Optimization - 2

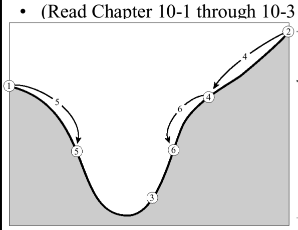
CMSC828 D

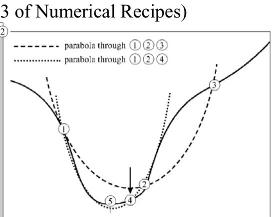
Outline

- Cost functions (last class)
- Given a cost function we can calculate
  - The global minimum
  - A local minimum
- Algorithms can be classified according to
  - Derivative information available/not available or expensive
    - Derivatives via finite-differences
  - Linear or nonlinear
  - Local minimum or global minimum
  - Differential or "statistical"
  - Constrained or Unconstrained
- Read Chapter 10-0 of Numerical Recipes.
- Focus will not be on details but educated use of these routines as black-boxes.

Bracketing methods in 1D

- Knowing the function value at 3 points bracket a minimum
- Find a better approximation to the minimum
  - Golden bisection
  - Parabola fitting
  - Methods using derivative information
- 1-D search methods important for multi-dimensional algorithms
- (Read Chapter 10-1 through 10-3 of Numerical Recipes)



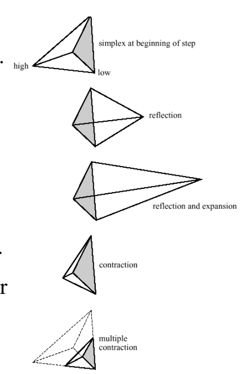


Bracketing a minimum in multiple dimensions

- Smallest region bounded by a group of points in
  - 1D is bounded by two points (a line segment)
  - 2D is bounded by three points (a triangle)
  - 3D by four points (a tetrahedron)
  - In *N*D by *N+1* points (a simplex)
- Can find a direction of a decreasing function in
  - 1D by the line from point with higher value to lower
  - 2D by joining point with highest value through point with average value on the opposite side of the triangle
  - And so on for *N*D
- However cannot guarantee a bracket of a minimum in *N*D

Downhill Simplex Method (Nelder-Mead)

- Reflection: Project along the direction of decrease with size 1.
- Reflection and expansion:If decrease is large try a step of size 2.
- Contraction: Result of reflection is bad, so try a simple reduction within simplex.
- Multiple contraction: If result of contraction does not give a better result than lowest point.
- Conclude: volume of simplex becomes below tolerance.



Basic calculus

- The direction of maximum increase of a function at a point **x** is along $\nabla f(\mathbf{x})$
- Critical points of a function f are at $df/dx=0$ or $\nabla f=0$.
  - One way of optimizing is to find **x** where $\nabla f=0$
  - However this can usually be done easily only in one dimension
- Taylor series
  - 1D $$f(x+h) = f(x) + h\left.\frac{df}{dx}\right|_x + \frac{h^2}{2}\left.\frac{d^2 f}{dx^2}\right|_x + O(h^3)$$
  - Multiple dimensions $$f(\mathbf{x}+\mathbf{h}) = f(\mathbf{x}) + h_i \frac{\partial f}{\partial x_i} + \frac{1}{2} h_i h_j \frac{\partial}{\partial x_i}\frac{\partial f}{\partial x_j} + O(|\mathbf{h}|^3)$$
  - Vector valued function $$f_j(\mathbf{x}+\mathbf{h}) = f_j(\mathbf{x}) + h_i \frac{\partial f_j}{\partial x_i} + \frac{1}{2} h_i h_k \frac{\partial}{\partial x_i}\frac{\partial f_j}{\partial x_k} + O(|\mathbf{h}|^3)$$
- Newton's method for solving $f(x)=0$.
  - Given $f(x)\neq 0$ seek a correction, h, to x, so that $f(x+h)=0$

$$f(x+h) = f(x) + hf'(x) = 0 \quad \text{so that} \quad h = -\frac{f(x)}{f'(x)}$$

Newton's Method

- If $f(\mathbf{x})$ is a scalar valued function of n variables $\mathbf{x}$

$$f(\mathbf{x}+\mathbf{h}) = f(x_i + h_i) = f(x_i) + h_i f_i(x_i) = 0$$

  - No way to get n equations from one equation above
  - Use steepest descent methods
- However in optimization problems we are usually solving for the minimum of a scalar valued function of multiple variables $f(\mathbf{x})$, where $\mathbf{x}$ is an n dimensional vector
  - We need to solve an equation of the type $\mathbf{g}(\mathbf{x}) = \nabla f = 0$
  - *Same prescription works but now* ∇g is a matrix called the Jacobian matrix

$$\mathbf{g}(\mathbf{x}+\mathbf{h}) = g_j(x_i + h_i) = g_j(x_i) + h_i \frac{\partial g_j}{\partial x_i} = 0$$

  - Solve the equation to get corrections and iterate
- However note that we are actually computing Hessian of f

Gradient Descent

- We have a function f and an estimate of its gradient ∇f
- Decrease f by a quantity along the direction of ∇f
  - **Begin initialize x,** tol, *k=0*

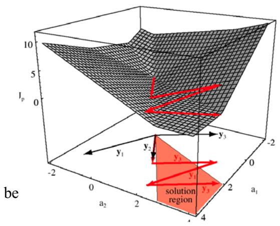
    do *k<-k+1*

    x **x-h**$_k$ ∇f

    until h$_k \nabla f$<tol`

    return x

    end



- Determining **h** is not easy
  - Called "learning rate" in AI
  - Hard to determine **h**
    - If **h** is too small algorithm will be procedure will diverge
    - Can select it using a line search or using a Newton method.

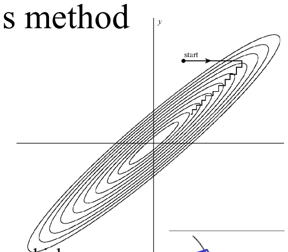
Selecting step size in Gradient Descent

- Recall $f(\mathbf{x}+\mathbf{h}) = f(x_i + h_i) = f(x_i) + h_i f_i(x_i) = 0$
- We cannot get h_i in general
- However we can minimize along a direction
  - Restrict to the direction of ∇f. Let $\mathbf{u}$ be a vector in this direction
  - Minimize the one dimensional function of t, $f(\mathbf{x}+t\mathbf{u})$ by using the one dimensional minimization techniques discussed earlier.
  - Recompute gradient at the new point and repeat the search in the new direction
  - Once t values become small we have converged
  - Each of the initial searches need not be performed with precision

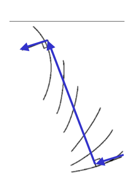
Function Evaluations

- Often evaluating the function is hard
  - Crash a car to measure a data point
- Analytical expressions for the derivatives are harder, and very much prone to programming error.
  - Analytical derivatives should always be compared with finite difference estimates for accuracy
- Often derivatives are evaluated using finite differences.
  - Recall $f' = h^{-1}(f(x+h)-f(x))$ => 2 function evaluations
  - For an n dimensional function we need at least $n+1$ function evaluations to get the derivative
  - However recall that this is the least accurate
- Promising research area*: Use chain rule and semantic parsing of functions to perform automatic differentiation*

Powell's method



- Sometimes it is not possible to estimate the derivative ∇f to obtain the direction in a steepest descent method
- First guess, minimize along one coordinate axis, then along other and so on.Repeat
- Can be very slow to converge



- Conjugate directions: Directions which are independent of each other so that minimizing along each one does not move away from the minimum in the other directions.
- Powell introduced a method to obtain conjugate directions without computing the derivative.

More complex methods

- Function can be approximated locally near a point **P** as

$$f(\mathbf{x}) = f(\mathbf{P}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \cdots$$

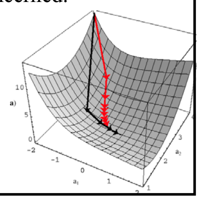
$$\approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x}$$

$$c \equiv f(\mathbf{P}) \qquad \mathbf{b} \equiv -\nabla f|_{\mathbf{P}} \qquad [\mathbf{A}]_{ij} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{P}}$$

- Gradient of above equation $\nabla f = \mathbf{A} \cdot \mathbf{x} - \mathbf{b}$
- Newton method set gradient equal zero and solve $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$.
- Conjugate directions:
  - Minimize along a direction **u**. In this case the change in ∇f as **x** changes by $\delta\mathbf{x}$ is $\mathbf{A}.\ \delta\mathbf{x}$
  - Minimization in a new direction **v** should not modify our previous minimization. Then **v** should be chosen so that **v.Au**=0
  - Any two directions that satisfy **v.Au**=0 are called conjugate directions.

Conjugate gradient and quasi-newton

- Use the fact that there is a routine available to calculate f and the Jacobian ∇f to calculate iteratively approaximations to the minimum
  - Conjugate gradients performs minimizations in conjugate directions without constructing **A**
  - Quasi Newton methods construct approximations to $\mathbf{A}^{-1}$ iteratively
- Black boxes, as far as this course is concerned.
- Generally only worth it when we are in the vicinity of a minumum.
- For nonlinear problems they often converge to a local minimum away from the true one.



Levenberg Marquardt

- Return to problem of model fitting by minimizing $\chi^2 \equiv \sum_{i=1}^{N} \left(\frac{y_i - y(x_i; a_1 \dots a_M)}{\sigma_i} \right)^2$
- As before set $\chi^2(\mathbf{a}) \approx \gamma - \mathbf{d} \cdot \mathbf{a} + \frac{1}{2} \mathbf{a} \cdot \mathbf{D} \cdot \mathbf{a}$
- Observation: steepest descent methods move faster (per function evaluation) far away from the minimum while Newton methods do well near it.
- Idea combine them so that the method adapts according to the location in parameter space.
- Usually for model fitting it is not too difficult to calculate derivatives

$$\frac{\partial \chi^2}{\partial a_k} = -2 \sum_{i=1}^{N} \frac{[y_i - y(x_i; \mathbf{a})]}{\sigma_i^2} \frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \qquad k = 1, 2, \dots, M$$

$$\frac{\partial^2 \chi^2}{\partial a_k \partial a_l} = 2 \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i; \mathbf{a})}{\partial a_k} \frac{\partial y(x_i; \mathbf{a})}{\partial a_l} - [y_i - y(x_i; \mathbf{a})] \frac{\partial^2 y(x_i; \mathbf{a})}{\partial a_l \partial a_k} \right]$$

Levenberg Marquardt

- Newton $\mathbf{a}_{\min} = \mathbf{a}_{\text{cur}} + \mathbf{D}^{-1} \cdot [-\nabla \chi^2(\mathbf{a}_{\text{cur}})]$
- Steepest Descent $\mathbf{a}_{\text{next}} = \mathbf{a}_{\text{cur}} - \text{constant} \times \nabla \chi^2(\mathbf{a}_{\text{cur}})$
- Define $\beta_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k}$ and $\alpha_{kl} \equiv \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l}$
- Then the Newton equation becomes $\sum_{l=1}^{M} \alpha_{kl} \, \delta a_l = \beta_k$
- Can combine the two equations by defining a new α matrix

$$\alpha'_{jj} \equiv \alpha_{jj}(1 + \lambda) \qquad \alpha'_{jk} \equiv \alpha_{jk} \qquad (j \neq k)$$

- Vary λ as the algorithm proceeds according to whether we are near the solution or away from it.

LM Algorithm

- Compute $\chi^2(\mathbf{a})$.
- Pick a modest value for λ, say $\lambda = 0.001$.
- (†) Solve the linear equations (15.5.14) for $\delta\mathbf{a}$ and evaluate $\chi^2(\mathbf{a} + \delta\mathbf{a})$.
- If $\chi^2(\mathbf{a} + \delta\mathbf{a}) \geq \chi^2(\mathbf{a})$, *increase* λ by a factor of 10 (or any other substantial factor) and go back to (†).
- If $\chi^2(\mathbf{a} + \delta\mathbf{a}) < \chi^2(\mathbf{a})$, *decrease* λ by a factor of 10, update the trial solution $\mathbf{a} \leftarrow \mathbf{a} + \delta\mathbf{a}$, and go back to (†).

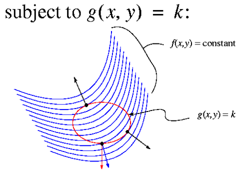
- When the algorithm has converged set $\lambda=0$ and compute the final solution

Constrained optimization

- We have to optimize *f(x)* subject to *g(x)=0*
  - Makes sense if *g(x)=0* leaves a few degrees of freedom (*N-M*)
- Approach 1 (Eliminate constraints)
  - Eliminate variables using constraint equations and solve a reduced problem $f(x^*)=0$
  - Not practical, except for simple problems
- Approach 2 (Penalty function)
  - Construct a new minimization function *f(x)+Pg(x)* where *P>>1*
  - If constraint is violated the minimization function increases rapidly, forcing the optimization routine to solutions where it is not violated
- Approach 3 (Lagrange Multipliers)
  - Solution has to lie on the surface of *g(x)=0*
  - Can't have $\nabla f=0$ anymore
  - However we require ∇f *parallel to* $\nabla g=0$

Lagrange Multipliers

Optimize $f(x, y)$ subject to $g(x, y) = k$:



Necessary conditions for a solution at $(\hat{x}, \hat{y})$:

$\nabla f(\hat{x}, \hat{y})$ is parallel to $\nabla g(\hat{x}, \hat{y})$ and $g(\hat{x}, \hat{y}) = k$

$\nabla f(\hat{x}, \hat{y}) = \lambda \nabla g(\hat{x}, \hat{y})$ and $g(\hat{x}, \hat{y}) = k$

$\nabla f(\hat{x}, \hat{y}) - \lambda \nabla g(\hat{x}, \hat{y}) = 0$ and $g(\hat{x}, \hat{y}) = k$

Linear programming

- Black box in this course
- Solve problems with systems of linear equality and inequality constraints

The subject of *linear programming*, sometimes called *linear optimization*, concerns itself with the following problem: For N independent variables $x_1, \ldots, x_N$, *maximize* the function

$$z = a_{01}x_1 + a_{02}x_2 + \cdots + a_{0N}x_N \qquad (10.8.1)$$

subject to the primary constraints

$$x_1 \geq 0, \quad x_2 \geq 0, \quad \ldots \quad x_N \geq 0 \qquad (10.8.2)$$

and simultaneously subject to $M = m_1 + m_2 + m_3$ additional constraints, m_1 of them of the form

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{iN}x_N \leq b_i \qquad (b_i \geq 0) \qquad i = 1, \ldots, m_1 \qquad (10.8.3)$$

m_2 of them of the form

$$a_{j1}x_1 + a_{j2}x_2 + \cdots + a_{jN}x_N \geq b_j \geq 0 \qquad j = m_1 + 1, \ldots, m_1 + m_2 \qquad (10.8.4)$$

and m_3 of them of the form

$$a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kN}x_N = b_k \geq 0 \qquad k = m_1 + m_2 + 1, \ldots, m_1 + m_2 + m_3 \qquad (10.8.5)$$