Non-linear Least Squares

Nick Henderson, AJ Friend
Stanford University

August 19, 2015

Non-linear least squares

$$\text{minimize} \quad \frac{1}{2}\sum_{i=1}^{m} f_i(x)^2$$

- Vector $x \in \mathbf{R}^n$ encodes the model parameters
- Functions f_i measures the residual or error of the model for data point i
- Linear least squares is a special case with

$$f_i(x) = a_i^T x - b_i$$

Gauss (1777-1855)

Oil painting of mathematician and philosopher Carl Friedrich Gauss by G. Biermann
(Public domain)

Cholesky (1875-1918)

Archives de l'Ecole polytechnique (Fonds A. Cholesky)

Triangulation

Outline

Triangulation

Huber loss

Triangulation

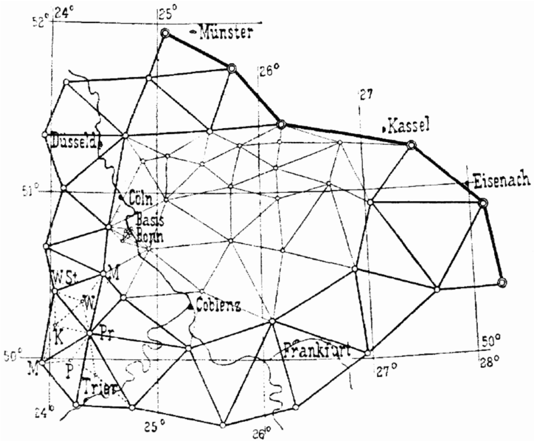


Fig. 4. Die rheiniſch-heſſiſche Kette und das nieder-rheiniſche Dreiecksnetz.

Nineteenth-century triangulation network for the triangulation of Rhineland-Hesse (Public Domain)

Simplified problem statement

- Desire to compute position of n points denoted (x_i, y_i)
- Have set of distances between points in triangulation network, denoted d_{ij}
- Have a few known anchor points
- Distance equation

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- To simplify the treatment, remove the square root

$$d_{ij}^2 = s_{ij} = (x_i - x_j)^2 + (y_i - y_j)^2$$

Non-linear least squares problem

- Let K be the set of pairs of indices for points with known (measured) distances
- Optimization problem

$$\text{minimize} \quad \sum_{(i,j)\in K} \left((x_i - x_j)^2 + (y_i - y_j)^2 - s_{ij}\right)^2$$

- This is a non-trivial, non-convex problem
- Modern interest in sensor network localization

Revisit Taylor series

- Let $f(x)$ be non-linear model of interest
- Have measurements (data) for $f(x^*)$
- Have approximate solution a
- Desire a better solution
- Look at Taylor series expansion of f

$$f(x^*) = f(a) + f'(a)(x^* - a) + \frac{1}{2}f''(a)(x^* - a)^2 + \cdots$$

- Method: when a is close enough to x^* we can disregard non-linear terms and solve for $\delta x = x^* - a$.

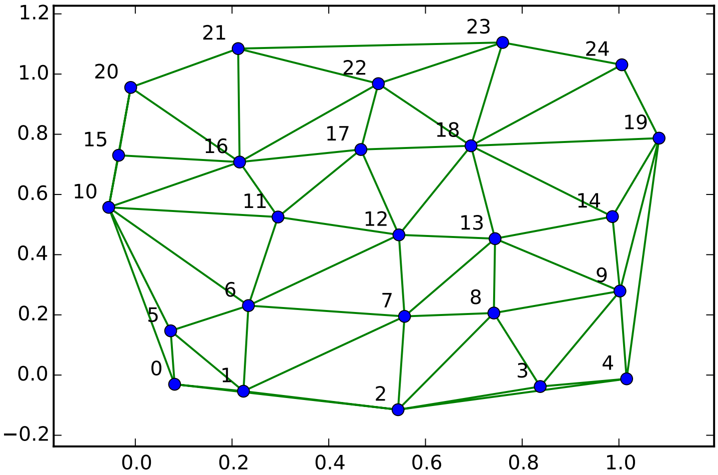
Method details

- s_{ij}: known distances (data)
- c_{ij}: computed distances between approximate point positions
- (x_i, y_i): approximate positions
- $(\delta x_i, \delta y_i)$: adjustment factor (problem variable)
- Linear equations: for each pair of points (i, j) with known distance

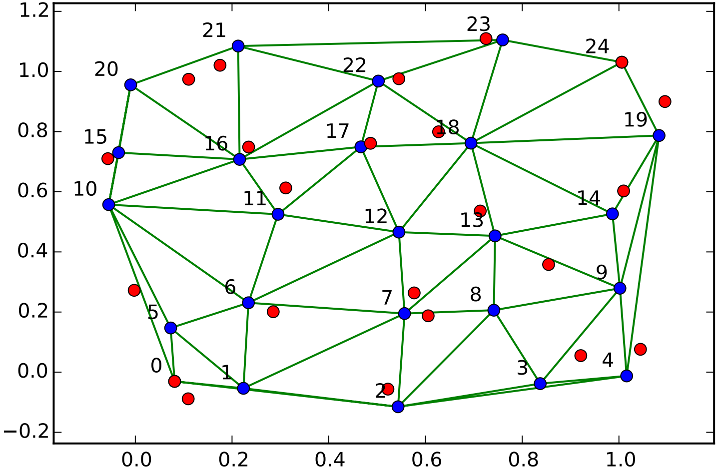
$$s_{ij} - c_{ij} = 2(x_i - x_j)\delta x_i - 2(x_i - x_j)\delta x_j + 2(y_i - y_j)\delta y_i - 2(y_i - y_j)\delta y_j$$

- Solve then update positions: $x_i \leftarrow x_i + \delta x_i$, $y_i \leftarrow y_i + \delta y_i$

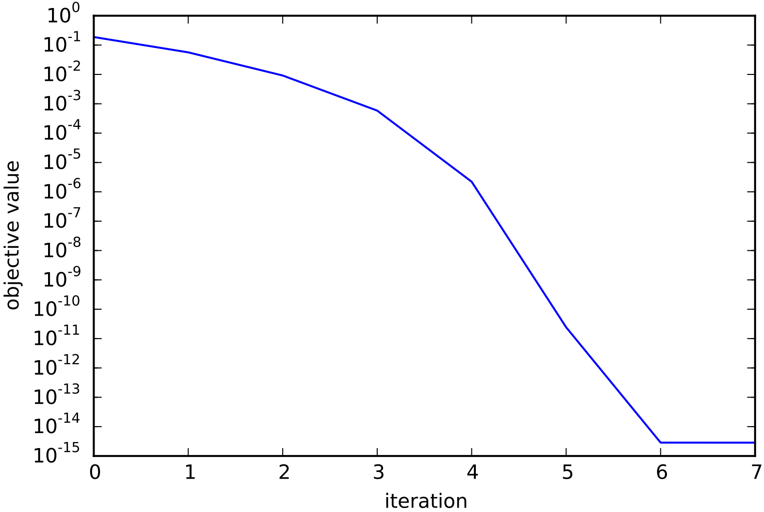
Triangulation example



Triangulation example



Triangulation example



Huber loss

Outline

Triangulation

Huber loss

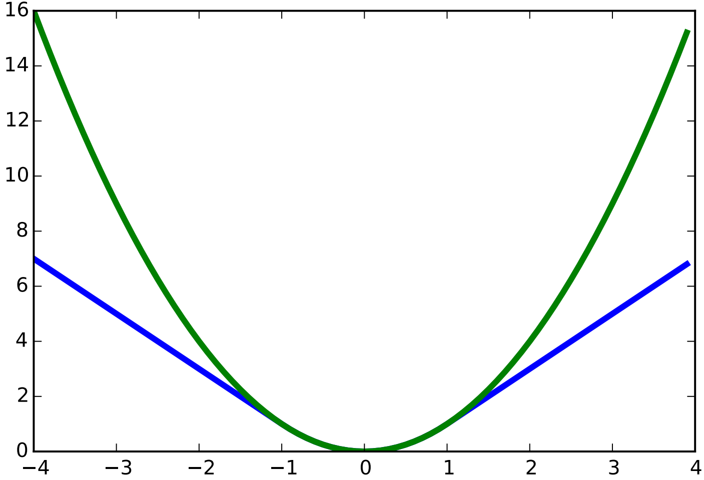
Huber loss

- Huber function allows us to better handle outliers in data
- defined as

$$h_M(x) = \begin{cases} x^2 & |x| \leq M \\ 2M|x| - M^2 & |x| > M \end{cases}$$

- usual quadratic loss in interval $[-M, M]$
- linear loss for $|x| > M$
- linear penalty for large errors is much less severe than quadratic
- large errors are better 'tolerated', have less influence on fit

Huber loss function



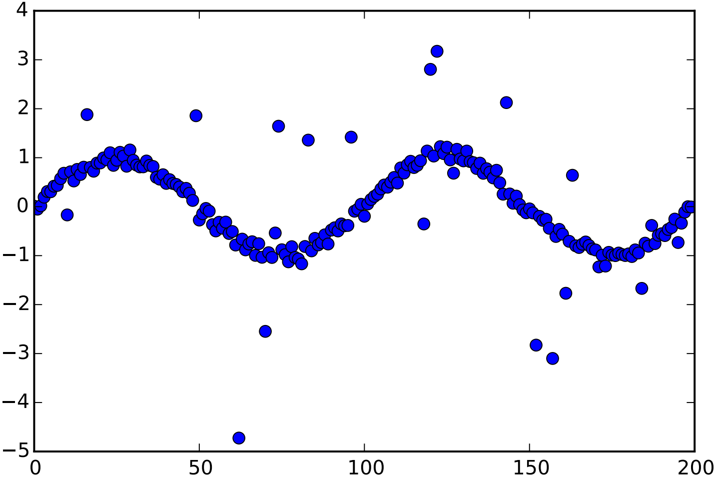
Huber example

- same curve fitting example as before, except data contains some extreme outliers
- penalize closeness to data with Huber function h_M to reduce influence of outliers in fit
- solve

$$\text{minimize} \quad \sum_{i=1}^{n} h_M(x_i - y_i) + \rho \|Dx\|_2^2$$

- M parameter controls width of quadratic region, or "non-outlier" errors

Huber data

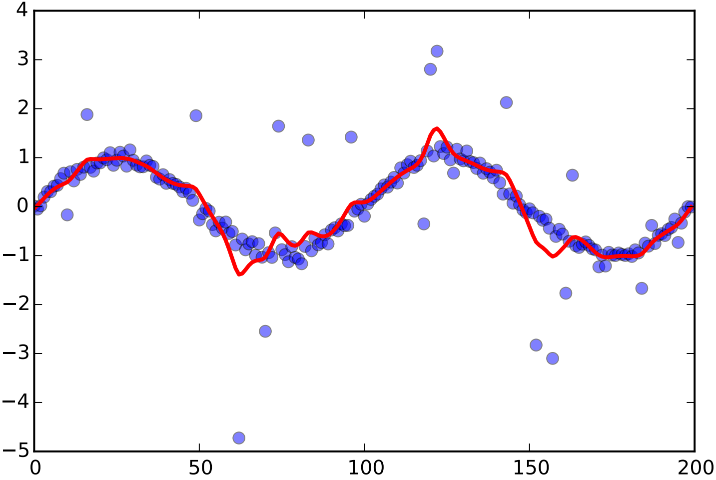


Linear smoothing

```
# get second-order difference matrix
D = diff(n, 2)
rho = 20

x = Variable(n)
obj = sum_squares(x-y) + rho*sum_squares(D*x)
Problem(Minimize(obj)).solve()
x = np.array(x.value).flatten()
```

Linear smoothing result



Huber smoothing

```
# get second-order difference matrix
D = diff(n, 2)
rho = 20
M = .15 # huber radius

x = Variable(n)
obj = sum_entries(huber(x-y, M)) + rho*sum_squares(D*x)
Problem(Minimize(obj)).solve()
x = np.array(x.value).flatten()
```

Huber smoothing result