Numerical Methods

Numerical Methods

The basic philosophy is to produce a sequence of improved approximations to the optimum according to the following scheme:

1. Start with an initial trial point $\boldsymbol{X}_1$.
2. Find a suitable direction $\boldsymbol{S}_i$ ($i = 1$ to start with) that points in the general direction of the optimum.
3. Find an appropriate step length λ_i^* for movement along the direction $\boldsymbol{S}_i$.
4. Obtain the new approximation $\boldsymbol{X}_{i+1}$ as
$$\boldsymbol{X}_{i+1} = \boldsymbol{X}_i + \lambda_i^* \boldsymbol{S}_i$$
5. Test whether $\boldsymbol{X}_{i+1}$ is optimum. If $\boldsymbol{X}_{i+1}$ is optimum, stop the procedure. Otherwise, set a new $i = i + 1$ and repeat step (2) onward

Direct Root Methods

The necessary condition for $f(\lambda)$ to have a minimum of λ^* is that $f(\lambda^*) = 0$.

The direct root methods seek to find the root (or solution) of the equation, $f(\lambda) = 0$.

Three root-finding methods—the Newton, the quasi-Newton, and the secant methods

Newton Method

Newton's method, which uses the Hessian of the function, has a quadratic rate of convergence.

The basic idea of the Newton's method is to use a second-order Taylor's expansion of the function about the current design point.

$$f(\boldsymbol{X} + \boldsymbol{\Delta X}) = f(\boldsymbol{X}) + \boldsymbol{c}^T \boldsymbol{\Delta X} + \frac{1}{2} \boldsymbol{\Delta X}^{\mathrm{T}} \, \mathbf{H} \boldsymbol{\Delta X}$$

If $\boldsymbol{H}$ is positive semidefinite, then there is a $\boldsymbol{\Delta X}$ that gives a global minimum for the function $f(\boldsymbol{X} + \boldsymbol{\Delta X})$

Newton Method

The optimality conditions $(\frac{\delta f}{\delta \boldsymbol{x}})$ for the Taylor expansion are

$$\boldsymbol{c} + \boldsymbol{H}\boldsymbol{\Delta X} = \mathbf{0}$$

Assuming $\boldsymbol{H}$ to be nonsingular, we get an expression for $\boldsymbol{\Delta X}$ as

$$\boldsymbol{d} = \boldsymbol{\Delta X} = -\mathbf{H}^{-1}\mathbf{c}$$

Which can be used to update the design variable

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$$

Newton Method

The Newton method was originally developed by Newton for solving nonlinear equations and later refined by Raphson, and hence the method is also known as Newton–Raphson method in the literature of numerical analysis.

The method requires both the first- and second-order derivatives of $f(\lambda)$.

If $f''(\lambda_i) \neq 0$ the Newton iterative method has a powerful (fastest) convergence property, known as quadratic convergence.

4. If the starting point for the iterative process is not close to the true solution λ^*, the Newton iterative process might diverge.

Quasi-Newton Method

Newton's method can be inefficient because it requires calculation of $\frac{n(n+1)}{2}$ second-order derivatives to generate the Hessian matrix.

Newton's method runs into difficulties if the Hessian of the function is singular at any iteration.

Quasi-Newton Methods generate an approximation for the Hessian matrix or its inverse at each iteration.

Only the first derivatives of the function are used to generate these approximations

Quasi-Newton Method

The Hessian is approximated by using two pieces of information:

change in design variables

change in gradient vectors

While updating, the properties of symmetry and positive definiteness are preserved.

The DFP method, builds an approximate inverse of the Hessian of $f(x)$ using only the first derivatives.

The BFGS method uses the approximates the Hessian of $f(x)$ rather than its inverse in every iteration.

Marquardt method

Marquardt (1963) suggested a modification to the direction finding process that has the desirable features of the steepest descent and Newton's methods.

Far away from the solution point, the method behaves like the steepest descent method.

Near the solution point it behaves like the Newton's method.

With the Marquardt modification the direction is given as

$$\boldsymbol{d}_k = -(\mathbf{H}_\mathrm{k} + \lambda_k \mathbf{I})\mathbf{c}_\mathrm{k}$$

I If the direction $\boldsymbol{d}_k$ does not reduce the cost function, then λ is increased and the search direction is recomputed

Generalized Reduced Gradient Method

In 1967, Wolfe developed the reduced gradient method based on a simple variable elimination technique for equality constrained problems.

The generalized reduced gradient (GRG) method is an extension of the reduced gradient method to accommodate nonlinear inequality constraints.

In this method, a search direction is found such that for any small move, the current active constraints remain precisely active.

Generalized Reduced Gradient Method

If some active constraints are not precisely satisfied because of nonlinearity of constraint functions, the Newton-Raphson method is used to return to the constraint boundary.

Thus, the GRG method can be considered somewhat similar to the gradient projection method.

Since inequality constraints can always be converted to equalities by adding slack variables, we can form an equality constrained NLP model.

Constrained Problems

Penalty functions

Unconstrained optimization methods can also be used to solve constrained design problems.

The basic idea is to construct a composite function using the cost and constraint functions.

Using the constraint functions, it is possible to construct a penalty function P.

The penalty function penalizes the composite function for violation of constraints.

Constrained Problems

Penalty functions

Unconstrained optimization methods can also be used to solve constrained design problems.

The basic idea is to construct a composite function using the cost and constraint functions.

Using the constraint functions, it is possible to construct a penalty function P.

The penalty function penalizes the composite function for violation of constraints.

Constrained Problems

Penalty and Barrier Methods

General classical constrained minimization problem

$$\begin{aligned} & \text{minimize } f(\boldsymbol{x}) \\ \text{subject to} \quad & g(\boldsymbol{x}) \leq 0 \\ & h(\boldsymbol{x}) = 0 \end{aligned}$$

Penalty methods are motivated by the desire to use unconstrained optimization techniques to solve constrained problems.

This is achieved by either

- adding a penalty for infeasibility and forcing the solution to feasibility and subsequent optimum, or
- adding a barrier to ensure that a feasible solution never becomes infeasible.

Constrained Problems

Penalty and Barrier Methods:

- minimize objective as unconstrained function
- provide penalty to limit constraint violations
- magnitude of penalty varies throughout optimization
- sequential unconstrained minimization techniques

$$F(\boldsymbol{x}, r_p) = f(\boldsymbol{x}) + r_p\, P(\boldsymbol{x})$$

$f(\boldsymbol{x})$: original objective function

$P(\boldsymbol{x})$: imposed penalty function

r_p : scalar multiplier to determine penalty magnitude

p : unconstrained minimization number

Constrained Problems

Penalty and Barrier Methods:

$$F(\boldsymbol{x}, r_p) = f(\boldsymbol{x}) + r_p\, P(\boldsymbol{x})$$

$$P(\boldsymbol{x}) = \sum_{j=1}^{n} \left(\max\left[0, g_j(\boldsymbol{x})\right]\right)^2 + \sum_{k=1}^{m} [h_k(\boldsymbol{x})]^2$$

Notes:

- if all constraints are satisfied, then $P(\boldsymbol{x}) = 0$
- penalty parameter, r_p starts as a small number but can increase with number of iterations.
- if r_p is small $F(\boldsymbol{x}, r_p)$ is easy to minimize but yields large constraint violations
- if r_p is large, constraints are all nearly satisfied but the function is numerically ill-conditioned