Merry Christmas 🎄
& Happy New Year ❄️

Eigen Letters

Location: Mumbai, India

VOL.I. . . No.3 *DECEMBER 25, 2022* Written & edited by: Puneet Panwar

Hey, the holiday season is here, and everyone is gearing up for Christmas. Merry Christmas and have a great festive season! With this, 2022 has also come to an end. Personally speaking, it was a great year in many aspects. If asked, I would sum it up with three words: *"Turbulence, Experience and Peace"*.

It's the third edition of Eigen Letters; in this edition, I will talk about *Optimization* in general and *Numerical Optimization* in particular. I am happy to report that the previous two editions saw good reach and as per the feedback the readers found the content helpful. It keeps me going and here is my Christmas and New year gift to all of you.

Optimization

Optimization can be defined as a process of selecting system variables/design parameters that optimizes (minimizes or maximizes) a performance-index/cost/objective function satisfying certain constraints and parametric bounds. Optimization problems are ubiquitous to all branches of engineering, design, economics, operations research and biology. Some common examples of optimization problems are as follows:

1. Design of an aircraft for minimum weight
2. Traveling salesman problem
3. Design of optimal control systems (min-fuel, min-time etc.)
4. Optimum design of mechanical components for cost minimization

Mathematically, an optimization problem can be formulated as:

$$\begin{aligned} \underset{x}{min} \quad & J(x) \\ \textit{subject to:} \quad & G_e(x) = 0, \; e = 1, \ldots.., p_e \\ & G_{ie}(x) \le 0, \; ie = 1, \ldots.., p_{ie} \\ x_l \le x \le x_u \end{aligned}$$

Where, x is the design parameter vector of length n, $J(x)$ is the cost/objective function and $G_e(x)$, $G_{ie}(x)$ are a vector function representing p_e, p_{ie} equality and inequality constraints respectively. x_l and x_u represents lower and upper bounds on design parameter values.

Based on various factors, optimization problems can be classified in several ways as shown in Table 1. No single method can be used to solve all optimization problems. To solve different types of optimization problems, several optimization methods have been developed, these optimum-seeking methods are also known as *mathematical programming techniques*.

For unconstrained optimization problems, classical methods of differential calculus can be used (with an assumption that the objective function is differentiable twice w.r.t. the design variables and derivatives are continuous). For problems with equality constraints, *the Lagrange Multiplier* method can be employed. For problems with inequality constraints, Karush-Kuhn-Tucker (KKT) conditions can be used to find the optimum point and the optimum point turns out to be the global minima/maxima if convexity condition on objective and constraint functions is satisfied.

Table 1: Types of optimization

Classification Criteria	**Type of optimization**
Existence of constraints	Constrained & Unconstrained
Nature of design variables	Static & Dynamic
Nature of Equations involved	Nonlinear & Linear programming Problems (NLP, LP)
Permissible values of design variables	Integer, real-valued programming problems
Number of objective functions	Single & multi-objective programming problems

The classical methods discussed above lead to a set of nonlinear simultaneous equations that may be difficult to solve. For problems with significant dimensions, finding the analytical solution becomes complicated. To solve such problems, people looked towards *Numerical Optimization*. Numerical optimization is an iteration based optimum finding method which gained popularity with the developments in computer technology and the availability of cheap-computing power.

Numerical Optimization

In numerical optimization, rather than looking for the exact analytical solution, an approximate solution is sought by proceeding iteratively by starting from an initial solution. To reach the optimal solution, the following philosophy is followed in numerical optimization:

- **Step 1 -** Begin with a meaningful initial guess value (x^1)
- **Step 2 -** Find a search direction p^k, $k = 1, 2, \ldots$
- **Step 3 -** Update the guess value, $x^{k+1} = x^k + \alpha p^k$, $\alpha > 0$
- **Step 4 -** Repeat step 2 & 3 until convergence is achieved:
  $$||J(x^{k+1}) - J(x^k)|| < \textit{tolerance}$$

α is known as *step-size*. The selection of a meaningful initial guess is a critical step. With the proper selection of initial guess values, the computational load is minimized (as less number of iterations are needed to reach the final solution), leading to faster convergence of the algorithm. The next big question is: *how do we find the search direction, p^k*. To find the answer, We will look at the following numerical optimization algorithms:

1. Steepest (Gradient) Descent Search
2. Line Search - Successive Parabolic Interpolation
3. Newton's Method

Here, we have only discussed unconstrained optimization problems, for optimization problems with equality constraints, philosophy of the above methods can be used on the augmented-cost function.

1. Steepest (Gradient) Descent Search: The steepest descent algorithm follows the update rule mentioned in step 3, at each iteration, the search direction p^k is the steepest direction we can take at that point. Using Taylor series expansion, the function at x_{k+1} may be expressed as,

$$J(x^{k+1}) = J(x^k) + \left[\nabla J(x^k)\right]^T (x^{k+1} - x^k) + H.O.T. \quad (1)$$

$$J(x^{k+1}) - J(x^k) \approx \left[\nabla J(x^k)\right]^T (x^{k+1} - x^k) \quad (2)$$

Using result from step (3), $x^{k+1} - x^k = \alpha p^k$, the equation (2) becomes:

$$J(x^{k+1}) - J(x^k) \approx \alpha \left[\nabla J(x^k)\right]^T p^k \quad (3)$$

If we choose, $p^k = -\nabla J(x^k)$ which also happens to be *Steepest descent direction*, right hand term in equation (3) becomes a quadratic term, which implies:

$$J(x^{k+1}) - J(x^k) \approx -\alpha \left[\nabla J(x^k)\right]^T \left[\nabla J(x^k)\right] \ (\alpha > 0) \quad (4)$$

$$J(x^{k+1}) - J(x^k) < 0 \quad (5)$$

This implies that if we choose $p^k = -\nabla J(x^k)$, the value of $J(x^{k+1})$ will always be less than $J(x^k)$ and with every iteration the solution will move closer to minimum value. Figure 1 shows pictorial view of steepest descent search algorithm. $\alpha > 0$ ensures that the solution always moves in the direction of p^k.

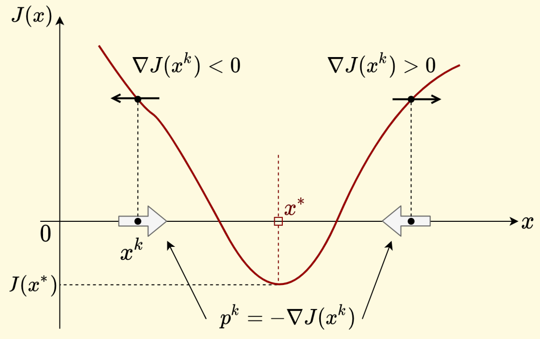

Fig. 1: Steepest descent search

2. Line Search - Successive Parabolic Interpolation: In this method, following steps are involved:

1. Find three guess values of p, such that there is an up-down-up behaviour
2. Fit a quadratic curve for these points
3. Minimum of this quadratic curve is the updated value
4. Find a new direction at this point
5. Repeat the procedure.

Figure 2 shows the visual representation of *Line Search Method*. Steepest descent and Line search are called first order methods because these methods only use gradient information to reach the optimum value. Execution of the algorithm proceeds without knowing how fast the minimum is approached. In other words, there is no mechanism available to take advantage of the function's curvature. The next method takes advantage of second-order derivatives, which improves the convergence rate of the algorithm.

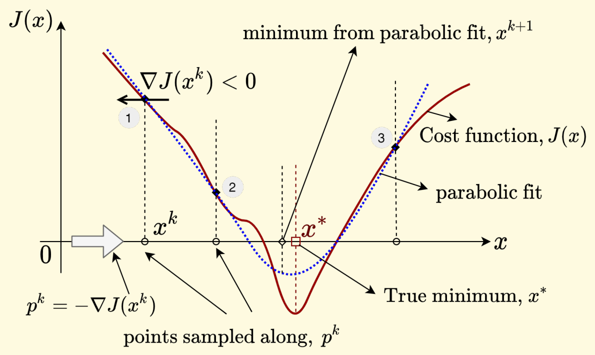

Fig. 2: Line Search - Parabolic Interpolation

3. Newton's Method: in Newton's method, a search direction that includes second-order derivative (curvature) will be generated. Using Taylor series expansion of the gradient, $J(x^{k+1})$:

$$\nabla J(x^{k+1}) = \nabla J(x^k) + \left[\nabla^2 J(x^k)\right] (x^{k+1} - x^k) + H.O.T. \quad (6)$$

$$0 \approx \nabla J(x^k) + \alpha \left[\nabla^2 J(x^k)\right] p^k \quad (7)$$

$$p^k = -\left(\frac{1}{\alpha}\right) \left[\nabla^2 J(x^k)\right]^{-1} \nabla J(x^k) \quad (8)$$

$$p^k = -\beta \left[\nabla^2 J(x^k)\right]^{-1} \nabla J(x^k), \quad \beta > 0 \quad (9)$$

At optimum point, $\nabla J(x^{k+1})$ will be zero. This result is used in equation (7). The advantage of this method is that it has fast convergence but the flip-side is that computing and storing $\left[\nabla^2 J(x^k)\right]^{-1}$ is very expensive. To avoid the drawbacks of Newton's method, quasi-Newton method is developed (not covered in the scope this article).
Engineering Optimization Theory and Practice by Singiresu S Rao is a good resource to learn more about this topic. In future, I will try to upload the code of these algorithms on my website.

The good stuff!

🎵 **Music:** *Tur Kalleyan*. This song is about courage and walking alone when you know it's the right path. There comes a time, when you feel it's getting too heavy. Always remember that you will be the first and last person to back yourself in any adversities of life. Never give up on yourself. Never, ever. If you fall, getup, analyze, work hard and **Win**! 🥇

🖊 **Quotes**: *"I don't stop when I'm tired, I stop when I'm done." - David Goggins*
"My success will not depend on what A or B thinks of me. My success will be what I make of my work." - Homi Jehangir Bhabha

🎬 **Video:***Using Caffeine to Optimize Mental & Physical Performance* by Prof. Andrew Huberman

I hope that this edition will add a little to your knowledge bank. If you liked the content of this edition, share it with your people. Most importantly, I would request to provide your valuable feedback and suggestions to improve future editions. To subscribe to *Eigen Letters*, visit *here*. Thanks for reading. Have a great year, Keep learning, and keep winning!

Best, Puneet Panwar