Project 1 GSI Notes

Newton's Method Intuition

Newton's method is an efficient algorithm for finding the roots of a function (i.e., points where the function evaluates to zero). It is widely used in scientific computing for tasks ranging from nonlinear solvers to optimization routines. For example, when a calculator computes the intersection of two curves, it is often using a variant of Newton's method under the hood. In practice, Newton steps are often combined with step-size control (e.g., line search or damping) to improve robustness and a numerical differentiation scheme if symbolic differentiation is impractical.

The method begins with an initial guess x_0. Suppose that at this point we can evaluate both the function value and its slope (or gradient in multiple dimensions). If the function were perfectly linear, then the slope at x_0 would remain constant, and we could extrapolate directly to where the function reaches zero. Given the value and slope at x_0, this extrapolated point is easy to compute and defines an updated approximation x_1.

If the function truly were linear, this single step would recover the exact solution. For nonlinear functions, the hope is that this linear approximation is locally accurate, so that the updated point x_1 is closer to the true root. The process is then repeated, producing a sequence of iterates that (under suitable conditions) converge rapidly to the solution.

In many applications, including this project, we are not interested in finding where a function equals zero, but rather where it is minimized. From calculus, we know that the extrema of a differentiable function occur where its derivative vanishes. As a result, minimizing a function can be reformulated as a root-finding problem applied to its derivative, making Newton's method a natural tool for optimization.

Newton's Method Equations

For a differentiable scalar function $g(x)$, Newton's method for finding a root of g is given by

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}. \tag{1}$$

To find extrema of a twice-differentiable function $f(x)$, we apply Newton's method to its derivative by setting $g(x) = f'(x)$. Substituting yields

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{2}$$

In higher dimensions, where the solution variable $x \in \mathbb{R}^n$ is a vector, the derivative generalizes to the gradient and Hessian. Newton's method for minimizing a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$ takes the form

$$x_{k+1} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k), \tag{3}$$

where $\nabla f(x_k)$ is the gradient vector and $\nabla^2 f(x_k)$ is the Hessian matrix evaluated at x_k.

Newton's Method Project-Specific Hints

1. Newton's method converges very rapidly when the initial guess is sufficiently close to a solution, but it may diverge or converge to an unintended stationary point if the initial guess is poor.

2. M=1 here. Your function will be more useful later on if it is capable of inputs beyond M=1, but we do not test that capability in this assignment.

3. In your implementation, Π_A and Π_b and their derivatives should be defined as separate hard coded functions which you can pass into your Newton's method function as arguments.

GA Intuition

The genetic algorithm is designed to mimic the process of evolution to explore a design space. It is substantially more expensive than Newton's method (many more design evaluations) but is capable of optimizing non-convex problems and problems with poorly defined gradients.

GA Process

1. Randomly generate S number of genetic strings as the first generation. In this project, `dv` is 1.

$$\Lambda^s = \{x_i^s\} = \{x^s\}, \quad s = 1, ..., S \text{ and } i = 1, ..., \texttt{dv} \tag{4}$$

2. Compute the fitness of each string using the objective function $\Pi(x_i)$.

3. Rank (sort) in decreasing order each genetic string's fitness.

4. Check if $f(x_{i+1}) \leq$ `TOL` or if the maximum number of generations `G` has been reached.

   - TRUE: Stop search. Found a local minimum.
   - FALSE: Continue to the next step.

5. Combine top `P` parents to produce `K` children. Note: To combine the first-ranked parent with the second-ranked parent, do the following:

$$\begin{cases} x_{c1} &= \phi_1 x_{p1} + (1 - \phi_1) x_{p2} \\ x_{c2} &= \phi_2 x_{p1} + (1 - \phi_2) x_{p2}, \end{cases} \tag{5}$$

   where vectors ϕ_1 and ϕ_1 have been randomly and independently generated for each mating, e.g. $0 \leq \phi_{1,i}, \phi_{2,i} \leq 1$. Generate new ϕ_1 and ϕ_2 for mating the third-ranked and the fourth-ranked string, and so forth.

6. Populate the new generation with the `P` parents and the `K` children. Eliminate all parents ranked below the `P`'th parent.

7. Fill up the new generation with N new random strings. Here, $N = S - P - K$. Return to Step 2.