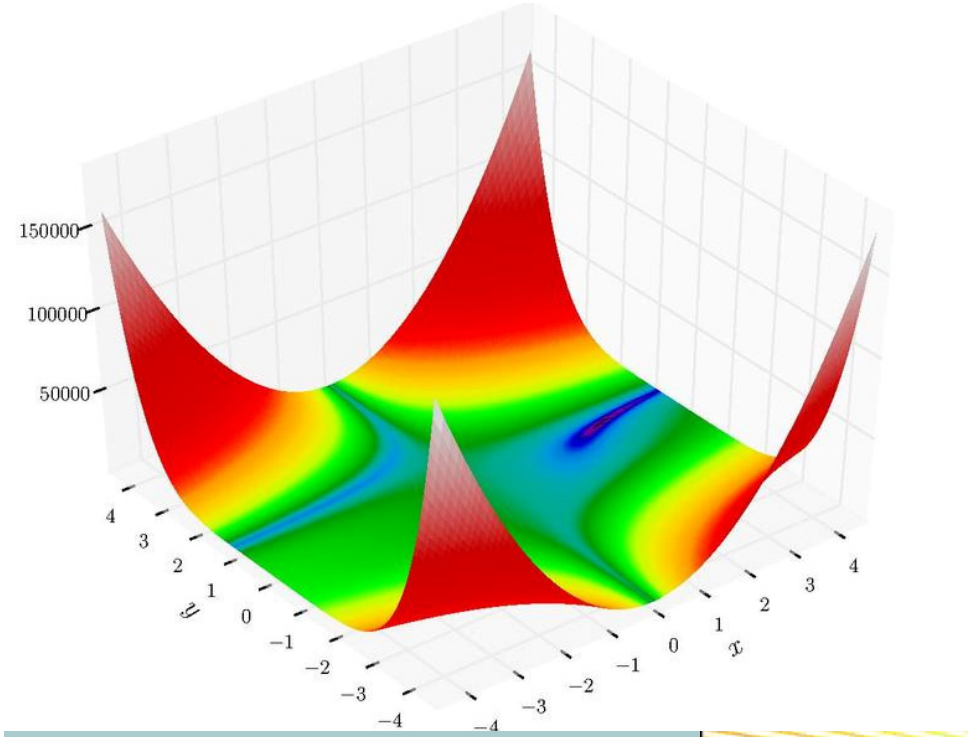
Recent proposals for training deep networks argue for the use of layer-wise pre-training. Optimization techniques for training these models include Contrastive Divergence, Conjugate Gradient, stochastic diagonal Levenberg-Marquardt and Hessian-free optimization. Convolutional neural networks have traditionally employed SGDs with the stochastic diagonal Levenberg-Marquardt, which uses a diagonal approximation to the Hessian.

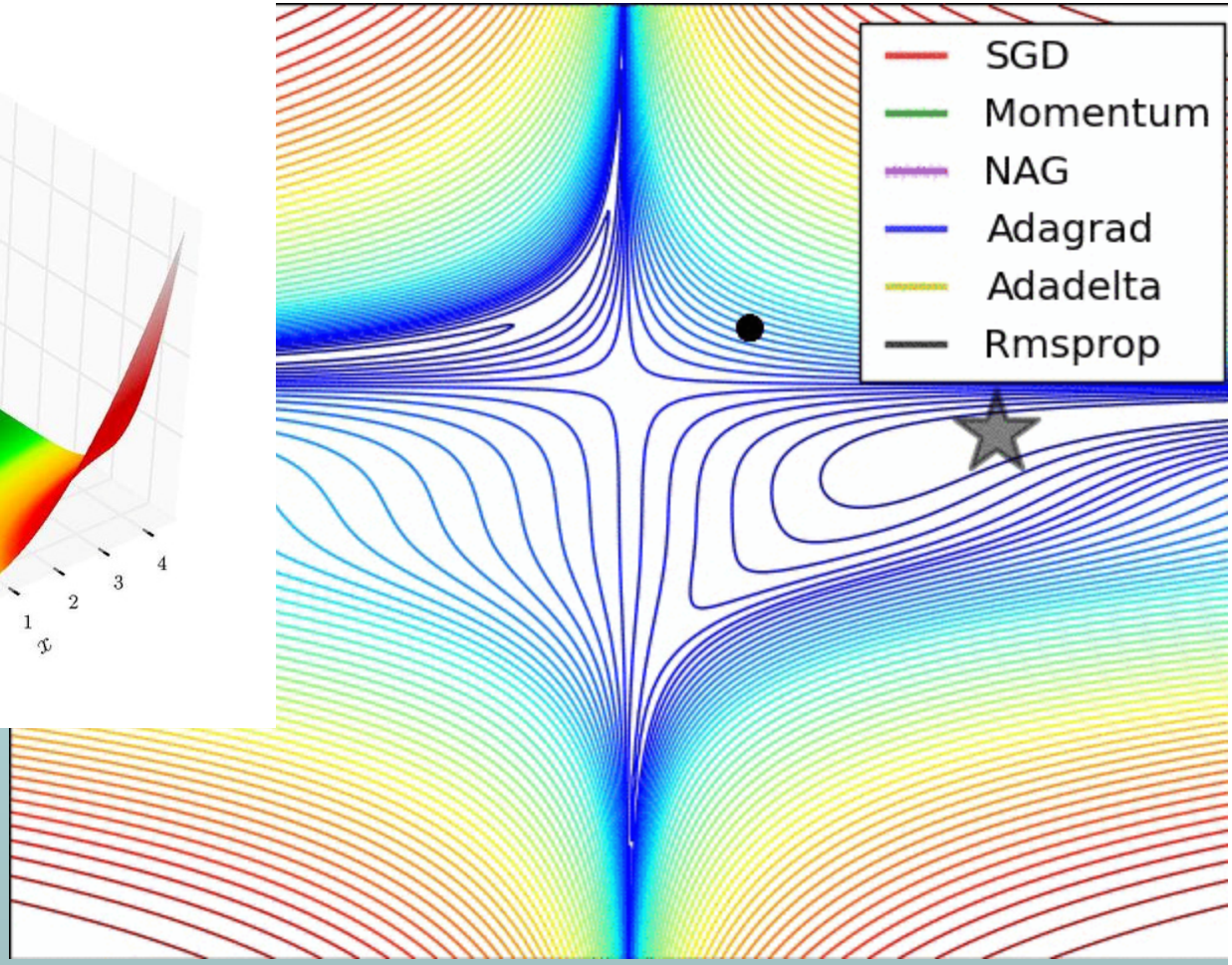
In this paper, it is our goal to empirically study the pros and cons of off-the-shelf optimization algorithms in the context of unsupervised feature learning and deep learning. In that direction, we focus on comparing L-BFGS, CG and SGDs.



Comparison of a few optimization methods (animation by Alec Radford). The star denotes the global minimum on the error surface. Notice that stochastic gradient descent (SGD) without momentum is the slowest method to converge in this example. We're using Nesterov's Accelerated Gradient Descent (NAG) throughout this tutorial.



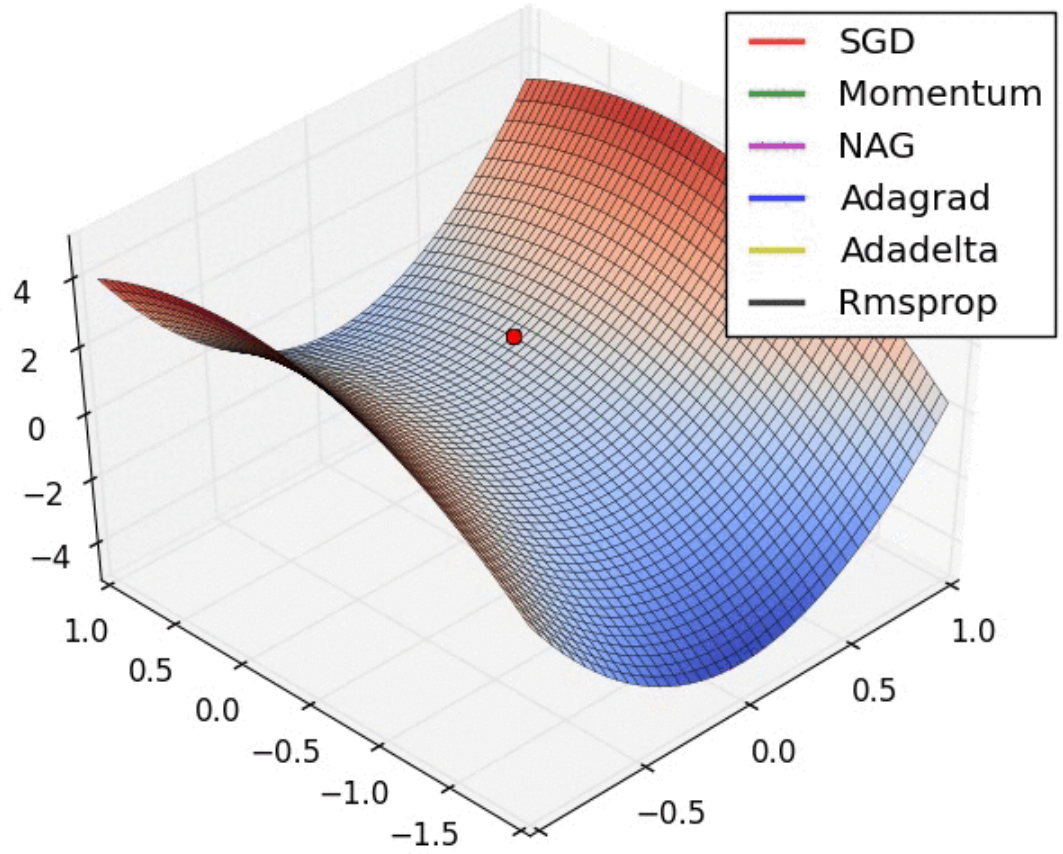




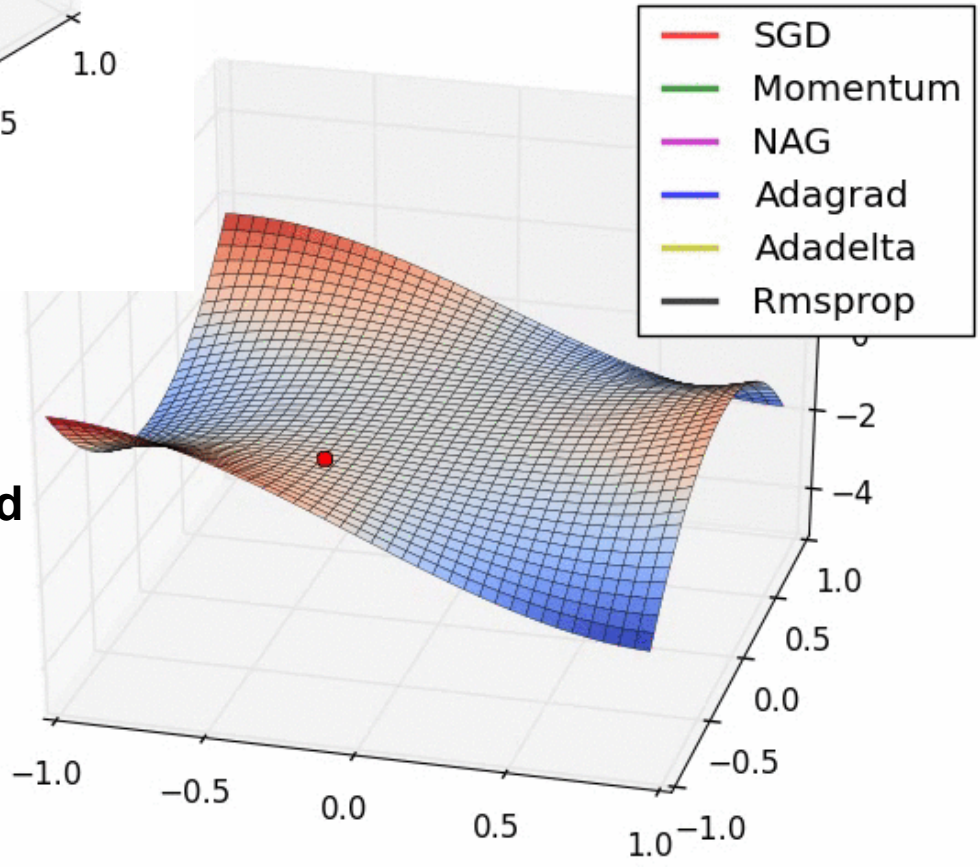
"Beale's function: Due to the large initial gradient, velocity based techniques shoot off and bounce around - adagrad almost goes unstable for the same reason. Algos that scale gradients/step sizes like adadelta and RMSProp proceed more like accelerated SGD and handle large gradients with more stability."

$$f(\mathbf{x}) = (1.5 - x_1 + x_1 x_2)^2 + (2.25 - x_1 + x_1 x_2^2)^2 + (2.625 - x_1 + x_1 x_2^3)^2$$

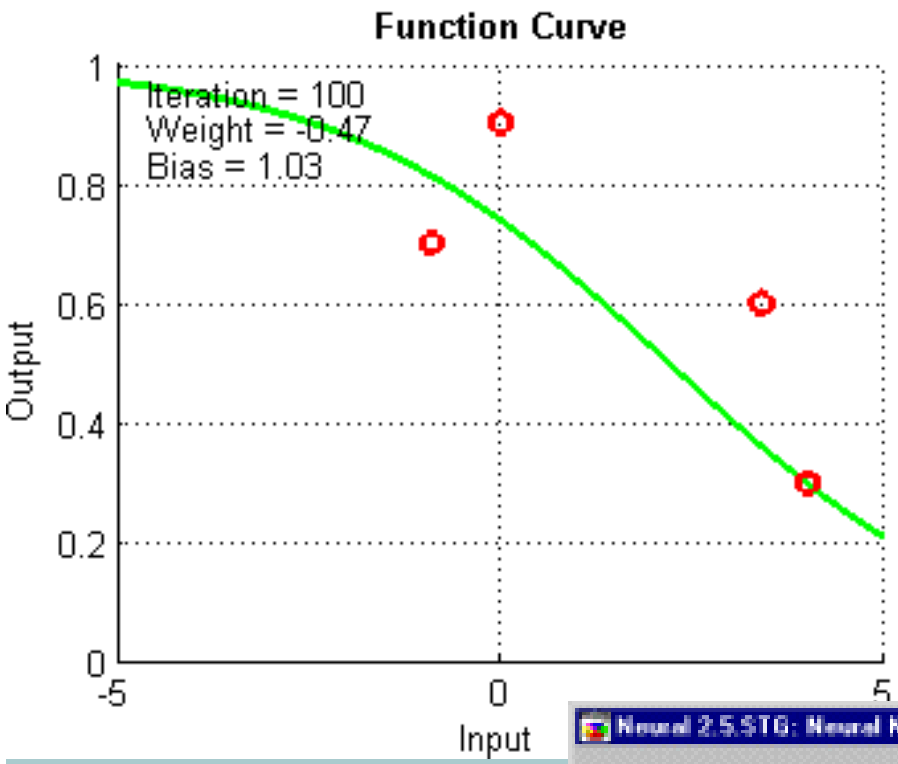
Global Minimum: $f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (3, 0.5)$

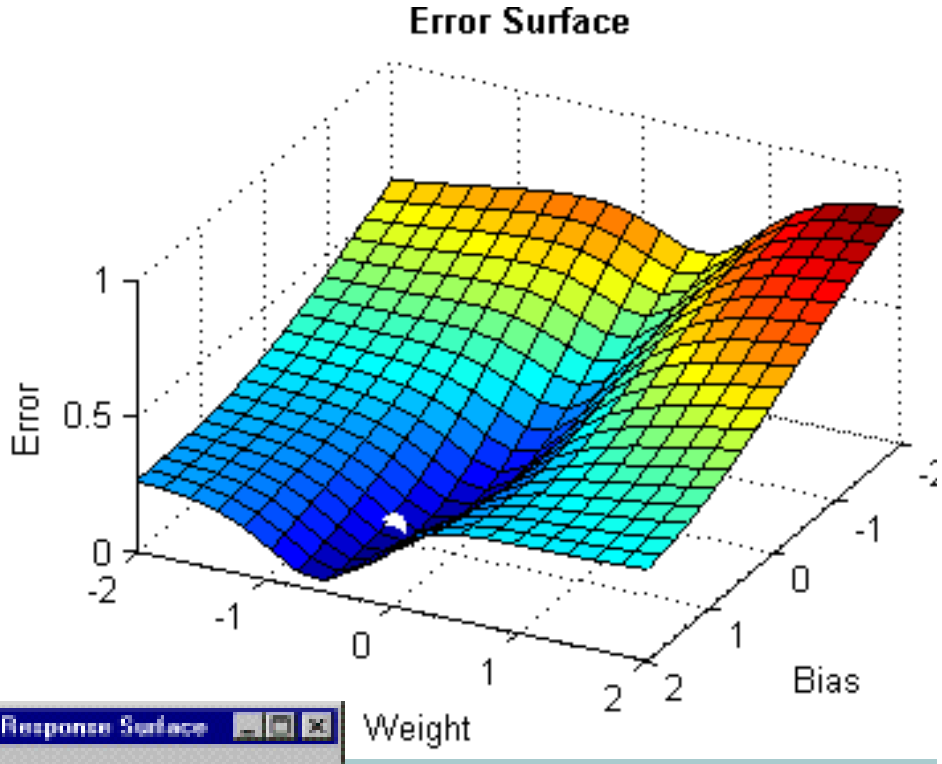


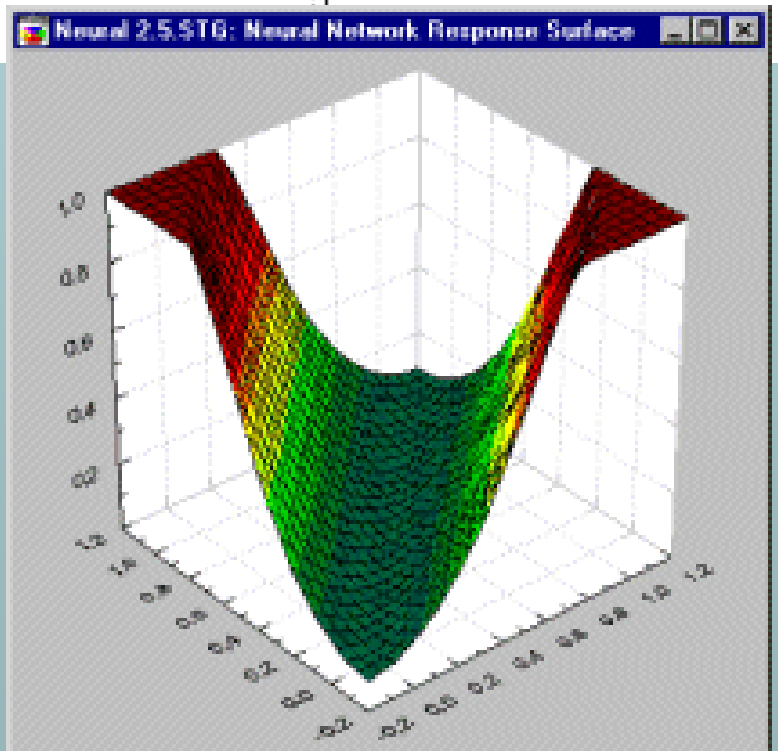
"Saddle point: Behavior around a saddle point. NAG/Momentum again like to explore around, almost taking a different path. Adadelta/Adagrad/RMSProp proceed like accelerated SGD.



"Long valley: Algos without scaling based on gradient information really struggle to break symmetry here - SGD gets no where and Nesterov Accelerated Gradient / Momentum exhibits oscillations until they build up velocity in the optimization direction. Algos that scale step size based on the gradient quickly break symmetry and begin descent."


Function Curve
Iteration = 100
Weight = -0.47
Bias = 1.03
Output
Input
1
0.8
0.6
0.4
0.2
0
-5
0
5


Error Surface
Error
Weight
Bias
1
0.5
0
-2
-1
0
1
2


Neural 2.5.STG: Neural Network Response Surface

With no further changes, the algorithm can iterate toward a solution, arriving at $w = -0.2365, b = 1.1449$. This solution is plotted in Figure 12.

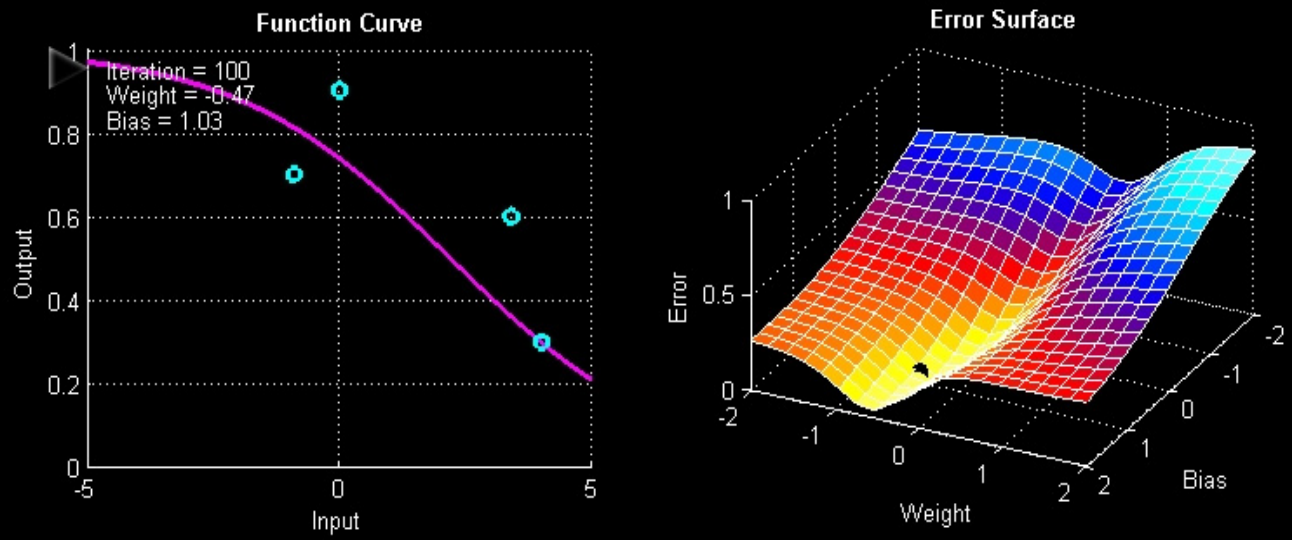


Figure 12 (interactive). Gradient descent on a one-node network, with multiple input/target pairs. Hover your cursor over either of the plots to watch the progress of the gradient descent algorithm. The left plot shows the curve of the network, and the four targets (in red). Simultaneously, the right plot shows the position of the network on its error surface.

We can make several observations in Figure 12.

- First, note that the four input/target pairs do not fit a sigmoid line, so there is no perfect solution to the problem. At best, we can expect gradient descent to find a curve somewhere in between all of the points.
- Second, note that the error surface appears to have a single deep valley, so gradient descent should be able to descend into it.
- Finally, when watching the function curve move toward the targets, note that its movement becomes erratic as it nears the targets. This is an example of the "overstepping" problem, where the algorithm misses a good solution because its step size is too large. We can address this by introducing a step coefficient. In the context of ANNs, this coefficient is often called a "learning rate". Typically, the coefficient is set up to decrease over time, so that the

Constrained Optimization

$$f(\mathbf{x}) : \mathbb{R}^N \longrightarrow \mathbb{R}$$

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

Subject to:

- Equality constraints: $a_i(\mathbf{x}) = 0 \quad i = 1, 2, \ldots, p$
- Nonequality constraints: $c_j(\mathbf{x}) \geq 0 \quad j = 1, 2, \ldots, q$

- Constraints define a feasible region, which is nonempty.
- The idea is to convert it to an unconstrained optimization.

Equality constraints

- Minimize $f(\mathbf{x})$ subject to: $a_i(\mathbf{x}) = 0$ for $i = 1, 2, \ldots, p$

- The gradient of $f(\mathbf{x})$ at a local minimizer is equal to the linear combination of the gradients of $a_i(\mathbf{x})$ with Lagrange multipliers as the coefficients.

$$\nabla f(\mathbf{x}^*) = \sum_{i=1}^{p} \lambda_i^* \nabla a_i(\mathbf{x}^*)$$

Inequality constraints

- Minimize $f(\mathbf{x})$ subject to: $c_j(\mathbf{x}) \geq 0$ for $j = 1, 2, \ldots, q$

- The gradient of $f(\mathbf{x})$ at a local minimizer is equal to the linear combination of the gradients of $c_j(\mathbf{x})$, which are active ($c_j(\mathbf{x}) = 0$)
- and Lagrange multipliers must be positive, $\mu_j \geq 0,\ j \in A$

$$\nabla f(\mathbf{x}^*) = \sum_{j \in A} \mu_j^* \nabla c_j(\mathbf{x}^*)$$

Lagrangien

- We can introduce the function (Lagrangien)

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i=1}^{p} \lambda_i a_i(\mathbf{x}) - \sum_{j=1}^{q} \mu_j c_j(\mathbf{x})$$

- The necessary condition for the local minimizer is

$$\nabla_x L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = 0$$

  and it must be a feasible point (i.e. constraints are satisfied).

- These are Karush-Kuhn-Tucker conditions

General Nonlinear Optimization

- Minimize $f(\mathbf{x})$

  subject to: $a_i(\mathbf{x}) = 0$

  $c_j(\mathbf{x}) \geq 0$

  where the objective function and constraints are nonlinear.

1. For a given $\{\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k\}$ approximate Lagrangien by Taylor series $\rightarrow$ QP problem
2. Solve QP $\rightarrow$ descent direction $\{\boldsymbol{\delta}_x, \boldsymbol{\delta}_\lambda, \boldsymbol{\delta}_\mu\}$
3. Perform line search in the direction $\boldsymbol{\delta}_x \rightarrow \mathbf{x}_{k+1}$
4. Update Lagrange multipliers $\rightarrow \{\boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}\}$
5. Repeat from Step 1.

General Nonlinear Optimization

Lagrangien $$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\mathbf{x}) - \sum_{i=1}^{p} \lambda_i a_i(\mathbf{x}) - \sum_{j=1}^{q} \mu_j c_j(\mathbf{x})$$

At the kth iterate: $\{\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k\}$

and we want to compute a set of increments: $\{\boldsymbol{\delta}_x, \boldsymbol{\delta}_\lambda, \boldsymbol{\delta}_\mu\}$

First order approximation of $\nabla_x L$ and constraints:

- $\nabla_x L(\mathbf{x}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}) \approx \nabla_x L(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) +$

  $$+\nabla_x^2 L(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\boldsymbol{\delta}_x + \nabla_{x\lambda}^2 L(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\boldsymbol{\delta}_\lambda + \nabla_{x\mu}^2 L(\mathbf{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)\boldsymbol{\delta}_\mu = 0$$

- $c_i(\mathbf{x}_k \boldsymbol{\delta}_x) \approx c_i(\mathbf{x}_k) + \boldsymbol{\delta}_x^T \nabla_x c_i(\mathbf{x}_k) \geq 0$
- $a_i(\mathbf{x}_k \boldsymbol{\delta}_x) \approx a_i(\mathbf{x}_k) + \boldsymbol{\delta}_x^T \nabla_x a_i(\mathbf{x}_k) = 0$

These approximate KKT conditions corresponds to a QP program

Quadratic Programming (QP)

- Like in the unconstrained case, it is important to study quadratic functions. Why?
- Because general nonlinear problems are solved as a sequence of minimizations of their quadratic approximations.
- QP with constraints

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

subject to linear constraints.

- $\mathbf{H}$ is symmetric and positive semidefinite.

QP with Equality Constraints

- Minimize $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{x}^T\mathbf{p}$

  Subject to: $\mathbf{A}\mathbf{x} = \mathbf{b}$

- Ass.: $\mathbf{A}$ is $p \times N$ and has full row rank ($p<N$)
- Convert to unconstrained problem by variable elimination:

$$\mathbf{x} = \mathbf{Z}\phi + \mathbf{A}^+\mathbf{b}$$

$\mathbf{Z}$ is the null space of $\mathbf{A}$

$\mathbf{A}^+$ is the pseudo-inverse.

Minimize $\hat{f}(\phi) = \frac{1}{2}\phi^T\hat{\mathbf{H}}\phi + \phi^T\hat{\mathbf{p}}$

$$\hat{\mathbf{H}} = \mathbf{Z}^T\mathbf{H}\mathbf{Z}$$
$$\hat{\mathbf{p}} = \mathbf{Z}^T(\mathbf{H}\mathbf{A}^+\mathbf{b} + \mathbf{p})$$

This quadratic unconstrained problem can be solved, e.g., by Newton method.

QP with inequality constraints

- Minimize $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{x}^T\mathbf{p}$
  Subject to: $\mathbf{A}\mathbf{x} \geq \mathbf{b}$

- First we check if the unconstrained minimizer $\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{p}$ is feasible.
  If yes we are done.
  If not we know that the minimizer must be on the boundary and we proceed with an active-set method.

  - $\mathbf{x}_k$ is the current feasible point
  - $\mathcal{A}_k$ is the index set of active constraints at $\mathbf{x}_k$
  - Next iterate is given by $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$

Active-set method

$$\mathbf{A}^T = [\mathbf{a}_1 \ldots \mathbf{a}_p]$$

- $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ How to find $\mathbf{d}_k$?
  - To remain active $\mathbf{a}_j^T \mathbf{x}_{k+1} - b_j = 0$ thus $\mathbf{a}_j^T \mathbf{d}_k = 0 \quad j \in \mathcal{A}_k$
  - The objective function at $\mathbf{x}_k$+$\mathbf{d}$ becomes

$$f_k(\mathbf{d}) = \frac{1}{2}\mathbf{d}^T \mathbf{H}\mathbf{d} + \mathbf{d}^T \mathbf{g}_k + f(\mathbf{x}_k) \qquad \text{where} \quad \mathbf{g}_k = \nabla f(\mathbf{x}_k)$$

- The major step is a QP sub-problem

$$\mathbf{d}_k = \arg\min_{\mathbf{d}} \frac{1}{2}\mathbf{d}^T \mathbf{H}\mathbf{d} + \mathbf{d}^T \mathbf{g}_k$$

subject to: $\mathbf{a}_j^T \mathbf{d} = 0 \quad j \in \mathcal{A}_k$

- Two situations may occur: $\mathbf{d}_k = 0$ or $\mathbf{d}_k \neq 0$

Active-set method

- $\mathbf{d}_k = 0$

  We check if KKT conditions are satisfied

$$\nabla_x L(\mathbf{x}, \boldsymbol{\mu}) = \mathbf{H}\mathbf{x}_k + \mathbf{p} - \sum_{j \in \mathcal{A}_k} \mu_j \mathbf{a}_j = 0 \quad \text{and} \quad \mu_j \geq 0$$

  If YES we are done.

  If NO we remove the constraint from the active set $\mathcal{A}_k$ with the most negative μ_j and solve the QP sub-problem again but this time with less active constraints.

- $\mathbf{d}_k \neq 0$

  We can move to $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$ but some inactive constraints may be violated on the way.

  In this case, we move by $\alpha_k \mathbf{d}_k$ till the first inactive constraint becomes active, update $\mathcal{A}_k$, and solve the QP sub-problem again but this time with more active constraints.

SQP example

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

subject to: $1.5 - x_1^2 - x_2^2 \geq 0$