Fitting The Unknown

Joshua Lande

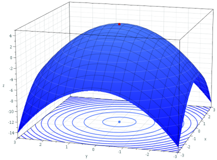
Stanford

September 1, 2010

Motivation: Why Maximize

It is frequently important in physics to find the maximum (or minimum) of a function

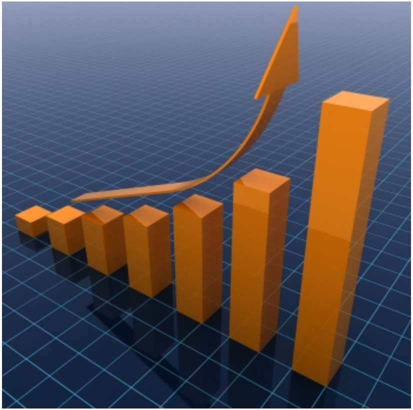
- Nature will maximize entropy
- Economists Maximize (Minimize?) the Cost Function
- In classical mechanics, minimizes the action
- Build experiments to maximize performance
- Model parameter estimation.



Parameter Estimation

- Common when analyzing data to fit a model to data
  - $\chi^2 = \sum (y_i - y(x_i))/\sigma_i)^2$
  - logLikelihood = log(Prob(data|model))
- Model is generally a function of free parameters
- Interesting to find parameters that maximize the likelihood.

Plan



- Typically, physicists pull out an off the shelf optimizer to fit their function and be done with it
- Today, lets dig under the hood and figure out how they work

Ad Hoc Methods

- Given an arbitrary function $F(\vec{x})$ of n variables $\vec{x}$,
  - how would you go about minimizing it?
- Grid Search
  - Divide space into an n dimensional grid
  - evaluate the function along the grid
  - avoids local minimum
  - Useful to seed other algorithms
- Bisection Algorithm
- Random points method
- These are slow/inefficient - $O(2^n)$

Alternating Variables

- Maximize one parameter at a time
- Ignore correlation between variables
- Algorithm is inefficient and unreliable
- Can cause oscillatory behavior

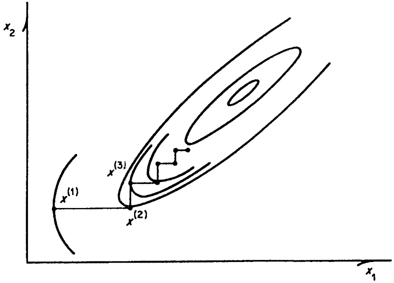
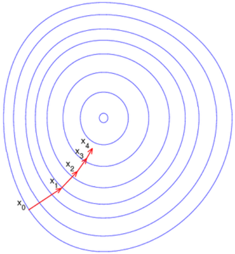


Figure 2.2.3 The method of alternating variables

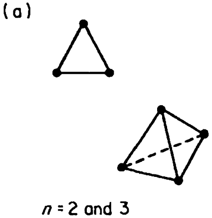
Gradient Descent

- Function decreases in the direction of the negative gradient
- The negative of the gradient should lead to the minimum
- $\vec{x}_{i+1} = \vec{x}_i - \gamma \vec{\nabla} F(\vec{x})$
- Iterate until $|\vec{\nabla} F(\vec{x}_i)| < \epsilon$
- Well suited when $\vec{\nabla} F$ is easily/analytically calculated
- Often, perform a grid search in the direction of $-\vec{\nabla} F$ before next iteration



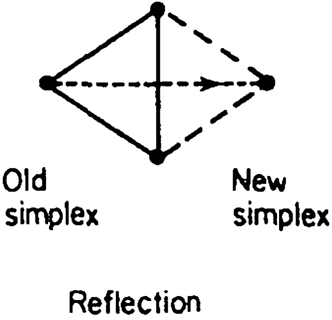
Simplex Fitting Algorithm (What's a Simplex???)

- A simplex is a generalization of a triangle or tetrahedron to arbitrary dimension
- An n-simplex has $n+1$ vertices in n dimensions
  - all equidistant
- For example,
  - a 2-simplex is a triangle
  - a 3-simplex is a tetrahedron
  - a 4-simplex is a pentachoron
  - a 5-simplex is a hexateron
  - a 6-simplex is a heptapeton



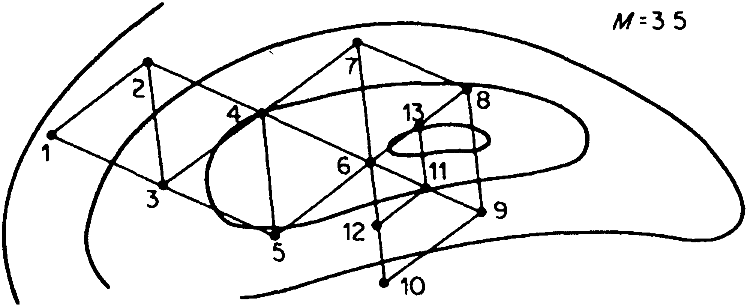
Simplex (continued)

- Define a simplex in the n dimensional fit space
- Evaluate the function at all points
- Reflect the highest point through the centroid of the other points



- If the reflected point is still the highest, reflect the second highest point
- When a certain vertex has remained in the current simplex for many iterations, contract all other vertices towards it by 1/2

Simplex Example



Simplex (continued)

- Pros:
  - Ignores the gradient/curvature of the function
  - Works well for noisy data,
  - Good for functions with local minimum
  - Works well when curvature varies rapidly
- Cons:
  - Requires an initial simplex choice
  - Slow convergence for smooth functions (compared to gradient descent)
  - Inflexible to changes in local function structure
    - E.G. wouldn't work well in a long valley

Nedler Mead Algorithm

- Improvement of Simplex algorithm
- “Adapts itself to the local landscape,
  - elongating down long inclined planes,
  - changing direction on encountering a valley at an angle,
  - and contracting in the neighborhood of a minimum”
- “Copies of the routine, written in Extended Mercury Autocode, are available from the authors”[1]
- Used by Minuit’s SIMPLEX algorithm and scipy’s fmin function

[1] J.A. Nedler and R. Mead ”A Simple Method for Function Minimization“

Nedler Mead Algorithm

- $\bar{P}$ is the simplex centroid. P_h is the largest F_h. P_l has the smallest F_h
  - *Reflection*: Evaluate the function $F*$ on the reflected part of the simplex $P^* = (1+\alpha)\bar{P} - \alpha P_h$
  - *Expansion*: If $F* < F_l$ (reflected point new minimum), then expland simplex futher in the direction by a ratio γ
    - $P^{**} = \gamma P^* + (1-\gamma)\bar{P}$
  - *Contraction*: If $F^* > F_i$ for $i \neq h$, then we contract by using as our new point
    - $P^{**} = \beta P_h + (1-\beta)\bar{P}$
- Replace P_h with P^{**}

Quit When. . .

- End when $\sqrt{\sum (F_i - \bar{F})^2/n} < \epsilon$
- End criteria is well suited for minimizing χ^2 or log likelihood, where curvature at minimum gives information about parameter uncertainty
- Fit error only has to be small compared to parameter uncertainty!

Newton-Raphson algorithm

- Assume your function is a parabola and calculate the extrema of the estimated parabola
- use curvature information to take a more direct route
- Taylor expand the derivative, set it to 0
- $f'(x + \Delta x) = f'(x) + \Delta x f''(x) = 0$
- $\Delta x = -f'(x)/f''(x)$
- $x_{i+1} = x_i - \gamma f'(x_i)/f''(x_i)$
- Iterate until $|f'(x_i) < \epsilon|$
- Excellent local convergence!
- Often, instead perform a grid search in direction of steepest descent

Newton Algorithm (Issues)

- May end up converging on a saddle point/local maximum
- May overshoot by quite a bit
- Formula undefined for $F'' = 0$.

Newton-Raphson in Many Dimensions

- Perform a n dimensional Taylor expansion
- $\vec{\nabla} F(\vec{x} + \Delta\vec{x}) = \vec{\nabla} F(\vec{x}) + H\Delta\vec{x} = 0$
- Where the Hessian matrix $H_{ij} = \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} F$
- The recursion condition is
  - $\vec{x}_{n+1} = \vec{x}_n - \gamma H_n^{-1} \vec{\nabla} F(\vec{x}_n)$
- Iterate until $|\vec{\nabla} F(\vec{x}_n)| < \delta$

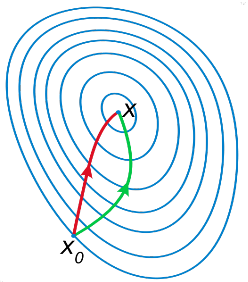


Figure: gradient descent (green) and Newton's method (red) for minimizing a function

Performance

- No reason that H_n has to be invertible
- Newton-Raphson works particularly well near the minimum
- Gradient descent (ignore curvature) works better when far from the minimum and higher order terms are more significant
- Gradient descent converges very slowly near the minimum

Levenberg-Marquardt

- Algorithm devised to naturally interpolate between Gradient and Newton-Raphson
- Replace equation to solve with $(H(\vec{x}) + \mu I)\Delta\vec{x} = -\vec{\nabla}F(\vec{x})$
- $\mu << 1$ reduces to the Newton-Raphson algorithm
- $\mu >> 1$ reduces to the Gradient algorithm with $\gamma = 1/\mu$
- Many different algorithms for adaptively changing μ based upon function

BFGS Method

- Often $H(\vec{x})$ is very costly to evaluate
- Desirable to find an intelligent approximation of the curvature
- BFGS is modification of Newton's algorithm that approximates the Hessian
- Uses Hessian at previous points and values of the derivative to estimate new one.

BFGS Method

- Same general formula as Newton's Method
  - $\vec{x}_{n+1} = \vec{x}_n - H_n^{-1}\vec{\nabla}F(\vec{x}_n)$
- Approximate the Hessian
  - $\vec{s}_{n+1} = \vec{x}_{n+1} - \vec{x}_n$
  - $\vec{y}_{n+1} = \vec{\nabla}F(\vec{x}_{n+1}) - \vec{\nabla}F(\vec{x}_n)$
- $H_{n+1} = H_n + \vec{y}_n\vec{y}_n^T/\vec{y}_n^T\vec{s}_n - H_n s_n(B_n s_n)^T/s_n^T B_n\vec{s}_n$
- Invert H_{n+1} Using the Sherman Morrison formula:

$$H_{n+1}^{-1} = H_n^{-1} + \frac{(\vec{s}_n^T\vec{y}_n + \vec{y}_n^T B_n^{-1}\vec{y}_n)(\vec{s}_n\vec{s}_n^T)}{(\vec{s}_n^T\vec{y}_n)^2} - \frac{H_n^{-1}\vec{y}_n\vec{s}_n^T + \vec{s}_n\vec{y}_n^T H_n^{-1}}{\vec{s}_k^T\vec{y}_k}$$

BFGS Method

- Advantageof BFGS:
  - the inevitability of the Hessian approximation is ensured directly
  - Well suited for problems where H is costly to compute
- Disadvantage: Convergence slower than Newton's Method[2]
- fmin_bfgs in scipy
- ROOT::Math::MinimizerOptions::SetDefaultMinimizer ("GSLMultiMin","BFGS")

[2] http://www.math.mtu.edu/~msgocken/ma5630spring2003/lectures/global2/

Physical Constrains

- Frequently, parameter values are constrained
  - E.G, experiment constrained by upper limit on cost
  - unable to observer negative counts
- A common strategy is to change to unconstrained variables
  - instead of fitting x, y on a circle, fit θ
- When a fit parameter must be positive, it is easy to instead fit the log of the parameter
  - Remember that you have to correct the fit error
  - To first order, $\sigma_{\log x} = \frac{\partial \log(x)}{\partial x}\sigma_x$
  - $\sigma_x = x\sigma_{\log x}$

Constrains

- Minuit fitter allows two sided limits of each fit parameters[3]
- It internally fits unconstrained variables but transformed them into constrained variables
- $P_{\text{int}} = \arcsin\left(2\frac{P_{\text{ext}}-a}{b-a} - 1\right)$
- $P_{\text{ext}} = a + \frac{b-a}{2}(\sin P_{\text{int}} + 1)$
- Mapping is non-linear, causes distortions in errors

[3] `http://wwwinfo.cern.ch/asdoc/minuit/minmain.html`

Penalty Functions

- Another strategy to for constrains are penalty functions
- Replace the function you are fitting with a function which increases rapidly in forbidden regions
- Want to minimize $F(\vec{x})$ such that
  - $g_i(\vec{x}) \leq 0$
  - $h_i(\vec{x}) = 0$
- g_i are inequalities (Flux > 0) and h_i are fixed constraints (cost $= 1,000$)
- Many types of penalty functions have been suggested

Static Penalty functions[4]

- Constant Penalty Functions
  - Replace function with $F_p(\vec{x}) = F(\vec{x}) + \sum C_i \delta_i$
  - where $\delta_i = \begin{cases} 1 \text{ if constrain } i \text{ is violated} \\ 0 \text{ if constrain } i \text{ is satisfied} \end{cases}$
  - No obvious way to pick the C_i
- "Cost to Completion" Penalty Function
  - Let penalty increase further farther from allowed region
  - $F_p(\vec{x}) = F(\vec{x}) + \sum C_i d_i^{\kappa}$
  - Where $d_i = \begin{cases} \delta_i g_i(\vec{x}) \\ |h_i(\vec{x})| \end{cases}$
  - Frequently κ is 1 or 2

[4] http://www.eng.auburn.edu/users/smithae/publications/bookch/chapter.pdf

Dynamic Penalty Functions

- static penalty functions lack a robust strategy for picking C_i
- Dynamic penalties use the length of time of search t
- $F_p(\vec{x}, t) = F(\vec{x}) + \sum s(t) d_i^{\kappa}$
- d_i is an increasing function of time
- Often have to tune $s_i(t)$ to particular problem
  - If $s_i(t)$ is too lenient, infeasible solution may result from fit
  - If $s_i(t)$ is too strict, search may converge to non-optimal feasible solution
- Lots of research into adaptive penalty functions...

Questions?