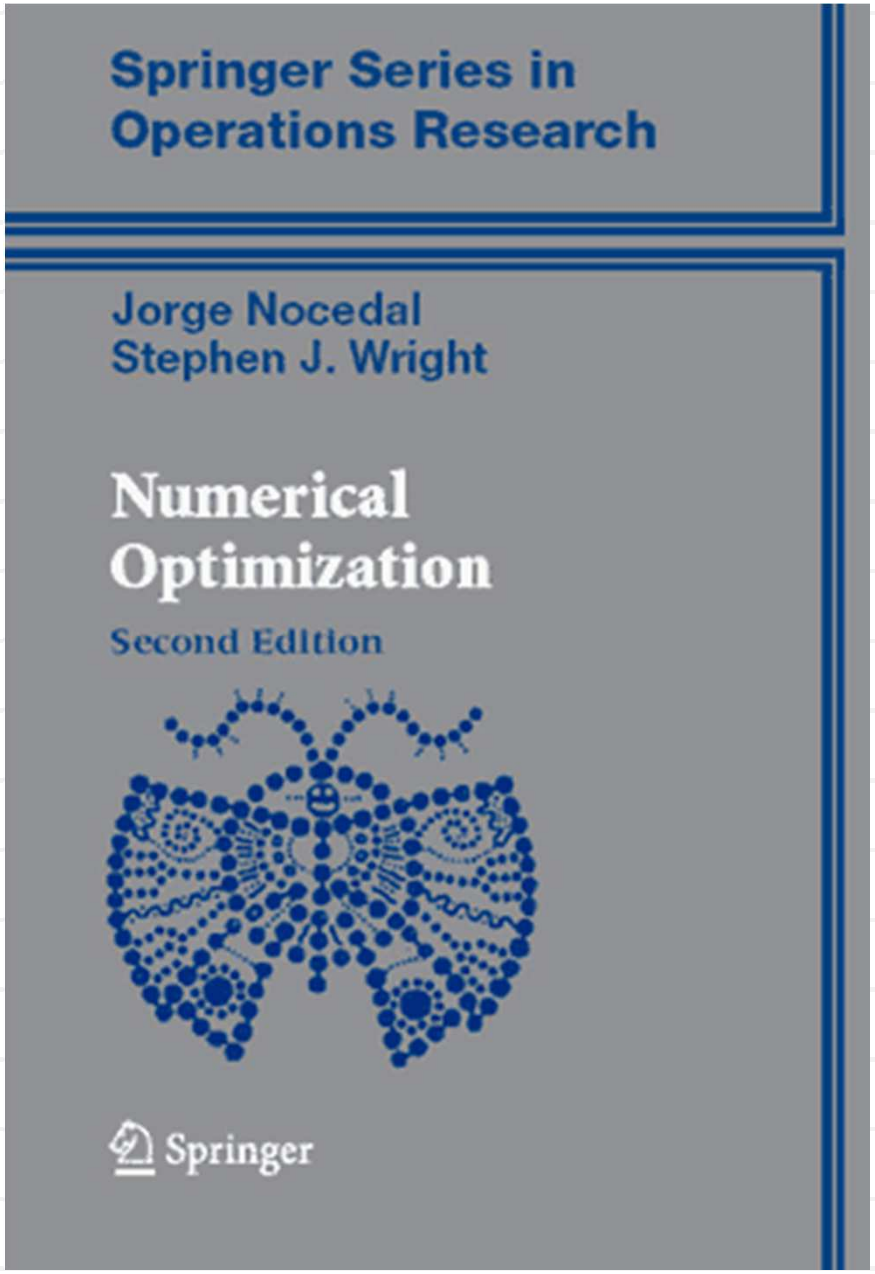
Lecture 11: Unconstrained optimization. Newton method. Quasi-Newton methods.

Textbook


Springer Series in Operations Research

Jorge Nocedal
Stephen J. Wright

Numerical Optimization

Second Edition

Springer

Nocedal, Wright

Recap: probabilistic least squares

$$Xw = y + \text{"noise"}$$

$$\min_w -\log P(X,y|w) - \log(P(w))$$

log-likelihood *log-prior*

We assume that the noise is independent between measurements and Gaussian. Then our likelihood is:

$$P(X,y|w) \propto \prod_i \exp\left(-\frac{\|y_i - x_i^T w\|_2^2}{2\sigma^2}\right)$$

We assume that w is drawn from an isotropic zero-mean Gaussian prior: $w \sim \frac{1}{Z}\exp(-\lambda \|w\|_2^2)$

Thus we get:

$$\mu = 2\lambda\sigma^2$$

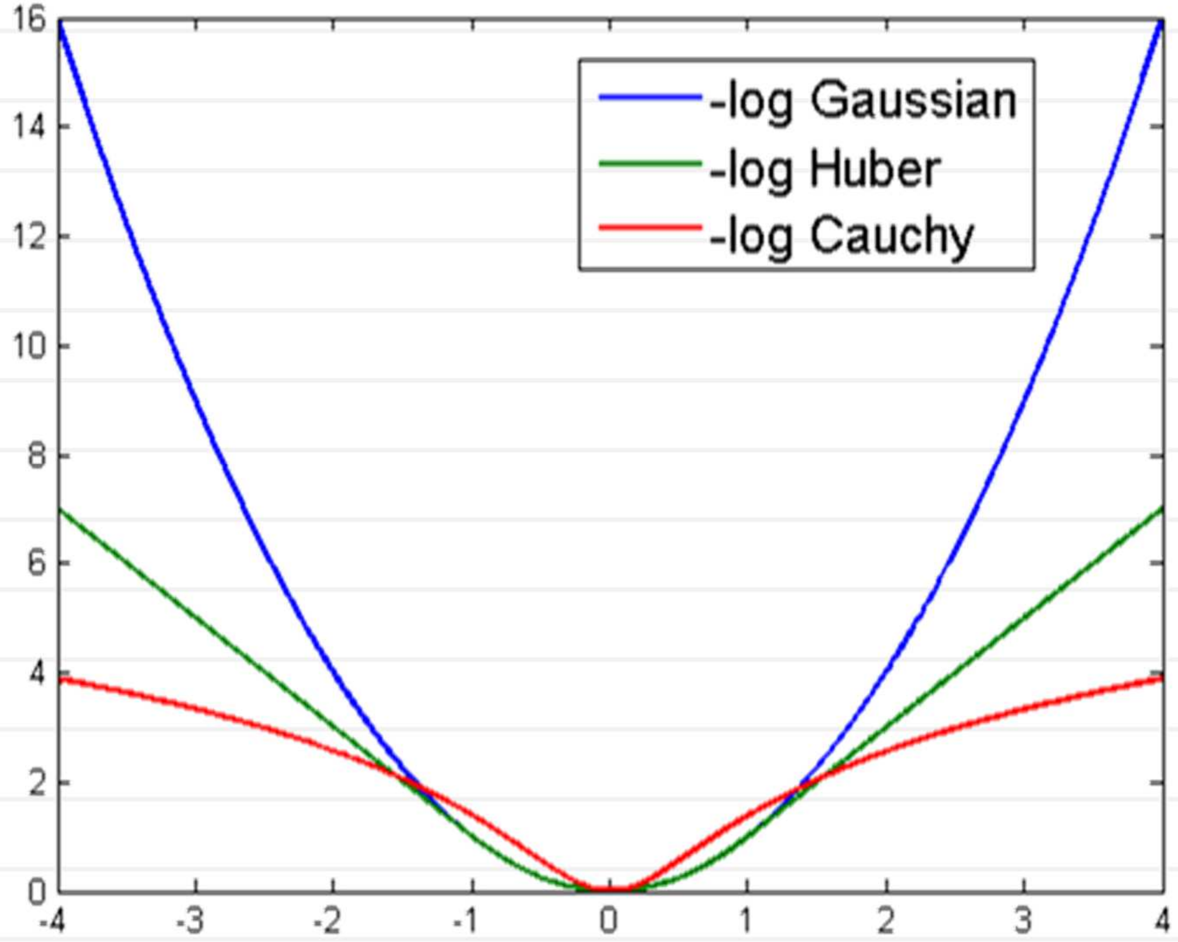
$$\min_w \frac{1}{2}\|Xw - y\|_2^2 + \frac{\mu}{2}\|w\|_2^2$$

Other choices for likelihood

$$\min_{w} \underbrace{-\log P(X, y \mid w)}_{\textit{log-likelihood}} \underbrace{- \log(P(w))}_{\textit{log-prior}}$$

- The noise can be non-Gaussian. E.g. data can contain *outliers*, i.e. data points where results strongly deviate from the model predictions
- Gaussian likelihood penalizes such large deviations very strongly (its *–log* is quadratic, hence grows very fast when we move from zero). Even a single outlier is then likely to perturb our estimate of ***w*** by a lot.
- When outliers are present, we have to use *robust ("heavy-tailed")* distributions to model the posterior.

Robust likelihood functions



Cauchy (**non-convex**, smooth)

Huber (**convex**, smooth)

$$\frac{1}{\pi}\ \frac{\gamma^2}{x^2+\gamma^2}$$

$$\frac{1}{Z}\exp\left(\begin{cases} x^2/2, & |x|<\gamma \\ \gamma(|x|-\gamma/2), & |x|\geq\gamma \end{cases}\right)$$

Adding robustness to Least Squares

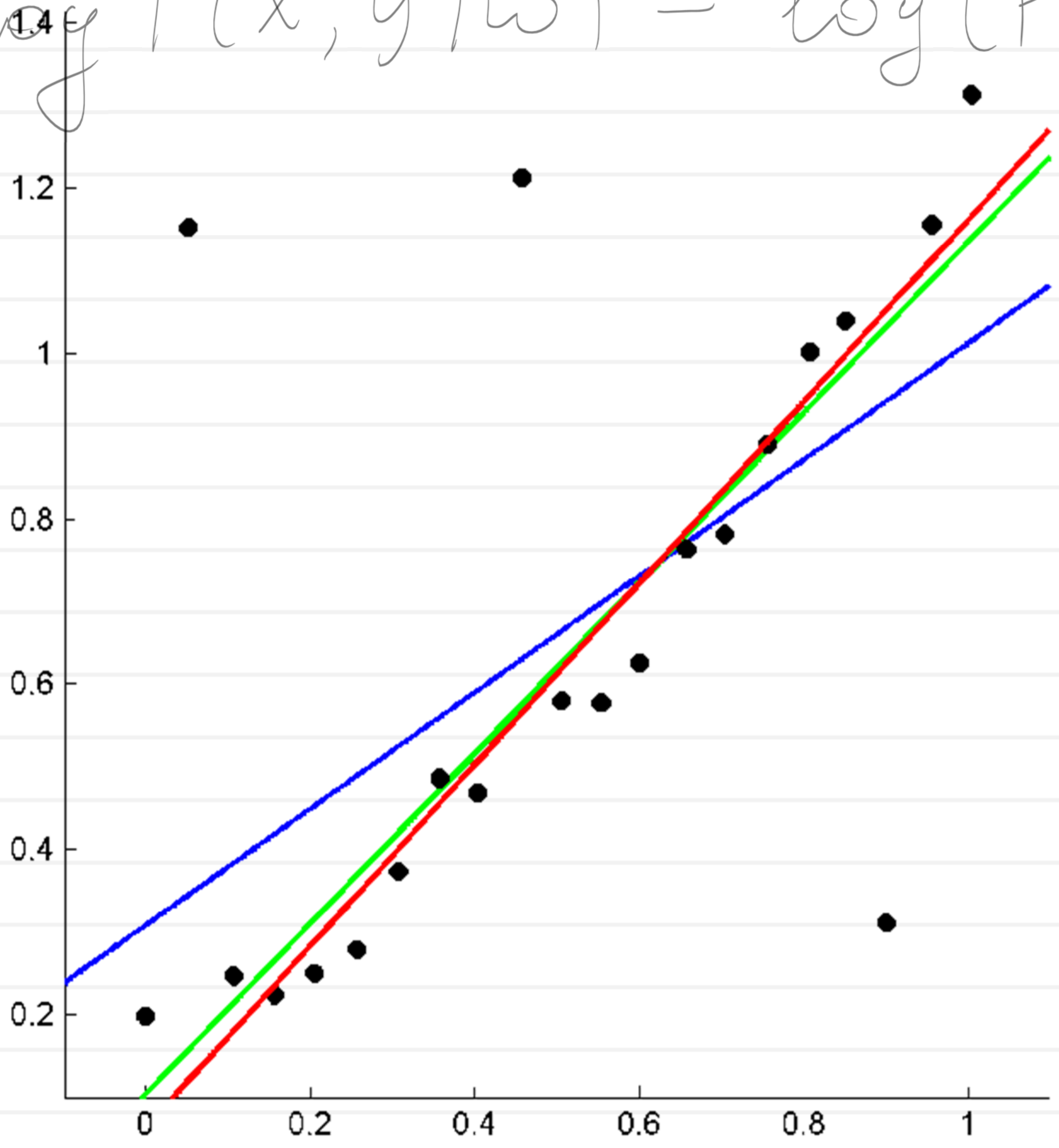
$$\min_{\omega} \underbrace{-\log P(X, y \mid \omega)}_{\text{log-likelihood}} \underbrace{- \log(P(\omega))}_{\text{log-prior}}$$

$$\min_{\omega} \sum_{i=1}^{n} -\log \text{Huber}(x_i^T\omega - y_i) + \lambda \|\omega\|^2$$

$$\| \quad \begin{cases} \frac{1}{2}(x_i^T\omega - y_i)^2, & \text{if } |x_i^T\omega - y| < \gamma \\ \gamma\left(|x_i^T\omega - y_i| - \frac{\gamma}{2}\right), & \text{otherwise} \end{cases}$$

Robust fitting

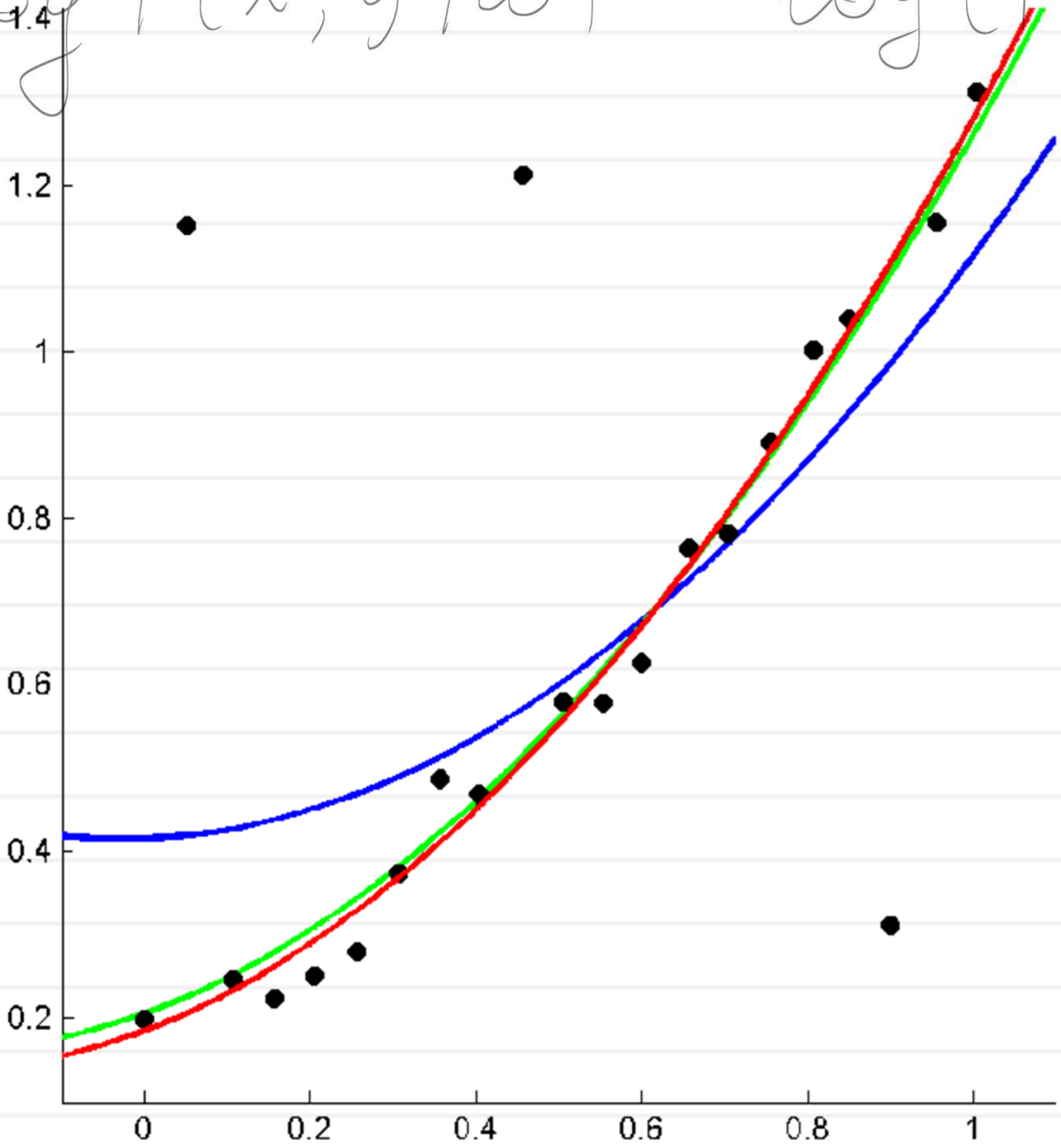
$$\min_{w} -\log P(x, y \mid w) - \log(P(w))$$

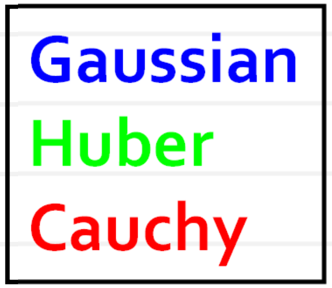


Gaussian
Huber
Cauchy

Robust fitting

$$\min_{w} -\log P(x, y \mid w) - \log(P(w))$$





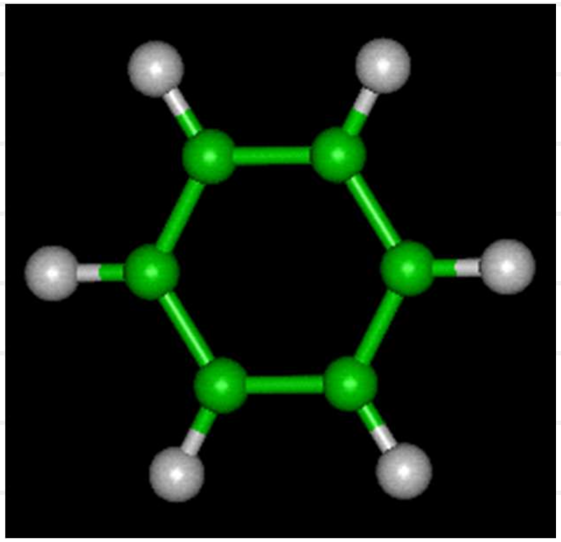
Other choices for prior

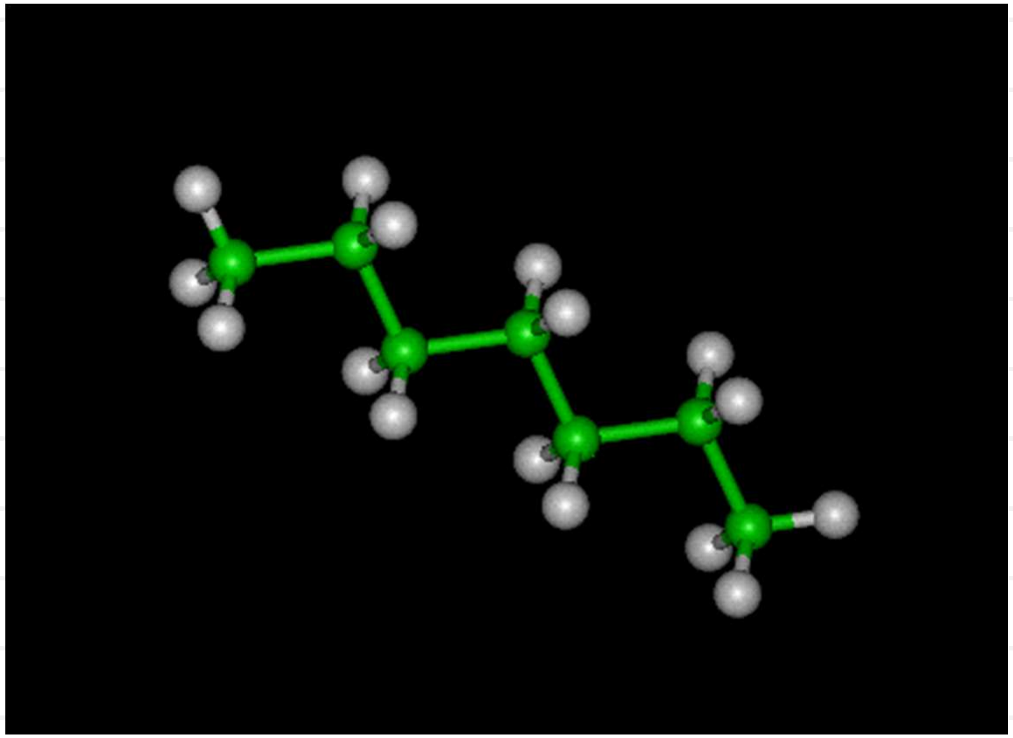
Popular choices for priors include:

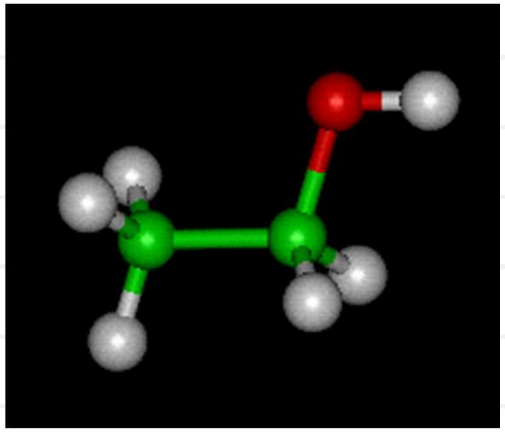
- L0-prior, $-\log P(w) = \lambda ||w||_0$, i.e. number of non-zeros. This lead to interpretable sparse results (in terms of recovered *w*).
- L1-prior, $-\log P(w) = \lambda ||w||_1$, i.e. sum of absolute values. Interestingly, this also lead to sparse results, and in some conditions to the same result as the previous prior.
- $||Lw||$, i.e. penalizing not the coordinates of *w* vector themselves but their linear combinations. Many imaging applications penalize large values of partial derivatives of the resulting image

Keywords (for sparsity inducing priors): sparse coding, compressed sensing,…

Another non-linear optimization problem

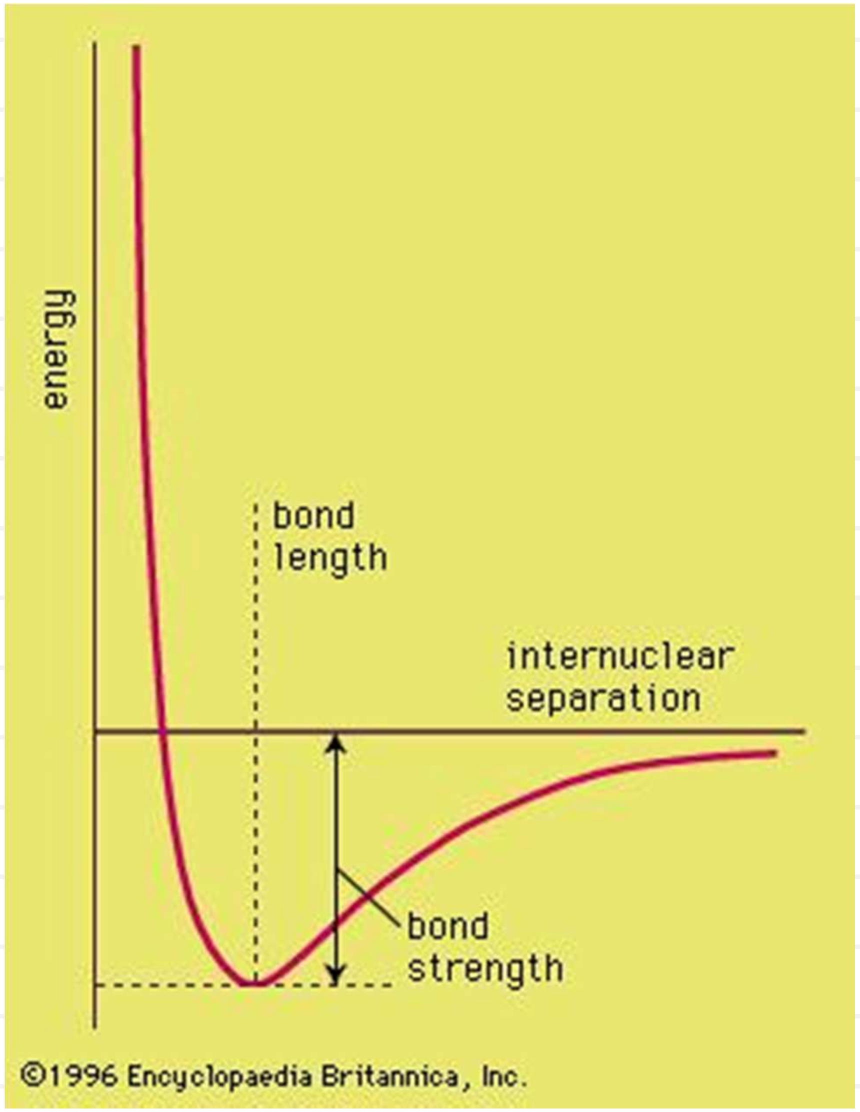

benzene


heptane

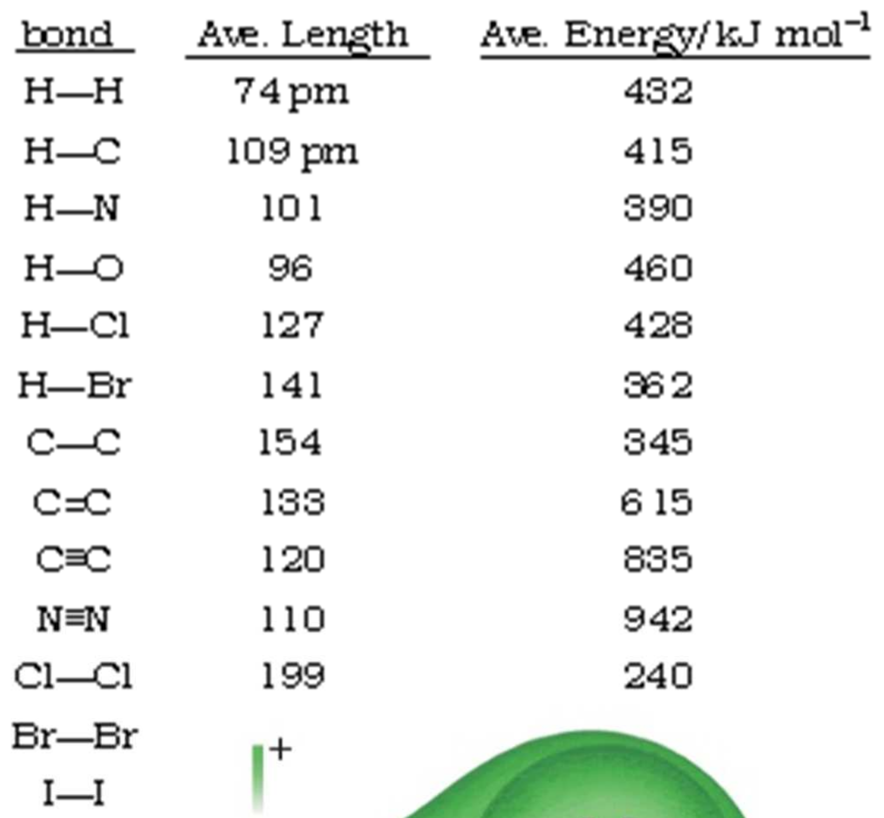

ethanol

Estimation of the molecular shape (from its formula) can be reduced to optimization.

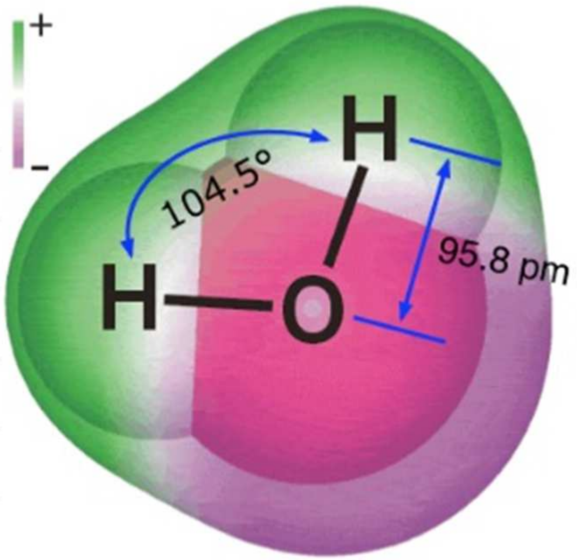
Some tabular data



Potential energy as a function of a bond length



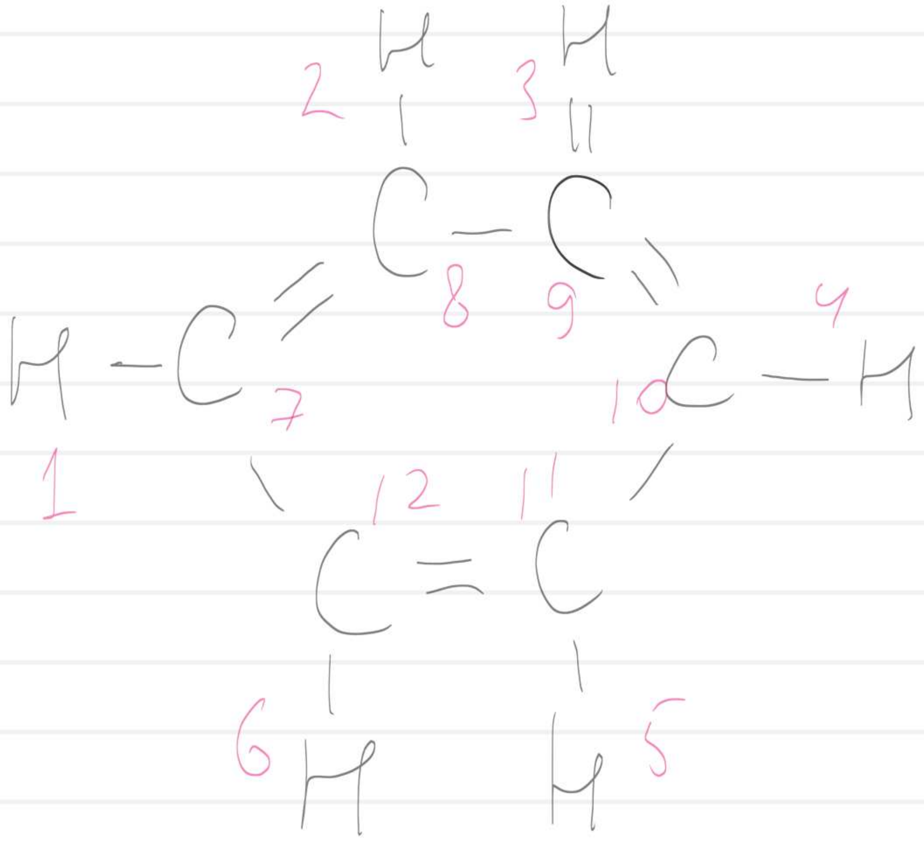
bond	Ave. Length	Ave. Energy/kJ mol^{-1}
H—H	74 pm	432
H—C	109 pm	415
H—N	101	390
H—O	96	460
H—Cl	127	428
H—Br	141	362
C—C	154	345
C=C	133	615
C≡C	120	835
N≡N	110	942
Cl—Cl	199	240
Br—Br		
I—I		



From formula to optimization

$x_i \in \mathbb{R}^3$

(the position of the *i*th atom)



$$\min_{x_1 \dots x_{12}} \sum_{i,j \in N} \Psi_{ij}(x_i\, x_j) + \sum_{i,j,k \in N'} \Psi_{i,j,k}(x_i, x_j, x_k)$$

Newton method (for optimization)

The most important method for $\min_x f(x)$

Main idea: iteratively solve $\nabla f(x) = 0$

Consider 2nd-order Taylor approximation around x_t:

$$f(x) \approx f(x_t) + \nabla f(x_t)^T (x - x_t) + \frac{1}{2}(x - x_t)^T \nabla^2 f(x_t)(x - x_t)$$

Then we need to solve:

$$\nabla f(x) = \nabla f(x_t) + \nabla^2 f(x_t)(x - x_t) = 0$$

We get: $x_{t+1} = x_t - (\nabla_2 f)^{-1} \nabla f(x_t)$

Newton method for non-linear equations

$$\begin{cases} g_1(x)=0 \\ \vdots \\ g_n(x)=0 \end{cases} \qquad g(x) = \vec{0} \qquad\qquad = \left[\frac{\partial g_i}{\partial x_j}\right]$$

"Jacobian"

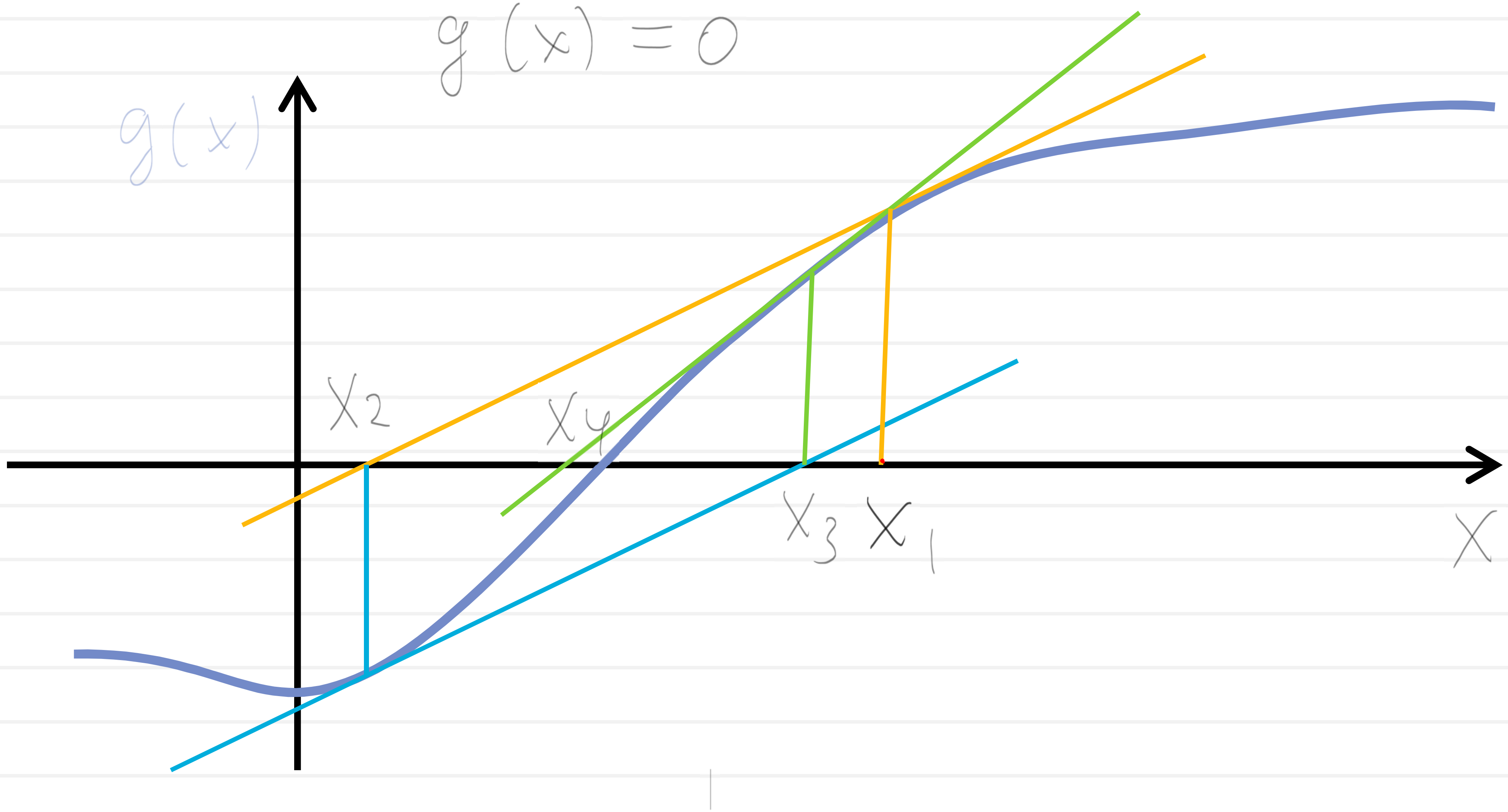
$$g(x) = g(x_t) + \frac{dg}{dx}(x_t)(x - x_t)$$

$$x_{t+1} = x_t - \frac{dg}{dx}(x_t)^{-1} \cdot g(x_t)$$

Newton method for optimization =
Newton method for non-linear equations ($\nabla F = 0$)

"Proof": $\quad \dfrac{d(\nabla f(x))}{dx} = \nabla^2 f$

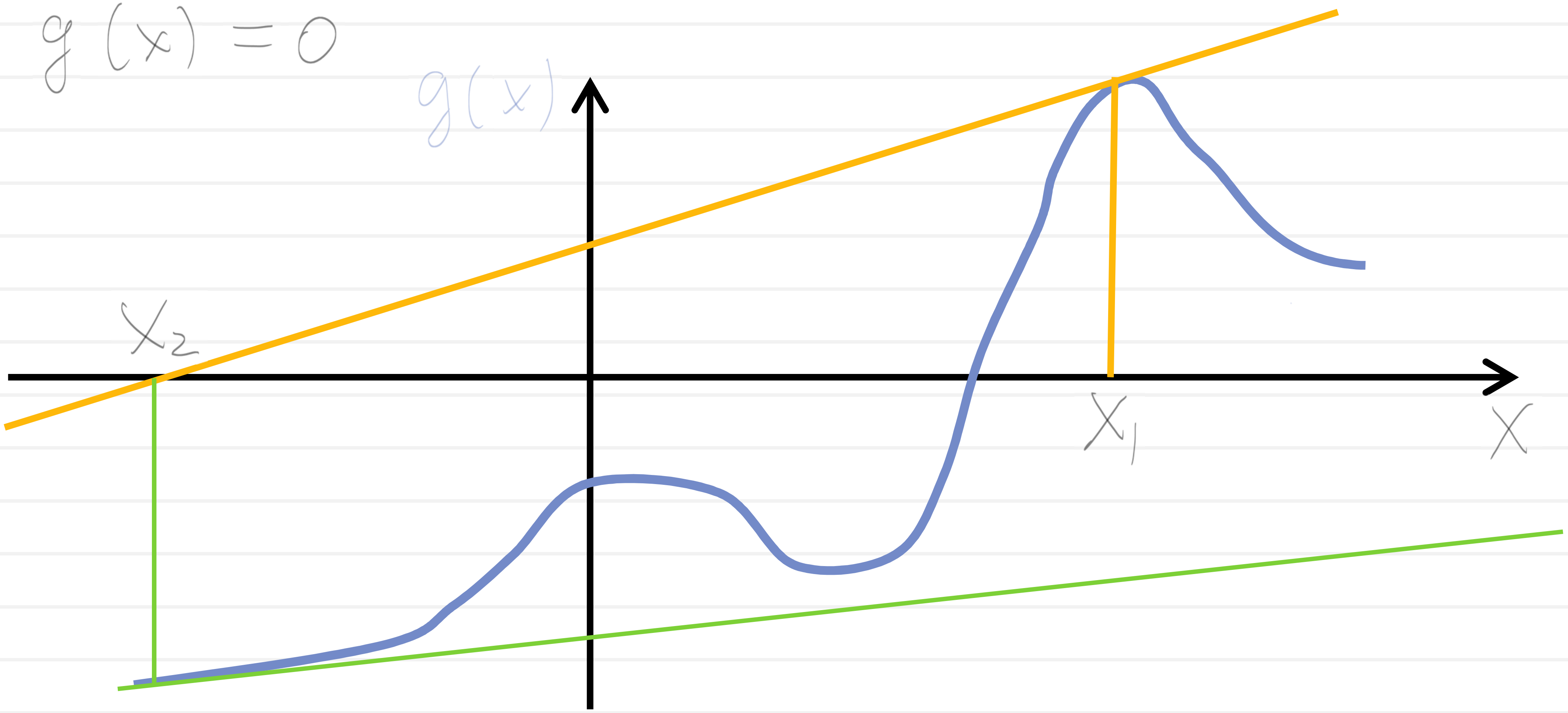
1D example (Newton method)



Observation: after x_4, the convergence will be extremely fast

1D example (Newton method)

$g(x) = 0$



Observation: before we get close to minimum, we can oscillate wildely (and even diverge)

Newton method: caveat 1

- $\nabla F = 0$ is a necessary condition for a minimum, not sufficient
- Thus, Newton steps may converge to a saddle point or even a maximum
- For convex functions, all points with zero gradient are local minima

Newton method: caveat 2

- Newton method may prescribe very big steps (especially if $\nabla^2 f$ is near-degenerate)
- For very big steps, the Taylor approximation is likely to be invalid (same overshooting behaviour as in the 2nd example)

Solution 1: line search

Solution 2: trust region

General Newton algorithm

$$\nabla f(x) = \nabla f(x_t) + \nabla^2 f(x_t)\underbrace{(x - x_t)}_{\Delta x} = 0$$

Vanilla Newton step:

$$x_{t+1} = x_t - (\nabla_2 f)^{-1} \nabla f(x_t)$$

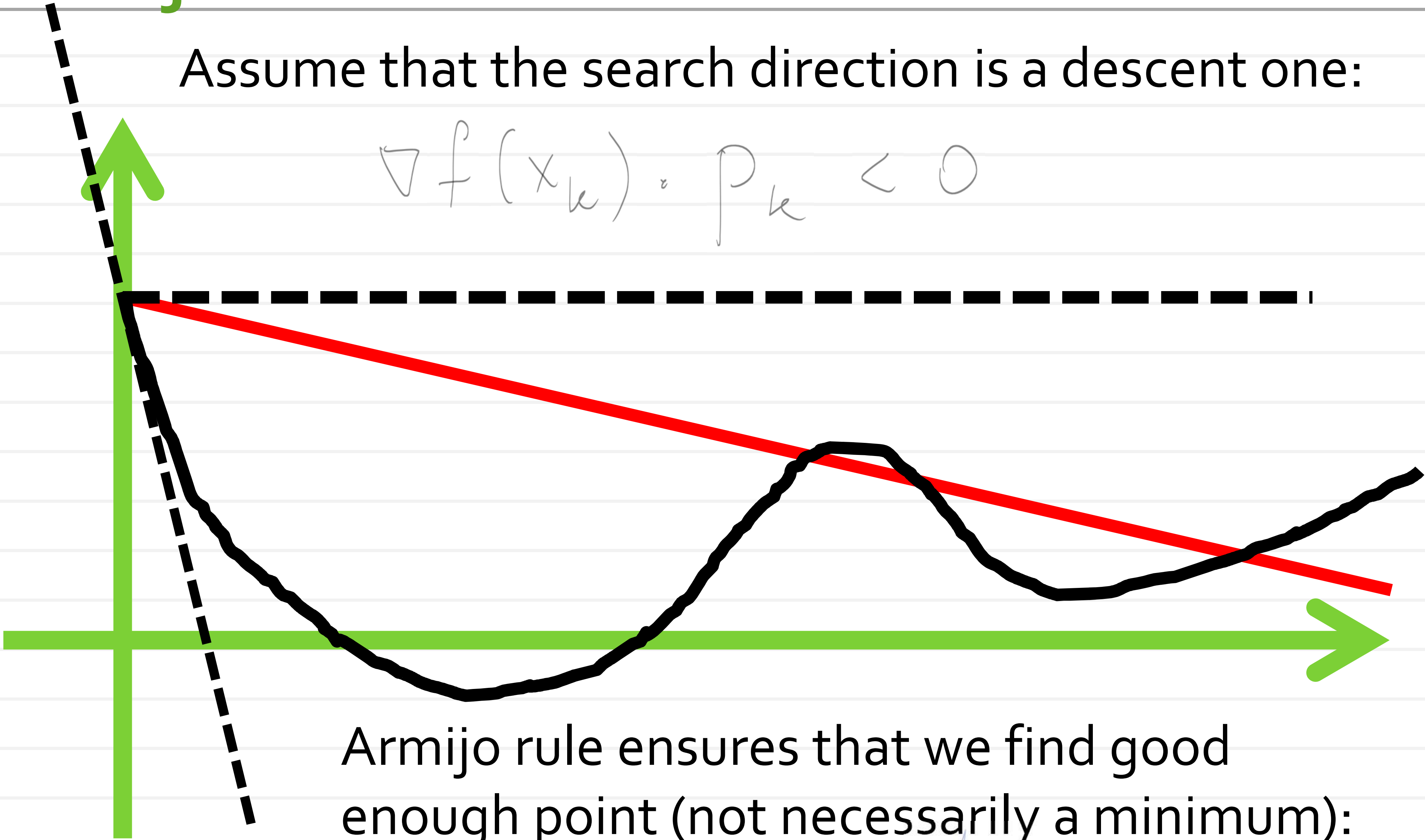
More realistic Newton involves line search:

```
Initialize xo
While $||\nabla f_k|| >$ eps
        $d_k = -\nabla^2 f(x_k)^{-1} * \nabla f_k$;
        $x_{k+1} = \text{LineSearch}(x_k, d_k)$;
end
```

Armijo rule for line search

Assume that the search direction is a descent one:

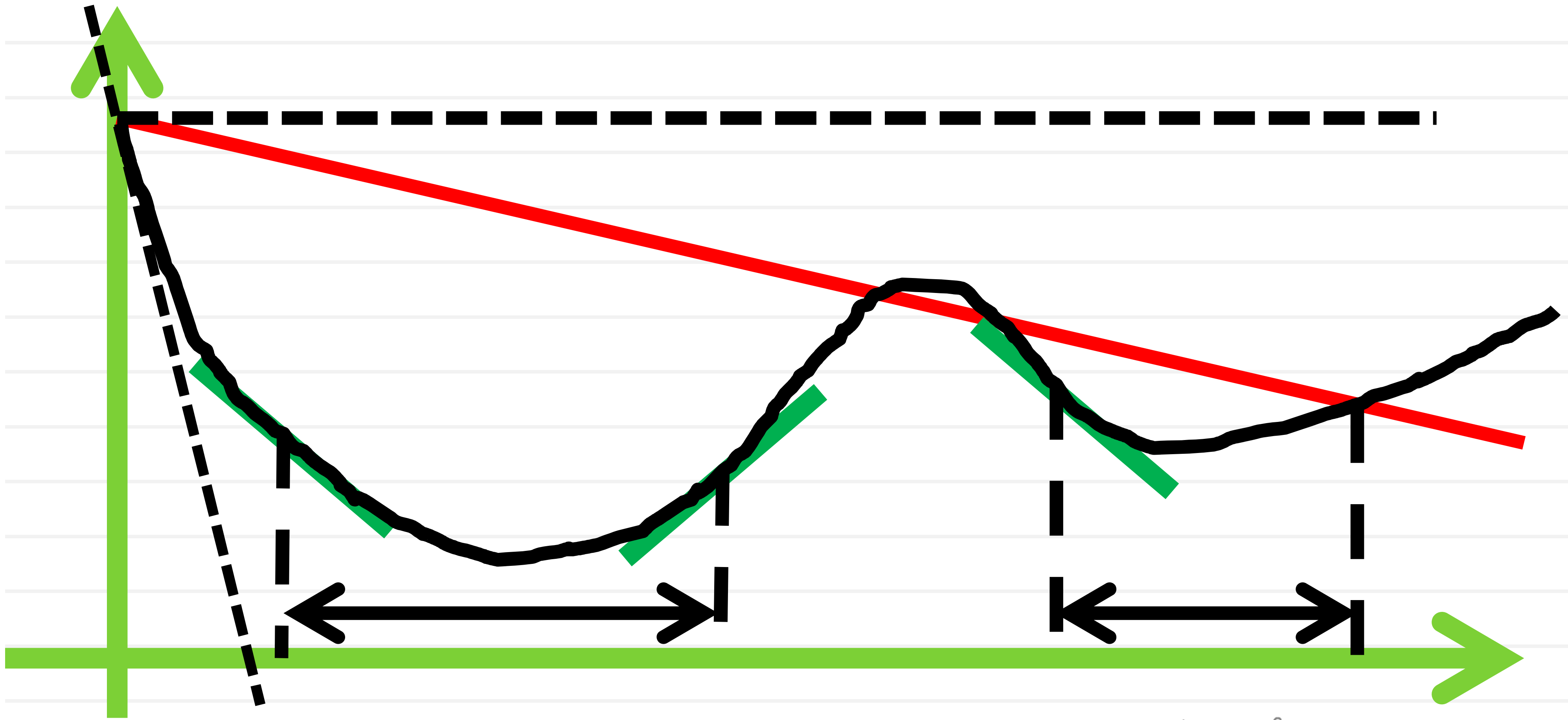
$$\nabla f(x_k) \cdot p_k < 0$$



Armijo rule ensures that we find good enough point (not necessarily a minimum):

$$f(x_k + \alpha_k p_k) \le f(x_k) + \overset{<1}{c_1} \alpha_k (\nabla f(x_k) \cdot p_k)$$

Strong Wolfe conditions for line search



$$f(x_k+\alpha_k p_k) \le f(x_k) + c_1 \alpha_k (\nabla f(x_k) \cdot p_k)$$

$$|p_k^T \nabla f(x_k+\alpha_k p_k)| \le |c_2 p_k^T \nabla f(x_k)|$$

Backtracking search for Armijo rule

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)$$

Simplest variant: start with α = 1 and divide it by 2, till Armijo rule is satisfied

More efficient variant: based on the evaluated points fit a polynomial approximation and go towards its minimum, given that it satisfies

$$0 < \overset{\overset{0.1}{\parallel}}{c_1} \leq \beta \leq \overset{\overset{0.5}{\parallel}}{c_2} < \alpha_k$$

Newton method: caveat 3

- In high dimensions, the Hessian is expensive to evaluate
- In high dimensions, the Hessian is expensive to invert
- In very high dimensions, the Hessian is even expensive to store

Recap: Hessian in Levenberg-Marquardt

$$\nabla^2 F(\omega) = \cancel{\frac{d\, J(\omega)^T}{d\omega} \cdot r(\omega)} + J(\omega)^T J(\omega) + \lambda I$$

- Easily computable
- Easily invertible (when Jacobian is sparse)

$$\nabla^2 F(\omega) \cdot \Delta\omega = -\nabla F(\omega)$$

- **Positive definite, well conditioned**

Quasi-Newton method

Quasi-Newton methods use the approximation:

$$f(x+\Delta x) = f(x) + \nabla f(x)^T \Delta x + \frac{1}{2}\Delta x^T B(x) \Delta x$$

Quasi-Newton step: $\Delta x = -B^{-1} \nabla f(x)$

If $B(x) = \nabla^2 f(x)$, then Quasi-Newton -> Newton.

Quasi-Newton idea: estimate B(x), which is similar to the true Hessian, without evaluating the second derivatives (just by observing gradients).

General quasi-Newton algorithm

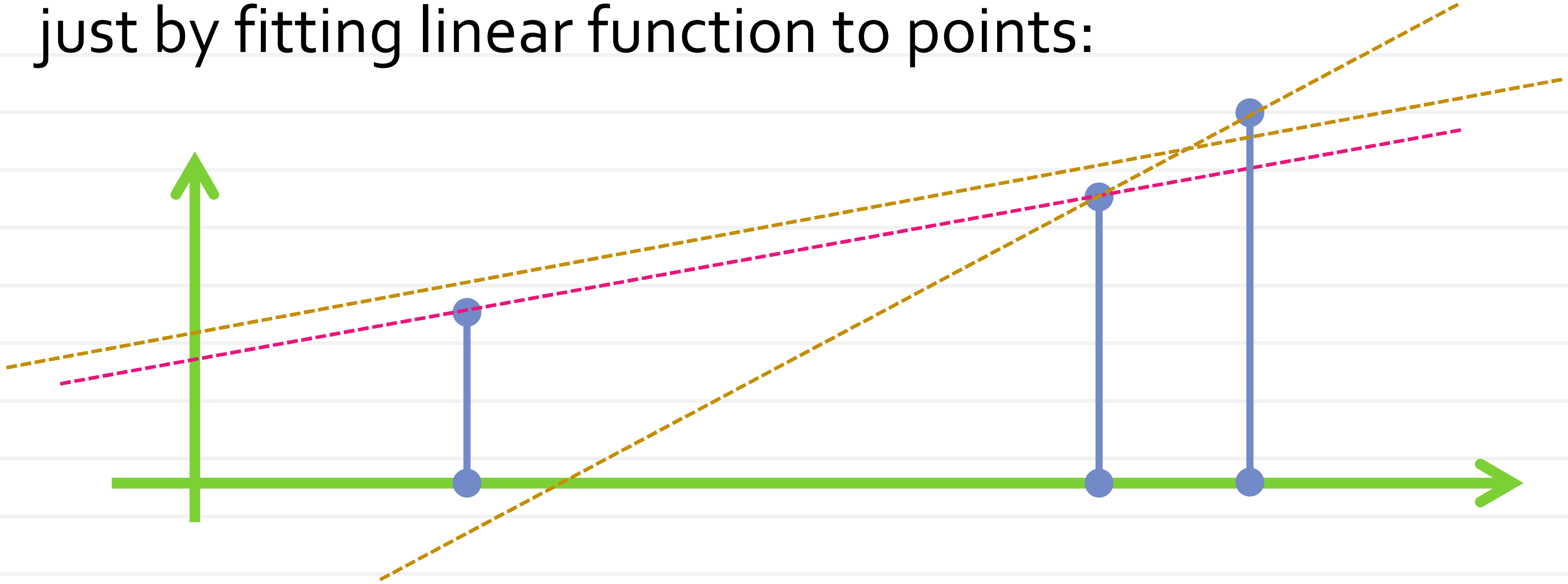
```
Initialize xo, $\nabla fo = \nabla f(x_o)$, $B_o = \lambda$ Identity
While $||\nabla f_k|| > eps$
        $d_k = -B_k^{-1} * \nabla f_k$;
        $x_{k+1} = \text{LineSearch}(x_k, d_k)$;
        $\nabla f_{k+1} = \nabla f(x_{k+1})$
        $B_{k+1} = \textbf{Update}(B_k, \nabla f_{k+1}, \nabla f_k)$;
end
```

Quasi-Newton idea: estimate B(x) which is similar to the true Hessian, without evaluating the second derivatives (just by observing gradients).

Estimating gradient without differentiation

- How to estimate Hessian without evaluating second derivatives?
- Analogous question: how to estimate the gradient without evaluating first derivatives?

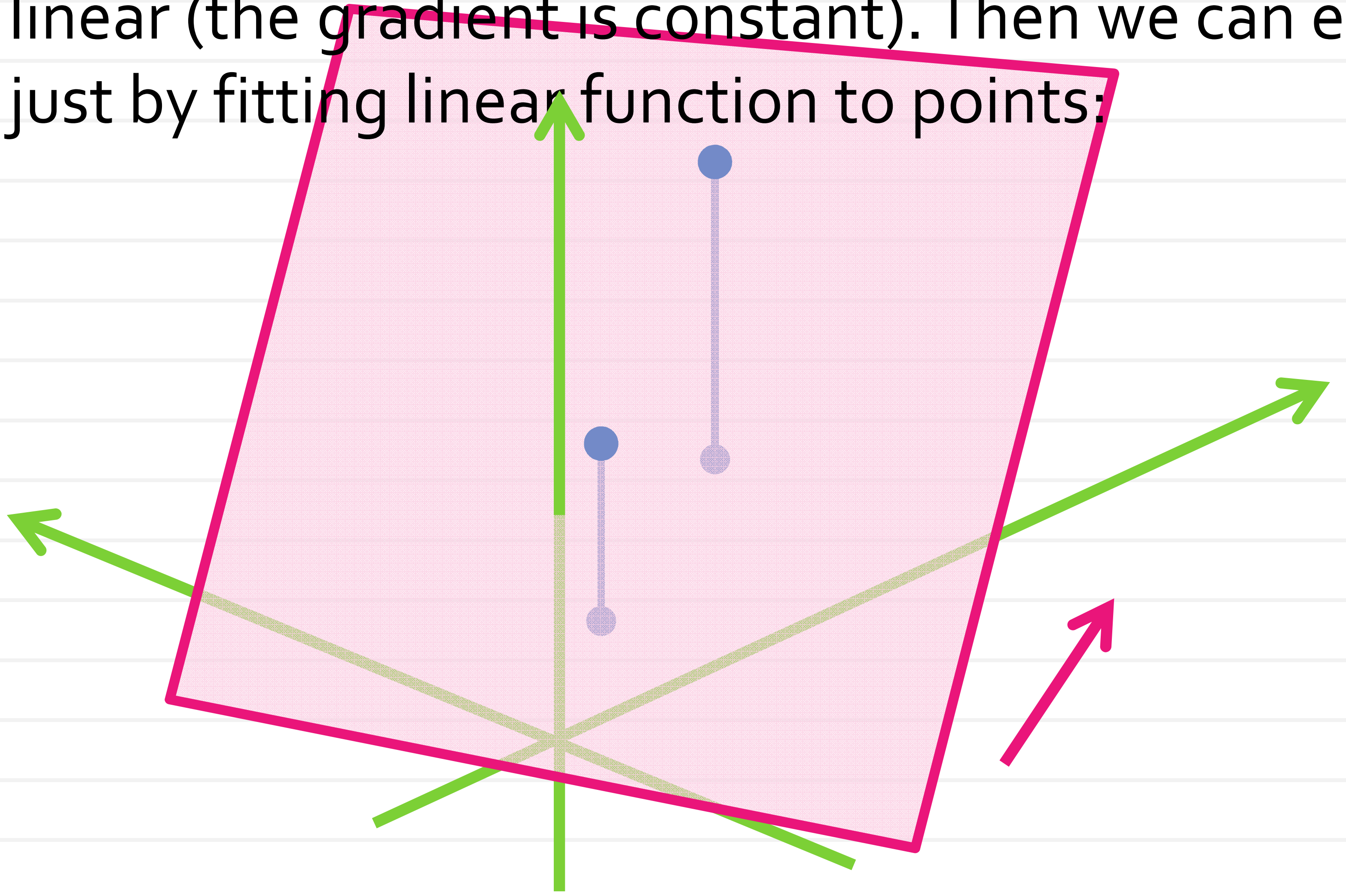
Assume the function is approximately linear (the gradient is constant). Then we can estimate it just by fitting linear function to points:



Estimating gradient without differentiation

- Analogous question: how to estimate the gradient without evaluating first derivatives?

Assume the function is approximately linear (the gradient is constant). Then we can estimate it just by fitting linear function to points:



Quasi-Newton update

B_k = **Update**(B_{k-1}, ∇f_k ,∇f_{k-1});

For a real Hessian and a quadratic function (where 2nd order approximation is perfect), we would have:

$$\nabla f(x) = \nabla f(x_t) + \nabla^2 f(x_t)(x - x_t)$$

Therefore, we seek B_k that fits:

$$\nabla f_k = \nabla f_{k-1} + B_k \cdot (x_k - x_{k-1})$$

$$y_k = B_k \cdot s_k$$

This is D equations on $D^2/2$ variables, so we need further assumptions to impose on B_k

"Symmetric Rank 1"-update

$$\nabla f_k = \nabla f_{k-1} + B_k \cdot (x_k - x_{k-1})$$

$$y_k = B_k \cdot s_k$$

Quasi-Newton with SR1 ("Symmetric Rank-1") updates:

$$B_k = B_{k-1} + \overset{\pm 1}{\overset{\shortparallel}{\sigma}} v \cdot v^T$$

$$y_k = \sigma v v^T \cdot s_k + B_{k-1} s_k$$

$$v = \delta (y_k - B_{k-1} s_k)$$

$$y_k - B_{k-1} s_k = \sigma \delta^2 (y_k - B_{k-1} s_k)(y_k - B_{k-1} s_k)^T s_k$$

“Symmetric Rank 1”-update

$$y_k - B_{k-1} s_k = \sigma \delta^2 (y_k - B_{k-1} s_k)(y_k - B_{k-1} s_k)^T s_k$$

$$\sigma \delta^2 (y_k - B_{k-1} s_k)^T s_k = 1$$

$$\sigma = \operatorname{sgn} (y_k - B_{k-1} s_k)^T s_k$$

$$\delta = \left| (y_k - B_{k-1} s_k)^T s_k \right|^{-1/2}$$

$$B_k = B_{k-1} + \frac{(y_k - B_{k-1} s_k)(y_k - B_{k-1} s_k)^T}{(y_k - B_{k-1} s_k)^T s_k}$$

"Symmetric Rank 1"-update

$$B_k = B_{k-1} + \frac{(y_k - B_{k-1}s_k)(y_k - B_{k-1}s_k)^T}{(y_k - B_{k-1}s_k)^T s_k}$$

- Works well unless the denominator vanishes
- Fix: skip the update if denominator ≈ 0
- More principled solution: Rank-2 updates
- A Rank-2 method (BFGS) is known as "the best universal" unconstrained optimization method for smooth functions

Final words about quasi-Newton methods

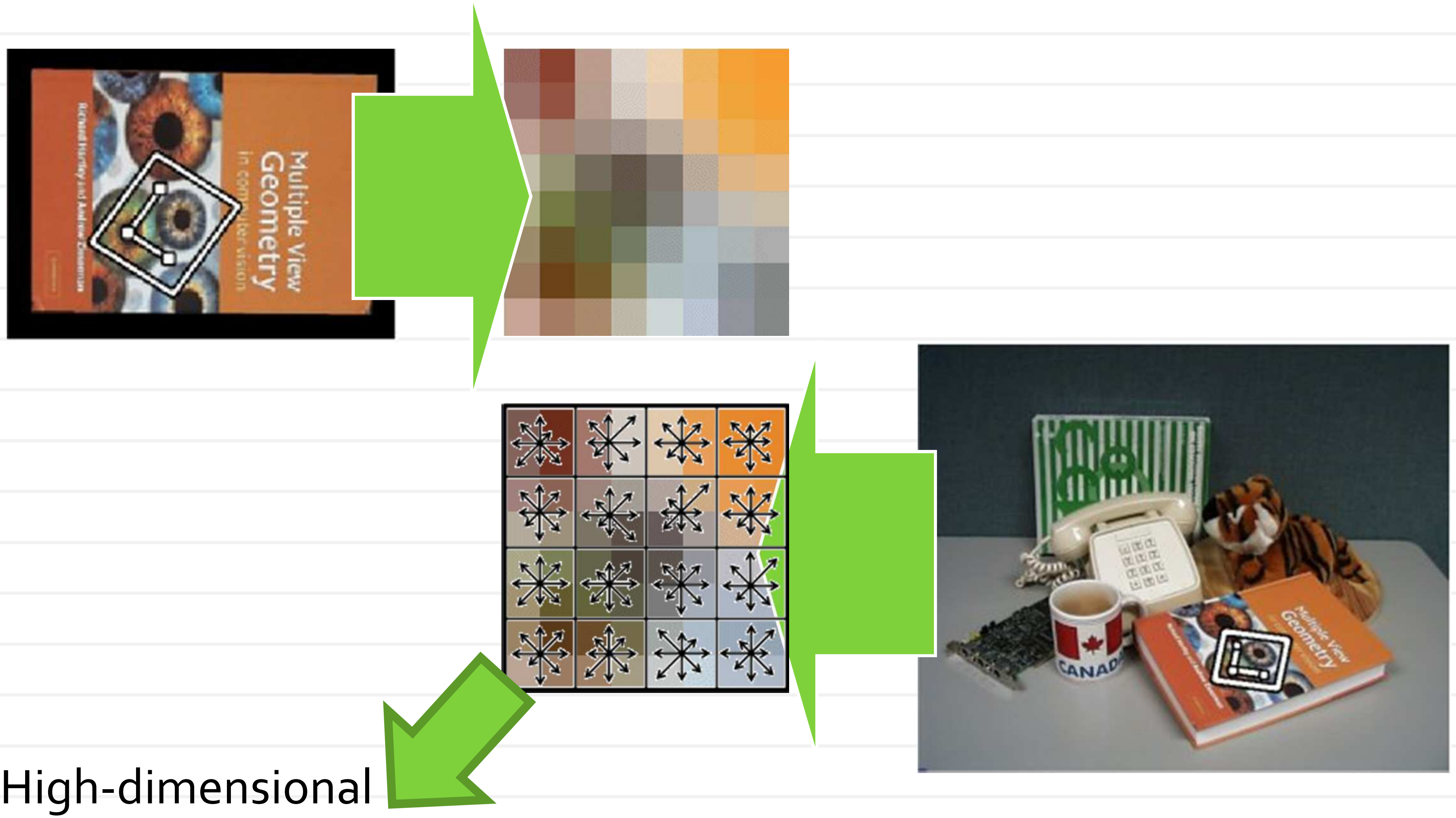
- Quasi-Newton (in particular, BFGS) are widely regarded as the most robust and versatile methods for unconstrained smooth optimization
- Another smart idea: maintain and update B_k^{-1}. In this way we do not need to invert matrices in order to solve for steps!
- For quadratic functions, some update rules converge to the true Hessian

Application: structure-and-motion

Ultimate goal (multiview stereo): given an unstructured collection of photographs of an object recover/build its 3D model

Intermediate step (structure-and-motion): estimate the position of cameras and a sparse set of points in a global reference frame

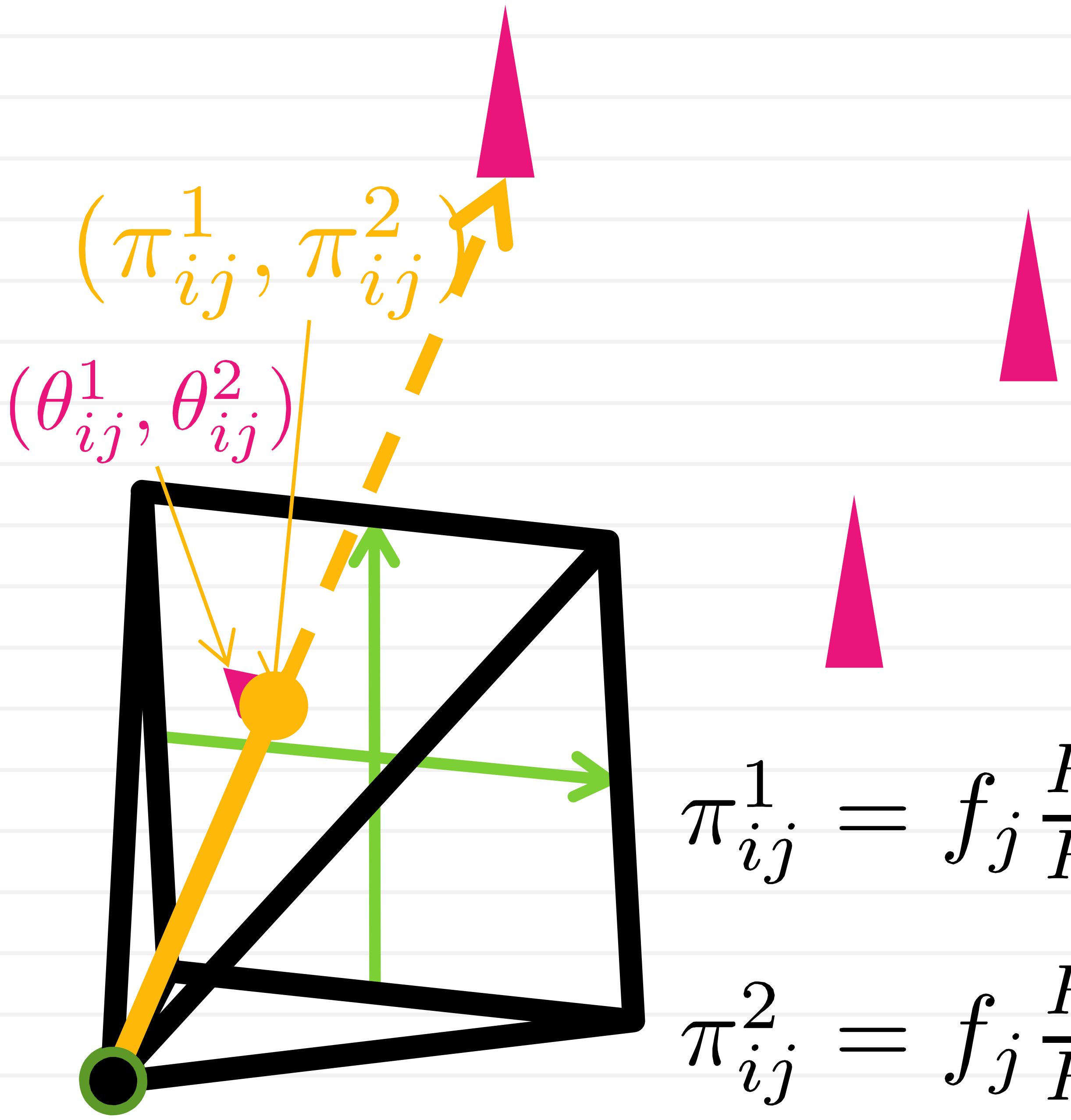
Automated feature detection

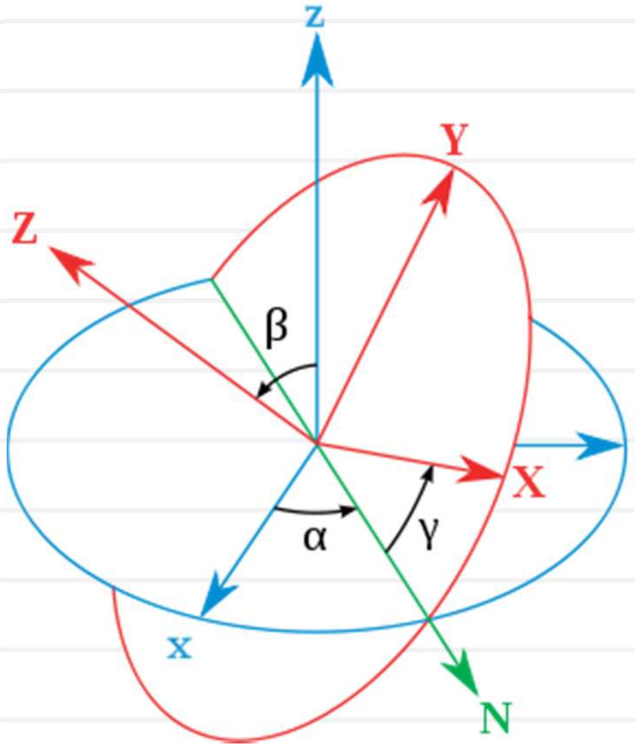


High-dimensional descriptor invariant to many transformations

Slide from Brown and Lowe

Math behind structure-and-motion

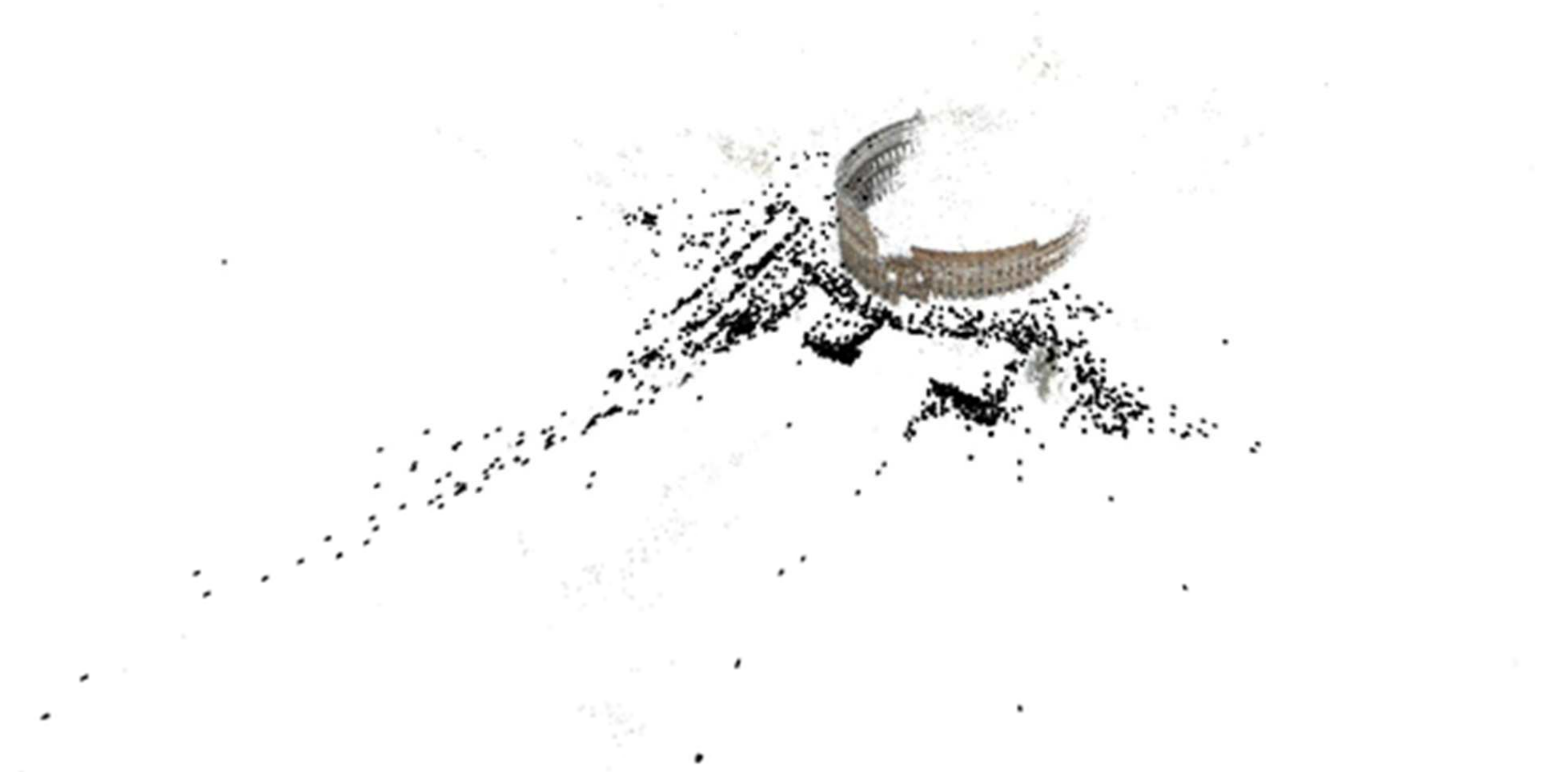




$$\pi_{ij}^1 = f_j \frac{R(\alpha_j,\beta_j,\gamma_j)^1 (\mathbf{b}_i - \mathbf{p}_j)}{R(\alpha_j,\beta_j,\gamma_j)^3 (\mathbf{b}_i - \mathbf{p}_j)}$$

$$\pi_{ij}^2 = f_j \frac{R(\alpha_j,\beta_j,\gamma_j)^2 (\mathbf{b}_i - \mathbf{p}_j)}{R(\alpha_j,\beta_j,\gamma_j)^3 (\mathbf{b}_i - \mathbf{p}_j)}$$

A result of structure-and-motion



[Agarwal, Snavely, Simon, Seitz, Szeliski, "Building Rome in a Day", CVPR 2009]

A result of structure-and-motion

[Agarwal, Snavely, Simon, Seitz, Szeliski, "Building Rome in a Day", CVPR 2009]

[Agarwal, Snavely, Simon, Seitz, Szeliski, "Building Rome in a Day", CVPR 2009