• **Lecture 1:** Local and global optima, unconstrained univariate and multivariate optimization, stationary points, steepest descent

• **Lecture 2:** Newton and Newton like methods – Quasi-Newton, Gauss-Newton; the Nelder-Mead (amoeba) simplex algorithm

• **Lecture 3:** Linear programming constrained optimization; the simplex algorithm, interior point methods; integer programming

• **Lecture 4:** Convexity, robust cost functions, methods for non-convex functions – grid search, multiple coverings, branch and bound, simulated annealing.
Holistic Scene Understanding

Reconstructing, Recognition and Segmentation of the Scene

Research:
- SLAM
- Stereo
- Recognition
- Segmentation
- Real time
- Tracking
- Mapping

Application:
- Augmented Reality
- Smart glasses for the partially sighted
- Displays
- Gaming (We did Augmented Reality for Sony Wonderbook & now work the the Kinect team)
Large Scale Reconstruction and Recognition

- Stereo Pairs
- Disparity
- Visual Odometry
- 2D Features
- Random Forest Unary Potential in 3D
- Depth
- Camera Pose
- X, Y, Z, roll, pitch, yaw
- Labelling in 3D
- Surface
- Raycasted Normals
- Raycasted Labels
- Semantics

Images and diagrams illustrating the process of large scale reconstruction and recognition.
Textbooks

• **Practical Optimization**

Covers unconstrained and constrained optimization. Very clear and comprehensive.
Background 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  
  CUP 1992/2002
  
  - Good chapter on optimization  
  - Available on line at  
    http://www.nrbook.com/a/bookcpdf.php
Background reading and web resources

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

- Further reading, web resources, and the lecture notes are on
  http://www.robots.ox.ac.uk/~az/lectures/b1
Lecture 1

Topics covered in this lecture

• Problem formulation
• Local and global optima
• Unconstrained univariate optimization
• Unconstrained multivariate optimization for quadratic functions:
  • Stationary points
  • Steepest descent
Optimization is used to find the best or optimal solution to a problem

Steps involved in formulating an optimization problem:

• Conversion of the problem into a mathematical model that abstracts all the essential elements

• Choosing a suitable optimization method for the problem

• Obtaining the optimum solution.
Introduction: Problem specification

Suppose we have a cost function (or objective function)

\[ f(x) : \mathbb{R}^n \rightarrow \mathbb{R} \]

Our aim is find the value of the parameters \( x \) that minimize this function

\[ x^* = \arg \min_x f(x) \]

subject to the following constraints:

- equality \( c_i(x) = 0, \quad i = 1, \ldots, m_e \)
- inequality \( c_i(x) \geq 0, \quad i = m_e + 1, \ldots, m \)

We will start by focusing on unconstrained problems
How to know if you are at a minimum?
Recall: One dimensional functions

- A differentiable function has a **stationary point** when the derivative is zero: \( df/dx = 0 \).
- The second derivative gives the **type** of stationary point

\[
\begin{align*}
  f''(x) &\leq 0 & \text{maximum} \\
  f''(x) &= 0 & \text{inflection} \\
  f''(x) &\geq 0 & \text{minimum}
\end{align*}
\]
Unconstrained optimization

function of one variable

\[ \min_x f(x) \]

- down-hill search (gradient descent) algorithms can find local minima
- which of the minima is found depends on the starting point
- such minima often occur in real applications
Example: template matching in 2D images

Model, $\mathcal{M}$

Input:
Two point sets
$\mathcal{M} = \{M_i\}$ and $\mathcal{D} = \{D_j\}$

Task:
Determine the transformation $T$ that minimizes the error between $\mathcal{D}$ and the transformed $\mathcal{M}$
Cost function

\[ f(\theta, t_x, t_y) = \sum_j \min_i \| R(\theta) M_i + t - D_j \|^2 \]

for each data point find closest model point

Model point: \( M_i = (x_i, y_i)^\top \)

Transformation parameters:
- rotation angle \( \theta \)
- translation \( t = (t_x, t_y)^\top \)
Unconstrained \textit{univariate} optimization

For the moment, assume we can start close to the global minimum

\[ \min_{x} f(x) \]

We will look at three basic methods to determine the minimum:

1. gradient descent
2. polynomial interpolation
3. Newton’s method

These introduce the ideas that will be applied in the \textit{multivariate} case
Simplest Rule for Minimization?
A typical 1D function

As an example, consider the function

\[ f(x) = 0.1 + 0.1x + \frac{x^2}{0.1 + x^2} \]
1. Gradient descent

Given a starting location, $x_0$, examine $\frac{df}{dx}$ and move in the *downhill* direction to generate a new estimate, $x_1 = x_0 + \delta x$

How to determine the step size $\delta x$?
Approximate $f(x)$ with a simpler function which reasonably approximates the function in a neighbourhood around the current estimate $x$. This neighbourhood is the trust region.

- 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.
Trust region is a term used in mathematical optimization to denote the subset of the region of the objective function that is approximated using a model function (often a quadratic).

If an adequate model of the objective function is found within the trust region then the region is expanded; conversely, if the approximation is poor then the region is contracted.

Wikipedia
Quadratic interpolation using 3 points, 2 iterations
Other methods to interpolate a quadratic?
• e.g. 2 points and one gradient
3. Newton’s method

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

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

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

- Find the $\delta x$ which minimizes this local quadratic approximation

$$f'(x + \delta x) = f'(x) + \delta xf''(x) = 0$$

- and rearranging

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

- Update for $x$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$
• avoids the need to bracket the root
• quadratic convergence (decimal accuracy doubles at every iteration)
When would it fail?
- global convergence of Newton’s method is poor
- often fails if the starting point is too far from the minimum

\[ f(x) \]

\[ x \]

- in practice, must be used with a globalization strategy which reduces the step length until function decrease is assured
How do we determine if we are a minimum?
Stationary Points for Multidimensional functions

A function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \),

has a \textit{stationary point} when the gradient

\[
\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right)^\top = 0
\]

\[
\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^\top = 0
\]
Is a stationary point a minimum?
Extension to N dimensions

• How big can N be?
  • problem sizes can vary from a handful of parameters to many thousands

In the following we will first examine the properties of stationary points in N dimensions

and then move onto optimization algorithms to find the stationary point (minimum)

We will consider examples for N=2, so that cost function surfaces can be visualized
Taylor expansion in 2D

A function may be approximated locally by its Taylor series expansion about a point $x_0$

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

This is a generalization of the 1D Taylor series

$$f(x_0 + x) = f(x_0) + xf'(x_0) + \frac{x^2}{2}f''(x_0) + \text{h.o.t}$$

The expansion to second order is a quadratic function in $x$

$$f(x) = a + g^\top x + \frac{1}{2} x^\top H x$$
Taylor expansion in ND

A function may be approximated locally by its Taylor series expansion about a point \( x_0 \)

\[
f(x_0 + x) \approx f(x_0) + \nabla f^\top x + \frac{1}{2}x^\top H x + \text{h.o.t}
\]

where the gradient \( \nabla f(x) \) of \( f(x) \) is the vector

\[
\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_N} \end{bmatrix}^\top
\]

and the Hessian \( H(x) \) of \( f(x) \) is the symmetric matrix

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

The expansion to second order is a quadratic function

\[
f(x) = a + g^\top x + \frac{1}{2}x^\top H x
\]
Properties of Quadratic functions

Taylor expansion

\[ f(x_0 + x) = f(x_0) + g^\top x + \frac{1}{2}x^\top H x \]

Expand about a stationary point \( x_0 = x^* \) in direction \( p \)

\[ f(x^* + \alpha p) = f(x^*) + g^\top \alpha p + \frac{1}{2}\alpha^2 p^\top H p \]

\[ = f(x^*) + \frac{1}{2}\alpha^2 p^\top H p \]

since at a stationary point \( g = \nabla f|_{x^*} = 0 \)

At a stationary point the behaviour is determined by \( H \)
H is a symmetric matrix, and so has orthogonal eigenvectors

\[ H u_i = \lambda_i u_i \quad \text{choose } ||u_i|| = 1 \]

\[
f(x^* + \alpha u_i) = f(x^*) + \frac{1}{2} \alpha^2 u_i^\top H u_i
\]

\[= f(x^*) + \frac{1}{2} \alpha^2 \lambda_i\]

As \( |\alpha| \) increases, \( f(x^* + \alpha u_i) \) increases, decreases or is unchanging according to whether \( \lambda_i \) is positive, negative or zero.

H has rotation and axis elements
Examples of Quadratic functions

Case 1: both eigenvalues positive

\[ f(x) = a + g^\top x + \frac{1}{2}x^\top H x \]

with

\[ a = 0 \]
\[ g = \begin{bmatrix} -50 \\ -50 \end{bmatrix} \]
\[ H = \begin{bmatrix} 6 & 4 \\ 4 & 6 \end{bmatrix} \]
Case 2: eigenvalues have different signs

\[ f(x) = a + g^\top x + \frac{1}{2} x^\top H x \]

with

\[ a = 0 \quad g = \begin{bmatrix} -30 \\ +20 \end{bmatrix} \quad H = \begin{bmatrix} 6 & 0 \\ 0 & -4 \end{bmatrix} \]

saddle surface: extremum but not a minimum
Case 3: one eigenvalue zero

\[ f(x) = a + g^\top x + \frac{1}{2} x^\top H x \]

with

\[ a = 0 \quad g = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad H = \begin{bmatrix} 6 & 0 \\ 0 & 0 \end{bmatrix} \]

parabolic cylinder
Types of Stationary Point

Hessian positive definite
Convex function.
Minimum point.

Hessian negative definite
Concave function
Maximum point.

Hessian mixed.
Surface has negative curvature.
Saddle point.
Optimization in N dimensions – line search

- Reduce optimization in N dimensions to a series of (1D) line minimizations
- Use methods developed in 1D (e.g. polynomial interpolation)
An Optimization Algorithm

Start at \( x_0 \) then repeat

1. compute a search direction \( p_k \)

2. compute a step length \( \alpha_k \), such that \( f(x_k + \alpha_k p_k) < f(x_k) \)

3. update \( x_{k+1} \leftarrow x_k + \alpha_k p_k \)

4. check for convergence (termination criteria) e.g. \( \nabla f = 0 \)

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

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

\[ x_{n+1} = x_n + \alpha_n p_n \]

The steepest descent method chooses \( p_n \) to be parallel to the negative gradient

\[ p_n = -\nabla f(x_n) \]

Step-size \( \alpha_n \) is chosen to minimize \( f(x_n + \alpha_n p_n) \). For quadratic forms there is a closed form solution:

\[ \alpha_n = -\frac{p_n^\top p_n}{p_n^\top H p_n} \]

[exercise]
What is the problem of Steepest Ascent?
Example

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

Steepest descent \((x_0 = [0, 14])\)

- 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.
• Newton’s method
  • Line search

• Quasi-Newton methods

• Least-Squares and Gauss-Newton methods

• Downhill simplex (amoeba) algorithm
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 is at \([1, 1]\)
What problems with this one?

\[ f(x, y) = 100(y - x^2)^2 + (1 - x)^2 \]

Minimum is at \([1, 1]\)
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 (100 iterations)
- The algorithm crawls down the valley
Performance issues for optimization algorithms

1. Number of iterations required

2. Cost per iteration

3. Memory footprint

4. Region of convergence
Recall from lecture 1: Newton’s method in 1D

Fit a quadratic approximation to $f(x)$ using both first and second derivatives 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 $\delta x$ which minimizes this local quadratic approximation.

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

- Update $x$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$
Recall from lecture 1: Taylor expansion in 2D

A function may be approximated locally by its Taylor series expansion about a point $x_0$

$$f(x_0 + \delta x) \approx f(x_0) + \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \left( \begin{array}{c} \delta x \\ \delta y \end{array} \right) + \frac{1}{2} (\delta x, \delta y) \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \begin{array}{c} \delta x \\ \delta y \end{array}$$

The expansion to second order is a quadratic function

$$f(x_0 + \delta x) = a + g^\top \delta x + \frac{1}{2} \delta x^\top H \delta x$$
Newton’s method in ND

Expand $f(x)$ by its Taylor series about the point $x_n$

$$f(x_n + \delta x) \approx f(x_n) + g_n^\top \delta x + \frac{1}{2} \delta x^\top H_n \delta x$$

where the gradient is the vector

$$g_n = \nabla f(x_n) = \left[ \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_N} \right]^\top$$

and the Hessian is the symmetric matrix

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

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

$$\nabla f(x) = g_n + H_n \delta x = 0$$

with solution $\delta x = -H_n^{-1} g_n$. This gives the iterative update

$$x_{n+1} = x_n - H_n^{-1} g_n$$
Assume that $H$ is positive definite (all eigenvalues greater than zero)

\[ x_{n+1} = x_n + \delta x = x_n - H_n^{-1}g_n \]

- If $f(x)$ is quadratic, then the solution is found in one step.
- The method has quadratic convergence (as in the 1D case).
- The solution $\delta x = -H_n^{-1}g_n$ is guaranteed to be a downhill direction (provided that $H$ is positive definite)
- For numerical reasons the inverse is not actually computed, instead $\delta x$ is computed as the solution of $H\delta x = -g_n$.
- Rather than jump straight to $x_n - H_n^{-1}g_n$, it is better to perform a line search which ensures global convergence
  \[ x_{n+1} = x_n - \alpha_n H_n^{-1}g_n \]
- If $H = I$ then this reduces to steepest descent.
Newton’s method - example

- 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 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, Fletcher, Goldfarb and Shanno (BFGS), and also Davidson, Fletcher and Powell (DFP).

**e.g. in 1D**

First derivatives

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

Second derivative

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

For $H_{n+1}$ build an approximation from $H_n, g_n, g_{n+1}, x_n, x_{n+1}$.
Quasi-Newton: BFGS

- Set $H_0 = I$.

- Update according to

$$H_{n+1} = H_n + \frac{q_n q_n^\top}{q_n^\top s_n} - \frac{(H_n s_n) (H_n s_n)^\top}{s_n^\top H_n s_n}$$

where

$$s_n = x_{n+1} - x_n$$
$$q_n = g_{n+1} - g_n$$

- 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.
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
Quick Recap
Optimization algorithm – key ideas

- Find $\delta x$ such that $f(x + \delta x) < f(x)$

- This leads to an iterative update $x_{n+1} = x_n + \delta x$

- Reduce the problem to a series of 1D line searches $\delta x = \alpha p$
Choosing the direction 1: axial iteration

Alternate minimization over x and y
Choosing the direction 2: steepest descent

Move in the direction of the gradient $\nabla f(x_n)$
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.
A harder case: Rosenbrock’s function

\[ f(x, y) = 100(y - x^2)^2 + (1 - x)^2 \]

Minimum is at \([1, 1]\)
Steepest descent on Rosenbrock function

- The zig-zag behaviour is clear in the zoomed view (100 iterations)
- The algorithm crawls down the valley
Conjugate Gradients – sketch only

The method of **conjugate gradients** chooses successive descent directions \( p_n \) such that it is guaranteed to reach the minimum in a finite number of steps.

- Each \( p_n \) is chosen to be conjugate to all previous search directions with respect to the Hessian \( H \):

\[
p_n^\top H p_j = 0, \quad 0 = < j < n
\]

- The resulting search directions are mutually linearly independent.

- Remarkably, \( p_n \) can be chosen using only knowledge of \( p_{n-1}, \nabla f(x_{n-1}) \) and \( \nabla f(x_n) \) (see Numerical Recipes)

\[
p_n = \nabla f_n + \left( \frac{\nabla f_n^\top \nabla f_n}{\nabla f_{n-1}^\top \nabla f_{n-1}} \right) p_{n-1}
\]
Choosing the direction 3: conjugate gradients

Again, uses first derivatives only, but avoids “undoing” previous work

• 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.
Choosing the direction 4: Newton’s method

Start from Taylor expansion in 2D

A function may be approximated locally by its Taylor series expansion about a point $x_0$

$$f(x + \delta x) \approx f(x) + \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \begin{pmatrix} \delta x \\ \delta y \end{pmatrix} + \frac{1}{2} (\delta x, \delta y) \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \begin{pmatrix} \delta x \\ \delta y \end{pmatrix}$$

The expansion to second order is a quadratic function

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

Now minimize this expansion over $\delta x$:

$$\min_{\delta x} f(x + \delta x) = a + g^\top \delta x + \frac{1}{2} \delta x^\top H \delta x$$
\[
\min_{\delta x} f(x + \delta x) = a + g^\top \delta x + \frac{1}{2} \delta x^\top H \delta x
\]

For a minimum we require that \( \nabla f(x + \delta x) = 0 \), and so

\[
\nabla f(x + \delta x) = g + H\delta x = 0
\]

with solution \( \delta x = -H^{-1}g \) (Matlab \( \delta x = -H\backslash g \)).

This gives the iterative update

\[
x_{n+1} = x_n - H_n^{-1}g_n
\]

What if \( H = I \)?
If \( f(x) \) is quadratic, then the solution is found in one step.

The method has quadratic convergence (as in the 1D case).

The solution \( \delta x = -H_n^{-1}g_n \) is guaranteed to be a downhill direction provided that \( H \) is positive definite.

Rather than jump straight to the predicted solution at \( x_n - H_n^{-1}g_n \), it is better to perform a line search

\[
x_{n+1} = x_n - \alpha_n H_n^{-1}g_n
\]

If \( H = I \) then this reduces to steepest descent.
Newton’s method - example

- The algorithm converges in only 15 iterations – far superior to steepest descent
- However, the method requires computing the Hessian matrix at each iteration – this is not always feasible
Performance issues for optimization algorithms

1. Number of iterations required

2. Cost per iteration

3. Memory footprint

4. Region of convergence
Special structure for cost function - non-linear least squares

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

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

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

- We also assume that the residuals $r_i$ are: (i) small at the optimum, and (ii) zero-mean.
Non-linear least squares

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

**Gradient**

\[ \nabla f(x) = 2 \sum_{i} r_i(x) \nabla r_i(x) \]

**Hessian**

\[ H = \nabla \nabla^T f(x) = 2 \sum_{i} \nabla \left( r_i(x) \nabla^T r_i(x) \right) \]

\[ = 2 \sum_{i} \nabla r_i(x) \nabla^T r_i(x) + r_i(x) \nabla \nabla^T r_i(x) \]

which is approximated as

\[ H_{\text{GN}} = 2 \sum_{i} \nabla r_i(x) \nabla^T r_i(x) \]

This is the **Gauss-Newton** approximation
Non-linear least squares

\[ x_{n+1} = x_n - \alpha_n H_n^{-1} g_n \quad \text{with} \quad H_n(x) = H_{GN}(x_n) \]

• For this least squares problem: minimization with the Gauss-Newton approximation with line search takes only 14 iterations
Comparison

**Newton**
- Gradient < 1e-3 after 14 iterations

**Gauss-Newton**
- Gradient < 1e-3 after 15 iterations

- Requires computing Hessian
- Exact solution if quadratic
- Approximates Hessian by product of gradient of residuals
- Requires only derivatives
Summary of minimizations methods

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

1. **Newton.**
   $$H \delta x = -g$$

2. **Gauss-Newton.**
   $$H_{GN} \delta x = -g$$

3. **Gradient descent.**
   $$\lambda \delta x = -g$$
Levenberg-Marquardt algorithm

- Away from the minimum, in regions of negative curvature, the Gauss-Newton approximation is not very good.

- In such regions, a simple steepest-descent step is probably the best plan.

- The Levenberg-Marquardt method is a mechanism for varying between steepest-descent and Gauss-Newton steps depending on how good the $H_{GN}$ approximation is locally.
• The method uses the modified Hessian

\[ H(x, \lambda) = H_{GN} + \lambda I \]

• When \( \lambda \) is small, \( H \) approximates the Gauss-Newton Hessian.

• When \( \lambda \) is large, \( H \) is close to the identity, causing steepest-descent steps to be taken.
**LM Algorithm**

\[ H(x, \lambda) = H_{GN}(x) + \lambda I \]

1. Set \( \lambda = 0.001 \) (say)

2. Solve \( \delta x = -H(x, \lambda)^{-1} g \)

3. If \( f(x_n + \delta x) > f(x_n) \), increase \( \lambda (\times 10 \text{ say}) \) and go to 2.

4. Otherwise, decrease \( \lambda (\times 0.1 \text{ say}) \), let \( x_{n+1} = x_n + \delta x \), and go to 2.

**Note:** This algorithm does not require explicit line searches.
Example

- Minimization using Levenberg-Marquardt (no line search) takes 31 iterations.

Matlab: lsqnonlin
Comparison

Gauss-Newton
Gauss-Newton method with line search
gradient < 1e-3 after 14 iterations

Levenberg-Marquardt
Levenberg-Marquardt method
gradient < 1e-3 after 31 iterations

- more iterations than Gauss-Newton, but
- no line search required,
- and more frequently converges
Case study – Bundle Adjustment (non-examinable)

Problem statement

- **Given:** \( n \) matching image points \( x^i_j \) over \( m \) views
- **Find:** the cameras \( P^i \) and the 3D points \( X_j \) such that \( x^i_j = P^i X_j \)

\[
\min_{P^i, X_j} \sum_{j \in \text{points}} \sum_{i \in \text{views}} d(x^i_j, P^i X_j)^2
\]

Notation:

- A 3D point \( X_j \) is imaged in the “i” th view as \( x^i_j = P^i X_j \)
- \( P \) : 3 \( \times \) 4 matrix
- \( x \) : 4-vector
- \( x \) : 3-vector
Number of parameters

\[
\min_{P^i X_j} \sum_{j \in \text{points}} \sum_{i \in \text{views}} d \left( x^i_j, P^i X_j \right)^2
\]

- for each camera there are 6 parameters
- for each 3D point there are 3 parameters

A total of 6 \( m \) + 3 \( n \) parameters must be estimated
  - e.g. 50 frames, 1000 points: 3300 unknowns

For efficient optimization:

- exploit the Gauss-Newton approximation (squared residual cost function)
  - and additional sparsity which means the Hessian has blocks of zeros
Example

image sequence

cameras and points
Application: Augmented reality

original sequence
Augmentation