Intro. Comp. for Data Science (FMI08)

Dr. Nono Saha

June 21, 2023

Max Planck Institute for Mathematics in the Sciences
University of Leipzig/ScaDS.AI

Spring 2023

Course plan

1. 2d gradient descent w/ backtracking
2. Newton's method
3. Conjugation gradient algorithm
4. Numerical optimization - `scipy`

Gradient descent method in 2d

A 2d cost function

We will be using `mk_quad()` to create quadratic functions with varying conditioning (as specified by the epsilon parameter).

$$f(x, y) = 0.33(x^2 + \epsilon^2 y^2)$$

$$\nabla f(x, y) = \begin{bmatrix} 0.66x \\ 0.66\epsilon^2 y \end{bmatrix}$$

$$\nabla^2 f(x, y) = \begin{bmatrix} 0.66 & 0 \\ 0 & 0.66\epsilon^2 \end{bmatrix}$$

Similarly, write a `Python` function that implements the Rosenbrock function, its first and second derivative. f is defined as follows:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Exercise

Plotting exercise

- Similar to the previous exercise, write a function called `plot_2d_traj` that takes as inputs: x, y, f, a title (`tle`) for your plot and a vector (or array) `traj`

2D Gradient descent

- Update your 1D gradient descent function to 2D
- Apply your 2D gradient descent on the two previously mentioned functions. Vary the parameters, plot the result and let's comment on them.

Newton's Method in 1d

Newton's Method in 1d

Lets simplify things for now and consider just the 1d case and write αp_k as Δ,

$$f(x_k + \Delta) \approx f(x_k) + \Delta f'(x_k) + \frac{1}{2}\Delta^2 f''(x_k)$$

To find the Δ that minimizes this function, we can take a derivative with respect to Δ and set the equation equal to zero, which gives,

$$0 = f'(x_k) + \Delta f''(x_k) \Rightarrow \Delta = \frac{f'(x_k)}{f''(x_k)}$$

Which then suggests an iterative update rule of

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Newton's Method: generalizing to nd

Based on the same argument, we can see the following result for a function in $\mathbb{R}^n$,

$$f(x_k + \Delta) \approx f(x_k) + \Delta^T \nabla f(x_k) + \frac{1}{2}\Delta^T \nabla^2 f(x_k)$$

To find the ∇ that minimizes this function, we can take a derivative with respect to ∇ and set the equation equal to zero, which gives,

$$0 = \nabla f(x_k) + \nabla^2 f(x_k) \Rightarrow \Delta = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k) f(x_k)$$

Which then suggests an iterative update rule of

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k) f(x_k)$$

Conjugate gradients

Conjugate gradients

This is a general approach for solving a system of linear equations with the form $Ax = b$ where A is an n×n symmetric positive definite matrix and b is $n1$ with x unknown.

This type of problem can also be expressed as a quadratic minimization problem of the form,

$$\min_x f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

The goal is then to find n conjugate vectors ($p_i^T Ap_j = 0$ for all $i \neq j$) and their coefficients such that.

$$x_* = \sum_{i=1}^{n} \alpha_i p_i$$

Conjugate gradient algorithm

Given x_0, we set the following initial values,

$$
\begin{aligned}
r_0 =& \nabla f(x_0) \\
p_0 =& -r_0 \\
k =& 0
\end{aligned}
$$

while $||r_k||_2 > tol$,

$$
\begin{aligned}
\alpha_k =& \frac{r_k^T p_k}{p_k^T \nabla^2 f(x_k) p_k} \\
x_{k+1} =& x_k + \alpha_k p_k \\
r_{k+1} =& \nabla f(x_{k+1}) \\
\beta_k =& \frac{r_{k+1}^T \nabla^2 f(x_k) p_k}{p_{k+1}^T \nabla^2 f(x_k) p_k} \\
p_{k+1} =& -r_{k+1} + \beta_k p_k; k = k+1
\end{aligned}
$$

Numerical optimisation methods in `scipy`

CG in `scipy`

`Scipy`'s optimize module implements the conjugate gradient algorithm by Polak and Ribiere, a variant that does not require the hessian,

Differences

- α_k is calculated via a line search along the direct
- β_{k+1} is replaced with

$$\beta_{k+1}^{PR} = \frac{\nabla f(x_{k+1})(\nabla f(x_{k+1}) - \nabla f(x_k))}{\nabla f(x_k)^T \nabla f(x_k)}$$

Other methods in `scipy`

Method: Newton-CG
It is a variant of Newton's method but does not require inverting the hessian, or even a hessian function - in which case it can be estimated by finite differencing of the gradient.

Method: BFGS
The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is a quasi-newton that iteratively improves its approximation of the hessian,

Method: Nelder-Mead
This is a gradient-free method that uses a series of simplexes which are used to iteratively bracket the minimum.