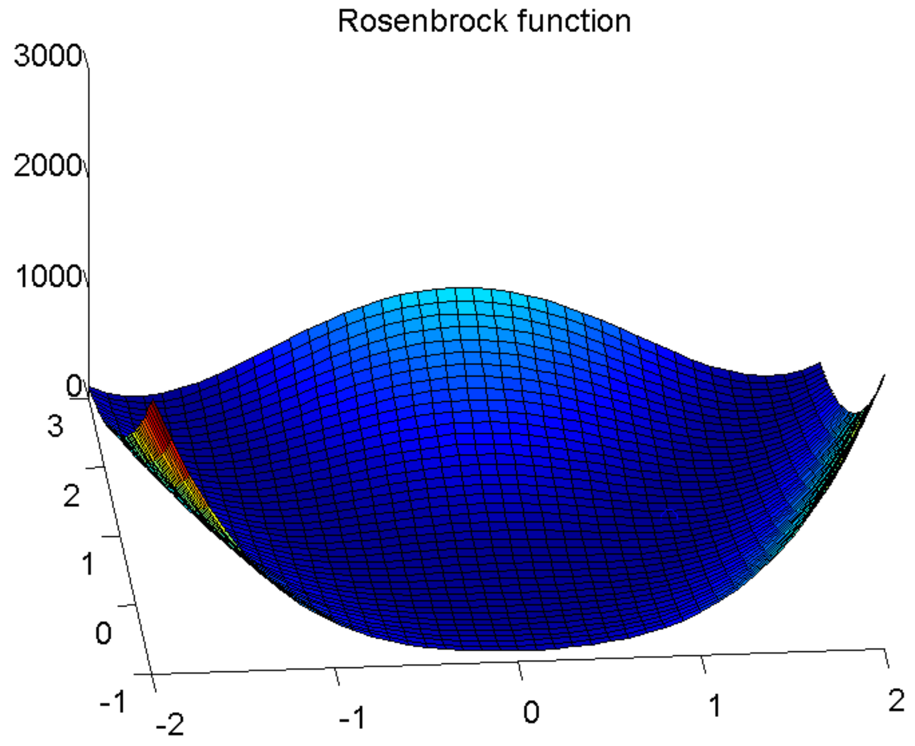
Lecture 2

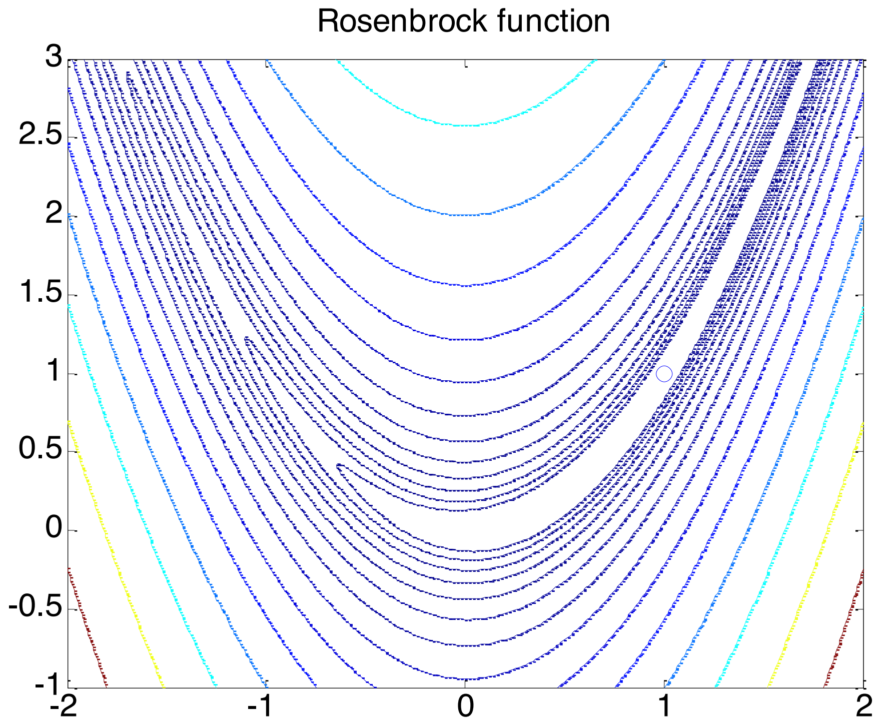
B1 Optimization - Michaelmas 2023 - V. A. Prisacariu

- Rosenbrock’s function
- Newton’s method
  - Line search
- Quasi-Newton methods
- Least-Squares and Gauss-Newton methods and a few words on Levenberg-Marquardt.
- Downhill simplex (amoeba) algorithm

Rosenbrock’s function

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





Minimum is at [1, 1]

An Optimization Algorithm (from lecture 1)

Start at $\mathbf{x}_0$ then repeat:

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

4. Check for convergence (termination criteria), e.g. $\nabla f \approx 0$.

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

Steepest descent (from lecture 1)

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

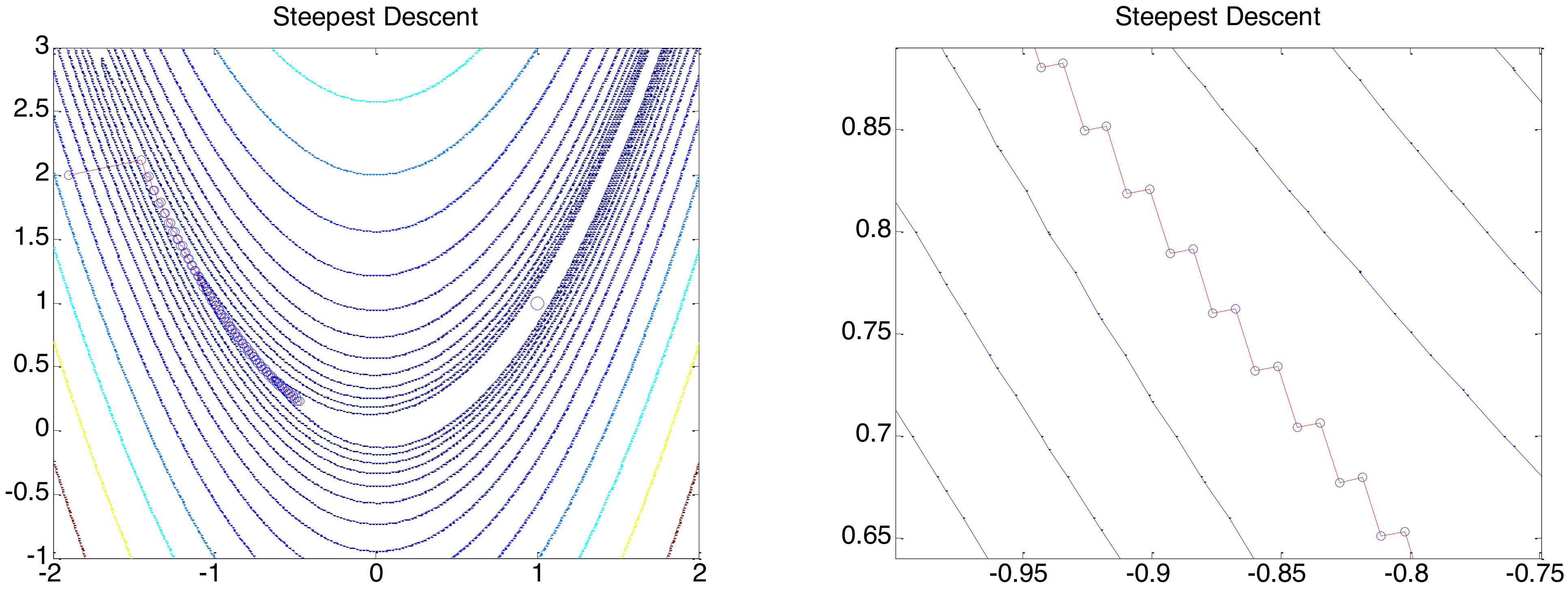
$$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{p}_n$$

The steepest decent method choses $\mathbf{p}_n$ to be parallel to the negative gradient:

$$\mathbf{p}_n = -\nabla f(\boldsymbol{x}_{\mathrm{n}})$$

Steepest descent

The 1D line minimization must be performed using one of the earlier methods



- The zig-zag behavior is clear in the zoomed view (100 iterations);
- The algorithm crawls along the valley.

Newton’s method in 1D (from lecture 1)

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

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

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

- Find the δx such that $x + \delta x$ is a stationary point of f:

$$\frac{d}{d\delta x}\left(f(x) + \delta x f'(x) + \frac{\delta x^2}{2} f''(x) \right) = f'(x) + \delta x f''(x) = 0$$

- and rearranging:

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

- Update for x:

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

Taylor expansion in 2D (from lecture 1)

A function can be approximated locally by its Taylor series expansion about a point x_0.

$$f(x_0 + x) \approx f(x_0) + \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)\binom{x}{y} + \frac{1}{2}(x, y)\begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}\binom{x}{y} + \text{h.o.t.}$$

- This is a generalization of the 1D Taylor series:

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

- The expansion to second order is a quadratic function in **x.**

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

Newton’s method in ND

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

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

where the gradient is the vector

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

and the Hessian is the symmetric matrix

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

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

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

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

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \mathrm{H}_n^{-1}\mathbf{g}_n$$

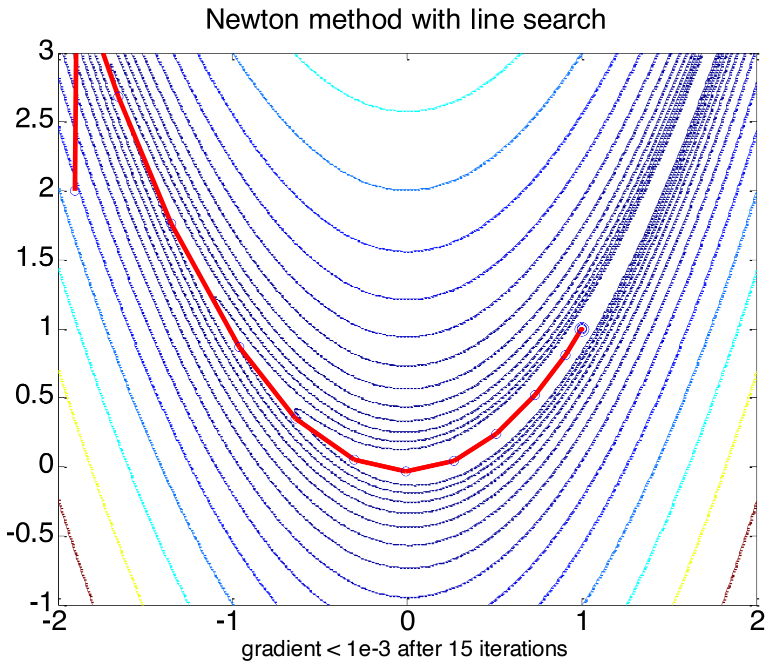
Newton's method in ND

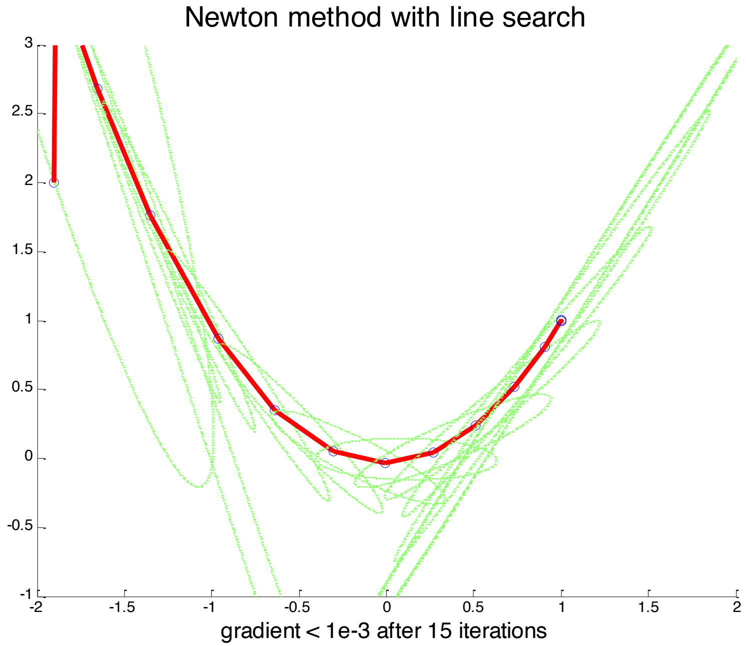
$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta\mathbf{x} = \mathbf{x}_n - \mathrm{H}_n^{-1}\mathbf{g}_n$$

- If $f(\mathbf{x})$ is quadratic, the solution is found in one step.
- The method has quadratic convergence (as in the 1D case).
- The solution $\delta\mathbf{x} = -\mathrm{H}_n^{-1}\mathbf{g}_n$ is guaranteed to be a downhill direction (provided that H is positive definite).
- For numerical reasons, the inverse is not actually computed, instead $\delta\mathbf{x}$ is computed as the solution of $\mathrm{H}\delta\mathbf{x} = -\mathbf{g}_n$.
- Rather than jump straight to $\mathbf{x}_n - \mathrm{H}_n^{-1}\mathbf{g}_n$, it is often better to perform a line search which ensures (more) global convergence:

  $$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha\mathrm{H}_n^{-1}\mathbf{g}_n$$
- If $\mathrm{H} = \mathrm{I}$, Newton reduces to steepest descent.

Newton’s method in ND Example





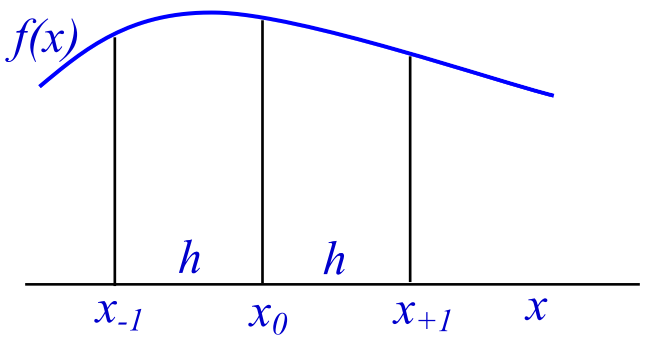
ellipses show successive quadratic approximations

- The algorithm converges in only 15 iterations compared to hundreds for steepest descent.
- However, the method requires computing the Hessian matrix at each iteration – this is not always feasible

Quasi-Newton Methods

- If the problem size is large and the Hessian matrix is dense, then it may be in infeasible / inconvenient to compute it directly.
- Quasi-Newton method avoid this problem by keeping “rolling estimate” of $\mathrm{H}(\mathbf{x})$, updated at each iteration using new gradient information.
- Common schemes are Broyden–Fletcher–Goldfarb–Shanno (BFGS) and Davidon–Fletcher–Powell (DFP).

e.g. in 1D



First derivatives:

$$f'\left(x_0+\frac{h}{2}\right)=\frac{f_1-f_0}{h} \text{ and } f'\left(x_0-\frac{h}{2}\right)=\frac{f_0-f_{-1}}{h}$$

Second derivatives:

$$f''(x_0)=\frac{\frac{f_1-f_0}{h}-\frac{f_0-f_{-1}}{h}}{h}=\frac{f_1-2f_0+f_{-1}}{h^2}$$

For H_{n+1} we can build an approximation from $\mathrm{H}_n, \mathbf{g}_n, \mathbf{g}_{n+1}, \mathbf{x}_n, \mathbf{x}_{n+1}$

Quasi-Newton BFGS

Set $\mathrm{H}_0 = \mathrm{I}$.

Update according to

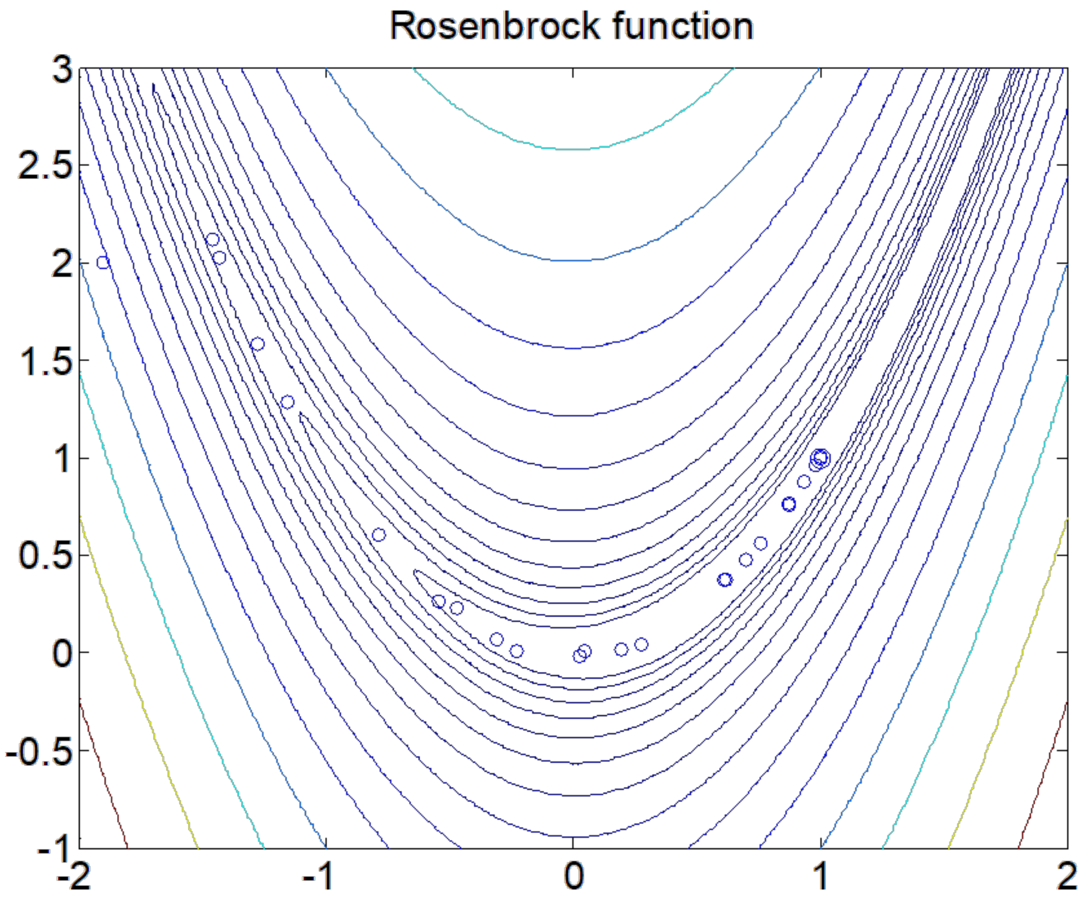
$$\mathrm{H}_{n+1} = \mathrm{H}_n + \frac{\mathbf{q}_n \mathbf{q}_n^T}{\mathbf{q}_n^T \mathbf{s}_n} - \frac{(\mathrm{H}_n \mathbf{s}_n)(\mathrm{H}_n \mathbf{s}_n)^T}{\mathbf{s}_n^T \mathrm{H}_n \mathbf{s}_n}$$

where

$$\mathbf{s}_n = \mathbf{x}_{n+1} - \mathbf{x}_n$$
$$\mathrm{q}_n = \mathbf{g}_{n+1} - \mathbf{g}_n$$

- The matrix itself is not stored, but rather represented compactly by a few stored vectors.
- The estimate H_{n+1} is used to form a local quadratic approximation as before.

Quasi-Newton BFGS Example



- The method converges in 25 iterations, compared to 15 for the full-Newton method
- In Matlab the optimization function ‘fminunc’ uses a BFGS quasi-Newton method for medium scale optimization problems.

Matlab – fminunc

```
>> f='100*(x(2)-x(1)^2)^2+(1-x(1))^2';
>> GRAD='[100*(4*x(1)^3-4*x(1)*x(2))+2*x(1)-2;  100*(2*x(2)-2*x(1)^2) ]';
```

Choose options for BFGS quasi-Newton

```
>> OPTIONS=optimset('LargeScale','off', 'HessUpdate','bfgs' );
>> OPTIONS = optimset(OPTIONS,'gradobj','on');
```

Start point

```
>> x = [-1.9; 2];
>> [x,fval] = fminunc({f,GRAD},x,OPTIONS);
```

This produces

```
x = 0.9998, 0.9996        fval = 3.4306e-008
```

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:

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

- 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.

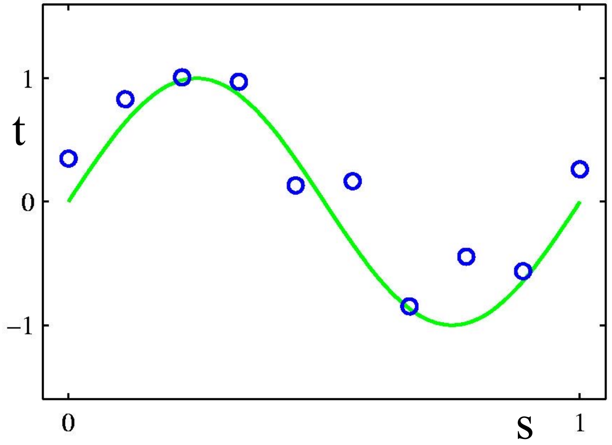
- Examples arise in non-linear regression (fitting) of data.

Linear Least Squares Reminder

The goal is to fit a function to measured data points $\{s_i, t_i\}$ by minimizing the cost:

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

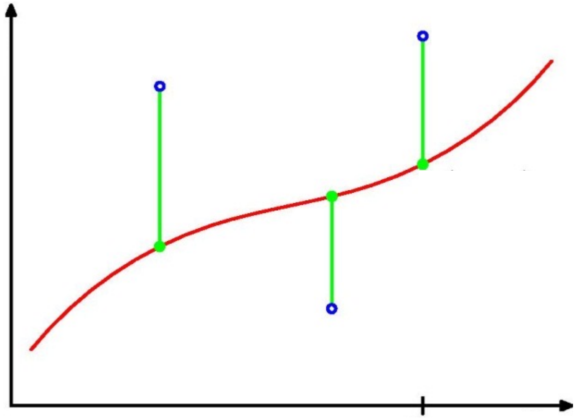
$$= \sum_{i=1} (y(s_i, \mathbf{x}) - t_i)^2 \text{, where } t_i \text{ is the target value}$$



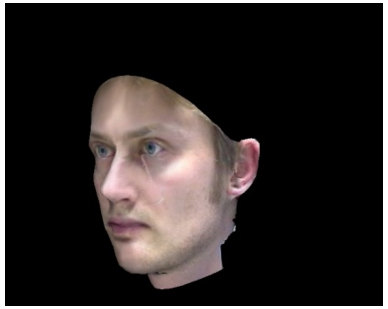
For example, the regression functions $y(s_i, \mathbf{x})$ might be the polynomial

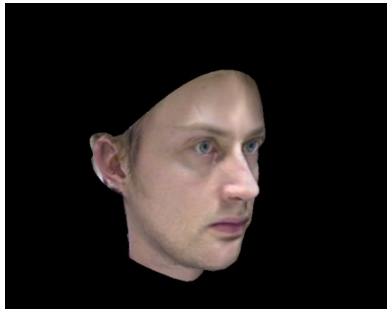
$$y(s, \mathbf{x}) = x_0 + x_1 s + x_2 s^2 + \cdots$$

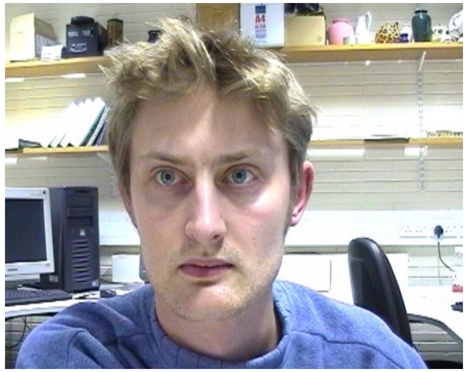
In this case the function is linear in the parameter $\mathbf{x}$ and there is a closed form solution. Most often, there will be no closed form solution to non-linear $y(s, \mathbf{x})$.



Non-Linear Least Squares Example: Aligning a 3D Model to an Image







Input:
3D textured face model, camera model, image $I(x, y)$.

Task:
Determine the 3D rotation and 3D translation that minimizes the error between the image $I(x, y)$ and the projected 3D model.

Cost Function

$$f(\mathrm{R}, \mathbf{T}) = \sum_{x,y}^{M} \left| \hat{I}_{\mathrm{R},\mathbf{T}}(x, y) - I(x, y) \right|^2$$

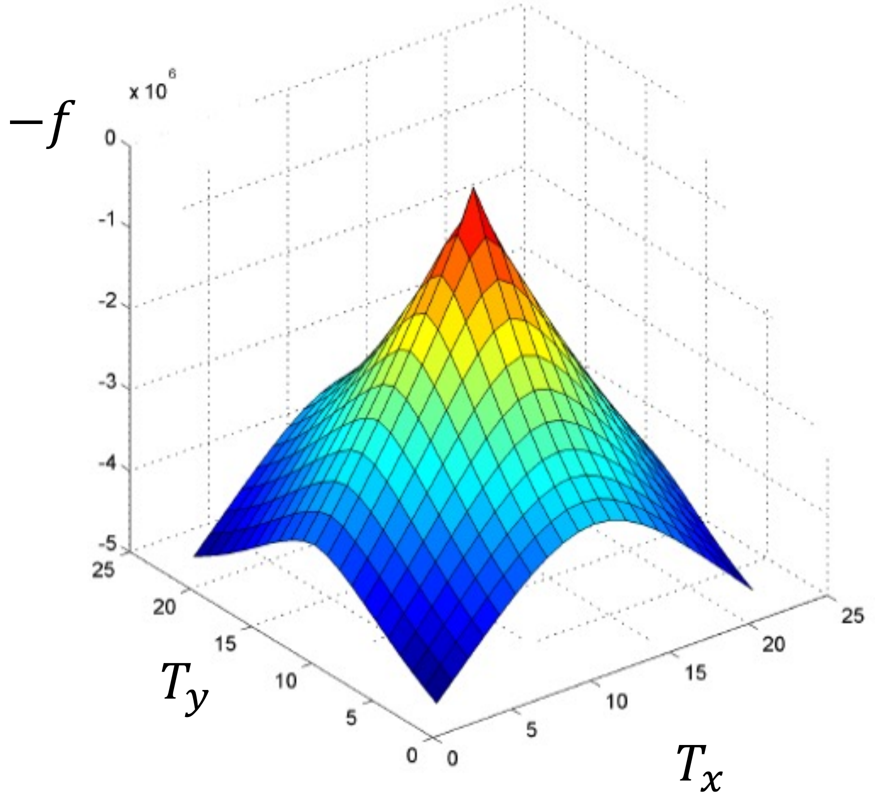
N=6, M = 10^6

Transformation parameters:

- 3D rotation matrix R
- Translation vector $\mathbf{T} = \left(T_x, T_y, T_z\right)^T$

Image generation:

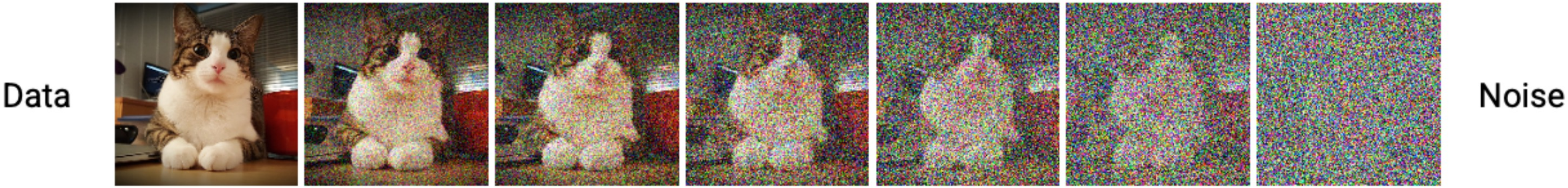
- Rotate and translate 3D model by R and $\mathbf{T}$
- Project to generate image $\hat{I}_{\mathrm{R},\mathbf{T}}(x, y)$



A4

Non-Linear Least Squares Example: Diffusion Models

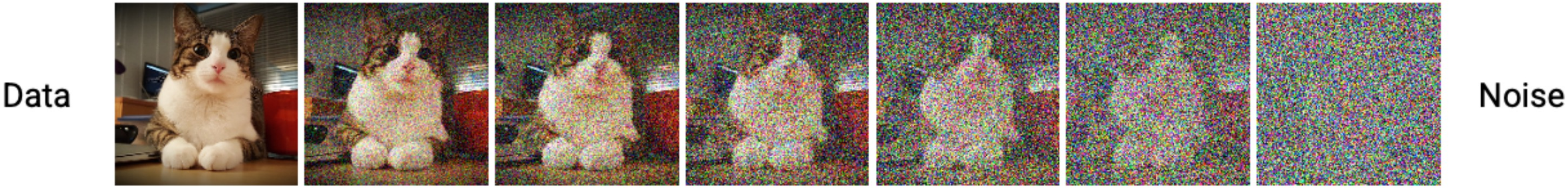
Forward Model



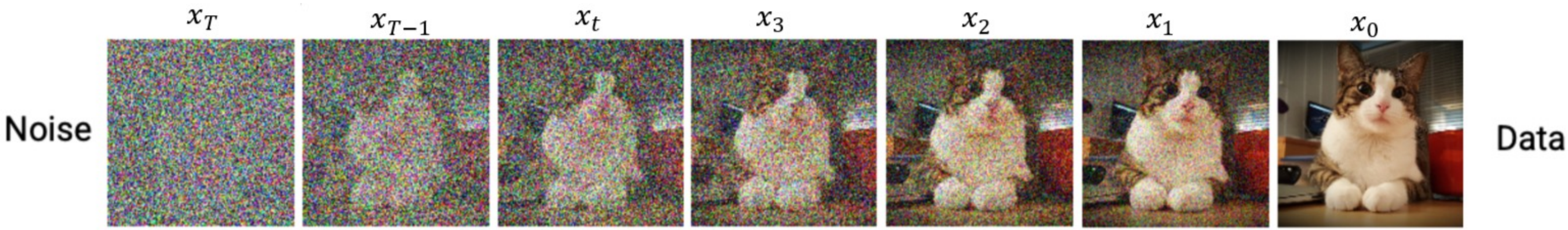
Denoising Diffusion-based Generative Modeling: Foundations and Applications (cvpr2022-tutorial-diffusion-models.github.io)

Non-Linear Least Squares Example: Diffusion Models

Forward Model



Backward Model



L2 between the predicted noise and the actual noise added at step t

https://stablediffusionweb.com

Denoising Diffusion-based Generative Modeling: Foundations and Applications (cvpr2022-tutorial-diffusion-models.github.io)

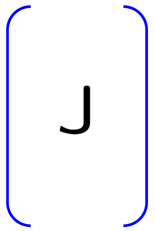
Non-Linear Least Squares

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

The $M \times N$ Jacobian of the vector of residuals $\mathbf{r}$ is defined as:

assume
M > N

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

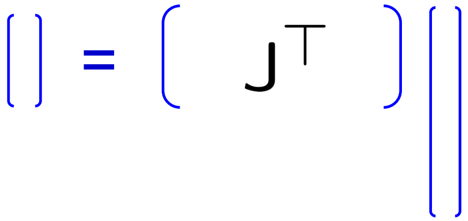


Consider

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

Hence

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



Non-Linear Least Squares

For the Hessian we require

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

$$\left(\ \mathrm{J}^\top\ \right)\left(\qquad\right)\qquad\qquad\left(\mathrm{R_i}\right)$$

Hence

$$\mathrm{H}(\mathbf{x}) = 2\mathrm{J}^T\mathrm{J} + 2\sum_i^M r_i \mathrm{R}_i$$

Non-Linear Least Squares

Note that the second-order term in the Hessian $\mathrm{H}(\mathbf{x})$ 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, giving the Gauss-Newton approximation of the Hessian:

$$\mathrm{H}(\mathbf{x}) = 2\mathrm{J}^T\mathrm{J}$$

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

Gauss-Newton Summary

For a cost function $f(\mathbf{x})$ that is the sum of squared residuals

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

The Hessian can be approximated as

$$\mathrm{H}(\mathbf{x}) = 2\mathrm{J}^T\mathrm{J}$$

and the gradient is given by

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

So the Newton update step

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta\mathbf{x} = \mathbf{x}_n - \mathrm{H}_n^{-1}\mathbf{g}_n$$

Computed as $\mathrm{H}\delta\mathbf{x} = -\mathbf{g}_n$, becomes

$$\mathrm{J}^T\mathrm{J}\delta\mathbf{x} = -\mathrm{J}^T\mathbf{r}$$

These are called the normal equations.

Gauss-Newton Example

The minimization of the Rosenbrock function.

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

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

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

$$\mathrm{J}(\mathbf{x}) = \begin{pmatrix} \dfrac{\partial r_1}{\partial x} & \dfrac{\partial r_1}{\partial y} \\ \dfrac{\partial r_2}{\partial x} & \dfrac{\partial r_2}{\partial y} \end{pmatrix} = \begin{pmatrix} -20x & 10 \\ -1 & 0 \end{pmatrix}$$

Gauss-Newton Example

The true Hessian is:

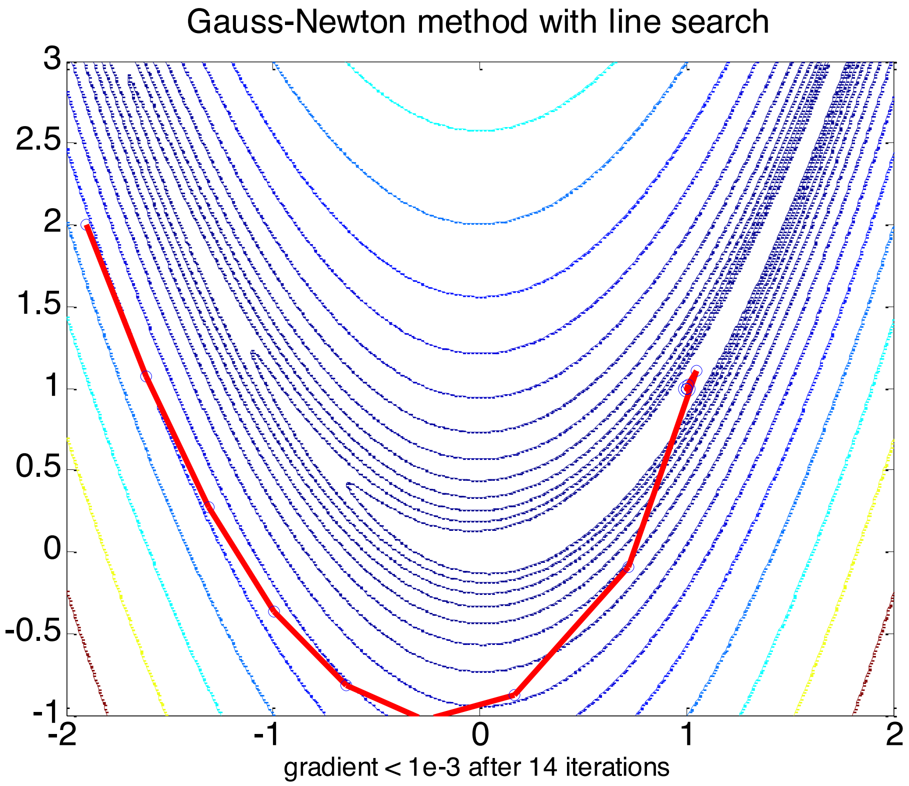
$$\mathrm{H}(\mathbf{x}) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x^2} & \dfrac{\partial^2 f}{\partial x \partial y} \\ \dfrac{\partial^2 f}{\partial x \partial y} & \dfrac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 1200x^2 - 400y + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

The Gauss-Newton approximation of the Hessian is:

$$2\mathrm{J}^T\mathrm{J} = 2\begin{bmatrix} -20x & -1 \\ 10 & - \end{bmatrix}\begin{bmatrix} -20x & 10 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} 800x^2 + 2 & -400x \\ -400x & 200 \end{bmatrix}$$

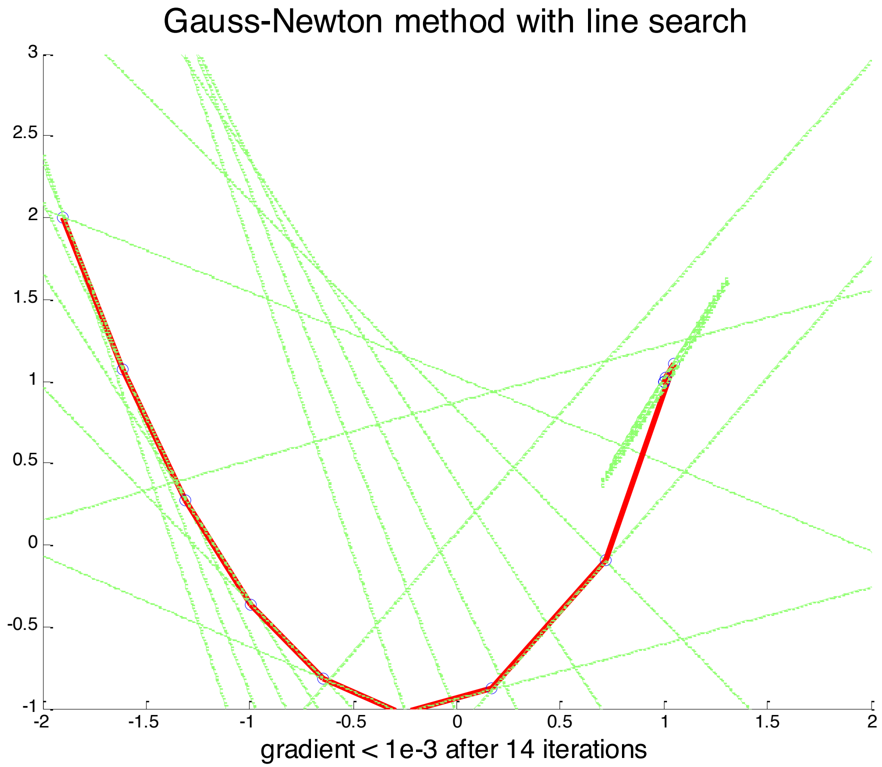
Gauss-Newton Rosenbrock

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



Gauss-Newton method with line search

gradient < 1e-3 after 14 iterations



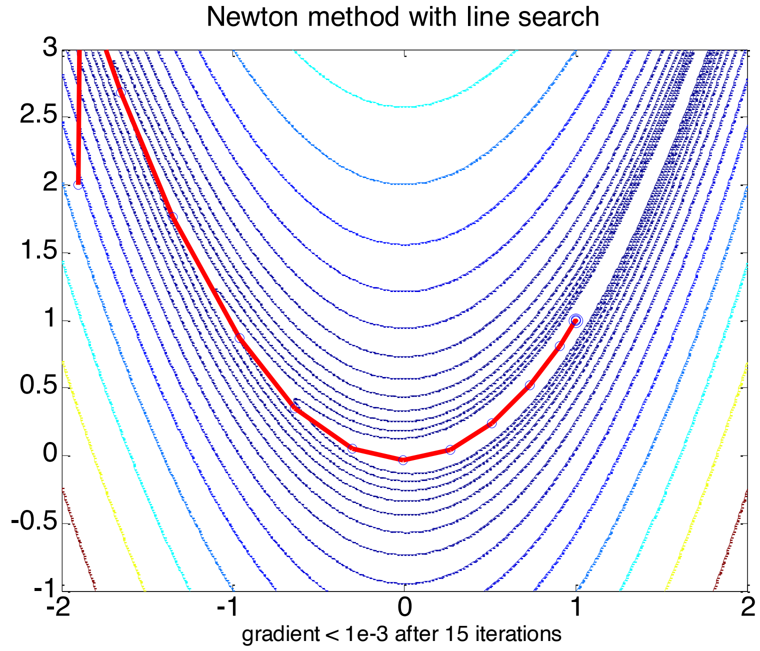
Gauss-Newton method with line search

gradient < 1e-3 after 14 iterations

minimization with the Gauss-Newton approximation with line search takes only 14 iterations

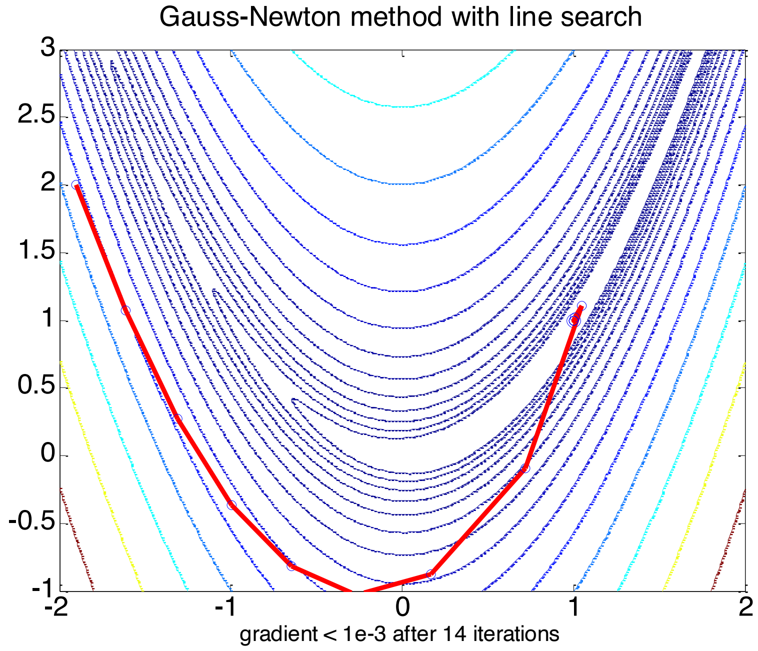
Comparison

Newton



- requires computing Hessian (i.e. n^2 second derivatives)
- exact solution if quadratic

Gauss-Newton



- approximates Hessian by Jacobian product
- requires only n x M first derivatives

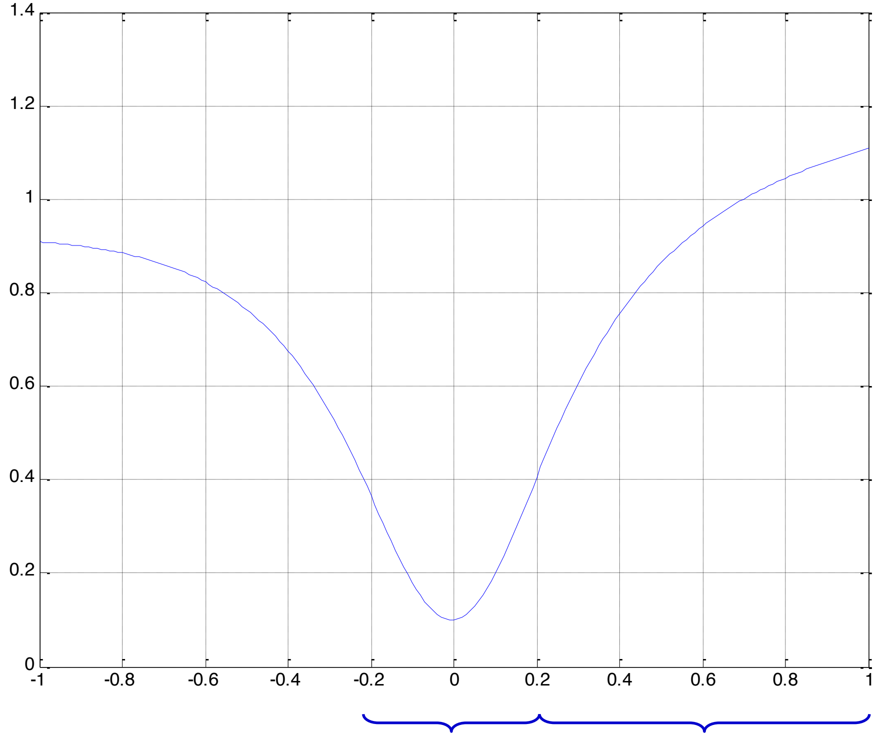
Properties of Methods

Update: $\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{p}_n$

- Gradient descent $[\mathbf{p}_\mathrm{n} = -\mathbf{g}]$
  - Will zig-zag – each new increment is perpendicular to the previous.
  - Requires 1D search (unless you hack it).
  - Slow to converge.
- Newton's method $[\mathbf{p}_\mathrm{n} = -\mathrm{H}^{-1}\mathbf{g}_n]$
  - Requires computation of Hessian.
  - Can converge to maximum or saddle as well as minimum.
  - Can be unstable.
- Gauss-Newton $[\mathrm{p}_\mathrm{n} = -\mathrm{H}_{GN}^{-1}\mathbf{g}_n]$
  - Is a downhill method, so will not converge to maximum or saddle.
  - Can be unstable, usually needs line search.

Levenberg-Marquardt [more in C25]

- 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-decent and Gauss-Newton steps depending on how good the H_{GN} approximation is locally.



Newton GD

Performance issues for optimization algorithms

1. Number of iterations required

2. Cost per iteration

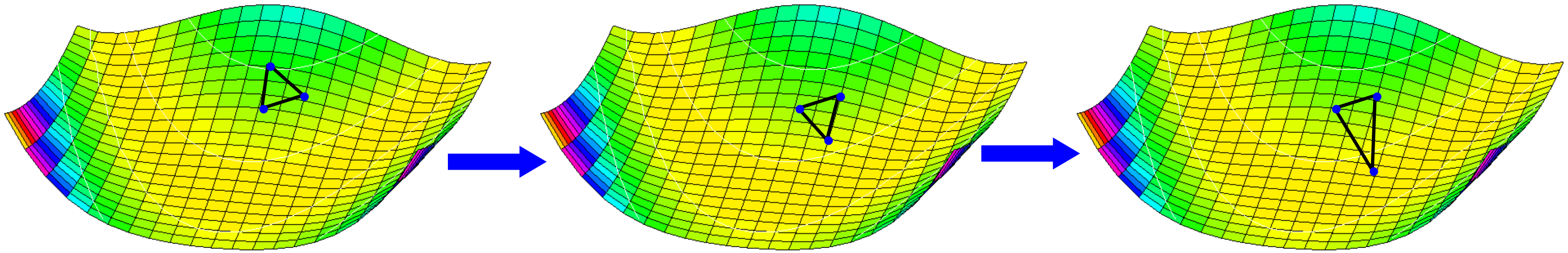
3. Memory footprint

4. Region of convergence

The downhill simplex (amoeba) algorithm

- Due to Nelder and Mead (1965)
- A *direct* method: only uses function evaluations (no derivatives)
- A simplex is the polytope in *N* dimensions with *N+1* vertices, e.g.
  - 2D: triangle
  - 3D: tetrahedron
- Basic idea: move by reflections, expansions or contractions

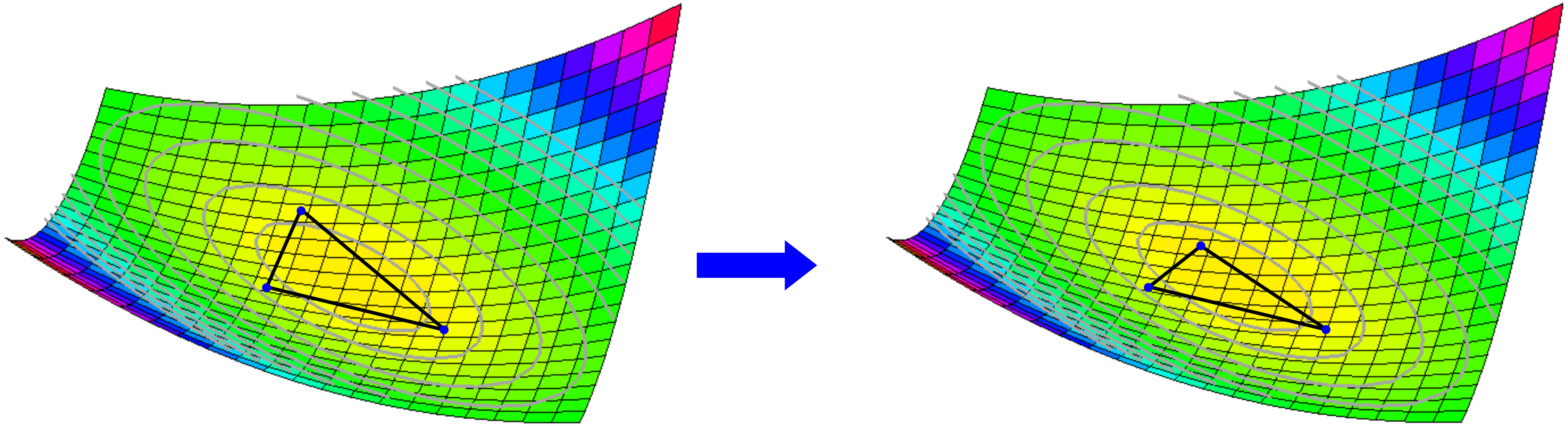
The downhill simplex (amoeba) algorithm



start

reflect

expand



contract

- Reorder the points so that $f(\mathbf{x}_{n+1}) > \cdots > f(\mathbf{x}_2) > f(\mathbf{x}_1)$ (i.e. $\mathbf{x}_{n+1}$ is the worst point).

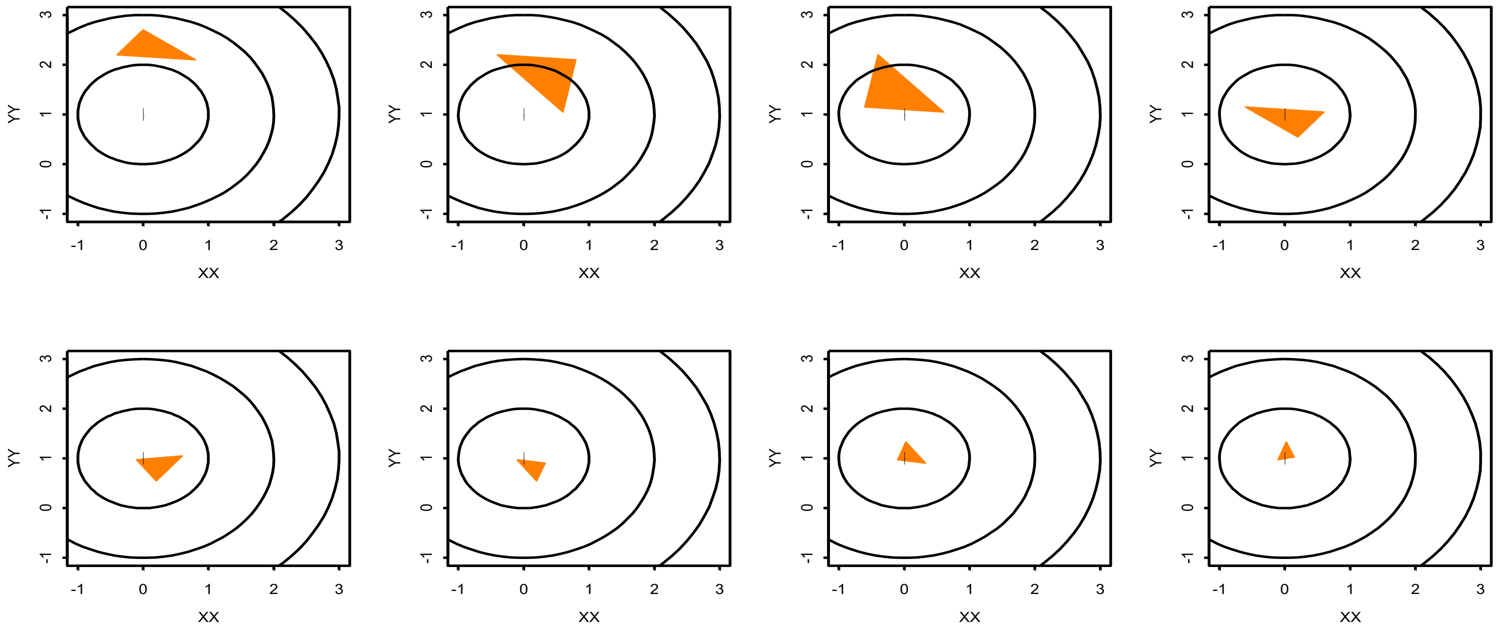
- Calculate $\bar{\mathbf{x}} = (\sum_i \mathbf{x}_i)/N$ as the centroid of all points except $\mathbf{x}_{n+1}$.

- Generate a trial point by reflection $\mathbf{x}_r = \bar{\mathbf{x}} + \alpha(\bar{\mathbf{x}} - \mathbf{x}_{n+1})$, where and $\alpha > 0$.
- If the reflected point $\mathbf{x}_r$ is neither the new best or worst point, i.e. if $f(\mathbf{x}_1) < f(\mathbf{x}_r) < f(\mathbf{x}_n)$ replace the worst point $\mathbf{x}_{n+1}$ by $\mathbf{x}_r$.

- If the reflected point $\mathbf{x}_r$ is the new best point, i.e. if $f(\mathbf{x}_r) < f(\mathbf{x}_1)$, generate a new point by expansion $\mathbf{x}_e = \mathbf{x}_r + \beta(\mathbf{x}_r - \bar{\mathbf{x}})$, where $\beta > 0$.
- If the expanded point is better than the reflected point $f(\mathbf{x}_e) < f(\mathbf{x}_r)$ then replace $\mathbf{x}_{n+1}$ by $\mathbf{x}_e$, otherwise replace $\mathbf{x}_{n+1}$ by $\mathbf{x}_r$.

- If $f(\mathbf{x}_r) > f(\mathbf{x}_n)$ assume the polytope is too large and generate a new point by contraction $\mathbf{x}_c = \bar{\mathbf{x}} + \gamma(\mathbf{x}_{n+1} - \bar{\mathbf{x}})$, where $\gamma, (0 < \gamma < 1)$ is the contraction coefficient.
- If the contracted point is better than the worst point, so $f(\mathbf{x}_c) < f(\mathbf{x}_{n+1})$, the contraction has succeeded, and replace $\mathbf{x}_{n+1}$ by $\mathbf{x}_c$, otherwise contract again (or shrink all points).

Downhill Simplex Example 2: Contraction About a Minimum



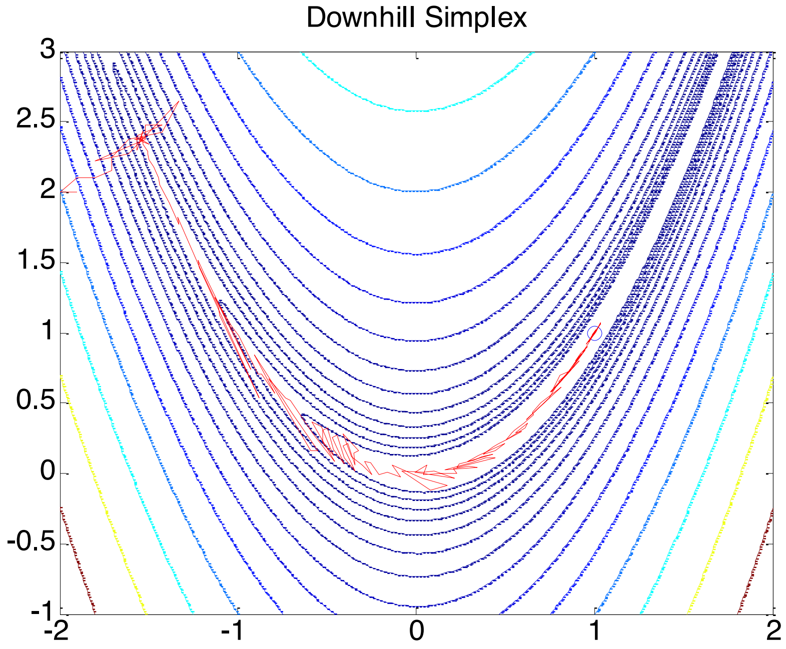
reflections

contractions

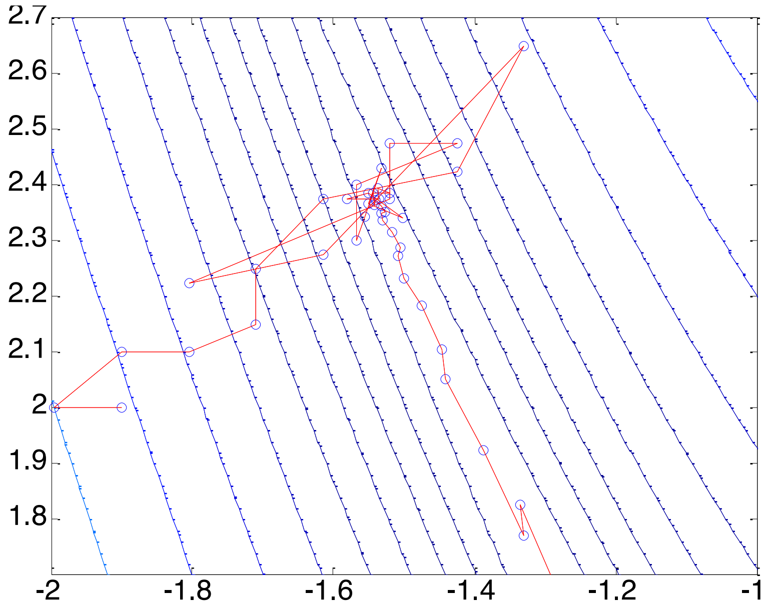
Summary

- no derivatives required
- deals well with noise in the cost function
- is able to crawl out of some local minima (though, of course, can still get stuck)

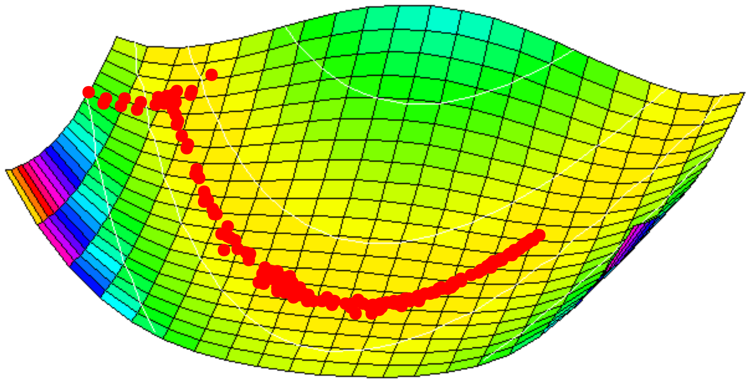
Downhill Simplex Example 1



Path of best vertex



Detail



Matlab fminsearch
with 200 iterations

Matlab – fminsearch

Nelder-Mead simplex direct search

```
>> banana = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2;
```

Pass the function handle to fminsearch:

```
>> [x,fval] = fminsearch(banana,[-1.9, 2])
```

This produces

```
   x = 1.0000  1.0000
   fval =4.0686e-010
```

Google to find out more on using this function

What is next?

- Move from general and quadratic optimization problems to linear programming.
- Constrained optimization.