APPM 4650 — Steepest descent

Motivation: Newton is very sensitive to the initial guess so we want something that is "less sensitive" to the initial guess. i.e. We don't know the basin of convergence for Newton in practice.

Idea: Our goal is to find $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)^T$ such that

$$\begin{cases} f_1(\alpha_1, \ldots, \alpha_n) &= 0 \\ \vdots & \\ f_n(\alpha_1, \ldots, \alpha_n) &= 0 \end{cases}$$

We will recast this problem as finding the minimum of a quadratic

$$g(x_1, \ldots, x_n) = \sum_{j=1}^{n} (f_j(\boldsymbol{x})^2_)$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^T$. The minimum of $g(\boldsymbol{x})$ is $\boldsymbol{\alpha}$.

The thing about this approach that is tricky is the fact that $g(\boldsymbol{x})$ may have many minimum.

Insert figure of a surface with many min.

The technique for building our iteration scheme is to start with a point on the surface and follow the path of maximum decrease. Hence the name Steepest Descent.

What is the direction of maximum decrease? It is the direction of the negative gradient; i.e.

$$-\nabla g = -\left(\frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n}\right)^T$$

Rough outline of the algorithm

1. Start with an initial guess $\boldsymbol{x}_0$
2. Determine the direction to go that results in a maximum decrease in g.
3. Move an appropriate amount β in that direction. Call the new location $\boldsymbol{x}_1$
4. Continue (i.e. go back to 1.)

Next we need to figure out how to evaluate ∇g. Recall that $g(\boldsymbol{x}) = \sum_{j=1}^{n} (f_j(\boldsymbol{x}))^2$. Let's see if we can write the gradient in terms of the Jacobian J and $\boldsymbol{F} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots f_n \end{bmatrix}$.

We know that $\nabla g = \left(\frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n}\right)^T$ and that

$$\frac{\partial g}{\partial x_i} = 2\left(f_1(\boldsymbol{x})\frac{\partial f_1}{\partial x_i} + \cdots + f_n(\boldsymbol{x})\frac{\partial f_n}{\partial x_i}\right)$$

So

$$\nabla g = 2 \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_1}{\partial x_n} & \cdots & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \boldsymbol{F}(\boldsymbol{x}) = \mathrm{J}^T \boldsymbol{F}$$

So up to now we know that we want an iteration of the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \lambda_k \nabla g(\boldsymbol{x}_k)$$

The question remains: how far should we go in the direction $\nabla g(\boldsymbol{x}_k)$? i.e. What is λ_k?

The idea is creative. This is a 1 dimensional minimization problem instead of an $n-$dimensional minimization problem. We seek λ_k such that the function $h(\lambda_k) := g(\boldsymbol{x}_k - \lambda_k \nabla g(\boldsymbol{x}_k))$ is minimized.

The function h is difficult to evaluate. So we will do a quadratic approximation and use the minimum of that.

More specifically, let's pick 3 values of β such that $\beta_1 \leq \beta_2 \leq \beta_3$ and create a quadratic approximation $p(\beta)$ of $h(\beta)$. We will find where $p(\beta)$ is minimized an call that λ_k. We want λ_k to between $[\beta_1, \beta_3]$ and will set $\beta_1 = 0$.

Our quadratic approximation is gong to be

$$p(\beta) = h_1 + s_1\beta + s_3\beta(\beta - 2)$$

where $h_1 = h(0)$, $h_2 = h(\beta_2)$ and $h_3 = h(\beta_3)$ $s_1 = \frac{h_2 - h_1}{\beta_2}$, $s_2 = \frac{h_3 - h_2}{\beta_3 - \beta_2}$ and $s_3 = \frac{s_2 - s_1}{\alpha_3}$.

This is the Newton Divided difference form of the approximation. You will learn this next week in detail. :)

The minimum of this quadratic is $\beta = \frac{-s_1 + \beta_2 s_2}{2s_3}$; where $p'(\beta) = 0$.

So we know how far we want to go but we do not know how to pick the β_j.
We already decided that $\beta_1 = 0$. Let's choose $\beta_2 = 0.5(\beta_1 + \beta_3)$ to be the midpoint. So how do we pick β_3?

The idea is to start with $\beta_3 = 1$ and if $h(\beta_3) \geq h(\beta_1)$ then set $\beta_3 = 0.5\beta_3$ until $h(\beta_1) > h(\beta_3)$. In other words, we are forcing the quadratic to have the desired concavity.

Possible troubles with this approach:

- g may have many minimums or you may get stuck.
  If g has any minimums, you need to verify that the output satisfies $\boldsymbol{F}(\alpha) = \boldsymbol{0}$.
- It is possible that you go right back to where you started or to places you have been.
- If it converges, the convergence is slow (approximately linear).

Possible extensions:

- Stochastic gradient descent. i.e. choose random directions in which to descend. This used often in machine learning, etc.
- A method called Conjugate gradient can be used which is similar to Steepest Descent but it keeps track of all the places you have been. (Most often used for solving symmetric linear systems)