CogSci 109: Lecture 20

Mon, Nov. 26, 2007: Review of gradient descent, conjugate gradient

Outline for today

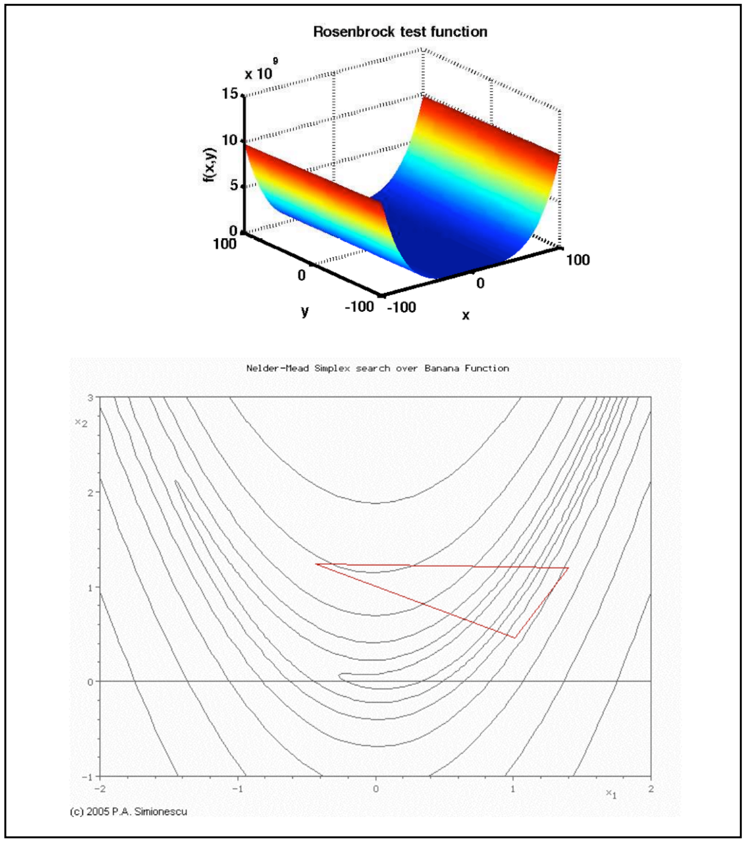
- Announcements
- Review of gradient descent
- Introduction to conjugate gradient
- Introduction to PLU's

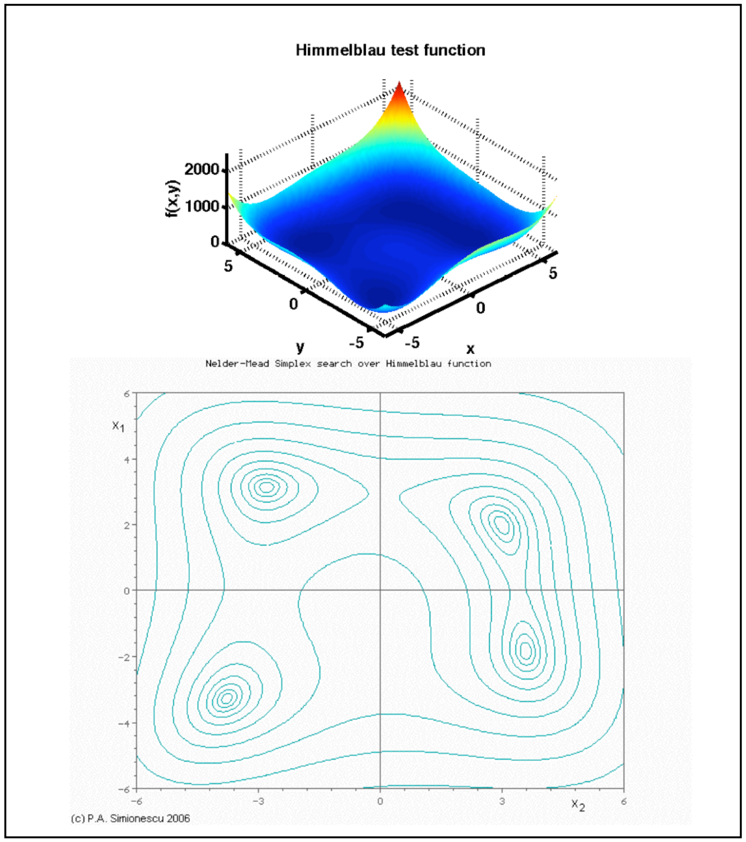
Announcements

- Homework 5 due today
- Hw 6 will be assigned this week
- Demo code
  - **gradient descent**
  - **Nonlinear function fits**
- Readings

About the Nelder-Mead Simplex algorithm

- Last time we showed examples of the NM algorithm and implementation details in matlab





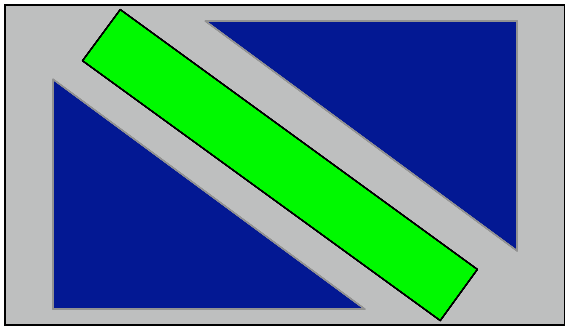
Before we go on, a few definitions

- **Positive definite matrix**
  - **All eigenvalues are positive**

$$(M_{ij} = M_{ji})$$

- **Symmetric matrix** (review) - symmetric about the diagonal

$$\begin{bmatrix} 10 & 2 & 3 & 4 \\ 2 & 4 & 9 & 7 \\ 3 & 9 & 2 & 8 \\ 4 & 7 & 8 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 10 & 2 & 3 & 4 \\ 2 & 4 & 9 & 7 \\ 3 & 9 & 2 & 8 \\ 4 & 7 & 8 & 1 \end{bmatrix}$$



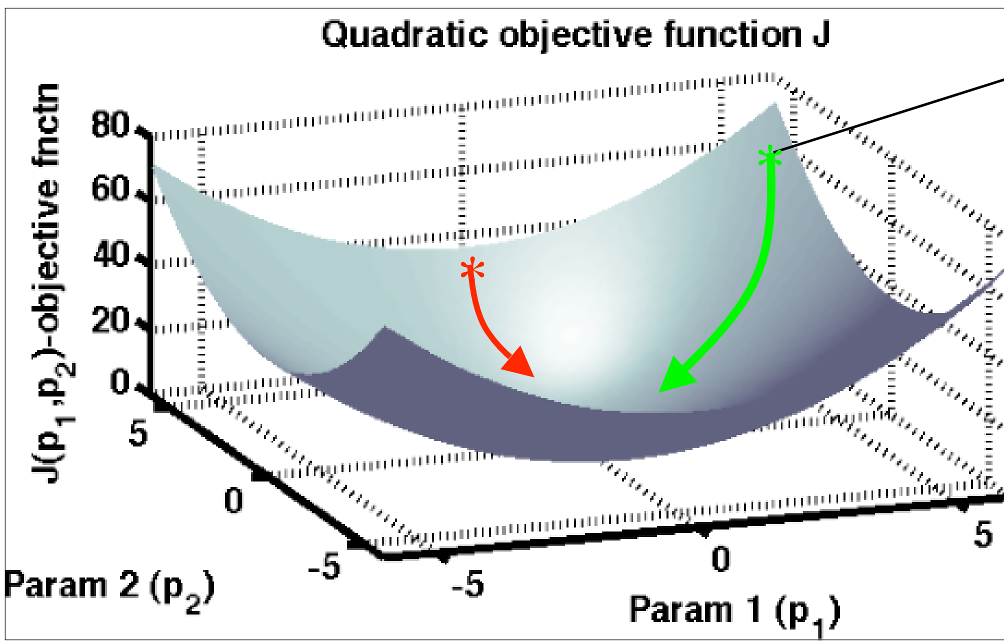
Last time we also introduced the gradient descent method

- Intuitive algorithm - go 'downhill' for the parameters in the objective function you want to minimize
- Useful for
  - **Solution of a large linear system of equations**
  - **Solution of a nonlinear systems of equations**
    - Special note - some Artificial Neural Network Algorithms use gradient descent on the weights (more on this later)
  - **Optimization and control of dynamic systems**

How does the gradient descent algorithm work?

- Consider first the objective of gradient descent
  - **You want to get to the bottom of the hill**
  - **Start somewhere, then you ski down the hill**

$$J(\mathbf{x}) = \frac{1}{2}x^T Ax - b^T x$$



Start here,

Then move towards the minimum

How do we do this mathematically?

- We want to minimize (A is assumed symmetric positive definite)

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x}$$

- We do this by starting with some initial guess for our parameters, and then 'skiing' downhill along the direction r with some 'speed' alpha at each iteration k

$$x_{k+1} = x_k + \alpha_k r_k$$

- So we'll proceed iteratively toward the minimum of *J(x)*
  - **We want to move down the opposite of the gradient of *J***

Computing the gradient of *J(x)*

- r at iteration k is given by taking the gradient of J(x) with respect to x

$$r_k = -\nabla J(\mathbf{x}_k) = -(A\mathbf{x}_k - \mathbf{b})$$

- With the gradient of J computed by

Note that
Since A is symmetric positive semi-definite

$$A\mathbf{x} = \mathbf{x}^T A$$

$$J(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - b^T\mathbf{x} \Rightarrow \nabla J(\mathbf{x}) = \frac{1}{2}A\mathbf{x} + \frac{1}{2}\mathbf{x}^T A - b$$

$$= A\mathbf{x} - b$$

So now we have the direction to move at iteration k…

Computing alpha

- We have to determine the step size (distance to go) at the iteration k
- We will compute the alpha at iteration k that minimizes

$$J(\mathbf{x}_k + \alpha_k r_k)$$

After a little work, we find alpha...

$$J(\mathbf{x}+\alpha r) = \frac{1}{2}(\mathbf{x}+\alpha\mathbf{r})^T A(\mathbf{x}+\alpha\mathbf{r}) - b^T(\mathbf{x}+\alpha\mathbf{r})$$

$$\frac{\partial J(\mathbf{x}+\alpha r)}{\partial\alpha} = \frac{1}{2}r^T A(\mathbf{x}+\alpha\mathbf{r}) + \frac{1}{2}(\mathbf{x}+\alpha\mathbf{r})^T A\mathbf{r} - b^T\mathbf{r}$$

$$= \alpha r^T Ar + r^T Ax - r^T b$$

$$= \alpha r^T Ar + r^T(Ax - b)$$

$$= \alpha r^T Ar - r^T r$$

The minimum occurs at 0

$$\frac{\partial J(\mathbf{x}+\alpha r)}{\partial\alpha} = 0$$

$$0 = \alpha r^T Ar - r^T r$$

$$\alpha r^T Ar = r^T r$$

We can divide here because these are all scalars (one number)

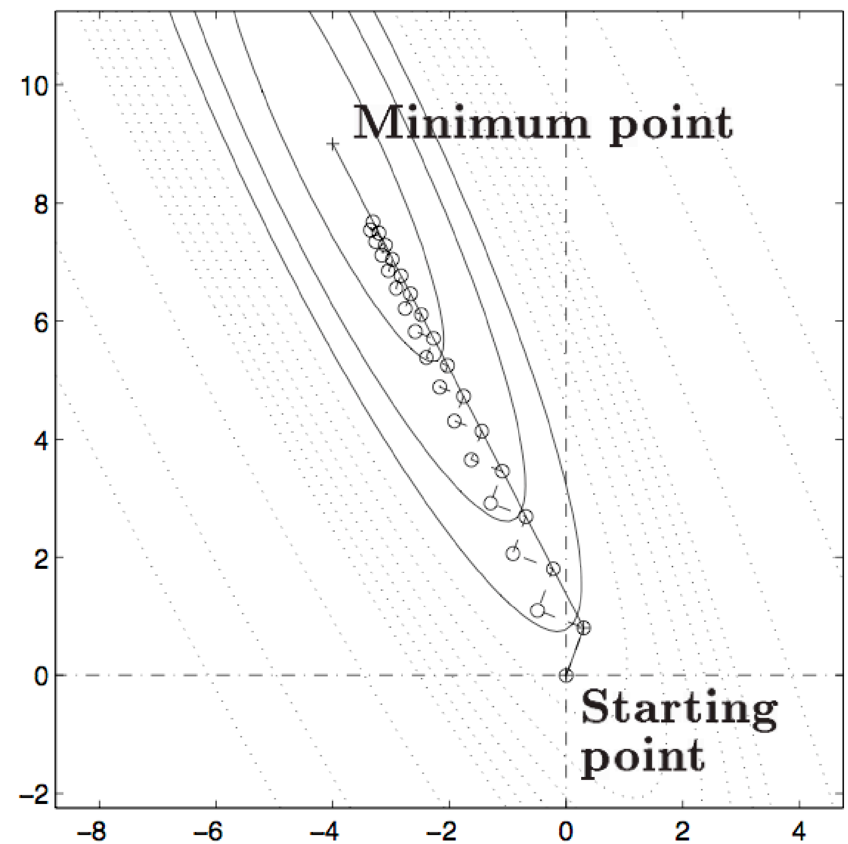
$$\boxed{\alpha = \frac{r^T r}{r^T Ar}}$$

So finally we have each part…

- Given an initial condition, we can iteratively head towards the minimum of a function J
  - **We compute the direction r and step size alpha at each k**
  - **If we have a small enough error between Ax-b, we stop**
  - **Or we stop if we've iterated too many times, as a convergence check**

But…

- There are issues with this method when the objective function is more challenging - with very steep sides and long flat valleys (*poorly conditioned*)
- This method also is a bit inefficient since it must 'tack' back and forth at 90 degree increments
  - **Due to successive line minimization and lack of momentum from one iteration to the next**
  - **THERE *HAS* TO BE A BETTER WAY!!!**



THERE IS - *Conjugate Gradient Descent*

- When you ski, you don’t instantaneously tack back and forth, you have some momentum from the previous moment leading you to the next
- With a slight modification to the previous method we can arrive at a method that doesn’t get hindered by long narrow valleys

How CG improves over steepest descent

- Instead of minimizing over a single alpha, which does one direction at a time for that iteration, we minimize our function in every direction simultaneously while only searching in one direction at a time
  - **In other words, to converge in exactly m iterations to the answer, we should minimize over all the steps we'll take at once**
  - **(ie we can think of this as minimizing in m directions simultaneously)**
- We can do this in any number of search directions
  - **Prevents that 'tacking' phenomena exhibited by the gradient descent method**

How it's done…

- We can reduce this problem to minimizing each direction individually provided the different directions are independent of each other, or ***conjugate*** in the following sense

$$p^{(i)^T} A p^{(j)} = 0, i \neq j$$

- We can choose our p's so they are conjugate in the following way

Solving in m iterations

- Consider that we start with our initial guess x_0, then move to our solution, x_m

$$x_m = x_0 + \sum_{j=0}^{m-1} \alpha_j p_j$$

- Substitute that into J, then compute the partial derivative with respect to each alpha, set that equal to zero

$$J(x_m) \Rightarrow \frac{\partial J(x_m)}{\partial \alpha_k} = 0$$

What does it boil down to?

- We compute a sequence of p's which are conjugate
  - **We redefine the descent direction at each iteration after the first to be a linear combination of the direction of steepest descent r and the previous descent direction**

$$\mathbf{p}^{(k)} = \mathbf{r}^{(k)} + \beta\,\mathbf{p}^{(k-1)} \quad \text{and} \quad \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\,\mathbf{p}^{(k)}$$

$$\beta = \frac{\mathbf{r}^{(k)\,T}\mathbf{r}^{(k)}}{\mathbf{r}^{(k-1)\,T}\mathbf{r}^{(k-1)}}, \qquad \alpha = \frac{\mathbf{r}^{(k)\,T}\mathbf{r}^{(k)}}{\mathbf{p}^{(k)\,T} A\,\mathbf{p}^{(k)}}.$$

The result