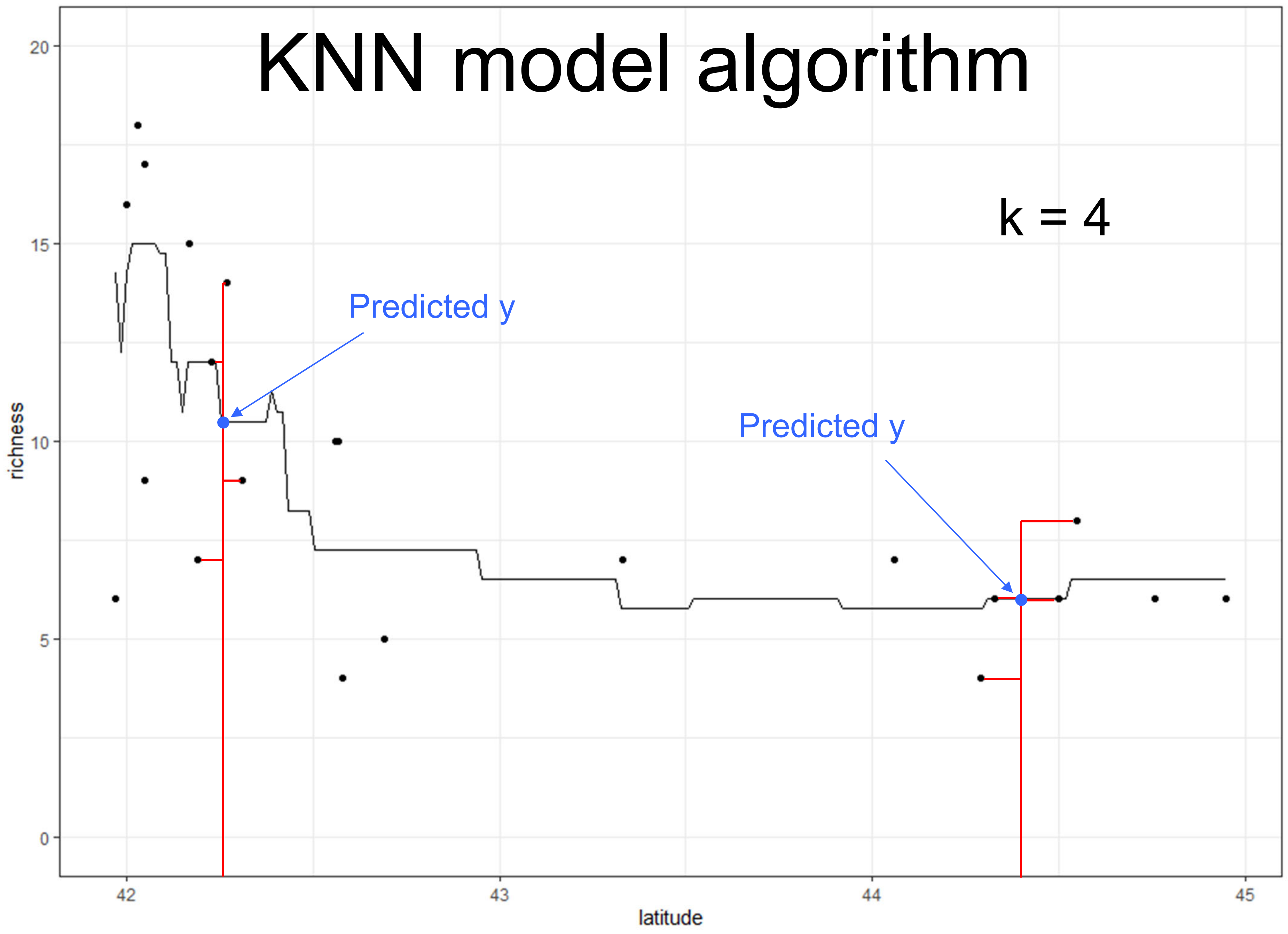
Today

- So far: focused on inference algorithm; 2 model algorithms (smoothing spline, KNN)
- Brief: smoothing spline model algorithm
- Basic full setup for machine learning
  - model + training + inference algorithms
  - illustrated with ants data, polynomial model
- Training algorithm: optimization review

KNN model algorithm

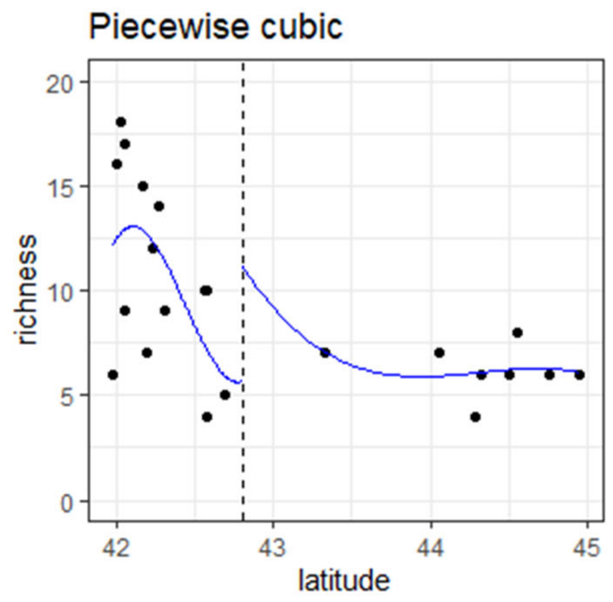


Smoothing spline model algo

- James et al. Ch 7.1 - 7.5
- Cubic smoothing spline
  - piecewise cubic polynomial
- Knots (joins) at data
- Constraints: continuous first and second derivates at knots

Building a smoothing spline

Ants data

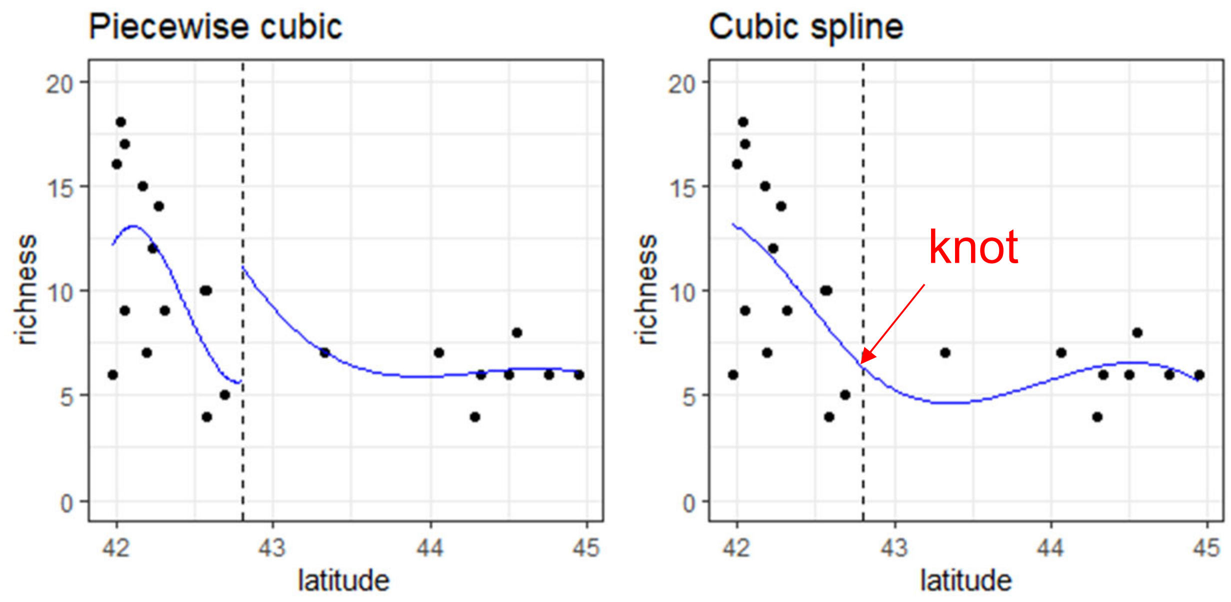


Cubic polynomials:

$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

Building a smoothing spline

Ants data

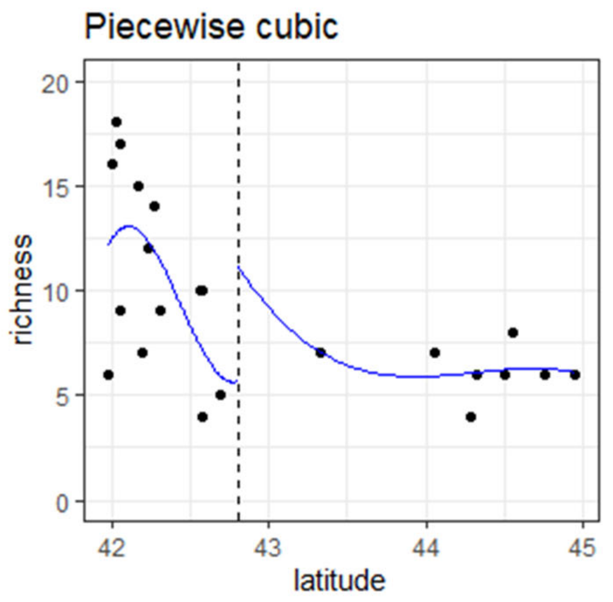


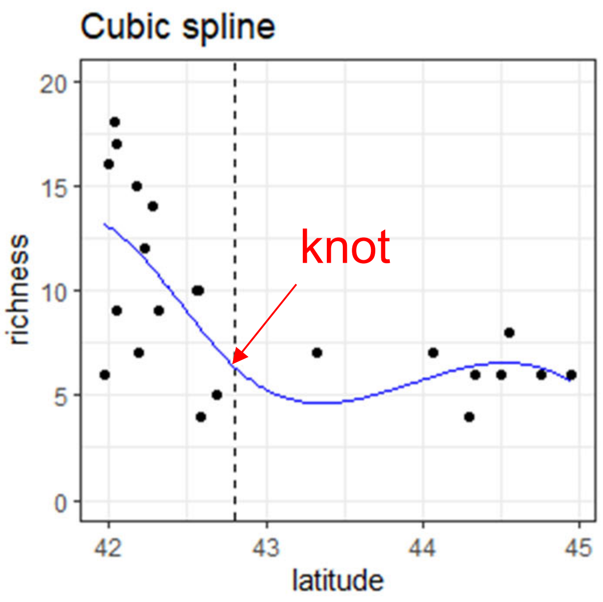
Cubic polynomials:
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

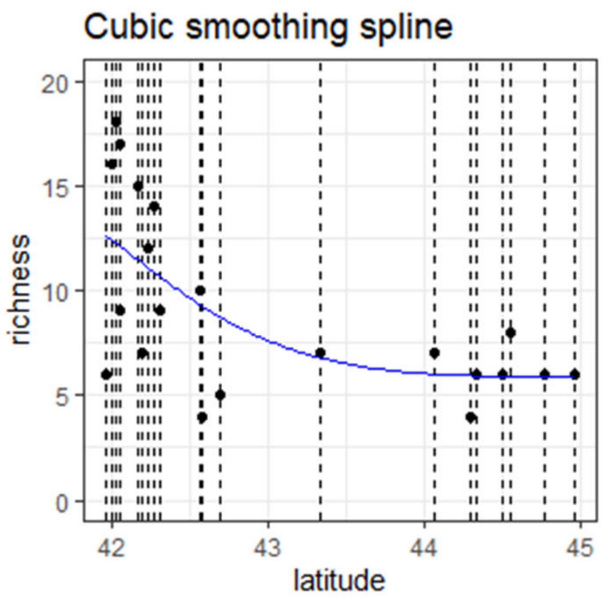
+ continuous 1st & 2nd derivatives at knot

Building a smoothing spline

Ants data







Cubic polynomials:
$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

+ continuous 1st & 2nd derivatives at knot

+ knots at each datum

Basic full ML setup

Overall algorithm:

1. Create a model algorithm for $\hat{f}(x)$
2. Use a training algorithm to find parameter values of $\hat{f}(x)$
3. Use an inference algorithm to compare predictive performance among models (model families, tuning parameters, x sets, etc).

Basic full ML setup

- Polynomial example, 3 algorithms:
  - model: flexible function $\hat{f}(x)$; polynomial linear model

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_m x^m$$

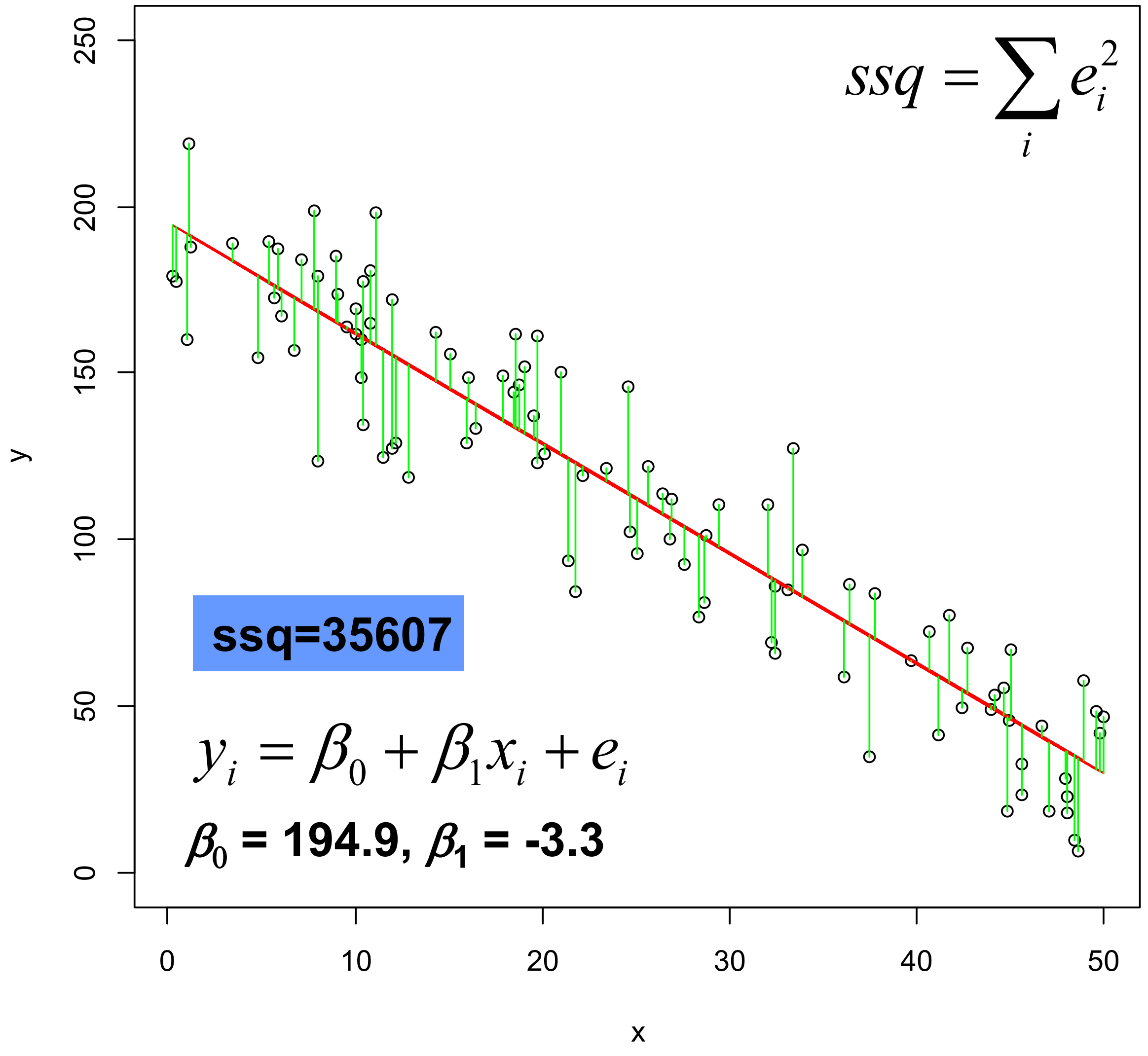
m=order

Basic full ML setup

- Polynomial example, 3 algorithms:
  - model: flexible function $\hat{f}(x)$; polynomial linear model

    $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ldots + \beta_m x^m$ m=order
  - training: optimize least squares objective function
  - minimize $SSQ = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$ for training data

Least squares optimization



General algorithmic idea:

Vary model parameters until we find the parameter values that minimize the distance of the model from the data

Optimization algorithms

Strategies

1. Systematically try all combinations of parameters - Grid search algorithms
2. Narrowing in: keep changing parameters in the direction that leads to lower SSQ - Descent algorithms
3. Try random values for parameter combinations - Monte Carlo algorithms
4. Solve for parameters using math - Analytical or numerical algorithms

Linear regression in R uses strategy 4

lm(y ~ x) solves a system of linear equations using linear algebra

Mathematical theory shows what to do (QR decomposition)

Numerical algorithm is needed to do it (householder algorithm)

Fast, guaranteed to find the minimum SSQ. Only works for SSQ: limited to ordinary linear regression.

Basic full ML setup

- Polynomial example, 3 algorithms:
  - model: flexible function $\hat{f}(x)$; polynomial linear model

    $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \ \ldots + \beta_m x^m$ m=order
  - training: optimize least squares objective function
  - minimize $SSQ = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ for training data
  - inference: tuning parameter (order of poly); k-fold cross validation

Tuning parameters

- Examples so far
  - order of polynomial
  - df of smoothing spline
  - k of KNN
- Different values of tuning parameters give different models
- Use CV inference algorithm to choose model with best predictive performance

KNN training algorithm

- There is no training algorithm!
- Automatic
- No parameters

Smoothing spline training algo

Penalized least squares

Optimize this objective function:

$$\underbrace{\sum_{i=1}^{n}\left(y_i - f(x_i)\right)^2}_{\text{SSQ}} + \underbrace{\lambda \int f''(t)^2 dt}_{\text{Penalty}}$$

Penalty term = “wiggliness”

λ is held constant to optimize

Tuning parameter d.f. is a function of λ and we later vary d.f. in an inference algorithm to find the d.f. (hence λ) with the best predictive performance.

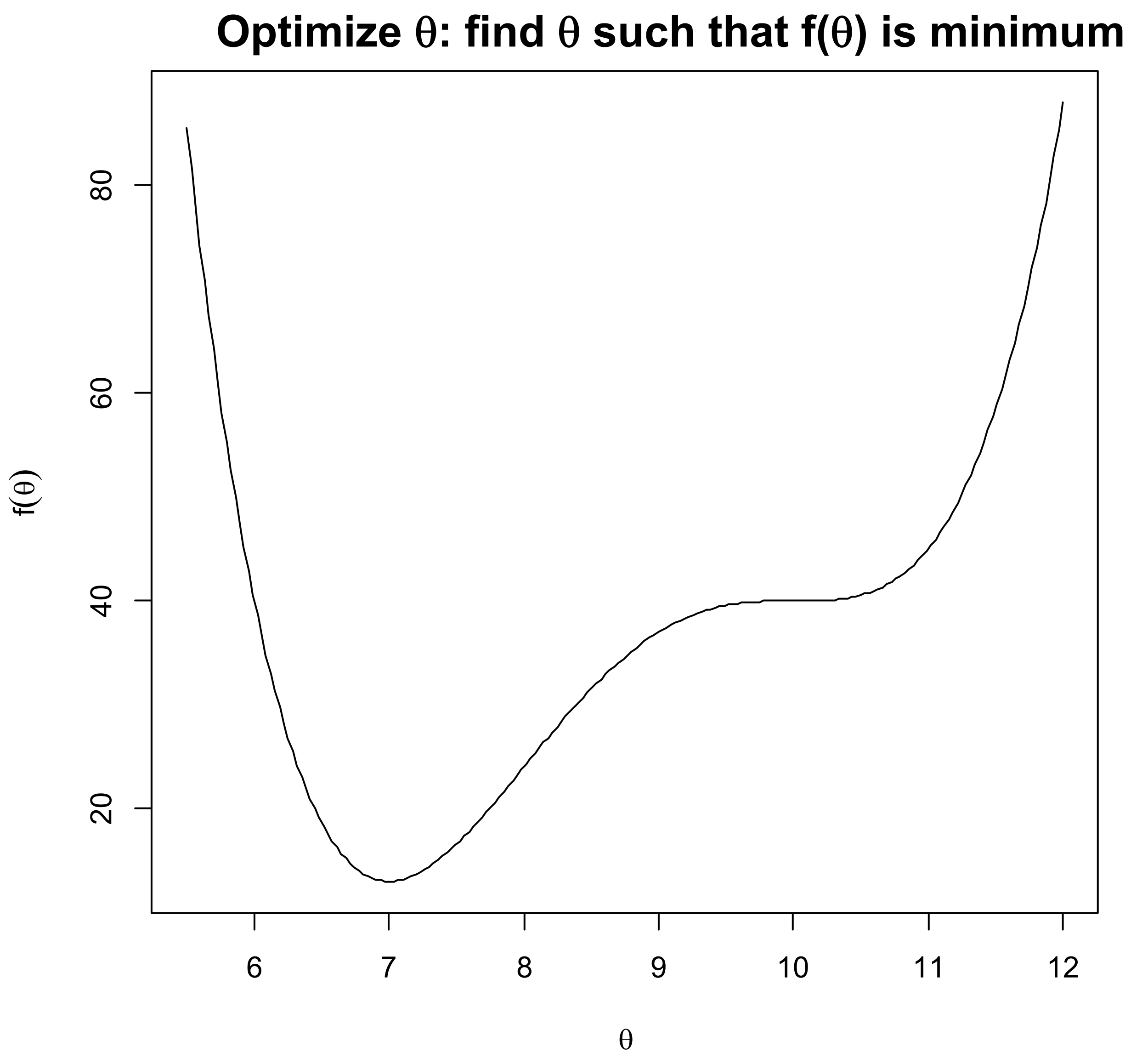
Smoothing spline training algo

Optimizing algorithm
Strategy 2: descent algorithm

Diving deep into the source code for smooth.spline() we come to the file sbart.c where we find the exact algorithm is due to Forsythe, Malcom & Moler, presumably circa 1977, in turn a slightly modified version of the Algol 60 (!!) procedure localmin from Brent (1973) *Algorithms for minimization without derivatives*, Prentice-Hall. Such legacy procedures and code are the case for much numerical work!

The algorithm uses a combination of golden section search and successive parabolic interpolation. We don’t need to worry about these details but the following slides give a sense for descent algorithms in general using the simple bisection algorithm as an example.

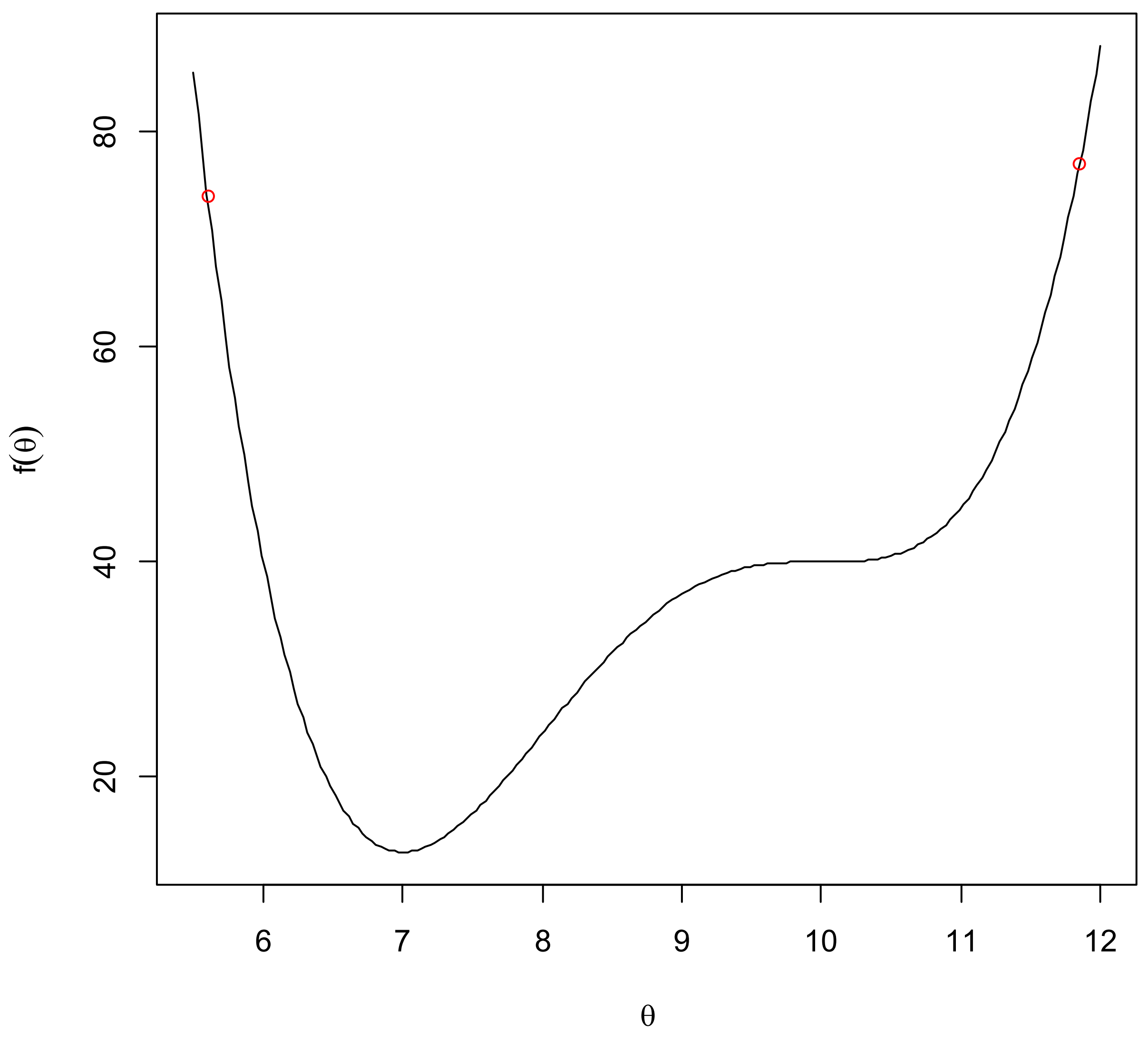
A feel for descent algorithms



Bisection algorithm

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum

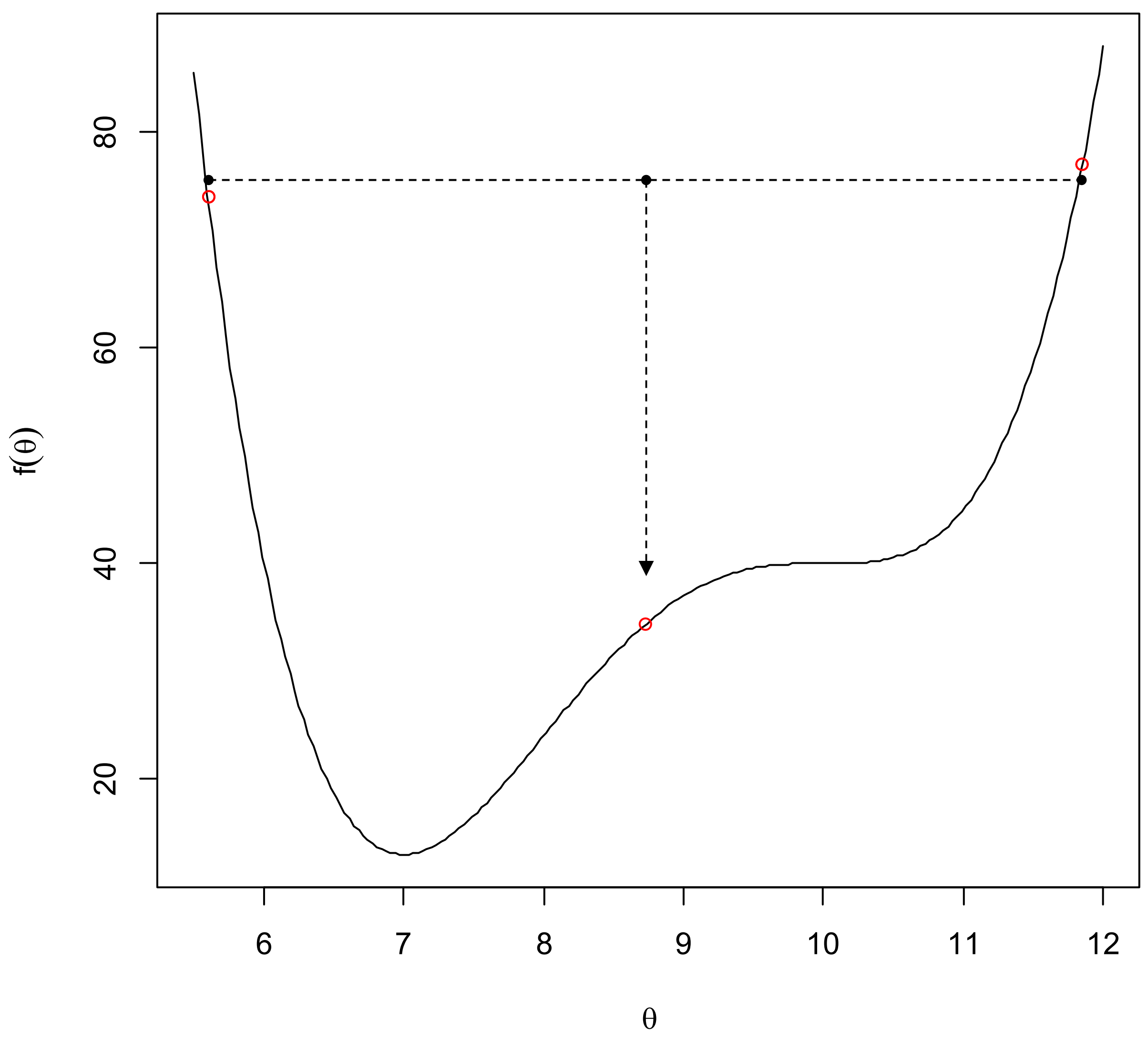


Bisection algorithm

Start with 2 points

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum



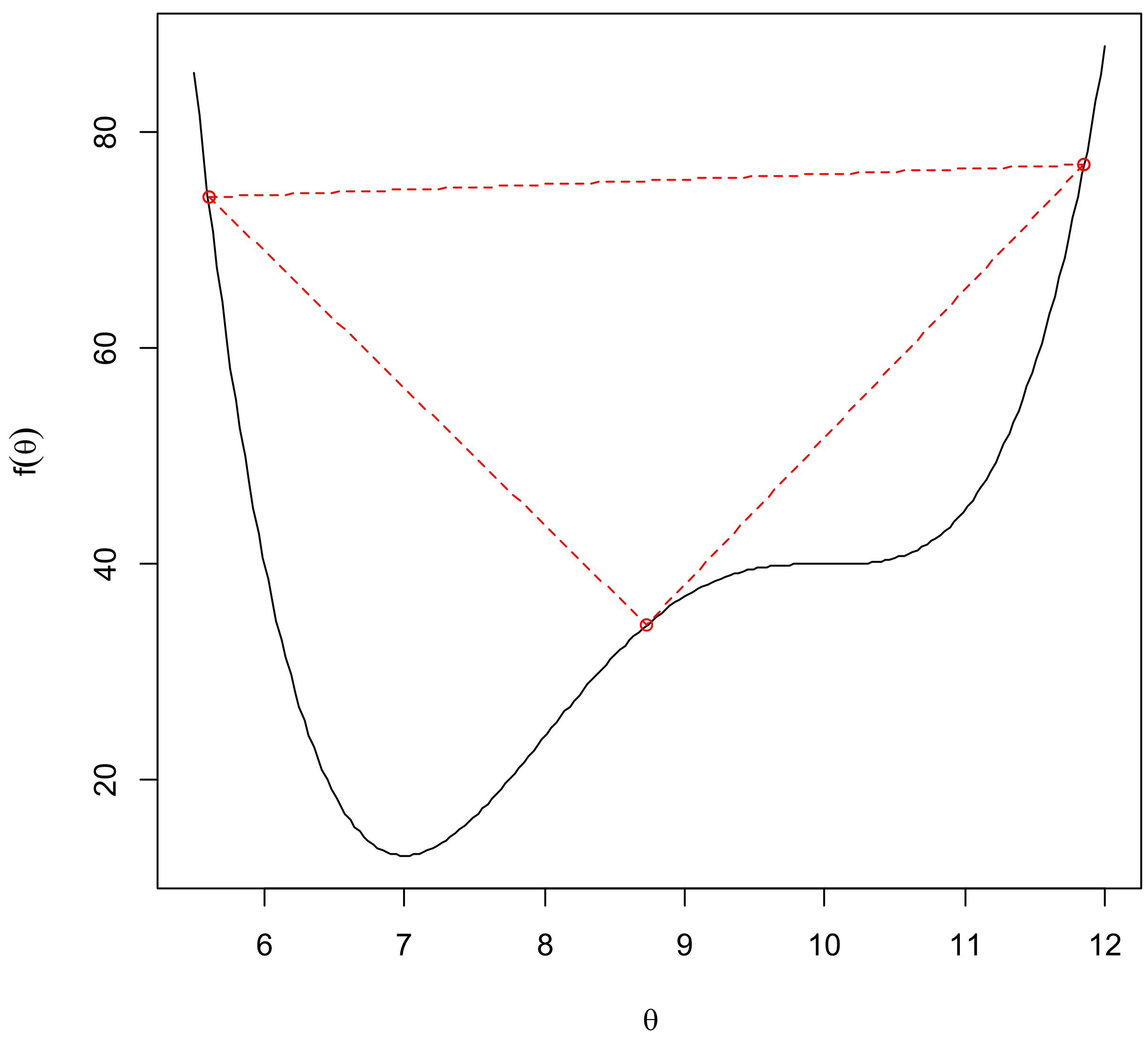
Bisection algorithm

Start with 2 points

Bisect

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum



Bisection algorithm

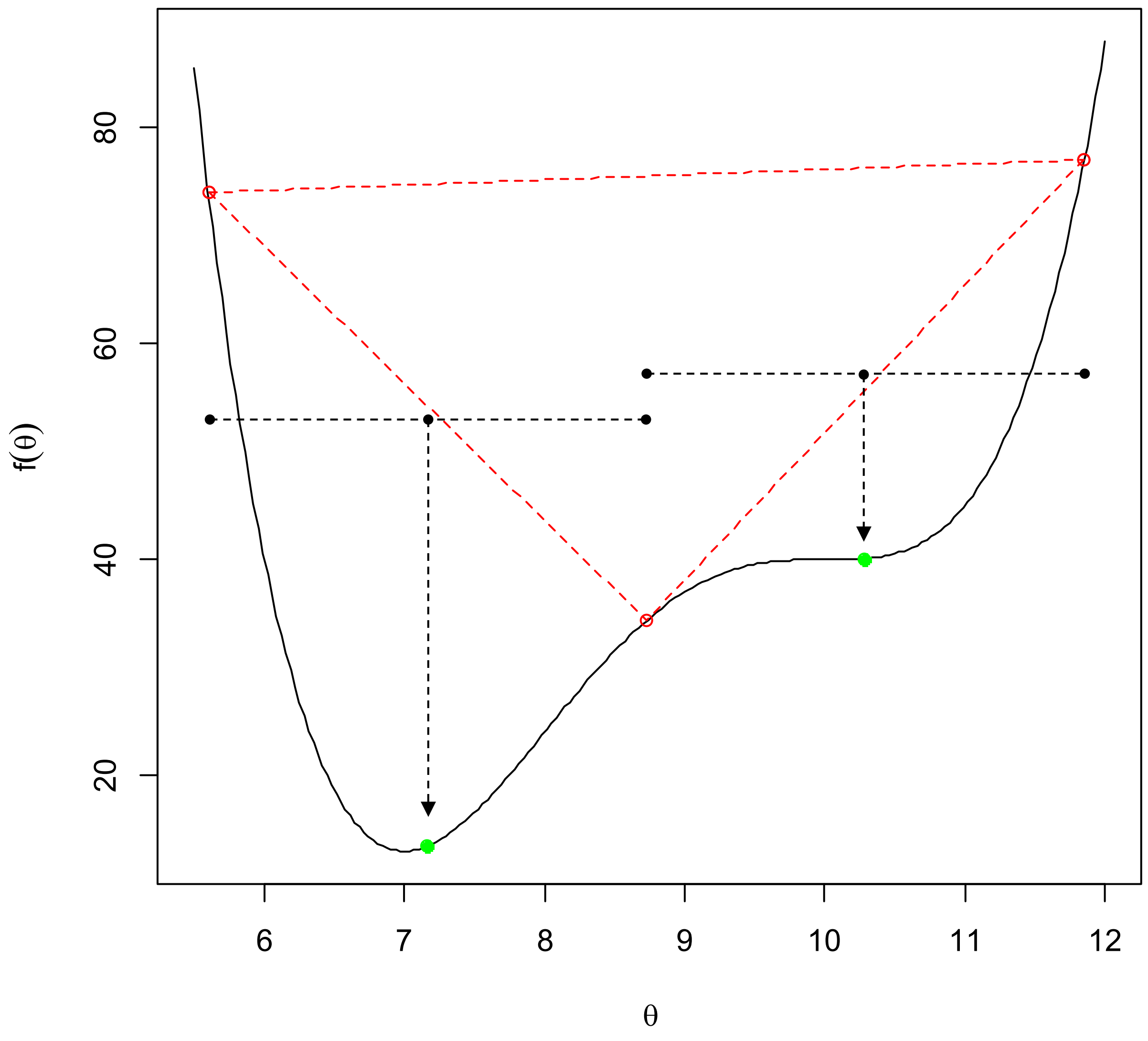
Start with 2 points

Bisect

Make triangle

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum



Bisection algorithm

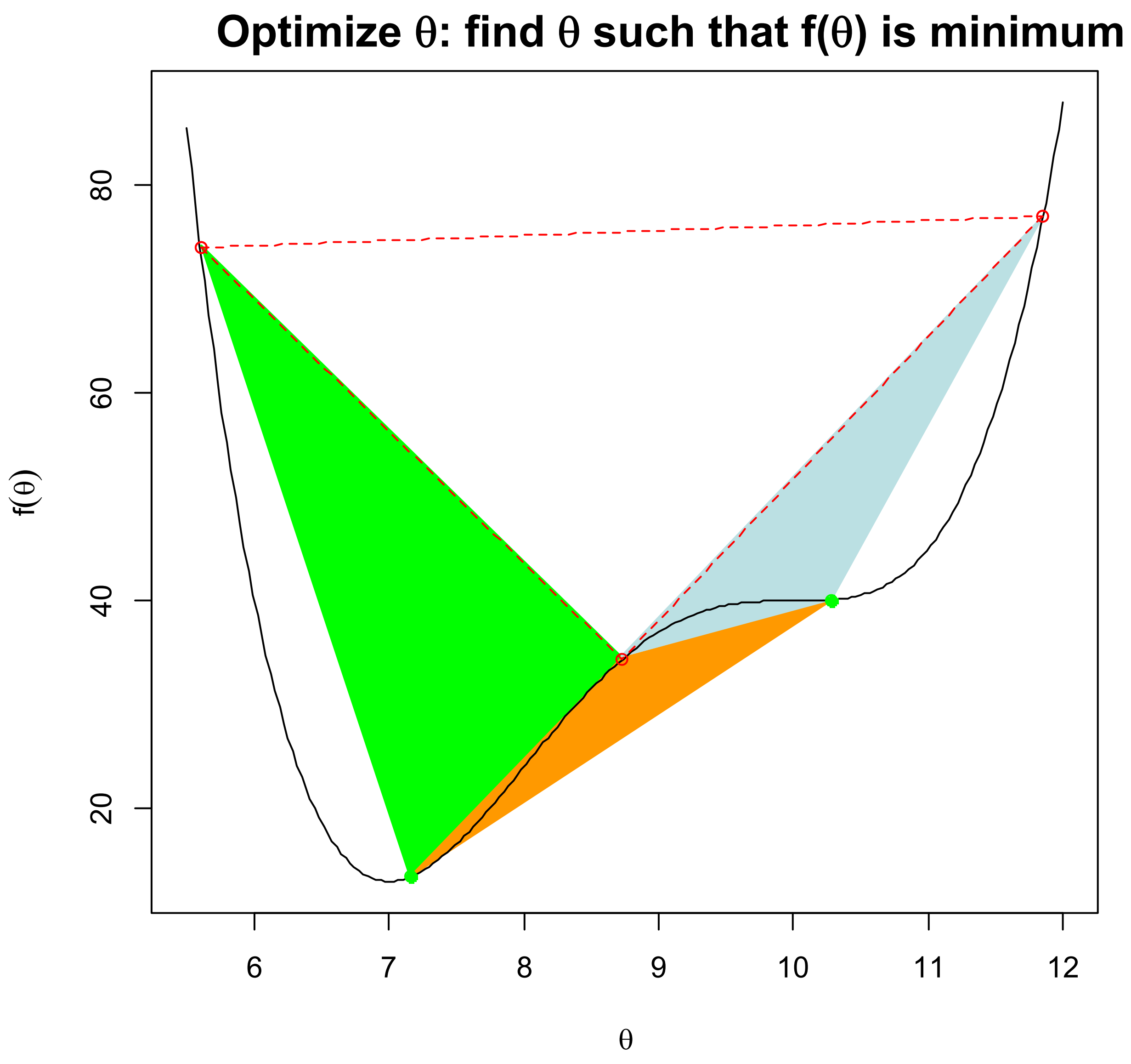
Start with 2 points

Bisect

Make triangle

Bisect lower sides

A feel for descent algorithms



Bisection algorithm

Start with 2 points

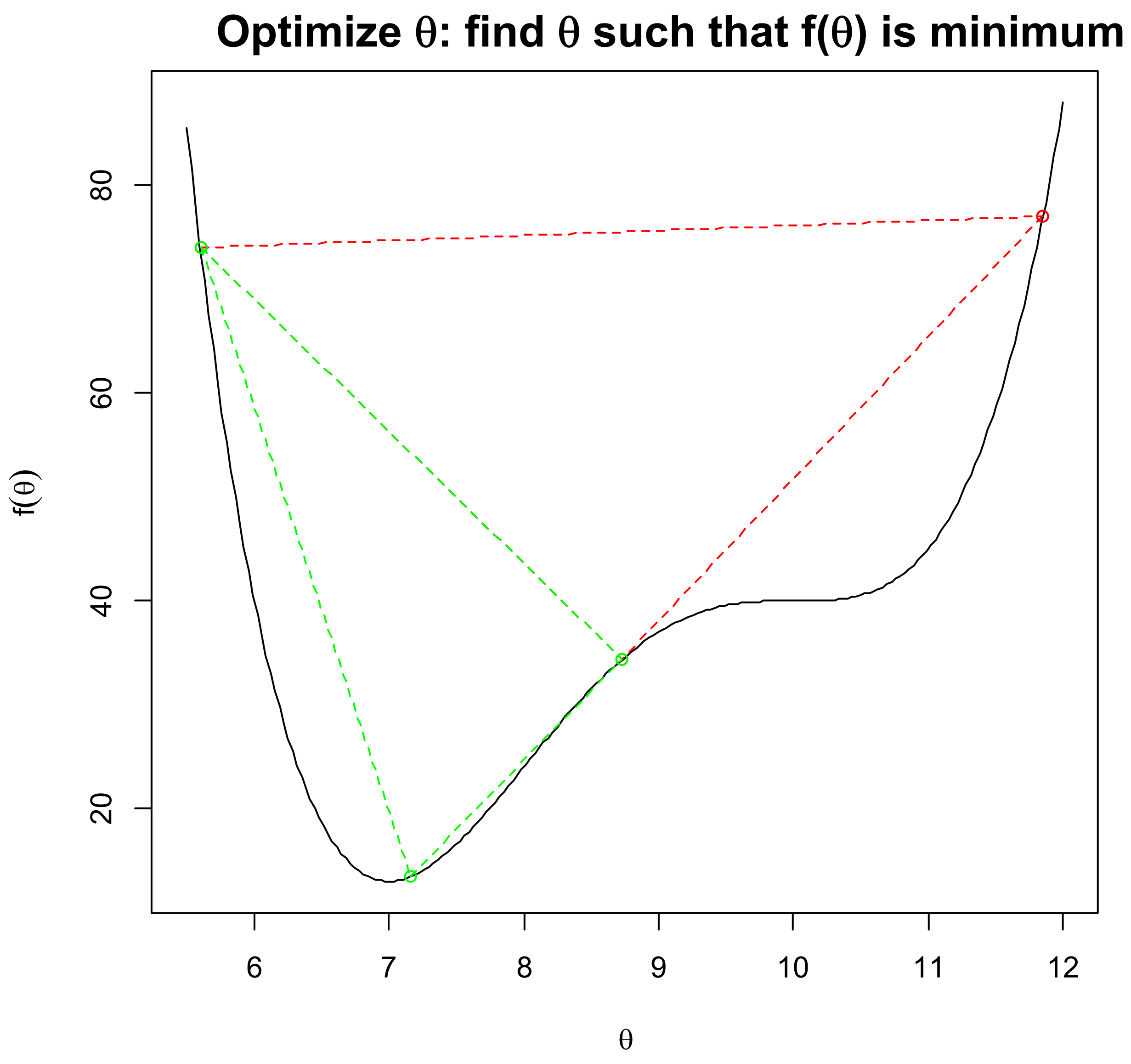
Bisect

Make triangle

Bisect lower sides

Make lowest triangle

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

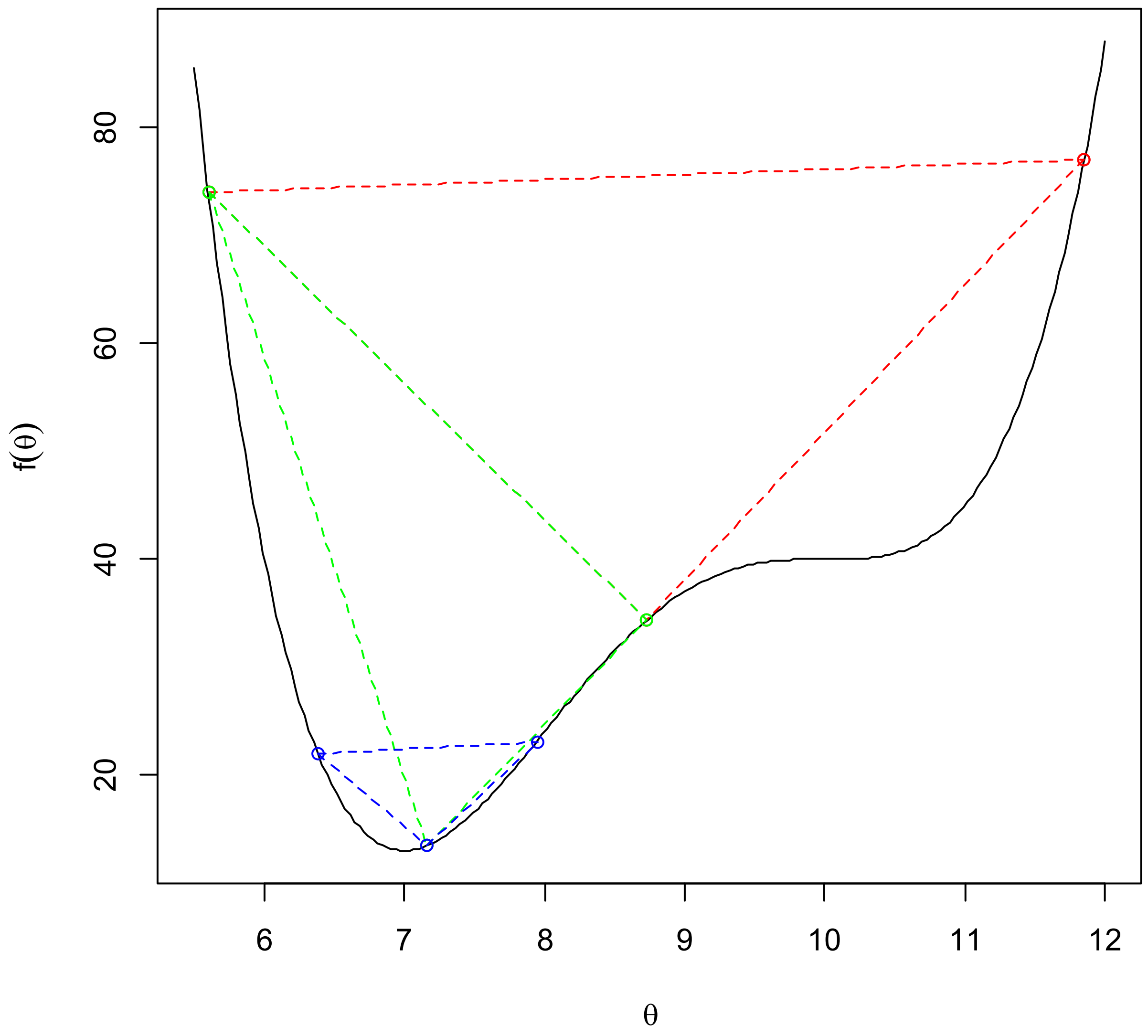
Make triangle

Bisect lower sides

Make lowest triangle

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum



Bisection algorithm

Start with 2 points

Bisect

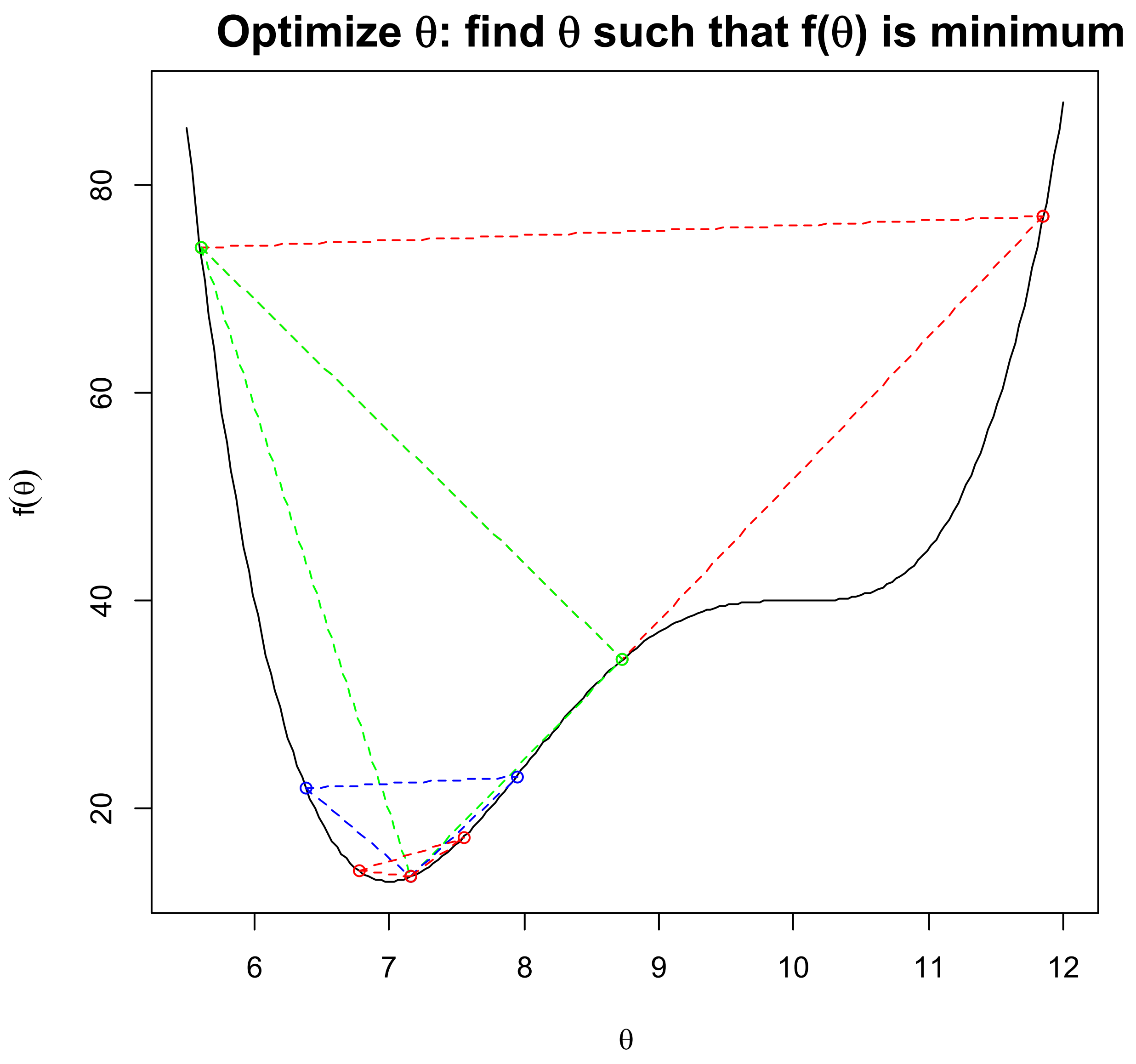
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

Make triangle

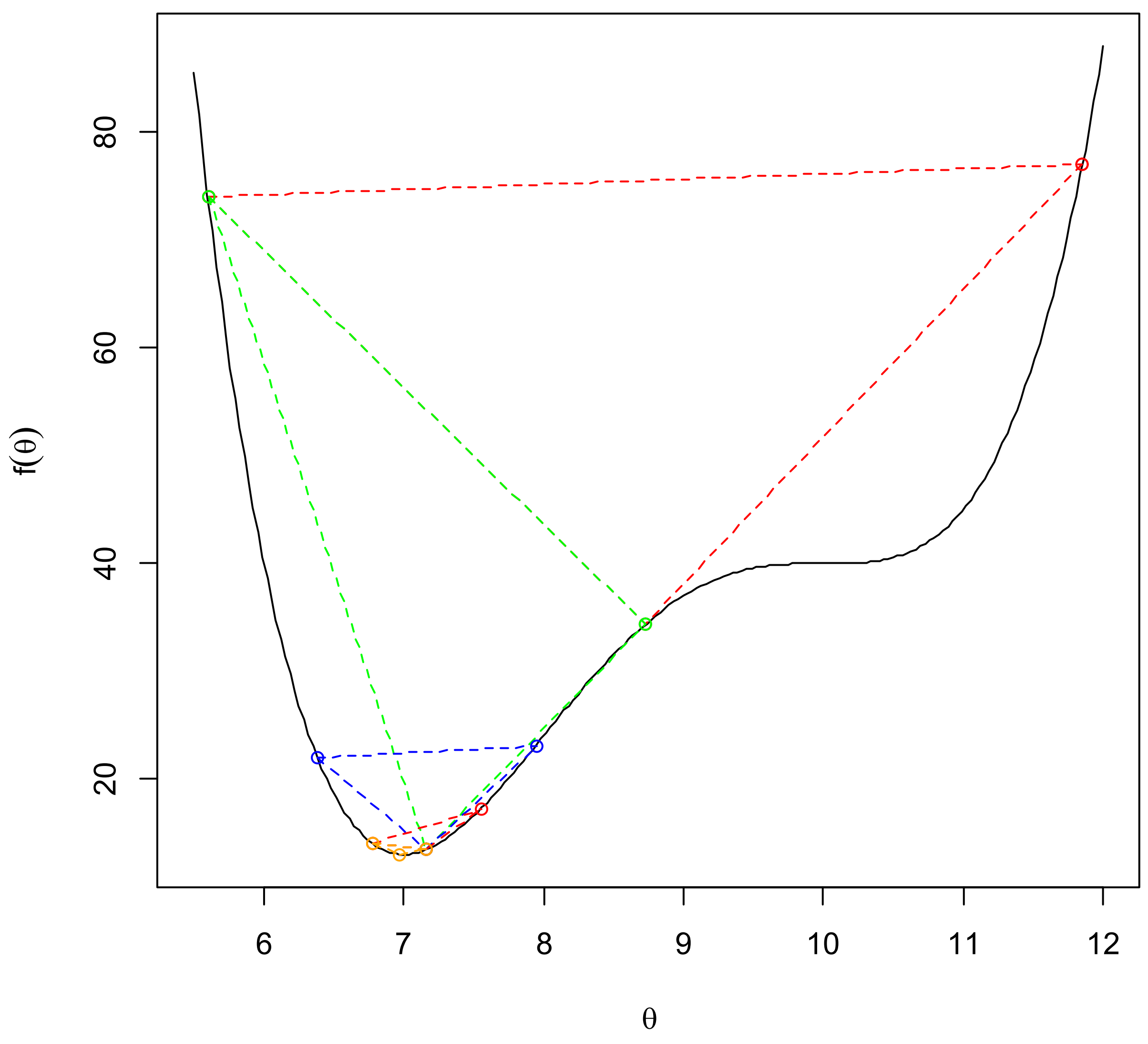
Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms

Optimize θ: find θ such that f(θ) is minimum



Bisection algorithm

Start with 2 points

Bisect

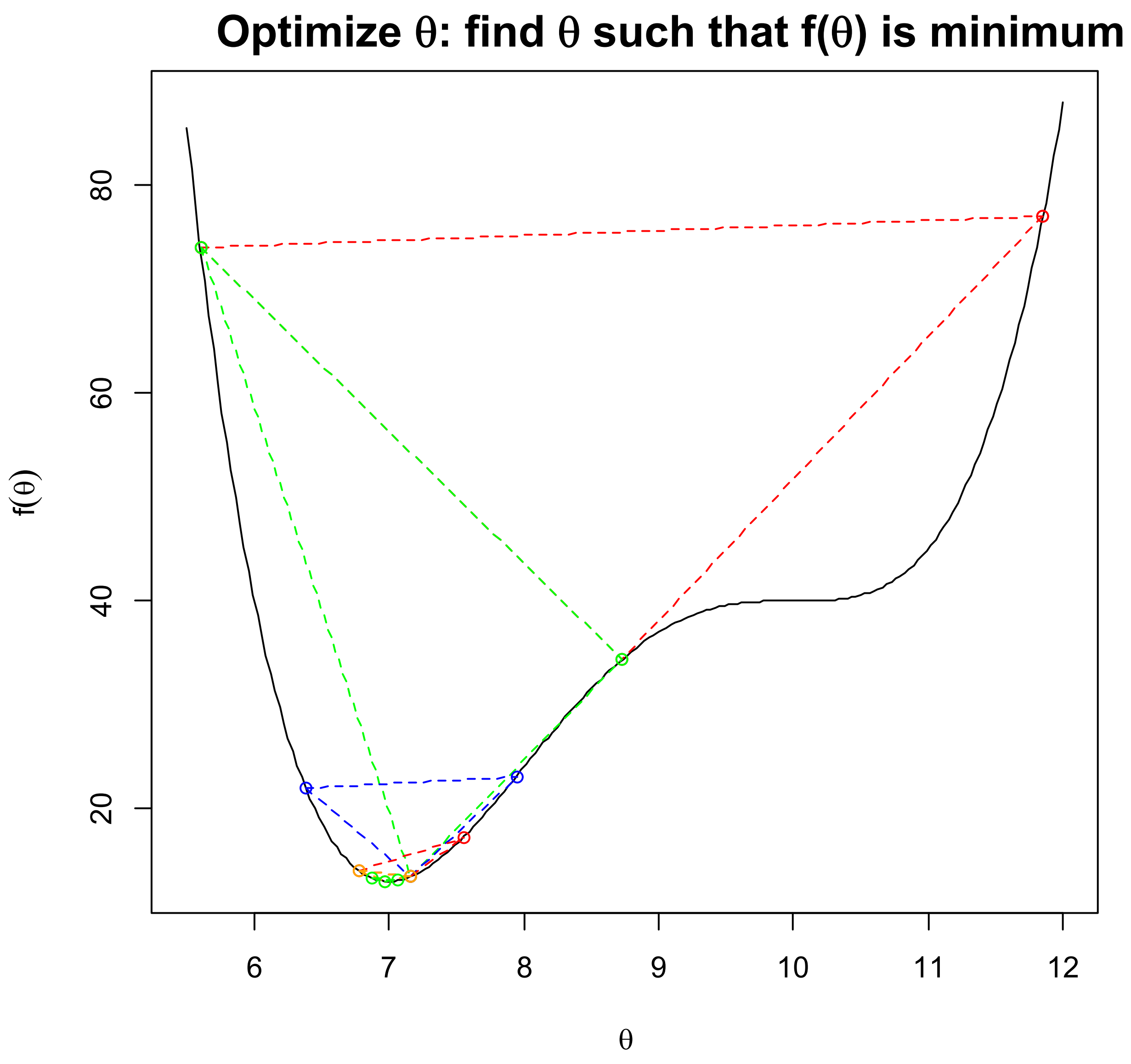
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

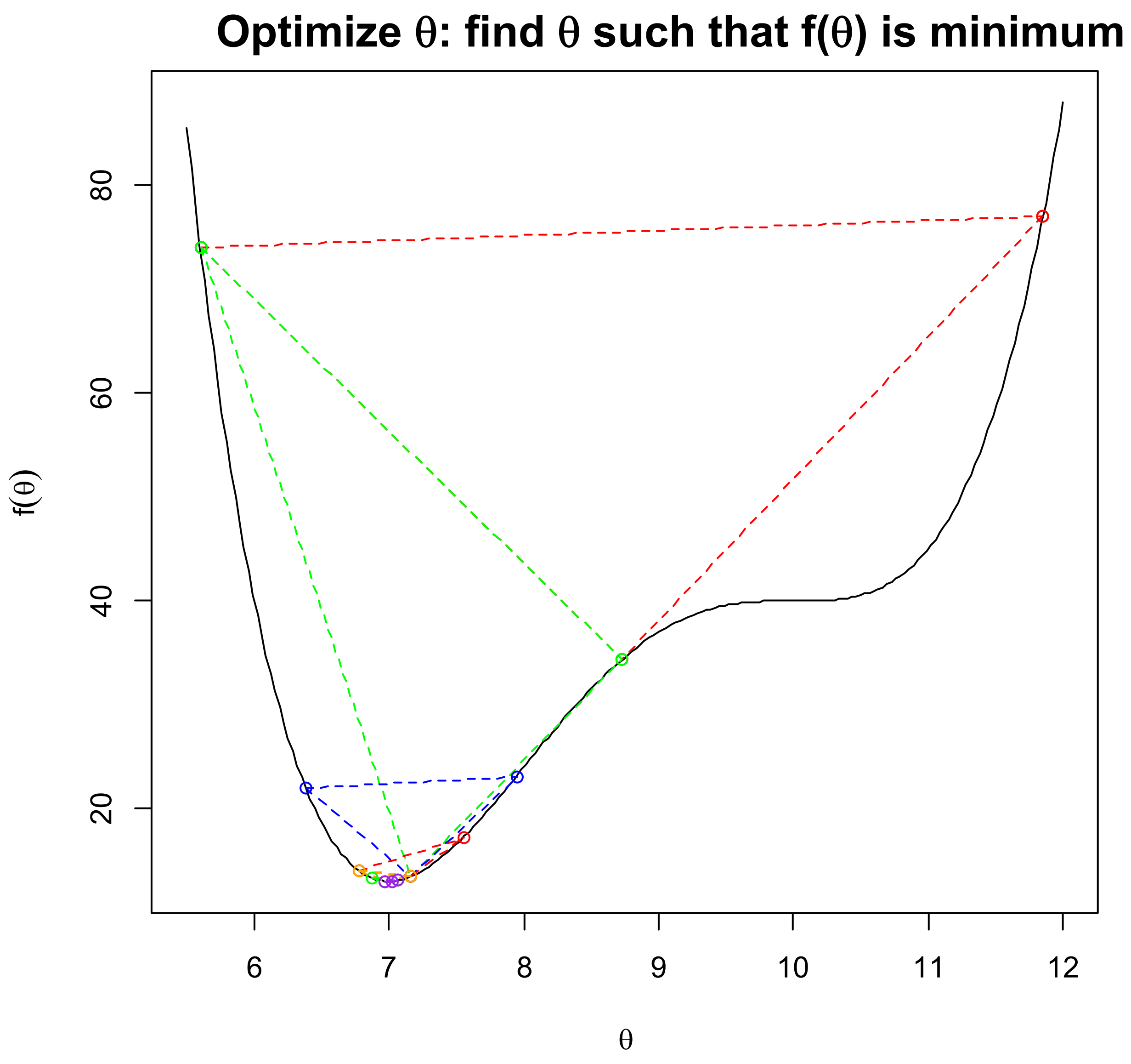
Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

A feel for descent algorithms



Bisection algorithm

Start with 2 points

Bisect

Make triangle

Bisect lower sides

Make lowest triangle

Keep repeating

Roll your own

Almost always we will be using whatever optimization algorithm is implemented within the R function that trains the model, e.g. within lm() or smooth.spline(). Usually the function authors will have made an excellent choice for that particular model!

If you need to roll your own, an excellent starting place is the versatile Nelder-Mead simplex algorithm, a descent algorithm, and the default in the R function optim().