Numerical Optimization in DROP

v3.92 1 December 2018

Introduction

Framework Glossary

1. Hyperspace Search: Hyperspace search is a search to determine whether the entity is inside the zone of a range, e.g., bracketing search.
2. Hyperpoint Search: Hyperpoint searches are hard searches that search for an exact specific point (to within an appropriately established tolerance).
3. Iterate Nodes: This is the set of the traveled nodes (variate/Objective Function ordered pairs) that contain the trajectory traveled.
4. Iteration Search Primitives: The set of variate iteration routines that generate the subsequent iterate nodes.
5. Compound iterator search scheme: Search schemes where the primitive iteration routine to be invoked at each iteration are evaluated.
6. RunMap: Map that holds the program state at the end of each iteration, in the generic case, this is composed of the Wengert iterate node list, along with the corresponding root finder state.
7. Cost of Primitive (cop): This is the cost of invocation of a single variate iterator primitive.

Document Layout

1. Base Framework
2. Search Initialization
    a. Bracketing
    b. Objective Function Failure
    c. Bracketing Start Initialization

Framework

1. The root search given an objective function and its goal is achieved by iteratively evolving the variate, and involves the following steps:
   - Search initialization and root reachability determination: Searched is kicked off by spawning a root variate iterator for the search initialization process (described in detail in the next section).
   - Absolute Tolerance Determination.
   - Root Search Iteration: The root is searched iteratively according to the following steps:
     1. The iterator progressively reduces the bracket width.
     2. Successive iteration occurs using either a single primitive (e.g., using the bisection primitive), or using a selector scheme that picks the primitives for each step (e.g., Brent's method).
     3. For Open Method, instead of 1 and 2, the routine drives towards convergence iteratively.
   - Search Termination Detection: The search termination occurs typically based on the following:
     - Proximity to the Objective Function Goal
     - Convergence on the variate
     - Exhaustion if the number of iterations
2. The flow behind these steps is illustrated in Figure 1.
3. The "Flow Control Variate" in root search is the "Objective Function Distance to Goal" Metric.

Search Initialization

1. Broadly speaking, root finding approaches can be divided into a) those that bracket roots before they solve for them, and b) those that don't need to bracket, opting instead to pick a suitable starting point.
2. Depending upon the whether the search is a bracketing or an open method, the search initialization does one the following:
    - Determine the root brackets for bracketing methods
    - Locate root convergence zone for open methods
3. Initialization begins by a search for the starting zone. A suitable starting point/zone is determined where, by an appropriate choice for the iterator, you are expected to reach the fixed-point target within a sufficient degree of reliability. Very general-purpose heuristics often help determine the search start zone.
4. Both bracketing and open initializers are hyperspace searches, since they search for something "IN", not "AT".

Bracketing

1. Bracketing is the process of localizing the fixed point to within a target zone with the least required number of Objective Function calculations. Steps are:
    - Determine a valid bracketing search start
    - Choose a suitable bracket expansion
    - Limit attention to where the Objective Function is defined (more on this below).
2. Figure 2 shows the flow for the Bracketing routine.
3. Bracketing methods require that the initial search interval bracket the root (i.e. the function values at interval end points have opposite signs).

4. Bracketing traps the fixed point between two variate values, and uses the intermediate value theorem and the continuity of the Objective Function to guarantee the presence/existence of the fixed point between them.
5. Unless the objective function is discontinuous, bracketing methods guarantee convergence (although may not be within the specified iteration limit).
6. Typically, they do not require the objective function to be differentiable.
7. Bracketing iteration primitives' convergence is usually linear to super-linear.
8. Bracketing methods preserve bracketing throughout computation and allow user to specify which side of the convergence interval to select as the root.
9. It is also possible to force a side selection after a root has been found, for example, in sequential search, to find the next root.
10. Generic root bracketing methods that treat the objective function as a black box will be slower than targeted ones – so much so that they can constitute the bulk of the time for root search. This is because, to accommodate generic robustness coupled with root-pathology avoidance (oscillating bracket pairs etc.), these methods have to perform a full variate space sweep without any assumptions regarding the location of the roots (despite this most bracketing algorithms cannot guarantee isolation of root intervals). For instance, naïve examination of the Objective Function's "sign-flips" alone can be misleading, especially if you bracket fixed-points with even numbered multiplicity within the brackets. Thus, some ways of analyzing the Black Box functions (or even the closed form Objective Functions) are needed to better target/customize the bracketing search (of course, parsimony in invoking the number of objective function calls is the main limitation).
11. Soft Bracketing Zone: One common scenario encountered during bracketing is the existence of a soft preferred bracketing zone, one edge of which serves as a "natural edge". In this case, the bracketing run needs to be positioned to be able to seek out starting variate inside soft zone in the direction AWAY from the natural edge.
12. The first step is to determine a valid bracketing search start. One advantage with univariate root finding is that objective function range validity maybe established

using an exhaustive variate scanner search without worrying about combinatorial explosion.

Objective Function Failure

1. Objective Function may fail evaluation at the specified variate for the following reason:
    - Objective Function is not defined at the specified variate.
    - Objective Function evaluates to a complex number.
    - Objective Function evaluation produces NaN/Infinity/Under-flow/Over-flow errors.
    - In such situations, the following steps are used to steer the variate to a valid zone.
2. Objective Function undefined at the Bracketing Candidate Variate: If the Objective Function is undefined at the starting variate, the starting variate is expanded using the variate space scanner algorithm described above. If the objective Function like what is seen in Figure 3, a valid starting variate will eventually be encountered.
3. Objective Function not defined at any of the Candidate Variates: The risk is that the situation in Figure 4 may be encountered, where the variate space scanner iterator "jumps over" the range over which the objective function is defined. This could be because the objective function may have become complex. In this case, remember that an even power of the objective function also has the same roots as the objective function itself. Thus, solving for an even power of the objective function (like the square) – or even bracketing for it – may help.

Bracketing Start Initialization

1. Figure 5 shows the flow behind a general-purpose bracket start locator.

2. Once the starting variate search is successful, and the objective function validity is range-bound, then use an algorithm like bisection to bracket the root (as shown in Figure 6 below).
3. However, if the objective function runs out of its validity range under the variate scanner scheme, the following steps need to be undertaken:
    - If the left bracketing candidate fails, bracketing is done towards the right using the last known working left-most bracketing candidate as the “left edge”.
    - Likewise, if the right bracketing candidate fails, bracketing is done towards the left using the last known working right-most bracketing candidate as the “right edge”.
4. The final step is to trim the variate zone. Using the variate space scanner algorithm, and the mapped variate/Objective Function evaluations, the tightest bracketing zones are extracted (Figure 7).

Open Search Initialization

1. Non-bracketing methods use a suitable starting point to kick off the root search. As is obvious, the chosen starting point can be critical in determining the fate of the search. In particular, it should be within the zone of convergence of the fixed-point root to guarantee convergence. This means that specialized methods are necessary to determine zone of convergence.
2. When the objective function is differentiable, the non-bracketing root finder often may make use of that to achieve quadratic or higher speed of convergence. If the non-bracketing root finder cannot/does not use the objective function’s differentiability, convergence ends up being linear to super-linear.
3. The typical steps for determining the open method starting variate are:
    - Find a variate that is proximal to the fixed point
    - Verify that it satisfies the convergence heuristic

4. Bracketing followed by a choice of an appropriate primitive variate (such as bisection/secant) satisfies both, thus could be a good starting point for open method searches like Newton's method.
5. Depending upon the structure of the Objective Function, in certain cases the chain rule can be invoked to ease the construction of the derivative – esp. in situations where the sensitivity to inputs are too high/low.

Search/Bracketing Initializer Heuristic Customization

1. Specific Bracketing Control Parameters
2. Left/Right Soft Bracketing Start Hints: The other components may be used from the bracketing control parameters.
3. Mid Soft Bracketing Start Hint: The other components may be used from the bracketing control parameters.
4. Floor/Ceiling Hard Bracketing Edges: The other components may be used from the bracketing control parameters.
5. Left/Right Hard Search Boundaries: In this case, no bracketing is done – brackets are used to verify the roots, search then starts directly.

Numerical Challenges in Search

1. Bit Cancellation
2. Ill-conditioning (e.g., see high order polynomial roots)
3. "domains of indeterminacy" – existence of sizeable intervals around which the objective function hovers near the target
4. Continuous, but abrupt changes (e.g., near-delta Gaussian objection function)
5. Under-flow/over-flow/round-off errors
6. root multiplicity (e.g., in the context of polynomial roots)
7. Typical solution is to transform the objective function to a better conditioned function – insight into the behavior of the objective can be used to devise targeted solutions.

Variate Iteration

1. $$\nu_{i+1} = I(\nu_i, \Im_i)$$

   where ν_i is the i[th] variate and $\Im_i$ is the root finder state after the i[th] iteration.
2. Iterate nodes as Wengert variables: Unrolling the traveled iterate nodes during the forward accumulation process, as a Wengert list, is a proxy to the execution time, and may assist in targeted pre-accumulation and check-pointing.
3. Cognition Techniques of Mathematical Functions:
   - Wengert Variate Analysis => Collection of the Wengert variates helps build and consolidate the Objective Function behavior from the variate iterate Wengert nodes – to build a behavioral picture of the Objective Function.
   - Objective Function Neighborhood Behavior => With every Wengert variable, calculation of the set of forward sensitivities and the reverse Jacobians builds a local picture of the Objective Function without having to evaluate it.
4. Check pointing: Currently implemented using a roving variate/OF iterate node "RunMap"; this is also used to check circularity in the iteration process.
5. Compound Iterator RunMap: For compound iterations, the iteration circularity is determined the doublet $(\nu_i, \Im_i)$, so the Wengert RunMap is really a doublet Multi-Map.
6. Hyperpoint univariate fixed point search proximity criterion: For hyperpoint checks, the search termination check needs to explicitly accommodate a "proximity to target" metric. This may not be then case for hyperspace checks.
7. Regime crossover indicator: On one side the crossover, the variate is within the fast convergence zone, so you may use faster Open techniques like the Newton's methods. On the other side, continue using the bracketing techniques.

   a. Fast side of the crossover must be customizable (including other Halley's method variants); robust side should also be customizable (say False Position).

8. Crossover indicator determination: Need to develop targeted heuristics needed to determine the crossover indicator.
   - Entity that determines the crossover indicator may be determined from the relative variate shift change $\frac{x_{N+1}-x_N}{x_N-x_{N-1}}$ and the relative objective function change $\frac{y_{N+1}-y_N}{y_N-y_{N-1}}$.
9. Types of bracketing primitives:
   - Bracket narrower primitives (Bisection, false position), and interpolator primitives (Quadratic, Ridder).
   - Primitive's COP determinants: Expressed in terms of characteristic compute units.
     a. Number of objective function evaluation (generally expensive).
     b. Number of variate iterator steps needed.
     c. Number of objective function invocation per a given variate iteration step.
   - Bracket narrower primitives => Un-informed iteration primitives, low invocation cost (usually single objective function evaluation), but low search targeting quality, and high COP.
   - Interpolator primitives => Informed iteration primitives, higher invocation cost (multiple objective function evaluations, usually 2), better search targeting quality, and lower COP.
10. Pre-OF Evaluation Compound Heuristic: Heuristic compound variates are less informed, but rely heavily on heuristics to extract the subsequent iterator, i.e., pre-OF evaluation heuristics try to guide the evolution without invoking the expensive OF evaluations (e.g., Brent, Zheng).
11. OF Evaluation Compound Heuristic: These compound heuristics use the OF evaluations as part of the heuristics algorithm to establish the next variate => better informed

Open Search Method: Newton's Method

1. Newton's method uses the objective function f and its derivative f' to iteratively evaluate the root.
2. Given a well-behaved function f and it's derivative f' defined over real x, the algorithm starts with an initial guess of x_0 for the root.
3. First iteration yields

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

4. This is repeated in

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

   till a value x_n that is convergent enough is obtained.
5. If α is a simple root (root with multiplicity 1), and

$$\epsilon_n = x_n - \alpha$$

   and

$$\epsilon_{n+1} = x_{n+1} - \alpha$$

   respectively, then for sufficiently large n, the convergence is quadratic:

$$\epsilon_{n+1} \approx \frac{1}{2}\left|\frac{f''(x_n)}{f'(x_n)}\right| {\epsilon_n}^2$$

6. Newton's method only works when f has continuous derivatives in the root neighborhood.
7. When analytical derivatives are hard to compute, calculate slope through nearby points, but convergence tends to be linear (like secant).
8. If the first derivative is not well behaved/does not exit/undefined in the neighborhood of a particular root, the method may overshoot, and diverge from that root.
9. If a stationary point of the function is encountered, the derivative is zero and the method will fail due to division by zero.
10. The stationary point can be encountered at the initial or any of the other iterative points.
11. Even if the derivative is small but not zero, the next iteration will be a far worse approximation.
12. A large error in the initial estimate can contribute to non-convergence of the algorithm (owing to the fact that the zone is outside of the neighborhood convergence zone).
13. If α is a root with multiplicity $m > 1$, then for sufficiently large n, the convergence becomes linear, i.e.,

$$\epsilon_{n+1} \approx \frac{m-1}{m}\epsilon_n$$

14. When there are two or more roots that are close together then it may take many iterations before the iterates get close enough to one of them for the quadratic convergence to be apparent.
15. However, if the multiplicity m of the root is known, one can use the following modified algorithm that preserves the quadratic convergence rate (equivalent to using successive over-relaxation)

$$x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$$

16. The algorithm estimates m after carrying out one or two iterations, and then use that value to increase the rate of convergence. Alternatively, the modified Newton's method may also be used:

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{f'(x_n)f'(x_n) - f(x_n)f''(x_n)}$$

17. It is easy to show that if

$$f'(x_n) = 0$$

and

$$f''(x_n) \neq 0$$

the convergence in the neighborhood becomes linear. Further, if

$$f'(x_n) \neq 0$$

and

$$f''(x_n) = 0$$

convergence becomes cubic.

18. One way of determining the neighborhood of the root. Define

$$g(x) = x - \frac{f(x)}{f'(x)}$$

$$p_n = g(p_{n-1})$$

where p is a fixed point of g, i.e.,

$$g \in C[a, b]$$

k is a positive constant,

$$p_0 \in C[a, b]$$

and

$$g(x) \in C[a, b] \; \forall \; x \in C[a, b]$$

19. One sufficient condition for p_0 to initialize a convergent sequence $\{p_0\}_{k=0}^{\infty}$, which converges to the root

$$x = p$$

of

$$f(x) = 0$$

is that

$$x \in (p - \delta, p + \delta)$$

and that δ be chosen so that

$$\frac{f(x_n)f''(x_n)}{f'(x_n)f'(x_n)} \leq k < 1 \; \forall \; x \in (p - \delta, p + \delta)$$

20. It is easy to show that under specific choices for the starting variate, Newton's method can fall into a basin of attraction. These are segments of the real number line such that within each region iteration from any point leads to one particular root - can be infinite in number and arbitrarily small. Also, the starting or the intermediate point

can enter a cycle - the n-cycle can be stable, or the behavior of the sequence can be very complex (forming a Newton fractal).

21. Newton's method for optimization is equivalent to iteratively maximizing a local quadratic approximation to the objective function. But some functions are not approximated well by quadratic, leading to slow convergence, and some have turning points where the curvature changes sign, leading to failure. Approaches to fix this use a more appropriate choice of local approximation than quadratic, based on the type of function we are optimizing. [13] demonstrates three such generalized Newton rules. Like Newton's method, they only involve the first two derivatives of the function, yet converge faster and fail less often.
22. One significant advantage of Newton's method is that it can be readily generalized to higher dimensions.
23. Also, Newton's method calculates the Jacobian automatically as part of the calibration process, owing to the reliance on derivatives – in particular, automatic differentiation techniques can be effectively put to use.

Closed Search Methods

Secant

1. Secant method results on the replacement of the derivative in the Newton's method with a secant-based finite difference slope.
2. Convergence for the secant method is slower than the Newton's method (approx. order is 1.6); however, the secant method does not require the objective function to be explicitly differentiable.
3. It also tends to be less robust than the popular bracketing methods.

Bracketing Iterative Search

1. Bracketing iterative root searches attempt to progressively narrow the brackets and to discover the root within.
2. The first set discusses the goal search univariate iterator primitives that are commonly used to iterate through the variate.
3. These goal search iterator primitives continue generating a new pair of iteration nodes (just like their bracketing search initialization counter-parts).
4. Certain iterator primitives carry bigger "local" cost, i.e., cost inside a single iteration, but may reduce global cost, e.g., by reducing the number iterations due to faster convergence.
5. Further, certain primitives tend to be inherently more robust, i.e., given enough iteration, they will find the root within – although they may not be fast.

6. Finally, the case of compound iterator search schemes, search schemes where the primitive iteration routine to be invoked at each iteration is evaluated on-the-fly, are discussed.
7. Iterative searches that maintain extended state across searches pay a price in terms of scalability – the price depending on the nature and the amount of state held (e.g., Brent's method carries iteration selection state, whereas Zheng's does not).

Univariate Iterator Primitive: Bisection

1. Bisection starts by determining a pair of root brackets a and b.
2. It iteratively calculates f at

$$c = \frac{a + b}{2}$$

   then uses c to replace either a or b, depending on the sign. It eventually stops when f has attained the desired tolerance.
3. Bisection relies on f being continuous within the brackets.
4. While the method is simple to implement and reliable (it is a fallback for less reliable ones), the convergence is slow, producing a single bit of accuracy with each iteration.

Univariate Iterator Primitive: False Position

1. False position works the same as bisection, except that the evaluation point c is linearly interpolated; f is computed at

$$c = \frac{bf(a) + af(b)}{f(a) + f(b)}$$

where $f(a)$ and $f(b)$ have opposite signs. This holds obvious similarities with the secant method.

2. False position method also requires that f be continuous within the brackets.
3. It is simple enough, more robust than secant and faster than bisection, but convergence is still linear to super-linear.
4. Given that the linear interpolation of the false position method is a first-degree approximation of the objective function within the brackets, quadratic approximation using Lagrange interpolation may be attempted as

$$f(x) = \frac{(x - x_{n-1})(x - x_n)}{(x_{n-2} - x_{n-1})(x_{n-2} - x_n)} f_{n-2} + \frac{(x - x_{n-2})(x - x_n)}{(x_{n-1} - x_{n-2})(x_{n-1} - x_n)} f_{n-1} + \frac{(x - x_{n-2})(x - x_{n-1})}{(x_n - x_{n-2})(x_n - x_{n-1})} f_n$$

where we use the three iterates, x_{n-2}, x_{n-1} and x_n, with their function values, f_{n-2}, f_{n-1} and f_n.

5. This reduces the number of iterations at the expense of the function point calculations.
6. Using higher order polynomial fit for the objective function inside the bracket does not always produce roots faster or better, since it may result in spurious inflections (e.g., Runge's phenomenon).
7. Further, quadratic or higher fits may also cause complex roots.

Univariate Iterator Primitive: Inverse Quadratic

1. Performing a fit of the inverse $\frac{1}{f}$ instead of f avoids the quadratic interpolation problem above. Using the same symbols as above, the inverse can be computed as

$$\frac{1}{f(y)} = \frac{(y-f_{n-1})(y-f_n)}{(f_{n-2}-f_{n-1})(f_{n-2}-f_n)}x_{n-2} + \frac{(y-f_{n-2})(x-f_n)}{(f_{n-1}-f_{n-2})(f_{n-1}-f_n)}x_{n-1} + \frac{(y-f_{n-2})(y-f_{n-1})}{(f_n-x_{n-2})(f_n-f_{n-1})}x_n$$

2. Convergence is faster than secant, but poor when iterates not close to the root, e.g., if two of the function values f_{n-2}, f_{n-1} and f_n coincide, the algorithm fails.

Univariate iterator primitive: Ridder's

1. Ridders' method is a variant on the false position method that uses exponential function to successively approximate a root of f.
2. Given the bracketing variates, x_1 and x_2, which are on two different sides of the root being sought, the method evaluates f at

$$x_3 = \frac{x_1 + x_2}{2}$$

3. It extracts exponential factor α such that $f(x)e^{\alpha x}$ forms a straight line across $x_1, x_2,$ and x_3. A revised x_2 (named x_4) is calculated from

$$x_4 = x_3 + (x_3 - x_1)\frac{sign[f(x_1) - f(x_2)]f(x_3)}{\sqrt{f^2(x_3) - f(x_1)f(x_2)}}$$

2. Ridder's method is simpler than Brent's method, and has been claimed to perform about the same.
3. However, the presence of the square root can render it unstable for many of the reasons discussed above.

Univariate compound iterator: Brent and Zheng

1. Brent's predecessor method first combined bisection, secant, and inverse quadratic to produce the optimal root search for the next iteration.
2. Starting with the bracket points a_0 and b_0, two provisional values for the next iterate are computed; the first given by the secant method

$$s = b_k - \frac{b_k - b_{k-1}}{f(b_k) - f(b_{k-1})} f(b_k)$$

   and the second by bisection

$$m = \frac{a_k + b_k}{2}$$

3. If s lies between b_k and m, it becomes the next iterate b_{k+1}, otherwise the m is the next iterate.
4. Then, the value of the new contra-point is chosen such that $f(a_{k+1})$ and $f(b_{k+1})$ have opposite signs.
5. Finally, if

$$|f(a_{k+1})| < |f(b_{k+1})|$$

   then a_{k+1} is probably a better guess for the solution than b_{k+1}, and hence the values of a_{k+1} and b_{k+1} are exchanged.
6. To improve convergence, Brent's method requires that two inequalities must be simultaneously satisfied.
   a) Given a specific numerical tolerance δ, if the previous step used the bisection method, and if

$$\delta < |b_k - b_{k-1}|$$

the bisection method is performed and its result used for the next iteration. If the previous step used interpolation, the check becomes

$$\delta < |b_{k-1} - b_{k-2}|$$

b) If the previous step used bisection, if

$$|s - b_k| < \frac{1}{2}|b_k - b_{k-1}|$$

then secant is used; otherwise the bisection used for the next iteration. If the previous step performed interpolation

$$|s - b_k| < \frac{1}{2}|b_{k-1} - b_{k-2}|$$

is checked instead.

7. Finally, since Brent's method uses inverse quadratic interpolation, s has to lie between $\frac{3a_k+b_k}{4}$ and b_k.
8. Brent's algorithm uses three points for the next inverse quadratic interpolation, or secant rule, based upon the criterion specified above.
9. One simplification to the Brent's method adds one more evaluation for the function at the middle point before the interpolation.
10. This simplification reduces the times for the conditional evaluation and reduces the interval of convergence.
11. Convergence is better than Brent's, and as fast and simple as Ridder's.

Polynomial Root Search

1. This section carries out a brief treatment of computing roots for polynomials.
2. While closed form solutions are available for polynomials up to degree 4, they may not be stable numerically.
3. Popular techniques such as Sturm's theorem and Descartes' rule of signs are used for locating and separating real roots.
4. Modern methods such as VCA and the more powerful VAS use these with Bisection/Newton methods – these methods are used in Maple/Mathematica.
5. Since the eigenvalues of the companion matrix to a polynomial correspond to the polynomial's roots, common fast/robust methods used to find them may also be used.
6. A number of caveats apply specifically to polynomial root searches, e.g., Wilkinson's polynomial shows why high precision is needed when computing the roots – proximal/other ill-conditioned behavior may occur.
7. Finally, special ways exist to identify/extract multiplicity in polynomial roots – they use the fact that $f(x)$ and $f'(x)$ share the root, and by figuring out their GCD.

Meta-heuristics

Introduction

1. Definition: Meta-heuristic is a higher-level procedure or heuristic designed to find, generate, or select a lower level procedure or heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity (Bianchi, Dorigo, Gambardella, and Gutjahr (2009)).
2. Applicability: Meta-heuristics techniques make only a few assumptions about the optimization problem being addressed, so are usable across a variety of problem (Blum and Roli (2003)).
3. Underpinning Philosophy: Many kinds of meta-heuristics implement some kind of stochastic optimization, so the solution depends upon the random variables being generated. As such it does not guarantee that a globally optimal solution can be found over some class of problems.
4. Search Strategy: By searching over a large set of feasible solutions meta-heuristics often finds good solutions with less computation effort than other algorithms, iterative methods, or simple heuristics (see Glover and Kochenberger (2003), Goldberg (2003), Talbi (2009)).
5. Literature: While theoretical results are available (typically on convergence and the possibility of locating global optimum, see Blum and Roli (2003)), most results on meta-heuristics are experimental, describing empirical results based on computer experiments with the algorithms.
    - While high quality research exists (e.g., Sorensen (2013)), enormously numerous meta-heuristics algorithms published as novel/practical have been of flawed quality – often arising out of vagueness, lack of conceptual elaboration, and ignorance of previous literature (Meta-heuristics (Wiki)).

Properties and Classification

1. Properties: This comes from Blum and Roli (2003):
    a. Meta-heuristics are strategies that guide the search process.
    b. The goal is to efficiently explore the search space in order to find near-optimal solutions.
    c. Techniques that constitute meta-heuristics range from simple local search procedures to complex learning processes.
    d. Meta-heuristic algorithms are approximate and usually non-deterministic.
    e. Meta-heuristics are not problem-specific.
2. Classification: These are taken from Blum and Roli (2003) and from Bianchi, Dorigo, Gambardella, and Gutjahr (2009):
    a. Classification based on the type of the search strategy
    b. Classification off-of single solution search vs. population based searches
    c. Classification off-of hybrid/parallel heuristics

Meta-heuristics Techniques

1. Simple Local Search Improvements: In this family of techniques, the search strategy employed is an improvement on simple local search algorithms; examples include simulated annealing, tabu search, iterated local search, variable neighborhood search, and GRASP (Blum and Roli (2003)).
2. Search Improvements with Learning Strategies: The other type of search strategy has a learning component to the search; meta-heuristics of this type include ant colony optimization, evolutionary computation, and genetic algorithms (Blum and Roli (2003)).
3. Single Solution Searches: These focus on modifying and improving a single candidate solution; single solution meta-heuristics include iterated local search, simulated annealing, variable neighborhood search, and tabu search (Talbi (2009)).

4. Population based Searches: Population based searches maintain and improve multiple candidate solutions using population characteristics to guide the search. These meta-heuristics include evolutionary computation, genetic algorithms, and particle swarm optimization (Talbi (2009)).
5. Swarm Intelligence: Swarm intelligence is the collective behavior of de-centralized, self-organized agents in a particle or a swarm. Ant colony optimization (Dorigo (1992)), particle swarm optimization (Talbi (2009)), artificial bee colony (Karaboga (2010)) are all example algorithms of swarm intelligence.
6. Hybrid meta-heuristic: These combine meta-heuristics with other optimization approaches (these could come from e.g., mathematical programming, constraint programming, machine learning etc.). Components of the hybrid meta-heuristic run concurrently and exchange information to guide the search.
7. Parallel meta-heuristic: This employs parallel programming techniques to run multiple meta-heuristics searches in parallel; these may range from simple distributed schemes to concurrent search runs that interact to improve the overall solution.

Meta-heuristics Techniques in Combinatorial Problems

1. Combinatorial Optimization Problems: In combinatorial optimization, an optimal solution is sought over a discrete search space. Typically the search-space of the candidate solution goes faster than exponentially as the problem-size increases, making an exhaustive search for the optimal solution infeasible (e.g., the Travelling Salesman Problem (TSP)).
2. Nature and Types of Combinatorial Problems: Multi-dimensional combinatorial problems (e.g., engineering design problems such as form-finding and behavior-finding (Tomoiaga, Chandris, Sumper, Sudria-Andrieu, and Villafafila-Robles (2013)) suffer from the usual curse of dimensionality, making them infeasible to analytical or exhaustive-search methods.
3. Meta-heuristics applied to Combinatorial Optimization: Popular meta-heuristic algorithms for combinatorial problems include genetic algorithms (Holland (1975)),

scatter search (Glover (1977)), simulated annealing (Kirkpatrick, Gelatt, and Vecchi (1983)), and tabu search (Glover (1986)).

Key Meta-heuristics Historical Milestones

1. Contributions: Many different meta-heuristics are in existence, and new variants are being continually developed. The following key milestone contributions have been extracted from Meta-heuristics (Wiki).
2. 1950s:
    a. Robbins and Munro (1951) work on stochastic optimization methods.
    b. Barricelli (1954) carries out the first simulation of the evolutionary process and uses it on general optimization problems.
3. 1960s:
    a. Rastrigin (1963) proposes random search.
    b. Matyas (1965) proposes random optimization.
    c. Nelder and Mead (1965) propose a simplex heuristic, which later shown to converge to non-stationary points on some problems.
    d. Fogel, Owens, and Walsh (1966) propose evolutionary programming.
4. 1970s:
    a. Hastings (1970) proposes the Metropolis-Hastings algorithm.
    b. Cavicchio (1970) proposes adaptation of the control parameters for an optimizer.
    c. Kernighan and Lin (1970) propose a graph-partitioning method that is related to variable-depth search and prohibition based tabu search.
    d. Holland (1975) proposes genetic algorithm.
    e. Glover (1977) proposes scatter search.
    f. Mercer and Sampson (1978) propose a meta-plan for tuning the optimizer's parameters by using another optimizer.
5. 1980s:
    a. Smith (1980) describes genetic programming.
    b. Kirkpatrick, Gelatt, and Vecchi (1983) propose simulated annealing.

c. Glover (1986) proposes tabu search, along with the first mention of the word meta-heuristic (Yang (2011)).
d. Moscato (1989) proposes memetic algorithms.

6. 1990s:
   a. Dorigo (1992) introduce ant colony optimization in his PhD thesis.
   b. Wolpert and MacReady (1995) prove the no free lunch theorems (these are later extended by Droste, Jansen, and Wegener (2002), Igel and Toussaint (2003), and Auger and Teytaud (2010)).

References

- Auger, A., and O. Teytaud (2010): Continuous Lunches are Free Plus the Design of Optimal Optimization Algorithms *Algorithmica* **57 (1)** 121-146.
- Barricelli, N. A (1954): Esempi Numerici di Processi di Evoluzione *Methodos* 45-68.
- Bianchi, L., M. Dorigo, L. M. Gambardella, and W. J. Gutjahr (2009): A Survey on Metaheuristics for Stochastic Combinatorial Optimization *Natural Computing: An International Journal* **8 (2)** 239-287.
- Blum, C., and A. Roli (2003): Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison *ACM Computing Surveys* **35 (3)** 268-308.
- Cavicchio, D. J. (1970): *Adaptive Search using Simulated Evolution* Technical Report **Computer and Communication Sciences Department, University of Michigan**
- Dorigo, M (1992): *Optimization, Learning, and Natural Algorithms* PhD Thesis **Politecnico di Milano**
- Droste, S., T. Jansen, and I. Wegener (2002): Optimization with Randomized Search Heuristics – The (A)NFL Theorem, Realistic Scenarios, and Difficult Functions *Theoretical Computer Science* **287 (1)** 131-144.
- Fogel, L., A. J. Owens, and M. J. Walsh (1966): *Artificial Intelligence Through Simulated Evolution* **Wiley**

- Glover, F. (1977): Heuristics for Integer Programming using Surrogate Constraints *Decision Sciences* **8 (1)** 156-166.
- Glover, F. (1986): Future Paths for Integer Programming and Links to Artificial Intelligence *Computers and Operations Research* **13 (5)** 533-549.
- Glover, F., and G. A. Kochenberger (2003): *Handbook of Metaheuristics* **57 Springer International Series in Operations Research and Management Science**.
- Goldberg, D. E. (1989): *Genetic Algorithms in Search, Optimization, and Machine Learning* **Kluwer Academic Publishers**
- Hastings, W. K. (1970): Monte Carlo Sampling Methods using Markov Chains and their Applications *Biometrika* **57 (1)** 97-109
- Holland, J. H. (1975): *Adaptation in Natural and Artificial Systems* **University of Michigan Press**
- Igel, C., and M. Toussaint (2003): On Classes of Functions for which the No Free Lunch Results hold *Information Processing Letters* **86 (6)** 317-321.
- Karaboga, D. (2010): Artificial Bee Colony Algorithm *Scholarpedia* **5 (3)** 6915.
- Kernighan, B. W., and S. Lin (1970): An Efficient Heuristic Procedure for Partitioning Graphs *Bell System Technical Journal* **49 (2)** 291-307
- Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi (1983): Optimization by Simulated Annealing *Science* **220 (4598)** 671-680.
- Matyas, J. (1965): Random Optimization *Automation and Remote Control* **26 (2)** 246-253.
- Mercer, R. E., and J. R. Sampson (1978): Adaptive Search using a Reproductive Meta-plan *Kybernetes* **7 (3)** 215-228.
- Moscato, P. (1989): *On Evolution, Search, Optimization, Genetic Algorithms, and Martial Arts: Towards Memetic Algorithms* Report 826 **Caltech Concurrent Computation Program**
- Nelder, J. A., and R. Mead (1965): A Simplex Method for Function Minimization *Computer Journal* **7** 308-313.

- Rastrigin, L. A. (1963): The Convergence of Random Search Method in the Extremal Control of a many Parameter System *Automation and Remote Control* **24 (10)** 1337-1342.
- Robbins, S., and H. Munro (1951): A Stochastic Approximation Method *Annals of Mathematical Statistics* **22 (3)** 400-407.
- Smith, S. F. (1980): *A Learning System based on Genetic Adaptive Algorithms* PhD Thesis **University of Pittsburgh**
- Sorensen, K. (2013): *Metaheuristics—the metaphor exposed*
- Talbi, E. G. (2009): *Metaheuristics: From Design to Implementation* **Wiley**.
- Tomoiaga, B., M. Chandris, A. Sumper, A. Sudria-Andrieu, and R. Villafafila-Robles (2013): Pareto Optimal Reconfiguration of Power Distribution Systems using a Genetic Algorithm based on NSGA-II *Energies* **6 (3)** 1439-1455.
- Wolpert, D. H., and W. G. MacReady (1995): *No Free Lunch Theorems for Search* Technical Report SFI-TR-95-02-010 **Santa Fe Institute**
- Yang, X. S. (2011): Metaheuristic Optimization *Scholarpedia* **6 (8)** 11472.

Introduction and Overview

Motivation, Background, and Setup

1. Mathematical Formulation and Framework: Convex Optimization is an important class of constrained optimization problems that subsumes linear and quadratic programming. Such problems are specified in the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$x \in S$$

where the feasible set

$$S \subseteq \mathbb{R}^n$$

is a *convex* closed subset of $\mathbb{R}^n$ and f is a *convex function* (Hauser (2012)).

2. Local Minima as Global Minima: Convex optimization problems are important because all local minima are also guaranteed to be globally minimal, although this value may not be reached necessarily at a unique point.

Convex Sets and Convex Hull

1. Definition of a Convex Set: A convex set S satisfies the following property. For any two points x and y in S the point $(1-u)\cdot x + u\cdot y$ for

$$u \in [0,1]$$

lies in S as well. In other words, the line segment between any two points in S must also lie in S.
2. Definition of a Convex Hull: The smallest convex set containing another set A is called the *convex Hull* of $C(A)$.
3. Definition of a Convex Function: A function of a scalar field f is convex if it satisfies the following property:

$$f\big((1-u)\cdot x + u\cdot y\big) \leq (1-u)\cdot f(x) + u\cdot f(y)$$

for all x and y and any

$$u \in [0,1]$$

In other words, the function value between any two points x and y lies below the line of f connecting $f(x)$ and $f(y)$. An equivalent definition is that on the line segment between x and y the area over the graph of f forms a convex set.
4. Minima of a Convex Function: Convex functions are always somewhat "bowl shaped", have no local maxima (unless constant), and have no more than one local minimum value. If a local minimum exists, then it is also a global minimum. Hence gradient descent and Newton methods (with line search) are guaranteed to produce the global minimum when applied to such functions.

Properties of Convex Sets/Functions

1. Connection Property: Convex sets are connected.
2. Intersection Property: The intersection of any number of convex sets is also a convex set.
3. Axiomatic Convex Sets: The empty set, the whole space, any linear sub-spaces, and half-spaces are also convex.
4. Convex Set Dimensionality Reduction: If a convex set of S is of a lower dimension than its surrounding space $\mathbb{R}^n$ then S lies on a linear sub-space of $\mathbb{R}^n$.
5. Typical Convex Set Closure Properties: Some common convex functions include e^x, $-\log x$, x^k where k is an even number. Convex functions are closed under addition, the max operator, monotonic transformations of the x variable, and scaling by a non-negative constant.
6. Convex Set from Ellipsoidal Constraints: Constraints of the form

$$x^T A x \leq c$$

   produce a closed convex set (an ellipsoid) if A is positive semi-definite, and c is a non-negative number.
7. Transformation onto Semi-definite Programs: The set of positive semi-definite matrices is a closed convex set. Optimization problems with these constraints are known as *semi-definite programs* (SDPs). A surprising number of problems, including LPs and QPs, can be transformed into SDPs.

Convex Optimization Problems

1. General Form of Convex Optimization: An optimization problem in general

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$g_i(x) \leq 0, i = 1, \cdots, m$$

and

$$h_j(x) = 0, j = 1, \cdots, p$$

is convex as long as f is convex, all of the g_i for

$$i = 1, \cdots, m$$

are convex, and all of the h_j for

$$j = 1, \cdots, p$$

are linear.

2. Elimination of the Equality Constraints: In convex optimization we will typically eliminate the equalities before optimizing, either by converting them into two inequalities

$$h_j(x) \leq 0$$

and

$$-h_j(x) \leq 0$$

or by performing a null-space transformation. Hence the h_j's can be dropped from typical treatments.

References

- Hauser (2012): Convex Optimization and Interior Point Methods.

Newton's Method in Optimization

Method

1. Iterative Root Finding vs Optimization: In calculus Newton's method is an iterative method for finding the roots of a differentiable function f, i.e., solutions to the equation

$$f(x) = 0$$

In optimization the Newton's method is applied to the derivative f' of a twice-differentiable function f to find the roots of the derivative, i.e., solutions to

$$f'(x) = 0$$

also known as the stationary points of f.

2. The Stationary Point of f: In the $1D$ problem the Newton's method attempts to construct a sequence x_n from an initial guess x_0 that converges towards some value x^* satisfying

$$f(x^*) = 0$$

This x^* is a stationary point of f (Newton's Method in Optimization (Wiki)).

3. Taylor Expansion of f around x_n: One wishes to find Δx such that $f(x_n + \Delta x)$ is a maximum. Thus one seeks to solve the equation that sets the derivative of the last expression with respect to Δx equal to zero:

$$0 = \frac{\partial}{\partial \Delta x}\left\{f(x_n) + f'(x_n)\Delta x + \frac{1}{2}f''(x_n)[\Delta x]^2\right\} = f'(x_n) + f''(x_n)\Delta x$$

4. Conditions for approximating the Stationary Point: For the value of

$$\Delta x = -\frac{f'(x_n)}{f''(x_n)}$$

which is a solution to the above equation, typically

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

will be closer to the stationary point x^*. Provided that $f(x)$ is a twice-differentiable function and other technical conditions are satisfied, the sequence $x_1, \cdots, x_n$ will converge to the point x^* satisfying

$$f(x^*) = 0$$

5. Geometric Interpretation of the Newton's Method: The geometric interpretation of the Newton's method is that at each iteration one approximates $f(\vec{x})$ by a quadratic function $\vec{x}_n$ and then takes a step towards a maximum/minimum of that quadratic function. In higher dimensions, this stationary point may also be a saddle point. Of course if $f(\vec{x})$ happens to *be* a quadratic function then the exact extremum is found in one step.

Higher Dimensions

1. Determination of the Iteration Increment: The above iteration scheme can be generalized to several dimensions by replacing the derivative with a gradient $\vec{\nabla} f(\vec{x})$

and the reciprocal of the second derivative with the inverse of the Hessian matrix $\mathbb{H}f(\vec{x})$. One thus obtains the iteration scheme

$$\vec{x}_{n+1} = \vec{x}_n + [\mathbb{H}f(\vec{x})]^{-1}\vec{\nabla}f(\vec{x}), n \geq 0$$

2. Adjustment to the Iteration Increment: Often Newton's method is modified to include a small step size

$$\gamma \in (0, 1)$$

instead of

$$\gamma = 1$$

to result in

$$\vec{x}_{n+1} = \vec{x}_n + \gamma[\mathbb{H}f(\vec{x})]^{-1}\vec{\nabla}f(\vec{x})$$

This is often done to ensure that the Wolfe conditions are satisfied at each step

$$\vec{x}_n \longrightarrow \vec{x}_{n+1}$$

of the iteration.

3. Newton's Method - Speed of Convergence: Where applicable Newton's method converges much faster towards the local maximum or minimum than gradient descent. In fact every local minimum has a neighborhood $\mathbb{N}$ such that if one starts with

$$x_0 \in \mathbb{N}$$

Newton's method with step size

$$\gamma = 1$$

converges quadratically. The only requirements are that the Hessian be invertible, and it be a Lipschitz-continuous function of $\vec{x}$ in that neighborhood.

4. By-passing Explicit Hessian Computation: Finding the inverse of the Hessian in high dimensions can be an expensive exercise. In such cases, instead of directly inverting the Hessian it may be better to calculate the vector

$$\Delta\vec{x}_n = \vec{x}_{n+1} - \vec{x}_n$$

as a solution to the system of linear equations

$$[\mathbb{H}f(\vec{x}_n)]\vec{\nabla}f(\vec{x}_{n+1}) = -\vec{\nabla}f(\vec{x}_n)$$

This may be solved by various factorizations or approximately – and to great accuracy – using iterative methods.

5. Caveats of the Matrix Methods: Many of these methods are only applicable to certain types of equations – e.g., the Cholesky factorization and the conjugate gradient method will work only if $\mathbb{H}f(\vec{x}_n)$ is a positive definite matrix. While this may seem like a limitation, it is often a useful indicator of something gone wrong. For instance if a maximization problem is being considered and $\mathbb{H}f(\vec{x}_n)$ is not positive definite, the iterations end up converging to a saddle point and not to a minimum.
6. Stable Variants of these Methods: On the other hand if constrained optimization is being considered – for example using Lagrange multipliers – the problem may become one of saddle point finding, in which case the Hessian will be symmetric indefinite and a solution for $\vec{x}_{n+1}$ needs to be done with approaches that will work for such situations, such as the LDL^T variant of the Cholesky factorization or the conjugate gradient method.

7. Techniques of Quasi-Newton Methods: There are exist various quasi-Newton methods where the approximation for the Hessian – or its direct inverse – is built up from the changes in the gradient.
8. Challenges with non-invertible Hessians: If a Hessian is close to a non-invertible matrix the inverted Hessian can be numerically unstable and the solution may diverge. In this case certain workarounds have been tried in the past, each of which have had varied success in differing setups.
9. Converting Hessian to Positive Definite: For example one can modify the Hessian by adding a correction matrix B_n so as to make $\mathbb{H}f(\vec{x}_n) + B_n$ positive definite. On approach is to diagonalize $\mathbb{H}f(\vec{x}_n)$ and choose B_n so that $\mathbb{H}f(\vec{x}_n) + B_n$ has the same eigenvectors as $\mathbb{H}f(\vec{x}_n)$ but with each negative eigenvalue replaced by

$$\epsilon > 0$$

10. Approach used by Levenberg-Marquardt: An approach exploited by the Levenberg-Marquardt algorithm – which uses an approximate Hessian – is to add a scaled identity matrix $\mu\mathbb{I}$ to the Hessian, with the scale adjusted at every iteration as needed.
11. Comparison with Gradient Descent Approach: For large μ and small Hessian, the iterations behave like gradient descent with a step size $\frac{1}{\mu}$. This results in slower but more reliable convergence when the Hessian doesn't provide useful information.

Wolf Conditions

1. Motivation, Rationale, and Primary Purpose: In the unconstrained minimization problem setting, the **Wolf Conditions** are a set of inequalities for performing *inexact* line search, especially in quasi-Newton methods (Wolf (1969, 1971)).
2. Problem Setup - Function Extremization: In these methods the idea is to find

$$\min_{\vec{x}} f(\vec{x})$$

for some smooth

$$f: \mathbb{R}^n \longrightarrow \mathbb{R}$$

Each step often involves approximately solving the sub-problem

$$\min_{\alpha} f(\vec{x}_k + \alpha \vec{p}_k)$$

where $\vec{x}_k$ is the current best guess

$$\vec{p}_k \in \mathbb{R}^n$$

is a search direction, and

$$\alpha \in \mathbb{R}$$

is the step length.

3. Application of the Wolfe Conditions: The inexact line search provides an efficient way of computing an acceptable step length α that reduces the objective function sufficiently rather than minimizing the objective function over

$$\alpha \in \mathbb{R}^+$$

exactly. A line search algorithm can use the Wolfe conditions as a requirement for any guessed α before finding a new search direction $\vec{p}_k$.

Armijo Rule and Curvature Condition

1. The Inequalities of Wolfe Condition: A step length α_k is said to satisfy the *Wolfe conditions*, restricted to the direction $\vec{p}_k$, if the following inequalities hold:

$$f(\vec{x}_k + \alpha_k \vec{p}_k) \leq f(\vec{x}_k) + c_1 \alpha_k \vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)$$

$$\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k + \alpha_k \vec{p}_k) \geq c_2 \vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)$$

with

$$0 < c_1 < c_2 < 1$$

In examining the second condition once recalls that to ensure $\vec{p}_k$ is a descent direction one has

$$\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k) < 0$$

2. Choices for c_1 and c_2: c_1 is usually chosen quite small while c_2 is much larger. Nocedal and Wright (2006) provide example values of

$$c_1 = 10^{-4}$$

and

$$c_2 = 0.9$$

for Newton or quasi-Newton methods, and

$$c_2 = 0.1$$

for the non-linear conjugate gradient method. The first inequality is known as the **Armijo Rule** (Armijo (1966)) and ensures that the step length α_k decreases f sufficiently. The second inequality is called the **Curvature Condition**; it ensures that the slope has reduced sufficiently.

3. Strong Wolfe Conditions on Curvature: Denoting by ϕ a univariate function restricted to the direction $\vec{p}_k$ as

$$\phi(\alpha) = f(\vec{x}_k + \alpha_k \vec{p}_k)$$

there are situations where the Wolfe conditions can result in a value for the step length that is not close to a minimizer of ϕ. If one modifies the curvature condition to

$$\left|\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k + \alpha_k \vec{p}_k)\right| \leq c_2 \left|\vec{p}_k \cdot \vec{\nabla} f(\vec{x}_k)\right|$$

then the curvature condition along with the Armijo rule form the so-called **strong Wolfe conditions**, and force α_k to lie close to a critical point of ϕ.

Rationale for the Wolfe Conditions

1. Convergence of the Gradient to Zero: The principal reason for imposing the Wolfe conditions in an optimization algorithm where

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \vec{p}_k$$

is to ensure the convergence of the gradient to zero. In particular if the cosine of the angle between $\vec{p}_k$ and the gradient

$$\cos\theta = \frac{\vec{\nabla} f(\vec{x}_k) \cdot \vec{p}_k}{\left\|\vec{\nabla} f(\vec{x}_k)\right\| \|\vec{p}_k\|}$$

is bounded away from zero, and Armijo rule and curvature conditions (modified) hold,

$$\vec{\nabla} f(\vec{x}_k) \longrightarrow 0$$

2. Positive Definiteness of Update Matrix: An additional motivation, in the case of a quasi-Newton method, is that if

$$\vec{p}_k = {B_k}^{-1} \vec{\nabla} f(\vec{x}_k)$$

where the matrix B_k is updated by the BFGS or the DFP formula, then if B_k is positive, the (modified) curvature condition implies that B_{k+1} is also positive definite.

References

- Armijo, L. (1966): Minimization of Functions having Lipschitz Continuous First Derivatives *Pacific Journal of Mathematics* **16 (1)** 1-3.
- Newton's Method in Optimization (Wiki).
- Nocedal, J., and S. Wright (2006): *Numerical Optimization 2nd Edition* **Springer**.
- Wolfe, P. (1969): Convergence Conditions for Ascent Methods *SIAM Review* **11 (2)** 226-235.
- Wolfe, P. (1971): Convergence Conditions for Ascent Methods II: Some Corrections *SIAM Review* **13 (2)** 185-188.
- Wolfe Conditions (Wiki).

Constrained Optimization

Constrained Optimization – Definition and Description

1. Constrained Optimization Definition – Variable Constraints: In mathematical optimization **constrained optimization** is the process of optimizing an objective function with respect to some variables in the presence of constraints on those variables.
2. Constrained Optimization Definition - Objective Function: The objective function is either a cost function or an energy function that is to be minimized, or a reward function or a utility function that has to be maximized.
3. Hard vs Soft Variable Constraints: Constraints can either be *hard constraints* that set conditions on variables required to be satisfied, or *soft constraints* that have some variable values that are penalized in the objective function if – and based on the extent that – the conditions on the variables are not satisfied (Constrained Optimization (Wiki)).

General Form

1. Constrained Optimization Problem – Mathematical Specification: A general constrained minimization problem may be written as follows:

$$\min_{x} f(x)$$

subject to

$$g_i(x) = c_i$$

for

$$i = 1, \cdots, n$$

equality constraints and

$$h_j(x) \geq d_j$$

for

$$j = 1, \cdots, m$$

inequality constraints where

$$g_i(x) = c_i$$

for

$$i = 1, \cdots, n$$

and

$$h_j(x) \geq d_j$$

for

$$j = 1, \cdots, m$$

are the constraints to be satisfied; these are called the hard constraints.

2. Constrained Optimizers - Soft Objective Function: In some problems, often called *constraint optimization problems*, the objective function is a sum of the cost functions, each of which penalized the extent if any to which a soft constraint – a constraint that is preferred but not required to be satisfied - is violated.

Solution Methods

1. Adaptation from Unconstrained Algorithms: Many unconstrained optimization algorithms can be adapted to the constrained case, often via the use of a penalty method. However, the search steps taken by the unconstrained method may be unacceptable for the constrained problem, leading to a lack of convergence. This is referred to as the Maratos effect (Sun and Yua (2010)).
2. Equality Constraints Lagrange Multiplier Technique: If the constrained problem has only equality constraints the method of Lagrange multipliers can be used to convert it to an unconstrained problem whose number of variables is the original number of variables plus the number of constraints.
3. Equality Constraints - Cross Variable Substitution: Alternatively if the constraints are all equality constraints and are all linear they can be solved for some of the variables in terms of the others, and the former can be substituted out of the objective function, leaving an unconstrained problem in a smaller number of variables.
4. Inequality Constraints - Conditions on Variables: With inequality constraints the problem can be characterized in terms of Geometric Optimality Conditions, Fritz-John Conditions, and Karush-Kuhn-Tucker Conditions in which simple problems may be directly solvable.
5. Linear Programming – Simplex/Interior Point: If the objective function and all of the hard constraints are linear, then the problem is a linear programming one. This can be solved by the Simplex method, which usually works in polynomial time in the

problem size but is not guaranteed to, or by interior point methods, which are guaranteed to work in polynomial time.

6. Quadratic Programming - Objective Function Constraints: If all the hard constraints are linear but the objective function is quadratic the problem is a quadratic programming problem.
7. Quadratic Programming Convex Objective Function: Quadratic Programming problems can be solved by the ellipsoid method in polynomial time if the objective function is convex; otherwise the problem is NP hard.

Constraint Optimization: Branch and Bound

1. Idea behind Branch and Bound: Constraint optimization can be solved by branch-and-bound algorithms. These are back-tracking algorithms that store the cost of the best solution found during execution and use it eventually to avoid part of the search.
2. Back Tracking vs. Solution Extension: More precisely whenever the algorithm encounters a partial solution that cannot be extended to form a solution with better cost than the best stored cost, the algorithm back tracks instead of trying to extend this solution.
3. Efficient of Back Tracking Algorithms: Assuming that the cost is to be minimized the efficiency of these algorithms depends on how the cost can be obtained from extending a partial solution is evaluated. Indeed if the algorithms can back track from a partial solution, part of the search can be skipped. The lower the estimated cost the better the algorithm, as a lower estimated cost is more likely to be lower than the best cost of the solution found so far.
4. Algorithm Lower and Upper Bounds: On the other hand this estimated cost cannot be lower than the effective cost obtained by extending the solution, as otherwise the algorithm could backtrack while a solution better than the best found so far exists. As a result the algorithm requires an upper bound on the cost that can be obtained by extending a partial solution and this upper bound should be as small as possible.

5. Hansen's Branch-and-Bound Variation: A variation of the above method called Hansen's method uses interval methods (Leader (2004)). It inherently implements rectangular constraints.

Branch-and-Bound: First-Choice Bounding Conditions

1. Separately treating each Soft Constraint: One way of evaluating this upper bound for a partial solution is to consider each soft constraint separately. For each soft constraint the maximal possible value for any assignment to the unassigned variables is assumed. The sum of these variables is an upper bound because the soft constraints cannot assume a higher value.
2. Exact Nature of Upper Bound: It is exact because the maximal nature of soft constraints may derive from different evaluations: a soft constraint may be maximal for

$$x = a$$

while another soft constraint is maximal for

$$x = b$$

Branch-and-Bound Russian Doll Search

1. Branch-and-Bound Sub-problems: This method runs a branch-and-bound algorithm on n problems where n is the number of variables (Verfaillie, Lemaitre, and Schiex (1996)). Each such sub-problem is the sub-problem obtained by dropping the sequence $x_1, \cdots, x_i$ from the original problem along with the constraints containing them.

2. Sub-problem Cost Upper Bound: After the problem on variables $x_{i+1}, \cdots, x_n$ is solved its optimal cost can be used as an upper bound while solving the other problems.
3. Assigned/Unassigned Variables Sub-problem: In particular the cost of estimating a solution having $x_{i+1}, \cdots, x_n$ is added to the cost that derives from the evaluated variables. Virtually this corresponds to ignoring the evaluate variables and solving the problem on the unassigned ones, except that the latter problem has been already solved.
4. Sub-problem Total Cost Update: More precisely the cost of the constraints containing both the assigned and the unassigned variables is estimated as above, or using another arbitrary method; the cost of soft constraints using unassigned variables is instead estimated using the optimal solution of the corresponding problem, which is already known at this point.
5. Similarities with Dynamic Programming Approach: Like Dynamic Programming Russian Doll Search solves the sub-problems in order to solve the whole problem.
6. Differences with Dynamic Programming Approach: However, whereas Dynamic Programming directly combines the results obtained on the sub-problems to get the result of the whole program, the Russian Doll Search only uses them as bounds during the search.

Branch-and-Bound – Bucket Elimination

1. Elimination of a Specified Variable: The bucket elimination algorithm can be used for constraint optimization. A given variable can be removed from the problem by replacing all the soft constraints containing it with a new soft constraint.
2. Removed Variable Constraint-Cost Expression: The cost of the constraint expression is estimated assuming the maximal objective function value for each of the removed variable. Formally if x is the variable to be removed, $C_1, \cdots, C_n$ are the constraints containing it, and $y_1, \cdots, y_m$ are the variables containing x, the new soft constraint is defined by

$$C(y_1 = a_1, \cdots, y_n = a_n) = \max_a \sum_i C_i(x = a, y_1 = a_1, \cdots, y_n = a_n)$$

3. Bucket of all Variable Constraints: Bucket elimination works with an arbitrary ordering of the variables. Every variable is associated with a bucket of constraints; the bucket of a variable contains all the constraints having the variable has the highest in the order.
4. Order of the Bucket Elimination: Bucket elimination proceeds from the last variable to the first. For each variable, all constraints of the bucket are replaced as above to remove the variable. The resulting constraint is then placed in the appropriate bucket.

References

- Constrained Optimization (Wiki).
- Leader, J. J. (2004): *Numerical Analysis and Scientific Computation* **Addison Wesley**.
- Sun, W., and Y. X. Yua (2010): *Optimization Theory and Methods: Non-linear Programming* **Springer**.
- Verfaillie, G., M. Lemaitre, and T. Schiex (1996): Russian Doll Search for solving Constraint Optimization Problems *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference* **Portland** 181-187.

Lagrange Multipliers

Motivation, Definition, and Problem Formulation

1. Purpose behind the Lagrange Methodology: In mathematical optimization the **method of Lagrange multipliers** (Lagrange (1811)) is a strategy for finding the local maxima and minima subject to equality constraints (Lagrange Multiplier (Wiki)).
2. Optimization Problem Mathematical Setup: Consider the optimization problem of maximizing $f(x, y)$ subject to

$$g(x, y) = 0$$

   A new variable λ is introduced (this is called the Lagrange multiplier) and the Lagrange function (or Lagrangian) defined by

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \lambda g(x, y)$$

   is studied. Here the term λ may be either added or subtracted.
3. Lagrange Multipliers as Stationary Points: If $f(x_0, y_0)$ is the maximum of $f(x, y)$ for the original constrained problem, then there exists a λ_0 such that (x_0, y_0, λ_0) is a stationary point for the Lagrangian function (stationary points are those where the partial derivatives of $\mathcal{L}$ are zero).
4. Necessary Conditions for Constrained Optimality: However not all stationary points yield a solution to the original problem. Thus, the method of Lagrange multipliers yields necessary conditions for optimality in constrained problems (Hiriart-Urruty and Lemarechal (1993), Bertsekas (1999), Vapnyarskii (2001), Lemarechal (2001), Lasdon (2002)).

5. Sufficient Conditions for Constrained Optimality: Sufficient conditions for a maximum or a minimum also exist. Sufficient conditions for a constrained local maximum or a minimum can be stated in terms of a sequence of principal minors, i.e., determinants of the upper-left-justified sub-matrices, of the bordered Hessian matrix of the second derivatives of the Lagrangian expression (Chiang (1984)).

Introduction, Background, and Overview

1. Advantage of the Lagrange Multiplier Formulation: One of the most common problems in calculus is that of finding the extrema of a function, but it is often difficult to find a closed form for the function being extremized. Such difficulties often arise when one wishes to extremize the function subject to fixed outside constraints or conditions. The method of Lagrange multipliers is a powerful tool for solving this class of problems without the need to explicitly solve for the conditions and use them to eliminate the extra variables.
2. Intuition behind the Lagrange Multiplier Methodology: Consider the 2D problem introduced above: maximize $f(x, y)$ subject to

$$g(x, y) = 0$$

The method of Lagrange multipliers relies on the intuition that at the maximum $f(x, y)$ cannot be increasing in the direction of any neighboring point where

$$g(x, y) = 0$$

If it did, one could walk along

$$g(x, y) = 0$$

to get higher, meaning that the starting point wasn't actually the maximum.

3. Function Realization/Constraint Value Contours: The contours of f given by

$$f(x,y) = d$$

for various values of d as well as the contours of g given by

$$g(x,y) = 0$$

may be visualized as being parallel/tangential to each other.

4. Invariance of $f(x,y)$ along the Constraint: Suppose one walks along the contour with

$$g(x,y) = 0$$

One is interested in funding points along $f(x,y)$ that do not change along the walk since these points may be maxima. There are two ways this can happen. First one could be walking along the contour line of $f(x,y)$ since by definition $f(x,y)$ does not change along the contour line. This would mean that the contour lines of $f(x,y)$ and $g(x,y)$ are parallel here. The second possibility is that one has reached a "level" part of $f(x,y)$ meaning that $f(x,y)$ cannot change in any direction.

5. Constraint/Objective Function Gradient Scaling: To check for the first possibility, one notices that the gradient of a function is perpendicular to its contour lines, and the contour lines of $f(x,y)$ and $g(x,y)$ are parallel if and only if the gradients of $f(x,y)$ and $g(x,y)$ are parallel. Thus one wants points (x,y) where

$$g(x,y) = 0$$

and

$$\vec{\nabla}_{x,y} f(x,y) = -\lambda \vec{\nabla}_{x,y} g(x,y)$$

for some λ where

$$\vec{\nabla}_{x,y} f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

$$\vec{\nabla}_{x,y} g(x,y) = \left[\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right]$$

are the respective gradients. The constant λ is required because although the two gradient vectors are parallel, the magnitudes of the gradient vectors are generally not equal. This constant is called the Lagrange multiplier (the negative sign is a convention).

6. The "Level" Plateau of f: This method also addresses the second possibility; if f is level then its gradient is zero, and setting

$$\lambda = 0$$

is a solution regardless of g.

7. Formulation of the Lagrangian Function: To incorporate these conditions into one equation one introduces an auxiliary function

$$\mathcal{L}(x,y,\lambda) = f(x,y) - \lambda g(x,y)$$

and solves for

$$\vec{\nabla}_{x,y,\lambda} \mathcal{L}(x,y,\lambda) = 0$$

This is the method of Lagrange multipliers. Note that

$$\vec{\nabla}_{\lambda}\mathcal{L}(x,y,\lambda)=0$$

implies

$$g(x,y)=0$$

To summarize

$$\vec{\nabla}_{x,y,\lambda}\mathcal{L}(x,y,\lambda)=0$$

results in

$$\vec{\nabla}_{x,y}f(x,y)=-\lambda\vec{\nabla}_{x,y}g(x,y)$$

and

$$g(x,y)=0$$

The constrained extrema of f are the *critical points* but they are not necessarily the *local extrema* of $\mathcal{L}$.

8. Alternate Formulations for the Lagrangian: One may re-formulate the Lagrangian as a Hamiltonian, in which case the solutions are the local minima for the Hamiltonian. This is done in optimal control theory in the form of Pontryagin's minimum principle.
9. Non-Extremum Solutions to Lagrangian: The fact that the solutions to the Lagrangian are not necessarily the extrema also poses difficulties for numerical optimization. This can be addressed by computing the *magnitude* of the gradient, as the zeroes of the magnitude are necessarily the local minima, as illustrated later in the part on numerical optimization.

Handling Multiple Constraints

1. Notation Used for Multiple Constraints: Throughout this section the independent variables are denoted $x_1, \cdots, x_N$ and, as a group denoted as

$$p = (x_1, \cdots, x_N)$$

   Also the function being analyzed will be denoted $f(p)$ and the constraints will be represented by the equations

$$g_1(p) = 0$$
$$\vdots$$
$$g_M(p) = 0$$

2. Basic Idea behind the Constraints: The basic idea remains essentially the same: if we consider only satisfy the constraints (i.e., are *in* the constraints) then a point $\big(p, f(p)\big)$ is a stationary point (i.e., a point in a *flat* region) of f if and only if the constraints at that point do not allow movement in a direction where f changes value. Once the stationary points are located further tests are needed to see if it is a minimum, a maximum, or just a stationary point that is neither.
3. Multi-dimensional Invariance of f: Consider the level set of f at $\big(p, f(p)\big)$. Let the set of vectors $\{v_L\}$ contain the directions in which one can move and still remain in the same level set, the directions where the value of f does not change (i.e., the change equals zero). For every vector v in $\{v_L\}$ the following relation must hold:

$$\frac{\partial f}{\partial x_1} v_1 + \cdots + \frac{\partial f}{\partial x_N} v_N = 0$$

   where the notation v_k above refers to the k^{th} component of the vector v.

4. Invocation of the Gradient of f: The equation above can be re-written in a more compact geometric form that helps the intuition:

$$\left[\frac{\partial f}{\partial x_1} \quad \cdots \quad \frac{\partial f}{\partial x_N}\right] \cdot \begin{bmatrix} v_1 \\ \vdots \\ v_N \end{bmatrix} = 0$$

i.e.

$$\vec{\nabla} f^T \cdot \vec{v} = 0$$

This makes it clear that if one is at p then *all* directions from this point that do *not* change the value of f *must be perpendicular to* $\vec{\nabla} f(p)$ (the gradient of f at p).

5. Usage of the Constraint Gradient: Each constraint limits the directions that one can move from a particular point and still satisfy the constraint. One uses the same set of procedures above to look for a set of vectors $\{v_C\}$ that contain the directions in which can move and still satisfy the constraint. As above, for every vector v in $\{v_C\}$ the following relation must hold:

$$\frac{\partial g}{\partial x_1} v_1 + \cdots + \frac{\partial g}{\partial x_N} v_N = 0$$

i.e.

$$\vec{\nabla} g^T \cdot \vec{v} = 0$$

From this it can be seen that at point p all directions from this point that will satisfy the constraint must be proportional to $\vec{\nabla} g(p)$.

6. Lagrange Multiplier Method Formal Definition: A point on f is a constrained stationary point if and only if the direction that changes f violates at least one of the constraints, i.e., it has no "component" in the legal space perpendicular to $\vec{\nabla} g(p)$.

Mathematically this means that the gradient of f at this constrained stationary point is perpendicular to the space spanned by the set of vectors $\{v_C\}$ which in turn is perpendicular to the gradients of the constraints g.

7. Single Constraint f/g Gradients: For a single constraint the above statement says that at stationary points the direction that changes f is the same as that violates the constraint. To determine if two vectors are in the same direction, note that if two vectors start from the same point then open vector can always reach the other by changing its length and/or flipping or the opposite way along the same direction line. This requires that

$$\vec{\nabla} f(p) = \lambda \vec{\nabla} g(p)$$

implying that

$$\vec{\nabla} f(p) - \lambda \vec{\nabla} g(p) = 0$$

8. Single Constraint Case - Compact Formulation: On adding another simultaneous equation to guarantee that this test is performed only at the point that satisfies the constraint two simultaneous equations result, which when solved identify all the constrained stationary points.

$$g(p) = 0$$

implies that p satisfies the constraint, thus

$$\vec{\nabla} f(p) - \lambda \vec{\nabla} g(p) = 0$$

is a stationary point.

9. Single Constraint Case Expanded Formulation: Fully expanded there are simultaneous equations that need to be solved for $N + 1$ variables, which are $x_1, \cdots, x_N$ and λ:

$$g(x_1, \cdots, x_N) = 0$$

$$\frac{\partial f(x_1, \cdots, x_N)}{\partial x_1} - \lambda \frac{\partial g(x_1, \cdots, x_N)}{\partial x_1} = 0$$
$$\vdots$$
$$\frac{\partial f(x_1, \cdots, x_N)}{\partial x_N} - \lambda \frac{\partial g(x_1, \cdots, x_N)}{\partial x_N} = 0$$

10. Multiple Constraints and their Gradients: For more than one constraint a similar reasoning applies. Each gradient function g has a space of allowable directions at p – the space of vectors perpendicular to $\vec{\nabla} g(p)$. The set of directions allowed by all constraints is thus the space of directions perpendicular to all of the constraint gradients. Denoting the space of allowable moves by A and the span of gradients by S, using the discussion above

$$A = S^{\perp}$$

the space of vectors perpendicular to every element in S.

11. Multiple Constraints Case Gradient Formulation: If p is an optimum any element not perpendicular to $\vec{\nabla} f(p)$ is not an allowable direction. One can show that this implies

$$\vec{\nabla} f(p) \in A^{\perp} = S$$

Thus there are scalars $\lambda_1, \cdots, \lambda_M$ such that

$$\vec{\nabla} f(p) = \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p)$$

implies

$$\vec{\nabla} f(p) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p) = 0$$

As before the simultaneous equations are added to guarantee that this test is performed only at a point that satisfies every constraint, and the resulting equations, when satisfied, identify all the constrained stationary points.

$$g_1(p) = \cdots = g_M(p) = 0$$

implies that p satisfies all constraints; further

$$\vec{\nabla} f(p) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(p) = 0$$

indicates that p is a stationary point.

12. Multiple Gradient Case Lagrangian Formulation: The method is complete now from the standpoint of finding the stationary points, but these equations can be condensed into a more succinct and elegant form. The equations above look like partial derivatives of a larger scalar function $\mathcal{L}$ that takes all the $x_1, \cdots, x_N$ and all the $\lambda_1, \cdots, \lambda_M$ as inputs. Setting every equation to zero is exactly what one would have to do to solve for the *unconstrained* stationary points of the larger function. Finally the larger function $\mathcal{L}$ with partial derivatives that are exactly the ones that we require can be constructed very simply as

$$\mathcal{L}(x_1, \cdots, x_N, \lambda_1, \cdots, \lambda_M) = f(x_1, \cdots, x_N) - \sum_{k=1}^{M} \lambda_k \vec{\nabla} g_k(x_1, \cdots, x_N)$$

Solving the above equation for its *unconstrained* stationary points generates exactly the same stationary points as solving for the *constrained* stationary points of f under the constraints $g_1, \cdots, g_M$

13. The Method of Lagrange Multipliers: In Lagrange's honor the above function is called a *Lagrangian*, the scalar multipliers $\lambda_1, \cdots, \lambda_M$ are called *Lagrange Multipliers*, and the method itself is called *The Method of Lagrange Multipliers*.
14. The Karush-Kuhn-Tucker Generalization: The method of Lagrange multipliers is generalized by the Karush-Kuhn-Tucker conditions, which also takes into account inequality constraints of the form

$$h(\vec{x}) \leq c$$

Modern Formulation via Differentiable Manifolds

1. Local Extrema of $\mathbb{R}^d \to \mathbb{R}^1$: Funding the local maxima of a function

$$f: \mathbb{U} \to \mathbb{R}$$

where $\mathbb{U}$ is an open subset of $\mathbb{R}^d$ is done by finding all points $x \in \mathbb{U}$ such that

$$\mathbb{D}_x f = 0$$

and then checking whether all of the eigenvalues of the Hessian $\mathbb{H}_x f$ are negative. Setting

$$\mathbb{D}_x f = 0$$

is a non-linear problem and in general arbitrarily difficult. After funding the critical points checking for the eigenvalues is a linear problem and thus easy.

2. Extrema under the Level-Set Constraint: When

$$g: \mathbb{R}^d \to \mathbb{R}^1$$

is a smooth function such that

$$\mathbb{D}_x g \neq 0$$

for all x in the level set

$$g(x) = c$$

then $g^{-1}(c)$ becomes an $n-1$ dimensional smooth manifold $\mathbb{M}$ by the level set theorem. Funding local maxima is by definition a local problem, so it can be done on the local charts of $\mathbb{M}$ after funding a diffeomorphism

$$\varphi: \mathbb{V} \to \mathbb{R}^{n-1}$$

from an open subset of

$$\mathbb{V} \subset \mathbb{M}$$

onto an open subset

$$\mathbb{U} \subset \mathbb{R}^{n-1}$$

and thus one can apply the algorithm in the previous part to the function

$$f' = f \circ \varphi^{-1} : \mathbb{U} \to \mathbb{R}$$

3. Approach of Lagrange Multiplier Method: While the above idea sounds good it is difficult to compute φ^{-1} in practice. *The entire method of Lagrange multipliers reduces to skipping that step and finding the zeroes of* $\mathbb{D}_x f'$ *directly.* It follows from the construction in the level set theorem that $\mathbb{D}_x \varphi^{-1}$ is the inclusion map

$$\ker \mathbb{D}_{\varphi^{-1}(x)} g \subseteq \mathbb{R}^n$$

Therefore

$$0 = \mathbb{D}_x f' = \mathbb{D}_x (f \circ \varphi^{-1}) = \mathbb{D}_{\varphi^{-1}(x)} f \circ \mathbb{D}_x \varphi^{-1}$$

if and only if

$$\ker \mathbb{D}_y g \subseteq \ker \mathbb{D}_y f$$

from writing y for $\varphi^{-1}(x)$

4. Existence of the Linear Map: By the first isomorphism theorem this is true if and only if there exists a linear map

$$\mathcal{L} : \mathbb{R} \to \mathbb{R}$$

such that

$$\mathcal{L} \circ \mathbb{D}_y g = \mathbb{D}_y f$$

As a linear map one must have that

$$\mathcal{L}(x) = \lambda x$$

for a fixed

$$\lambda \in \mathbb{R}$$

Therefore funding the critical point of f' is equivalent to solving the system of equations

$$\lambda \mathbb{D}_y g = \mathbb{D}_y f$$

$$g(y) = c$$

in the variables

$$y \in \mathbb{R}^{n-1}$$

and

$$\lambda \in \mathbb{R}$$

This is in general a non-linear system of n equations and n unknowns.

5. Multiple Constraints Surjective Linear Map: In the case of general constraints one works with

$$g: \mathbb{R}^n \to \mathbb{R}^m$$

and replaces the condition

$$\mathbb{D}_x g \neq 0$$

for all

$$x \in g^{-1}(c)$$

with the requirement that $\mathbb{D}_x g$ be surjective at all such points. In this case $\mathcal{L}$ will be a linear map $\mathbb{R}^m \to \mathbb{R}$ i.e., a row vector with m entries.

Interpretation of the Lagrange Multipliers

1. As a Rate of Change Quantity: Often the Lagrange multipliers have an interpretation as some quantity of interest. For example if the Lagrangian expression is

$$\begin{aligned}\mathcal{L}(x_1, \cdots, x_N, \lambda_1, \cdots, \lambda_M) \\ &= f(x_1, \cdots, x_N) + \lambda_1[c_1 - g_1(x_1, \cdots, x_N)] + \cdots \\ &+ \lambda_M[c_M - g(x_1, \cdots, x_N)]\end{aligned}$$

then

$$\frac{\partial \mathcal{L}}{\partial c_k} = \lambda_k$$

So λ_k is the rate of change of the quantity being optimized as a function of the constraint variable.

2. Examples of Lagrange Multiplier Roles: As examples, in Lagrangian mechanics the equations of motion are derived by finding stationary points of action, i.e., the time integral of the difference between the potential and the kinetic energies. Thus the force on a particle due to a scalar potential

$$\vec{F} = -\vec{\nabla}V$$

can be interpreted as a Lagrange multiplier determining the change in action – transfer of potential to kinetic energy – following a variation in the particle's constrained trajectory. In control theory this formulated instead as co-state equations.

3. Marginal Effect of the Constraint: Moreover by the control theorem the optimal value of a Lagrange multiplier has an interpretation as the marginal effect of the corresponding constraint constant upon on the optimal attainable value of the original objective function. Denoting the optimum with an asterisk it can be shown that

$$\frac{\partial f\left(x_1{}^*(c_1, \cdots, c_M), \cdots, x_N{}^*(c_1, \cdots, c_M)\right)}{\partial c_k} = \lambda_k{}^*$$

4. Lagrange Multiplier Usage in Economics: For example in economics the optimal profit to a player is calculated subject to a constrained space of actions, where the Lagrange multiplier is the change in the optimal value of the objective function (profit) due to the relaxation of a given constraint – e.g., through a change in the income. In such a context λ^* is the marginal cost of the constraint, and is referred to as the shadow price.

Lagrange Application: Maximal Information Entropy

1. Information Entropy Maximization – Problem Setup: Suppose on wishes to find the discrete probability distribution on the points $\{p_1, \cdots, p_n\}$ with maximal information entropy. This is the same as saying that one wishes to find the least biased probability on the points $\{p_1, \cdots, p_n\}$. In other words one wishes to maximize the Shannon entropy equation

$$f(p_1,\cdots,p_n) = -\sum_{j=1}^{n} p_j \log p_j$$

2. Cumulative Discrete Probability Normalization Constraint: For this to be a probability distribution the sum of the probabilities p_j at each point x_i must equal 1, so the constraint becomes

$$g(p_1,\cdots,p_n) = \sum_{j=1}^{n} p_j \log p_j$$

3. Application of the Lagrange Multipliers: Using Lagrange multipliers to find the point of maximum entropy $\vec{p}^*$ across all discrete probability distributions $\vec{p}$ on $\{x_1,\cdots,x_n\}$ requires that

$$\frac{\partial}{\partial \vec{p}}[f + \lambda(g-1)]\Big|_{\vec{p}=\vec{p}^*} = 0$$

which gives a system of n equations with indices

$$k = 1,\cdots,n$$

such that

$$\frac{\partial}{\partial p_k}\left[-\sum_{j=1}^{n} p_j \log p_j + \lambda\left(\sum_{j=1}^{n} p_j - 1\right)\right]\Bigg|_{\vec{p}=\vec{p}^*} = 0$$

4. Solution to the Discrete Probability: Carrying out the differentiation on these n equations one gets

$$-[1 + \log p_k{}^*] + \lambda = 0$$

This shows that all $p_k{}^*$ are the same because they depend only on λ. By using the constraint

$$\sum_{j=1}^{n} p_j = 1$$

one finds that

$$p_k{}^* = \frac{1}{n}$$

Hence the uniform distribution is the distribution with the greatest entropy, using distributions on n points.

Lagrange Application: Numerical Optimization Techniques

1. Local Minima vs. Saddle Points: The critical points of the Lagrangian occur at saddle points rather than the local minima or maxima (Heath (2005)). Unfortunately many numerical optimization techniques such as hill climbing, gradient descent, some of the quasi-Newton methods, among others, are designed to find local minima (or maxima) and not the saddle points.
2. Conversion to an Extremization Problem: For this reason one must modify the formulation to ensure that this is a minimization problem – for example by extremizing the square of the gradient of the Lagrangian – or use an optimization technique that finds stationary points and not necessarily extrema, e.g., such as Newton's method without an extremum seeking line search.

3. "Saddle" Critical Points - An Example: As a simple example consider the problem of finding the value of x that minimizes the function

$$f(x) = x^2$$

constrained such that

$$x^2 = 1$$

Clearly this problem is pathological because there are only two values that satisfy the constraint, but it is useful for illustration purposes because the corresponding unconstrained function can be visualized in three dimensions.

4. Application of the Lagrange Multipliers: Using Lagrange multipliers this problem can be converted into an unconstrained optimization problem

$$\mathcal{L}(x, \lambda) = x^2 + \lambda(x^2 - 1)$$

The two critical points occur at the saddle points where

$$x = +1$$

and

$$x = -1$$

5. Transformation to Local Extremum Problem: In order to solve this problem with a numerical optimization technique one must first transform this problem such that the critical points occur at the local minima. This is done by computing the magnitude of the gradient of the unconstrained optimization problem.

6. Objective Variate/Constraint Multiplier Jacobian: First one computes the partial derivative of the unconstrained problem with respect to each variable;

$$\frac{\partial \mathcal{L}}{\partial x} = 2x + 2\lambda x$$

$$\frac{\partial \mathcal{L}}{\partial \lambda} = x^2 - 1$$

If the target function is not easily differentiable the differential with respect to each variable can be approximated using the divided differences technique, i.e.

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial x} \approx \frac{\mathcal{L}(x+\epsilon,\lambda) - \mathcal{L}(x-\epsilon,\lambda)}{2\epsilon}$$

$$\frac{\partial \mathcal{L}(x,\lambda)}{\partial \lambda} \approx \frac{\mathcal{L}(x,\lambda+\epsilon) - \mathcal{L}(x,\lambda-\epsilon)}{2\epsilon}$$

where ϵ is a small value.

7. Computing the Magnitude of the Gradient: Next one computes the magnitude of the gradient, which is the square root of the sum of squares of the partial derivatives:

$$h(x,\lambda) = \sqrt{(2x + 2\lambda x)^2 + (x^2 - 1)^2}$$

$$\approx \sqrt{\left[\frac{\mathcal{L}(x+\epsilon,\lambda) - \mathcal{L}(x-\epsilon,\lambda)}{2\epsilon}\right]^2 + \left[\frac{\mathcal{L}(x,\lambda+\epsilon) - \mathcal{L}(x,\lambda-\epsilon)}{2\epsilon}\right]^2}$$

Since the magnitude is always non-negative optimizing over the squared magnitude is equivalent to optimizing over the magnitude. Thus the square root may be omitted from these equations with no expected differences in the results of the optimization.

8. Critical Points as Local Extrema: The critical points of h occur at

$$x = +1$$

and

$$x = -1$$

just as in $\mathcal{L}$. However the critical points in h occur at local minima, so the numerical optimization techniques can be used to find them.

Lagrange Multipliers – Common Practice Applications

1. Lagrange Multipliers Applied in Economics: Constrained optimization plays a central role in economics. For example, the choice problem for a customer is represented as one of maximizing a utility function subject to a budget constraint. The Lagrange multiplier has an interpretation as the shadow price associated with that constraint – in this instance the marginal utility of the income. Other examples include profit maximization for a firm, along with various macro-economic applications.
2. Lagrange Multipliers in Control Theory: In optimal control theory the Lagrange multipliers are represented as co-state variables and are re-formulated as the minimization of the Hamiltonian, e.g., they are the basis behind the Pontryagin's minimum principle.
3. Lagrange Multipliers in Non-linear Programming: The Lagrange multiplier method has several generalizations. In non-linear programming there are several multiplier rules, e.g., the Caratheodery-John Multiplier Rule and the Convex Multiplier Rule for inequality constraints (Pourciau (1980)).

References

- Bertsekas, D. P. (1999): *Nonlinear Programming 2nd Edition* **Athena Scientific** Cambridge MA.
- Chiang, A. C. (1984): *Fundamental Methods of Mathematical Economics 3rd Edition* **McGraw-Hill**.
- Heath, M. T. (2005): *Scientific Computing – An Introductory Survey* **McGraw-Hill**.
- Hiriart-Urruty, J. B., and C. Lemarechal (1993): XII Abstract Duality for Practitioners, in: *Convex Analysis and Minimization Algorithms, Volume II: Advanced Theory and Bundle Methods* **Springer-Verlag** Berlin.
- Lagrange, J. L. (1811): *Mecanique Analytique*.
- Lagrange Multiplier (Wiki).
- Lasdon, L. S. (2002): *Optimization Theory for Large Systems* **Dover Publications** Mineola NY.
- Lemarechal, C. (1993): Lagrangian Relaxation, in: *Computational Combinatorial Optimization: Papers from the Spring School held in Schloss Dagstuhl (Editors: M. Junger and D. Naddef)* **Springer-Verlag** Berlin.
- Pourciau, B. H. (1980): Modern Multiplier Rules *American Mathematical Monthly* **87 (6)** 433-452.
- Vapnyarskii, I. B. (2001): Lagrange Multiplier, in: *Encyclopedia of Mathematics (editor: M. Hazewinkel)* **Springer**.

Karush-Kuhn-Tucker Conditions

Introduction, Overview, Purpose, and Motivation

1. KKT Conditions - Necessity and Scope: In mathematical optimization the Karush-Kuhn-Tucker (KKT) Conditions – also known as Kuhn-Tucker Conditions – are the first order necessary conditions for a solution in non-linear programming to be optimal provided some regularity conditions are satisfied (Karush-Kuhn-Tucker Conditions (Wiki)).
2. KKT Conditions vs. Lagrange Multipliers: Allowing for inequality constraints the KKT approach to non-linear programming generalizes the method of Lagrange multipliers, which only allows for equality constraints.
3. Mathematical Foundation behind Optimization Algorithms: The system of equations corresponding to the KKT conditions is usually not solved directly except in a few special cases where a closed-form solution may be derived analytically. In general many optimization algorithms can be interpreted as methods for numerically solving the KKT systems of equations (Boyd and van den Berghe (2009)).
4. KKT Conditions - Historical Attribution Revision: The KKT conditions were originally named after Harold W. Kuhn and Albert W. Tucker who first published the conditions in 1951 (Kuhn and Tucker (1951)). Later scholars discovered that then necessary conditions for this problem had been stated by William Karush in his master's thesis (Karush (1939), Kjeldsen (2000)).

Necessary Conditions for Optimization Problems

1. KKT Statement - Non-linear Optimization Problem: Consider the following non-linear optimization problem: Maximize $f(x)$ subject to

$$g_i(x) \leq 0$$

and

$$h_j(x) = 0$$

where x is the optimization variable, f is the *objective* or the *utility* function, $g_i(i = 1, \cdots, m)$ are the inequality constraint functions, and $h_j(j = 1, \cdots, l)$ are the equality constraint functions. The numbers of the inequality and the equality constraints are denoted m and l respectively.

2. KKT Statement - The Necessary Condition: Suppose that the objective function

$$f: \mathbb{R}^n \to \mathbb{R}$$

and the constraint functions

$$g_i: \mathbb{R}^n \to \mathbb{R}$$

and

$$h_j: \mathbb{R}^n \to \mathbb{R}$$

are continuously differentiable at the point x^*. If x^* is a local optimum that satisfies some regularity conditions below, there exist constants $\mu_i(i = 1, \cdots, m)$ and $\lambda_j(j = 1, \cdots, l)$ called KKT multipliers that have the following properties.

3. KKT Statement - Optimal Stationary Conditions: For maximizing $f(x)$

$$\vec{\nabla} f(x^*) = \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

For minimizing $f(x)$

$$-\vec{\nabla} f(x^*) = \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

4. KKT Statement - Primal Feasibility Conditions:

$$g_i(x^*) \leq 0$$

for all

$$i = 1, \cdots, m$$

and

$$h_j(x^*) = 0$$

for all

$$j = 1, \cdots, l$$

5. KKT Statement - Dual Feasibility Conditions:

$$\mu_i \geq 0$$

for all

$$i = 1, \cdots, m$$

6. KKT Statement – Complementary Slackness Conditions:

$$\mu_i g_i(x^*) = 0$$

for all

$$i = 1, \cdots, m$$

7. Reduction to the Lagrange Criterion: In the particular case

$$m = 0$$

where there are no inequality constraints the KKT conditions turn into the Lagrange conditions and the KKT multipliers become the Lagrange multipliers.

8. Non-differentiable Version of KKT: If some of the functions are non-differentiable sub-differentiable versions of the KKT conditions are available (Eustaquio, Karas, and Ribeiro (2008)).

Regularity Conditions or Constraint Qualifications

1. The KKT Necessary Regularity Conditions: In order for a minimum point x^* to satisfy the above KKT conditions the problem should satisfy some regularity conditions. The most common ones are listed below.
2. Linear Constraint Qualification: If g_i and h_j are affine functions the no other condition is needed to be satisfied.

3. Linear Independence Constraint Qualification (LICQ): The gradients of the active inequality constraints and the gradients of the equality constraints are linearly independent at x^*.
4. Mangasarian-Fromovitz Constraint Qualification (MFCQ): The gradients of the active inequality constraints and the gradients of the equality constraints are positive-linearly independent at x^*.
5. Constant Rank Constraint Qualification (CRCQ): For each subset of the gradients of the active inequality constraints and the gradients of the active equality constraints the rank at the vicinity of x^* is constant.
6. Constant Positive Linear Dependence Constraint Qualification (CPLD): For each subset of the gradients of the active inequality constraints and the gradients of the equality constraints, if it is positive-linearly dependent at x^*, then it is positive-linearly dependent at the vicinity of x^*.
7. Quasi-Normality Constraint Qualification (QNCQ): If the gradients of the active inequality constraints and the gradients of the active equality constraints are positive-linearly dependent at x^* with the associated multipliers λ_i for equalities and μ_j for inequalities then there is no sequence

$$x_k \to x^*$$

such that

$$\lambda_i \neq 0$$

implies

$$\lambda_i h_i(x_k) > 0$$

AND

$$\mu_j \neq 0$$

implies

$$\mu_j g_j(x_k) > 0$$

8. Slater Condition: For a convex problem there exists a point x such that

$$h_j(x) = 0$$

and

$$g_i(x) < 0$$

9. Positive Linear Dependency Condition Definition: The set of vectors $\{v_1, \cdots, v_n\}$ is positive linearly dependent if there exists

$$a_1 \geq 0, \cdots, a_n \geq 0$$

not all zero such that

$$a_1 v_1 + \cdots + a_n v_n = 0$$

10. Strength Order of Constraint Qualifications: It can be shown that

$$LICQ \Longrightarrow MFCQ \Longrightarrow CPLD \Longrightarrow QNCQ$$

and

$$LICQ \Longrightarrow CRCQ \Longrightarrow CPLD \Longrightarrow QNCQ$$

– the converses are not true – although MFCQ is not equivalent to CRCQ (Ruszczynski (2006)). In practice weaker constraint qualifications are preferred since they provide stronger optimality conditions.

Sufficient Conditions

1. The Second Order Sufficiency Conditions: In some cases the necessary conditions are also sufficient for optimality. In general the necessary conditions are not sufficient for optimality and more information is needed, such as the Second Order Sufficiency Conditions (SOSC). For smooth functions SOSC involve the second derivatives, which explains its name.
2. When Necessary Conditions are Sufficient: The necessary conditions are sufficient for optimality of the objective function f of a maximization problem is a concave function, the inequality constraints g_i are continuously differentiable convex functions, and the equality constraints h_j are affine functions.
3. The Type 1 Invex Functions: The broader class of functions in which the KKT conditions guarantee global optimality are the so-called Type 1 **invex functions** (Martin (1985), Hanson (1999)).
4. Second Order Sufficiency Conditions Formulation: For smooth non-linear optimization problems the second order sufficiency conditions are given as follows. Consider x^*, λ^*, and μ^* that find a local minimum using the Karush-Kuhn-Tucker conditions above. With μ^* such that the strict complementary condition is held at x^*, i.e.

$$\mu > 0$$

for all

$$s \neq 0$$

such that

$$\left[\frac{\partial g(x^*)}{\partial x}, \frac{\partial h(x^*)}{\partial x}\right] s = 0$$

the following equation must hold:

$$s^T \nabla_{xx}{}^2 \mathcal{L}(x^*, \lambda^*, \mu^*) s > 0$$

If the above condition is strictly met, the function is a strict constrained local minimum.

KKT Conditions Application - Economics

1. KKT Models as Theoretical Tools: Often in mathematical economics the KKT approach is used in theoretical models in order to obtain qualitative results.
2. Minimum Profit Constraint Revenue Maximization: Consider a firm that maximizes its sales revenue subject to a minimum profit constraint. Letting Q be the quantity of output produced – this is to be chosen - $R(Q)$ be the sales revenue with a positive first derivative and with a zero value at zero output, $C(Q)$ be the production cost with a positive first derivative and with a non-negative value at zero output, and G_{min} be the positive minimal acceptable level of profit, then the problem is a meaningful one if the revenue function levels off so it is eventually less steep than the cost function.
3. Application of the KKT Condition: The problem can be expressed in the following minimization form: Minimize $-R(Q)$ subject to

$$G_{min} \leq R(Q) - C(Q)$$

$$Q \geq 0$$

and the KKT conditions are

$$Q\left[\frac{\partial R}{\partial Q}(1+\mu)-\mu\frac{\partial C}{\partial Q}\right]=0$$

$$R(Q)-C(Q)-G_{min} \geq 0$$

$$\mu \geq 0$$

$$\mu[R(Q)-C(Q)-G_{min}]=0$$

4. Revenue Growth vs. Cost Growth: Since

$$Q=0$$

would violate the minimum profit constraint

$$Q>0$$

must hold, and hence the third condition implies that the first condition holds with equality. Solving that equality gives

$$\frac{\partial R}{\partial Q}=\frac{\mu}{1+\mu}\frac{\partial C}{\partial Q}$$

5. Impact of the Minimum Profit Constraint: Because it was given that $\frac{\partial R}{\partial Q}$ and $\frac{\partial C}{\partial Q}$ are strictly positive, the inequality along with the non-negativity condition on μ is

positive and so the revenue-maximizing firm operates at a level of output at which the marginal revenue $\frac{\partial R}{\partial Q}$ is less than the marginal cost $\frac{\partial C}{\partial Q}$ - a result that is of interest because it contrasts the behavior of a profit maximizing firm which operates at a level at which they are equal.

KKT Conditions Application – Value Function

1. Equality/Inequality Function Space Constraints: Reconsidering the optimization problem as a maximization problem with constant inequality constraints $v_1, \cdots, v_n$ Maximize $f(x)$ subject to

$$g_i(x) \leq a_i$$

$$h_j(x) = 0$$

2. Definition of the Value Function: The value function is defined as

$$V(a_1, \cdots, a_n) = \sup_x f(x)$$

subject to

$$g_i(x) \leq a_i$$

$$h_j(x) = 0$$

$$j \in \{1, \cdots, l\}$$

$$i \in \{1, \cdots, m\}$$

So the domain of V is

$$a \in \mathbb{R}^m$$

for some

$$x \in X$$

$$g_i(x) \leq a_i$$

$$i \in \{1, \cdots, m\}$$

3. Interpretation of a_i and μ_i: Give this definition each coefficient μ_i is the rate at which the value of the function increases as a_i increases. Thus if each a_i is interpreteed as a resource constraint the coefficients indicate how much increasing the resource increases the optimum value of f. This interpretation is especially important in economics and is used, for example, in utility maximization problems.

Generalizations

1. KKT Extensions – Fritz John Conditions: With an extra constant multiplier μ_0 which may be zero, in front of $\vec{\nabla} f(x^*)$ the KKT stationary conditions turn into

$$\mu_0 \vec{\nabla} f(x^*) + \sum_{i=1}^{m} \mu_i \vec{\nabla} g_i(x^*) + \sum_{j=1}^{l} \lambda_i \vec{\nabla} h_j(x^*)$$

which are called the Fritz John conditions.

2. KKT Class as FONC Support: The KKT conditions belong to the wider class of the First Order Necessary Conditions (FONC) which allow for non-smooth functions using sub-derivatives.

References

- Boyd, S., and L. van den Berghe (2004): *Convex Optimization* **Cambridge University Press**.
- Eustaquio, R., E. Karas, and A. Ribeiro (2008): *Constraint Qualification under Non-linear Programming* Technical Report **Federal University of Parana**.
- Hanson, M. A. (1999): Invexity and the Kuhn-Tucker Theorem *Journal of Mathematical Analysis and Applications* **236 (2)** 594-604.
- Karush, W. (1939): *Minima of Functions of Several Variables with Inequalities as Side Constraints* Master's Thesis **University of Chicago** Chicago Illinois.
- Karush-Kuhn-Tucker Conditions (Wiki).
- Kjeldsen, T. H. (2000): A Contextualized Analysis of the Kuhn-Tucker Theorem in Non-linear Programming: The Impact of World War II *Historia Mathematica* **27 (4)** 331-361.
- Martin, D. H. (1985): The Essence of Invexity *Journal of Optimization Theory and Applications* **47 (1)** 65-76.
- Ruszczynski, A. (2006): *Nonlinear Optimization* **Princeton University Press** Princeton NJ.

Interior Point Method

Motivation, Background, and Literature Survey

1. Definition of Interior Point Methods: Interior Point Methods (also known as *barrier methods*) are a class of algorithms that solves linear and non-linear convex optimization problem (Interior Point Method (Wiki)).
3. John von Neumann's Early Approach: John von Neumann suggested an interior point method of linear programming that was neither a polynomial time method not an efficient method in practice (Dantzig and Thapa (2003)). In fact it turned out to be slower in practice than the simplex method which is not a polynomial time method.
4. Karmarkar's Extension to the Simplex Method: Karmarkar (1984) developed a method for linear programming called the Karmarkar's algorithm which runs in provably polynomial time, and is also very efficient in practice. It enabled solutions to linear programming problems that were beyond the capabilities of the simplex method.
5. Traversal across the Feasible Region: Contrary to the Simplex method Karmarkar's algorithm reaches the best solution by traversing the interior of the feasible region. The method can be generalized to convex programming based on a self-concordant barrier function used to encode the convex set.
6. Transformation of the Convex Function: Any convex optimization problem can be transformed into minimizing (or maximizing) a linear function over a convex set by converting to an epigraph form (Boyd and van den Berghe (2004)). The idea of encoding the feasible set using a barrier and designing barrier methods was studied by Anthony V. Fiacco, Garth P. McCormick, and others in the early '60s. These ideas were mainly developed for general non-linear programming, but they were later abandoned due to the presence of more competitive methods for this class of problems (e.g., sequential quadratic programming).

7. Barriers to Encode Convex Set: Yuri Nesterov and Arkadi Nemirovskii came up with a special class of such barriers that can be used to encode any convex set. They provide guarantees that the number of iterations of the algorithm is bounded by a polynomial in the dimension and as well for the accuracy of the solution (Wright (2004)).
8. Interior Point Methods with Barriers: Karmarkar's breakthrough re-vitalized the study of interior point methods and barrier problems showing that it was possible to create an algorithm for linear programming characterized by polynomial time complexity, and, moreover, that was competitive with the simplex method. Already Khachiyan's ellipsoid method was a polynomial time algorithm; however it was too slow to be of practical interest.
9. Interior Point Method Sub-classes: The class of primal dual path-following interior point methods is considered the most successful. Mehrotra's predictor-corrector algorithm provides the basis for most implementations of this class of methods (Mehrotra (1992), Potra and Wright (2000)).

Interior Point Methodology and Algorithm

1. Principle, Concept, and Approach Methodology: The main idea behind interior point methods is to iterate inside of the feasible set and progressively approach the boundary – if the minimum does lie on the boundary. This is done by transforming the original problem into a sequence of unconstrained optimization problems in which the objective function uses a *barrier function* that goes to ∞ at the boundaries of the feasible region. By reducing the strength of the barrier at each subsequent optimization the sequence of minima approach arbitrarily closely toward the minimum of the original problem.
2. Objective Function with Logarithmic Barriers: For the inequality constrained problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

such that

$$g_i(x) \leq 0, i = 1, \cdots, m$$

f is modified to obtain a *barrier function*

$$f_\alpha(x) = f(x) - \alpha \sum_{i=1}^{m} \log g_i(x)$$

3. Impact of the Barrier Strength Parameter: Since the logarithm goes to $-\infty$ as its argument approaches 0 the modified barrier function becomes arbitrarily large as x approaches the boundaries of the feasible region. The parameter α gives the barrier "strength". Its significance is that as α approaches 0 the optimum of $f_\alpha(x)$ approaches the optimum of $f(x)$. More precisely if f^* optimizes $f(x)$ and

$$x_\alpha{}^* \equiv \underset{x}{\arg min}\, f_\alpha(x)$$

then

$$\underset{\alpha \to 0}{Limit}\, f(x_\alpha{}^*) = f^*$$

4. Gradient of the Barrier Function: Newton's method is employed to find the minimum of $f_\alpha(x)$ The gradient is

$$\nabla f_\alpha(x) = \nabla f(x) - \alpha \sum_{i=1}^{m} \frac{\nabla g_i(x)}{g_i(x)}$$

5. Numerical Stability Close to the Boundary: One concern is that as $x_\alpha{}^*$ proceeds closer and closer to the boundary the numerical stability of the unconstrained optimization problem becomes worse and worse because the denominator approaches zero. Thus it would be very challenging to apply the gradient descent to a small tolerance.
6. Use of KKT Type Multipliers: Therefore another set of KKT multiplier like variables

$$\vec{\lambda} = (\lambda_1, \cdots, \lambda_m)$$

with

$$\lambda_i \equiv \frac{\alpha}{g_i(x)}$$

is introduced. So in the $\left(x, \vec{\lambda}\right)$ space one seeks the root of the equation

$$\nabla_x f_\alpha\left(x, \vec{\lambda}\right) = \nabla f(x) - \sum_{i=1}^{m} \lambda_i \nabla g_i(x) = 0$$

subject to the equality

$$\lambda_i g_i(x) = \alpha$$

for

$$i = 1, \cdots, m$$

7. Stability of the Modified Solution: This set of equations is far more numerically stable than

$$\nabla f_\alpha(x) = \nabla f(x) - \alpha \sum_{i=1}^{m} \frac{\nabla g_i(x)}{g_i(x)}$$

near the boundary. Note the similarity between these equations and the KKT equations. In fact if α were 0 one gets the KKT equations except with the equalities stripped away.

8. Objective/Constraint Function - Partial Derivatives: To find a root $\left(x_\alpha{}^*, \vec{\lambda}_\alpha{}^*\right)$ where both of the functions are satisfied the Newton's method is applied. To do so one needs the partial derivatives of the above functions.

$$\nabla_{xx}{}^2 f_\alpha\left(x, \vec{\lambda}\right) = \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$$

$$\nabla_{\vec{\lambda}} \nabla_x f_\alpha(x) = -\sum_{i=1}^{m} \nabla g_i(x)$$

$$\nabla_x(\lambda_i g_i(x) - \alpha) = \lambda_i \nabla g_i(x)$$

$$\frac{\partial}{\partial \lambda_i}[\lambda_i g_i(x) - \alpha] = g_i(x)$$

for

$$i = 1, \cdots, m$$

9. Variate/Constraint Multipliers newton Increment: At the current iterate $\left(x_t, \vec{\lambda}_t\right)$ the Newton step $\left(\Delta x_t, \Delta\vec{\lambda}_t\right)$ is derived from the following system of equations.

$$\begin{bmatrix} \mathcal{H} & -\mathcal{G} \\ Diagonal(\vec{\lambda}_t) \cdot \mathcal{G}^T & Diagonal(\vec{g}) \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta \vec{\lambda}_t \end{bmatrix} = \begin{bmatrix} -\nabla f(x_t) + \mathcal{G} \cdot \vec{\lambda}_t \\ \alpha \mathbb{I} - Diagonal(\vec{g}) \cdot \vec{\lambda}_t \end{bmatrix}$$

where $\mathcal{H}$ denotes the Hessian

$$\nabla_{xx}{}^2 f_\alpha(x, \vec{\lambda}) = \nabla^2 f(x) - \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x)$$

evaluated at $(x_t, \vec{\lambda}_t)$, $\vec{g}$ denotes the $m \times 1$ vector of g_i's, $\mathcal{G}$ denotes the $n \times m$ matrix whose i^{th} column is $\nabla g_i(x_t)$, $Diagonal(\vec{v})$ produces a square matrix whose diagonal is the vector $\vec{v}$, and $\mathbb{I}$ denotes the $m \times 1$ vector of all 1's.

10. Variate Retention inside Feasible Region: Solving this system of equations provides a search direction that is then update via line search to avoid divergence. Note that to keep x drifting out of the feasible set one must enforce for all i the condition

$$g_i(x) \geq 0$$

or equivalently

$$\lambda_i \geq 0$$

during the line search.

11. Progressive Reduction of the Barrier Strength: Once the unconstrained search has converged α may be reduced (e.g., by multiplication by a small number) and the optimization can begin again. There is a trade-off in the convergence thresholds for each unconstrained search; too small and the algorithm must perform a lot of work even when α is high; but too large and the sequence of unconstrained optima does not converge quickly. A more balanced approach is to choose the threshold proportional to α.

12. Application to Non-Convex Functions: The formulation above did not explicitly require that the problem be convex. Interior point methods can certainly be used in general non-convex problems, but like any local optimization problem they are not guaranteed to a minimum. Furthermore computing the Hessian matrix is quite often expensive.
13. Use in Linear/Quadratic Problems: They do, however, work quiet well in convex problems. For example in quadratic programming the Hessian is constant, and in linear programming the Hessian is zero.
14. Initialization at a Feasible Point: The interior point method must be initialized at an interior point, or else the barrier function is undefined. To find the initial feasible point x the following optimization may be used.

$$\underset{x \in \mathbb{R}^n, s_1, \cdots, s_m}{max} \sum_{i=1}^{m} s_i$$

such that

$$g_i(x) - s_i \geq 0, i = 1, \cdots, ms_i \geq 0$$

If the problem is feasible then the optimal s_i will all be 0. It is easy to find an initial set of s_i for any given x simply by setting

$$s_i = \min(g_i(x), 0)$$

References

- Boyd, S., and L. van den Berghe (2004): *Convex Optimization* **Cambridge University Press**.

- Dantzig, G. B., and M. N. Thapa (2003): *Linear Programming 2 – Theory and Extensions* **Springer-Verlag**.
- Interior Point Method (Wiki).
- Karmarkar, N. (1984): A New Polynomial-Time Algorithm for Linear Programming *Combinatorica* **4 (4)** 373-395.
- Mehrotra, S. (1992): On the Implementation of the Primal-Dual Interior Point Method *SIAM Journal on Optimization* **2 (4)** 575-601.
- Potra, F. A., and S. J. Wright (2000): Interior Point Methods *Journal of Computational and Applied Mathematics* **124 (1-2)** 281-302.
- Wright, M. H. (2004): The Interior-Point Revolution in Optimization; History, Recent Developments, and Lasting Consequences *Bulletin of the American Mathematical Society* **42 (1)** 39-56.

Optimizer

Constrained Optimization using Lagrangian

1. Base Set up: Use the Lagrangian objective function to optimize a multi-variate function $L(x, y)$ to incorporate the constraint

$$g(x, y) = c$$

as

$$\Lambda(x, y, z) = L(x, y) + \lambda[g(x, y) - c]$$

Here λ is called the Lagrange multiplier, and use one Lagrange multiplier per constraint.

2. Optimize (Maximize/Minimize) the Lagrangian: Optimize (i.e., maximize/minimize) the Lagrangian with respect to x, y, and λ – thus

$$\frac{\partial \Lambda(x, y, z)}{\partial x} = 0$$

$$\frac{\partial \Lambda(x, y, z)}{\partial y} = 0$$

and

$$\frac{\partial \Lambda(x, y, z)}{\partial \lambda} = 0$$

Notice that

$$\frac{\partial \Lambda(x, y, z)}{\partial \lambda} = 0$$

automatically implies the validity of the constraint

$$g(x, y) = c$$

thereby accommodating it in a natural way.

- Further

$$\frac{\partial^2 \Lambda(x, y, z)}{\partial \lambda^2} = 0$$

always, since $\Lambda(x, y, z)$ is linear in λ. Further, since the constraint is true by the optimizer construction, there should be no explicit dependence on λ.

3. Comparison with unconstrained optimization:
   - Unconstrained Optimization results in

$$\frac{\partial L(x, y)}{\partial x} = 0$$

and

$$\frac{\partial L(x, y)}{\partial y} = 0$$

Constrained Optimization results in

$$\frac{\partial \Lambda(x,y,z)}{\partial x} = \frac{\partial L(x,y)}{\partial x} + \lambda \frac{\partial g(x,y)}{\partial x}$$

and

$$\frac{\partial \Lambda(x,y,z)}{\partial y} = \frac{\partial L(x,y)}{\partial y} + \lambda \frac{\partial g(x,y)}{\partial y}$$

$$\lambda = 0$$

automatically reduces the constrained case to the unconstrained.

- Advantage of the Lagrange Multiplier Incorporation into Optimization => This ends up converting the constrained formulation space over on to the unconstrained, thereby providing with as many equations as the number of optimization unknowns, and one equation each per every constraint.
- Drawback of the Lagrange Multiplier Optimization Incorporation => While it lets you passively move to the unknowns' space from the constrained formulation space, this may end up impacting the application of certain boundary conditions, such as financial boundary, natural boundary conditions, Not-A-Knot boundary condition, etc.

4. Constraints of inequality $g(x,y) < c$ or $g(x,y) > c$: Solutions to these constraints exist either inside the unconstrained set, in which case the unconstrained equations can be solved, or they don't, in which case convert the inequality to an equality and give it the Lagrange multiplier treatment.
5. Comparison with Convex Optimization: Convex optimization is predicated on the presence of at least one minimum/maximum in the zone of interest. One example is variance/covariance constrained optimization, where both the variance/covariance and the constraints are both quadratic. Eigenization (just another type of constrained variance/covariance optimization) is another.
6. Penalizing Optimizer as a Constrained Optimization Setup: Naively put

$$\Lambda = Good - Bad$$

where, from a calibration point of view, "Good" refers to closeness of fit, and "Bad" to the curvature/smoothness penalty. Thus, constrained optimization here corresponds to "maximize the Good, and minimize the Bad".

- De-coupling "Good/Bad" from closeness of fit and curvature/smoothness penalty, respectively, can lead to alternate optimization framework formulations in finance, along with its insights.

Least Squares Optimizer

1. Least Squares Optimization Formulation:

$$\mu_i(x_i) = y_i$$

$$\hat{\mu}_i(x_i) = \sum_{j=1}^{n} a_j B_{ij}(x_i)$$

$$S = \sum_{i=1}^{m} [\mu_i(x_i) - \hat{\mu}_i(x_i)]^2 = \sum_{i=1}^{m} \left[y_i - \sum_{j=1}^{n} a_j B_{ij}(x_i) \right]^2$$

$$= \sum_{i=1}^{m} \left\{ {y_i}^2 - 2 y_i \sum_{j=1}^{n} a_j B_{ij}(x_i) + \left[\sum_{j=1}^{n} a_j B_{ij}(x_i) \right]^2 \right\}$$

$$\frac{\partial S}{\partial a_j} = 2 \left[\sum_{i=1}^{m} a_j B_{ij}(x_i) - y_j \right] \left[\sum_{i=1}^{m} a_j B_{ij}(x_i) \right]$$

2. Least Squares Matrix Formulation:

$$Y = [y_1, \cdots, y_m]^T$$

$$B_i(\vec{x}) = \left[B_{i,1}(\vec{x}), \cdots, B_{i,m}(\vec{x})\right]^T$$

$$a = [a_1, \cdots, a_m]^T$$

Then

$$\left[\frac{\partial \vec{S}}{\partial A}\right] = 2AB(\vec{x})B^T(\vec{x})A - 2YB(\vec{x})$$

As expected

$$y_i = \sum_{j=1}^{n} a_j B_{ij}(x_i)$$

is the optimized least squares tight fit.

Multi-variate Distribution

1. Mean/Variance Location Dependence: The mean is sensitive to both translation and rotation, unless the distribution is mean centered. The variance along a given fixed direction, however, is not sensitive to rotation of the basis.
   - This also implies that the maximal/minimal variances are invariant to representational basis changes. They are, however, sensitive to scaling, though, as PCA itself is.
2. Dimensional Independence vs. Dimensional Realization Independence: Dimensions are distinct (pressure, temperature etc.), but the realizations in those dimensions need not be. Therefore, correlated unit vectors only apply to actual realizations (and they are NOT scale invariant).
3. Orthogonal Data Set in the Native Basis Representation: If a data set is orthogonal under a given basis, then

$$\langle x_i x_j \rangle = 0$$

   (See Figure 1).
4. Non-orthogonal Data Set in the Native Basis Representation: In this case, the native representation basis results in

$$\langle x_i x_j \rangle \neq 0$$

   (See Figure 2). Thus, an orthogonalization operation needs to be performed such that under the new basis

$$\langle x_i' x_j' \rangle = 0$$

(See Figure 3).

5. Orthogonalization as Principal Components Extraction: As can be seen from figures 2 and 3, under the new schematic axes

$$\langle x_i' x_j' \rangle = 0$$

Further these correspond to the principal components, i.e., orthogonal components where the variance is an extremum.

6. Orthogonalized Representation: Upshot of all these is that, if the representation basis is structured such that

$$\langle x_i x_j \rangle = 0$$

for all

$$i \neq j$$

then these representations automatically correspond to principal components as well.

7. Full Rank Matrix: This is simply an alternate term for multi-collinear matrix.

Parallels between Vector Calculus and Statistical Distribution Analysis

1. DOT PRODUCT => The notion of dot product is analogous to covariance operation in statistical analysis, i.e., DOT PRODUCT is

$$\vec{x}_i \cdot \vec{x}_j = 0$$

if

$$\vec{x}_i \perp \vec{x}_j$$

and covariance is

$$\langle x_i x_j \rangle = 0$$

if x_i and x_j are orthogonal to each other.

2. Distance Metric => Vector Euclidean distance is $\sum[(x_1 - x_2)^2 + (y_1 - y_2)^2 + \cdots]$ is equivalent to the variance $\sum[(x_i - \mu_i)^2 + \cdots]$. Further, extremizing the Euclidean/Frobenius distance is analogous to variance minimization/maximization techniques.

Linear Systems Analysis and Transformation

Matrix Transforms

1. Co-ordinate Rotation and Translation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix} + B$$

where A is the rotating transformer, and B is the translator.

2. Scaling vs. Rotation: Say

$$A = \begin{bmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{bmatrix}$$

If

$$a_{11} = a_{22} \neq 1$$

and

$$a_{12} = a_{21} = 0$$

it becomes pure scaling.

$$a_{11} \neq a_{22}$$

and

$$a_{12} = a_{21} = 0$$

produces differential elongation/compression and

$$a_{12} \neq 0$$

or

$$a_{21} \neq 0$$

results in rotation.

3. Uses of Gaussian Elimination:
   - Linearization
   - Orthogonalization
   - Inversion
   - Diagonalization
   - Lower/Upper Triangle Decomposition (LU Decomposition)
   - Independent Component Extraction
   - Principal Component Analysis
4. Diagonal Identity Matrix Conception: Given a matrix A what matrix M should it be transformed by to get a diagonal identity matrix, i.e.

$$MA = I$$

Answer is

$$M = A^{-1}$$

Thus, diagonalization is also an inversion operation.

$$Diagonalization == Orthogonalization == Inversion == ICA$$

Diagonalization, of course, is unique only to a diagonal entry, whereas inversion corresponds to a specific choice of the diagonal entry.

Systems of Linear Equations

1. Importance of Diagonal Dominance in Gauss-Seidel: Is diagonal dominance important because the dominant diagonal's contribution to the RHS drives the given equation's value, and therefore the iterative accuracy?
2. Eigenization Square Matrix Inversion Conceptualization: Given a source matrix

$$F_{SOURCE,0} = \{f_{ij}\}_{i,j=0}^{n-1}$$

and an initialized target inverse identity matrix

$$F_{INV,0} = \{I\}_{n \times n}$$

achieve a suitable set of p transformations simultaneously on F_{SOURCE} and F_{INV} to eventually make

$$F_{SOURCE,p} = \{I\}_{n \times n}$$

so that the corresponding

$$F_{INV,p} = \{f_{INV,ij}\}_{i,j=0}^{n-1}$$

becomes the inverse.

3. Valid Inversion Rules: These are fairly straight forward application of the Gaussian elimination scheme:
   - Scale a single F_{SOURCE} row by a constant => scale the corresponding F_{INV} row by the same constant.
   - Add/subtract any pair of F_{SOURCE} rows from each other => add/subtract the same pair of F_{INV} rows from each other.
4. Matrix Inversion using Gaussian Elimination:
   - Scan across the diagonal entries.
   - Scan through the rows corresponding to each diagonal entry.
   - If a given row entry call value is zero, or its index corresponds to that of a diagonal, skip.
   - Calculate the $WorkColFactor$ as

$$WorkColFactor = \frac{DiagonalEntry}{CellValueEntry}$$

   - Scan all the cells in the column of the current cell.
   - Apply the $WorkColFactor$ product to each entry in the current working column of the source matrix.
   - Do the same as above to the inverse matrix.
   - Subtract the entries corresponding to the designated diagonal column from the working column to make the current entry zero.
   - Do the same as above to the inverse matrix as well.
   - After the completion of all such diagonal scans, row scans, and working column scans, re-scan the diagonal again.
   - Scale down each entry of the source matrix by itself, so that the source matrix entries now constitute an identity ($\{I\}_{n\times n}$) matrix.
   - Do the same as above to the inverse matrix as well.

5. Non-invertible Coefficient Matrix, but Solution exists: For unprocessed coefficient matrices, certain conditions (such as zero diagonal entries) may cause the coefficient matrix to be technically non-invertible, but that does not automatically mean that the system is unsolvable – a simply re-casting of the basic linear system set may be all that is required.
6. Linear Basis Re-arrangement: Sometimes, a singular coefficient matrix (with zero determinant, therefore non-invertible) may be re-arranged to create an invertible coefficient matrix. After inversion, it can be re-structured again to extract the inverse (which is just a coefficient Jacobian).
7. Rows/Columns as "Preferred Linear Basis Sequence" for Matrix Manipulation: Consider the solution to

$$AX = Z$$

where X and Z are columns. In this case, the notion of constraint linear representation is maintained exclusively in rows. Therefore, all elimination/scaling basis operations need to be applied on that basis. In considering the solution to

$$AX = Z$$

where X and Z are rows, columns now become the preferred linear basis sequence.

8. Regularization before Gauss Elimination: Before the Gauss Elimination can process, we need to diagonalize the matrix. Row swapping is a more robust way to diagonalize than row accumulation due to a couple of reasons:
    - Matrix Row Swap vs. Row Cumulate => In all cases, row swap can be transformed into row accumulation. When inverting, however, the row swapping AND accumulation should both be done on both the SOURCE and the TARGET matrices.
    - From a core linear operation set point-of-view, row accumulation is the inverse of the eventual of Gauss Elimination, so the danger is that the

diagonalization gains of row swap may be undone during the intermediate stages of Gauss Elimination.

- Even more important is that row swapping simply retains the original information, by just re-arranging the row set.

9. Row Swapping Caveats:
   - Always swap rows by retaining the directionality of the scan AND by retaining the scan initial node preceding the swap (to choose the pivot). One way to do this is by starting the scan at $row + 1$ - or from

$$row = 0$$

     if the edge has been reached - AND always keeping the scan sequence forward/backward.
   - Also ensure that the target swapped row is "valid", i.e., it's post-swapped diagonal entry should be non-zero. If this cannot be achieved through the scan, then that is an error condition.
10. Diagonalization/Inversion Algorithm: Work in terms of an intermediate transform variate Z produced by re-arranging the original coefficient matrix and Y such that the A in

$$AX = Z$$

    is now invertible. The following would be the steps:
    - First, re-arrange the equation system set to identify a suitable pair A and Z such that A is invertible, and Z is estimated from

$$AX = Z$$

    - The re-arranging linear operation set will produce A such that

$$BY = Z \Longrightarrow AX = BY \Longrightarrow X = A^{-1}BY$$

11. **More General Inverse Transformation Re-formulation:** Given the coefficient matrix $\{a_{qj}\}_{j,q=0}^{n-1}$, the set of unknown variables $\{x_j\}_{j=0}^{n-1}$, and the RHS $\{y_q\}_{q=0}^{n-1}$, y_p, y_q, and the corresponding z_p and z_q are given as

$$\sum_{j=0}^{n-1} a_{qj}x_j = y_q$$

$$\sum_{j=0}^{n-1} a_{pj}x_j = y_p$$

and

$$\{z_j = y_j\}_{j=0}^{n-1}$$

On transformation (i.e., adding row p to row q), you get

$$\sum_{j=0}^{n-1} (a_{pj} + a_{qj})x_j = y_p + y_q$$

and therefore

$$\{z_j = y_j\}_{j=0;j\neq q}^{n-1}$$

along with

$$z_q = y_p + y_q$$

This clearly implies that

$$\frac{\partial z_q}{\partial y_p} = 1$$

or

$$B_{qp} = B_{qp} + 1$$

Orthogonalization

1. 2D Orthogonalization: In 2D, you need to fix one 1D orthogonal axis to be able to orthogonalize the other.
2. 2D Equation System:

$$x_1 = a_{11}s_1 + a_{12}s_2$$

and

$$x_2 = a_{21}s_1 + a_{22}s_2$$

This has 4 unknowns, so one solution for this is as follows: Fix

$$a_{11} = 1$$

and

$$a_{12} = 0$$

This results in 2 unknowns. Setting

$$\langle {x_1}^2 \rangle = 1$$

$$\langle {x_2}^2 \rangle = 1$$

and

$$\langle x_1 x_2 \rangle = \rho$$

you get

$${a_{11}}^2 + {a_{12}}^2 = 1$$

and

$${a_{21}}^2 + {a_{22}}^2 = 1$$

resulting in

$$a_{21} = \rho$$

and

$$a_{21} = \sqrt{1 - \rho^2}$$

Thus, all unknowns are determined.

3. nD Orthogonalization:

$$\begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,n-1} \end{bmatrix} \begin{bmatrix} s_0 \\ \vdots \\ s_{n-1} \end{bmatrix}$$

- Number of diagonal entries => n
- Number of non-diagonal entries => $n^2 - n$
- Net Number of equations => Number of diagonal entries + Number of non-diagonal entries =>

$$n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$$

4. nD Unknowns Analysis:
   - Number of Unknowns => n^2.
   - Fix the first row, and the number of unknowns becomes => $n^2 - n$.
   - Fixing the row takes off one equation, so the number of equations => $\frac{n(n+1)}{2} - 1$
   - Number of equations = Number of Unknowns =>

$$\frac{n(n+1)}{2} - 1 = n^2 - n$$

   results in

$$(n-1)(n-2) = 0$$

Gaussian Elimination

1. n-D Gaussian Elimination: What does it work? If you fix a row, you can rotate the other rows to eliminate dependence on an ordinate of the fixed row.
2. Row Fixation as a Basis Choice: Row elimination does not automatically make the matrix diagonal or orthogonalize it – all it does is to eliminate dependence on a stochastic variate.

3. Number of Elimination Rotations: The first row fixation results in $n-1$ rotations, the second row fixation results in $n-2$ rotations, and so on. Thus, the total number of rotating transformations is $\frac{(n-1)(n-2)}{2}$ (i.e., the same as the number of unknowns seen earlier). The result of these transformations is a lower/upper triangular matrix.
4. Final Reversing Sweep: An additional reverse sweep would eliminate similar column dependencies as well – resulting in another $\frac{(n-1)(n-2)}{2}$ rotation choices.
5. Number of Sweep Operations: Total result of all the rotations and their corresponding choices =>

$$2 \cdot \frac{(n-1)(n-2)}{2} = (n-1)(n-2)$$

Rayleigh Quotient Iteration

Introduction

1. Idea behind Rayleigh Quotient Iteration: **Rayleigh Quotient Iteration** is an eigenvalue algorithm that extends the idea of inverse iteration by using the Rayleigh Quotient to obtain increasingly accurate eigenvalue estimates (Wiki - Rayleigh Quotient Iteration (2018)).
2. Progressive Navigation towards "True" Selection: Rayleigh Quotient Iteration is an iterative method, i.e., it delivers a sequence of approximate solutions that converge to a true solution in the limit. This is true for all algorithms that compute eigenvalues; since the eigenvalues can be irrational numbers, there can be no general method for computing them in a finite number of steps.
3. Practicality of the Iterative Approach: Very rapid convergence is guaranteed and no more than a few iterations are needed in practice to obtain a reasonable solution.
4. Cubic Convergence for Typical Matrices: The Rayleigh Quotient algorithm converges cubically for Hermitian or symmetric matrices, given an initial vector that is sufficiently close to an eigenvector of the matrix that is being analyzed.

The Algorithm

1. Update Eigenvalue using Rayleigh Quotient: The algorithm is very similar to inverse iteration, but replaces the estimated eigenvalues at the end of each iteration with the Rayleigh Quotient.

2. Initial Guess for Eigenvalue/Eigenvector: Begin by choosing a value μ_0 as an initial eigenvalue guess for the Hermitian matrix A. An initial vector b_0 must also be supplied as the initial eigenvector guess.
3. Iterative Eigenvalue and Eigenvector: Calculate the subsequent approximation of the eigenvector b_{i+1} by

$$b_{i+1} = \frac{[A - \mu_i I]^{-1} b_i}{\|[A - \mu_i I]^{-1} b_i\|}$$

where I is the identity matrix, and set the next approximation of the eigenvalue to the Rayleigh quotient of the current iteration equal to

$$\mu_i = \frac{b_i{}^* A b_i}{b_i{}^* b_i}$$

4. Deflation Techniques for Successive Eigenvalues: To compute more than one eigenvalue the algorithm can be combined with a deflation technique.
5. Treatment for Very Small Matrices: Note that for very small matrices it is beneficial to replace the matrix inverse with the adjugate, which will yield the same iteration because it is equal to the inverse to an irrelevant scale – specifically, the inverse of the determinant.
6. Advantages of using the Adjugate: The adjugate is easier to compute explicitly than the inverse – though the inverse is easier to apply to a vector for problems that aren't small – and is more numerically sound because it remains well-defined as the eigenvalue changes.

References

- Wikipedia – Rayleigh Quotient Iteration (2018)

Power Iteration

Introduction

1. Power Iteration – Problem Statement/Definition: In mathematics, **power iteration** – also known as the *power method* – is an eigenvalue algorithm. Given a diagonalizable matrix A, the algorithm will produce a number λ, which is the greatest – in absolute value – eigenvalue of A, and a non-zero vector v, the corresponding eigenvector of λ, such that

$$Av = \lambda v$$

   The algorithm is also known as the von Mises iteration (von Mises and Pollazek-Geiringer (1929)).
2. Characteristics of the Power Iteration Algorithm: Power iteration is a very simple algorithm, but it may converge slowly. It does not compute a matrix decomposition, and hence can be used when A is a very large sparse matrix.

The Method

1. Starting Choice for Principal Eigenvector: The power iteration method starts with a vector b_0, which may be either an approximation to the dominant eigenvector or a random factor.
2. Recurrence Relation for Power Iteration: The method is described by the recurrence relation

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

So, at every iteration, the vector b_k is mulplied by the matrix A and normalized.

3. Necessary Condition for Eigen-component Convergence: If one assumes that A has an eigenvalue that is greater in magnitude than its other eigenvalues and that the starting vector b_0 has a non-zero component in the direction of an eigenvector associated with the dominant eigenvalue, then a sub-sequence $\{b_k\}$ converges to an eigenvector associated with the dominant eigenvector.
4. Recast of $\{b_k\}$ using Phase Shift: Without the two assumptions above the sequence does not necessarily converge. In this sequence

$$b_k = e^{i\phi_k} v_1 + r_k$$

where v_1 is the eigenvector associated with the dominant eigenvalue, and

$$\|r_k\| \to 0$$

The presence of the term $e^{i\phi_k}$ implies that $\{b_k\}$ does not converge unless

$$e^{i\phi_k} \equiv 1$$

5. Convergence to the Dominant Eigenvalue: Under the two assumptions listed above, the sequence $\{\mu_k\}$ defined by

$$\mu_k = \frac{{b_k}^T A b_k}{{b_k}^T b_k}$$

converges to the dominant eigenvalue.

6. Eigenvector and Eigenvalue Sequences: The vector b_k is the associated eigenvector. Ideally one should use the Rayleigh Quotient in order to get the associated eigenvalue.
7. Spectral Radius using the Rayleigh Quotient: The method can also be used to calculate the spectral radius – the largest eigenvalue of a matrix – by computing the Rayleigh Quotient

$$\frac{{b_k}^T A b_k}{{b_k}^T b_k} = \frac{{b_{k+1}}^T b_k}{{b_k}^T b_k}$$

Analysis

1. Decomposition to Jordan Canonical Form: Let A be decomposed into its Jordan Canonical Form

$$A = VJV^{-1}$$

where the first column of V is an eigenvector of A corresponding to the dominant eigenvalue λ_1.

2. The Principal Component Jordan Block: Since the dominant eigenvalue of A is unique, the first Jordan block of J is the 1×1 matrix $[\lambda_1]$, where λ_1 is the largest eigenvalue of A in magnitude.
3. Initializing the Principal Component Eigenvector: The starting vector b_0 can be written as a linear combination of the columns of V:

$$b_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

By assumption, b_0 has a non-zero component in the direction of the dominant eigenvalue, so

$$c_1 \neq 0$$

4. k[th] Recurrence of the Initial Vector: The computationally recurrence relation for b_{k+1} can be written as:

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|} = \frac{A^{k+1}b_0}{\|A^{k+1}b_0\|}$$

where the expression $\frac{A^{k+1}b_0}{\|A^{k+1}b_0\|}$ is amenable to the analysis below.

5. Principal Component Dependence of b_k:

$$\begin{aligned} b_k = \frac{A^k b_0}{\|A^k b_0\|} = \frac{[VJV^{-1}]^k b_0}{\|[VJV^{-1}]^k b_0\|} &= \frac{VJ^kV^{-1}b_0}{\|VJ^kV^{-1}b_0\|} \\ &= \frac{VJ^kV^{-1}(c_1v_1 + c_2v_2 + \cdots + c_nv_n)}{\|VJ^kV^{-1}(c_1v_1 + c_2v_2 + \cdots + c_nv_n)\|} \\ &= \frac{VJ^k(c_1\hat{e}_1 + c_2\hat{e}_2 + \cdots + c_n\hat{e}_n)}{\|VJ^k(c_1\hat{e}_1 + c_2\hat{e}_2 + \cdots + c_n\hat{e}_n)\|} \\ &= \left(\frac{\lambda_1}{|\lambda_1|}\right)^k \frac{c_1}{|c_1|} \frac{v_1 + \frac{1}{c_1}V\left(\frac{1}{\lambda_1}J\right)^k (c_2\hat{e}_2 + \cdots + c_n\hat{e}_n)}{\left\|v_1 + \frac{1}{c_1}V\left(\frac{1}{\lambda_1}J\right)^k (c_2\hat{e}_2 + \cdots + c_n\hat{e}_n)\right\|} \end{aligned}$$

6. Scaling Down the Principal Eigenvector: As

$$k \to \infty$$

the expression above simplifies to

$$\left(\frac{1}{\lambda_1}J\right)^k = \begin{pmatrix} [\mathbb{I}] & \cdots & & \\ \cdots & \left(\frac{1}{\lambda_1}J_2\right)^k & \cdots & \cdots \\ \cdots & \cdots & \ddots & \cdots \\ \cdots & \cdots & \cdots & \left(\frac{1}{\lambda_1}J_n\right)^k \end{pmatrix} \to \begin{pmatrix} [\mathbb{I}] & \cdots & \cdots & \cdots \\ \cdots & 0 & & \\ \cdots & \cdots & \ddots & \cdots \\ \cdots & \cdots & \cdots & 0 \end{pmatrix}$$

7. Limit of $\left(\frac{1}{\lambda_1}J_i\right)^k$ as $k \to \infty$: The limit follows from the fact that the eigenvalue of $\left(\frac{1}{\lambda_1}J_i\right)^k$ is less than 1 in magnitude, so

$$\left(\frac{1}{\lambda_1}J_i\right)^k \to 0$$

as

$$k \to \infty$$

8. Expansion of Higher Order Eigen Terms: It follows that

$$\frac{1}{c_1}V\left(\frac{1}{\lambda_1}J\right)^k (c_2\hat{e}_2 + \cdots + c_n\hat{e}_n) \to 0$$

as

$$k \to \infty$$

9. Reduction of b_k for $k \to \infty$: Using this fact, b_k can be written in a form that emphasizes its relationship with v_1 when k is large:

$$b_k = \left(\frac{\lambda_1}{|\lambda_1|}\right)^k \frac{c_1}{|c_1|} \frac{v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} J\right)^k (c_2 \hat{e}_2 + \cdots + c_n \hat{e}_n)}{\left\| v_1 + \frac{1}{c_1} V \left(\frac{1}{\lambda_1} J\right)^k (c_2 \hat{e}_2 + \cdots + c_n \hat{e}_n) \right\|} = e^{i\phi_k} \frac{c_1}{|c_1|} \frac{v_1}{\|v_1\|} + r_k$$

where

$$e^{i\phi_k} = \left(\frac{\lambda_1}{|\lambda_1|}\right)^k$$

and

$$\|r_k\| \to 0$$

as

$$k \to \infty$$

10. Uniqueness of the $\{b_k\}$ Sequence: The sequence $\{b_k\}$ is bounded, so it contains a convergent sub-sequence. Note that the eigenvector corresponding to the dominant eigenvalue is unique only upto a scaler, so although the sequence $\{b_k\}$ may not converge, b_k is nearly an eigenvector of A for large k.
11. Alternative Proof of the Sequence Convergence: Alternatively, if A is diagonalizable, then the following proof yields the same result. Let $\lambda_1, \lambda_2, \cdots, \lambda_m$ be the m eigenvalues – counted with multiplicity – of A, and let $v_1, v_2, \cdots, v_m$ be the corresponding eigenvectors. Suppose that λ_1 is the dominant eigenvalue, so that

$$|\lambda_1| > |\lambda_j|$$

for

$$j > 1$$

12. Suitable Choice for the Initial Vector: As seen earlier, the initial vector b_0 is written as

$$b_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

If b_0 is chosen randomly – with uniform probability – the

$$c_1 \neq 0$$

with probability 1

13. Power Iteration over Initial Eigenvector: Now

$$A^k b_0 = c_1 A^k v_1 + c_2 A^k v_2 + \cdots + c_n A^k v_n = c_1 {\lambda_1}^k v_1 + c_2 {\lambda_2}^k v_2 + \cdots + c_n {\lambda_n}^k v_n$$
$$= c_1 {\lambda_1}^k \left[v_1 + \frac{c_2}{c_1} \left(\frac{\lambda_2}{\lambda_1} \right)^k v_2 + \cdots + \frac{c_n}{c_1} \left(\frac{\lambda_n}{\lambda_1} \right)^k v_n \right] \rightarrow c_1 {\lambda_1}^k v_1$$

as long as

$$\left| \frac{\lambda_j}{\lambda_1} \right| < 1$$

for

$$j > 1$$

14. Convergence to the Principal Eigenvector: On the other hand,

$$b_k = \frac{A^k b_0}{\|A^k b_0\|}$$

Therefore b_k converges to a multiple of the eigenvector v_1

15. Convergence Ratio of Power Iteration: The convergence ratio is geometric with the ratio $\left|\frac{\lambda_2}{\lambda_1}\right|$ where λ_2 denotes the second principal eigenvalue.
16. Consequence of Close Principal Eigenvectors: Thus, the method converges slowly if there is an eigenvalue close in magnitude to the dominant eigenvalue.

Applications

1. Identification of the Dominant Eigenvector/Eigenvalue: Although the power iteration method approximates only one eigenvalue of a matrix, it remains useful for several computational problems.
2. Use in Google and Twitter: For instance, Google uses it to calculate the PageRank of documents in their search engine (Ipsen and Wills (2005)), and Twitter uses it to show users recommendations of who to follow (Gupta, Goel, Lin, Sharma, Wang, and Zadeh (2013)).
3. Space Advantage of Power Iteration: The power iteration is especially suitable for sparse matrices, such as the web matrix, or as the matrix-free method that does not require storing the coefficient matrix A explicitly, but can instead access a function evaluating matrix-vector products Ax.
4. Power Iteration vs. Arnoldi Method: For non-symmetric matrices that are well-conditioned, the power iteration method can outperform the more complex Arnoldi method.
5. Power Iteration vs. Lanczos/LOBPCG: For symmetric matrices, the power iteration method is rarely used, since its convergence speed can be easily increased without sacrificing the smaller cost per iteration, e.g., Lanczos iteration and LOBPCG.

6. Variation in the Method - Inverse Iteration: Some of the more advanced algorithms can be understood as variations of the power iteration method. For instance, the inverse iteration method applies power iteration to the matrix A^{-1}.
7. Sub-space of Eigenvalues - Krylov Subspace: Other algorithms look at the whole subspace generated by the vectors b_k. This subspace is known as the Krylov subspace. It can be computed by Arnoldi iteration or Lanczos iteration.

References

- Gupta, P., A. Goel, J. Lin, A. Sharma, D. Wang, and R. Zadeh (2013): WTF – The Who-To-Follow Service at Twitter
- Ipsen, I., and R. Wills (2005): Analysis and Computation of Google's PageRank
- Von Mises, R., and H. Pollaczek-Geiringer (1929): Praktische Verfahren der Gleichingsauflosung *Zeitschrift fur Andewandte Mathematik und Mechanik* **9** 152-164.

Figure #1
Fixed Point Search SKU Flow

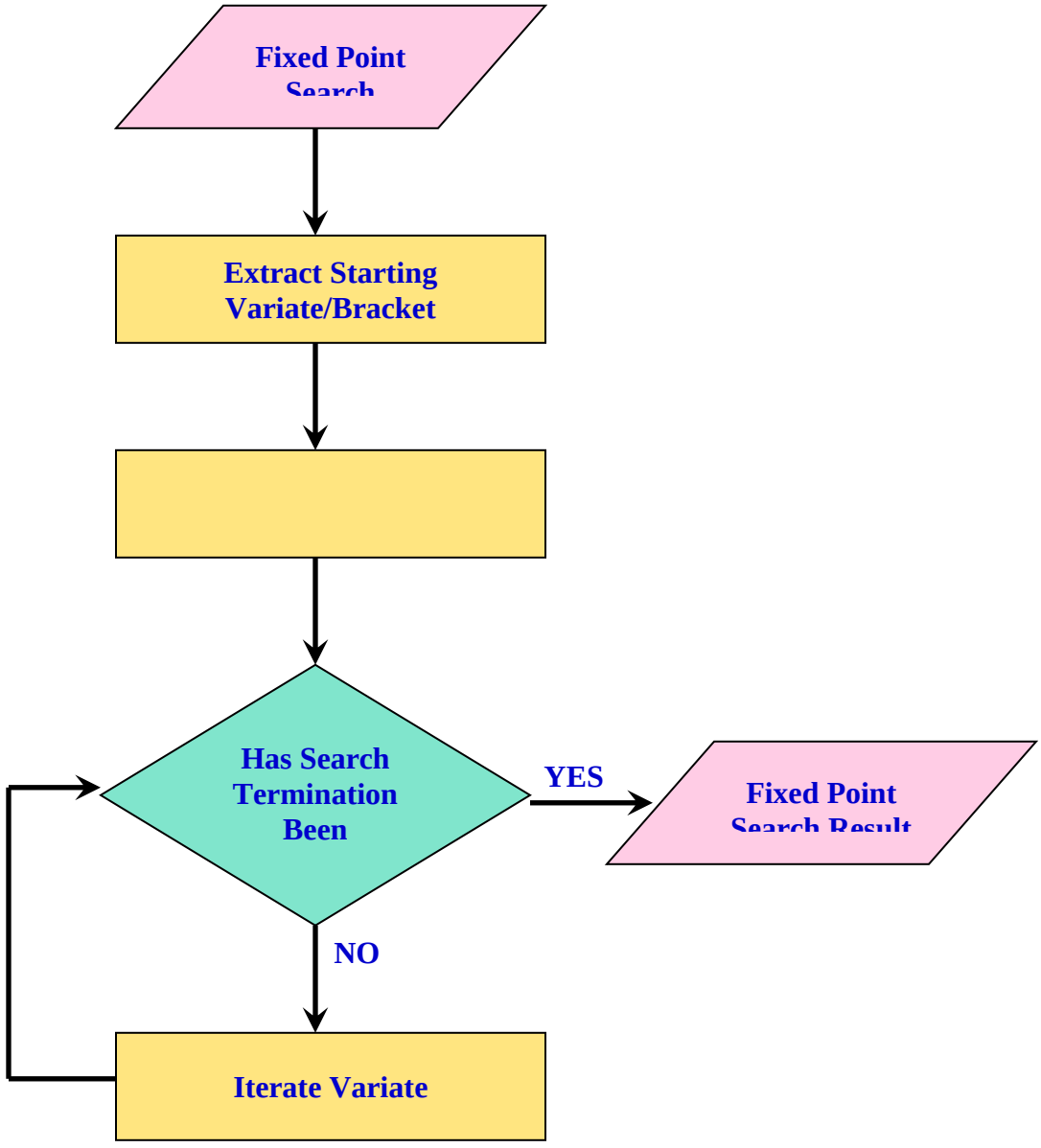

Figure #2
Bracketing SKU Flow

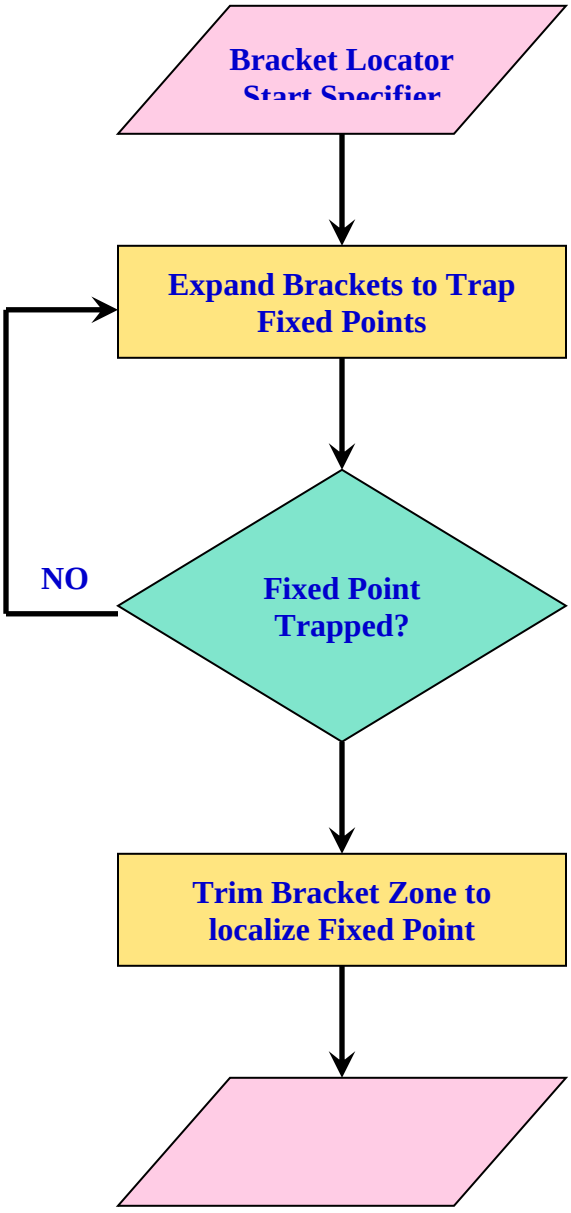

Figure #3
Objective Function Undefined at the Starting Variate

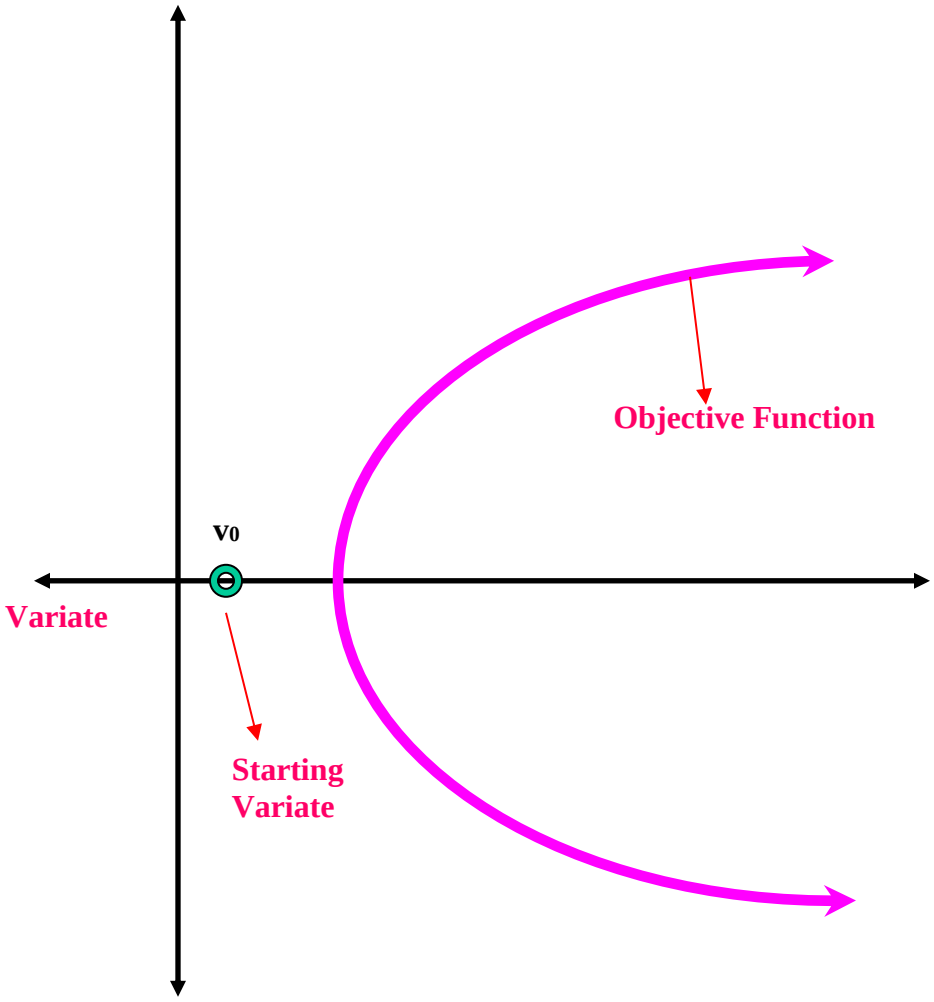

Figure #4
Objective Function Undefined at any of the Candidate Variates

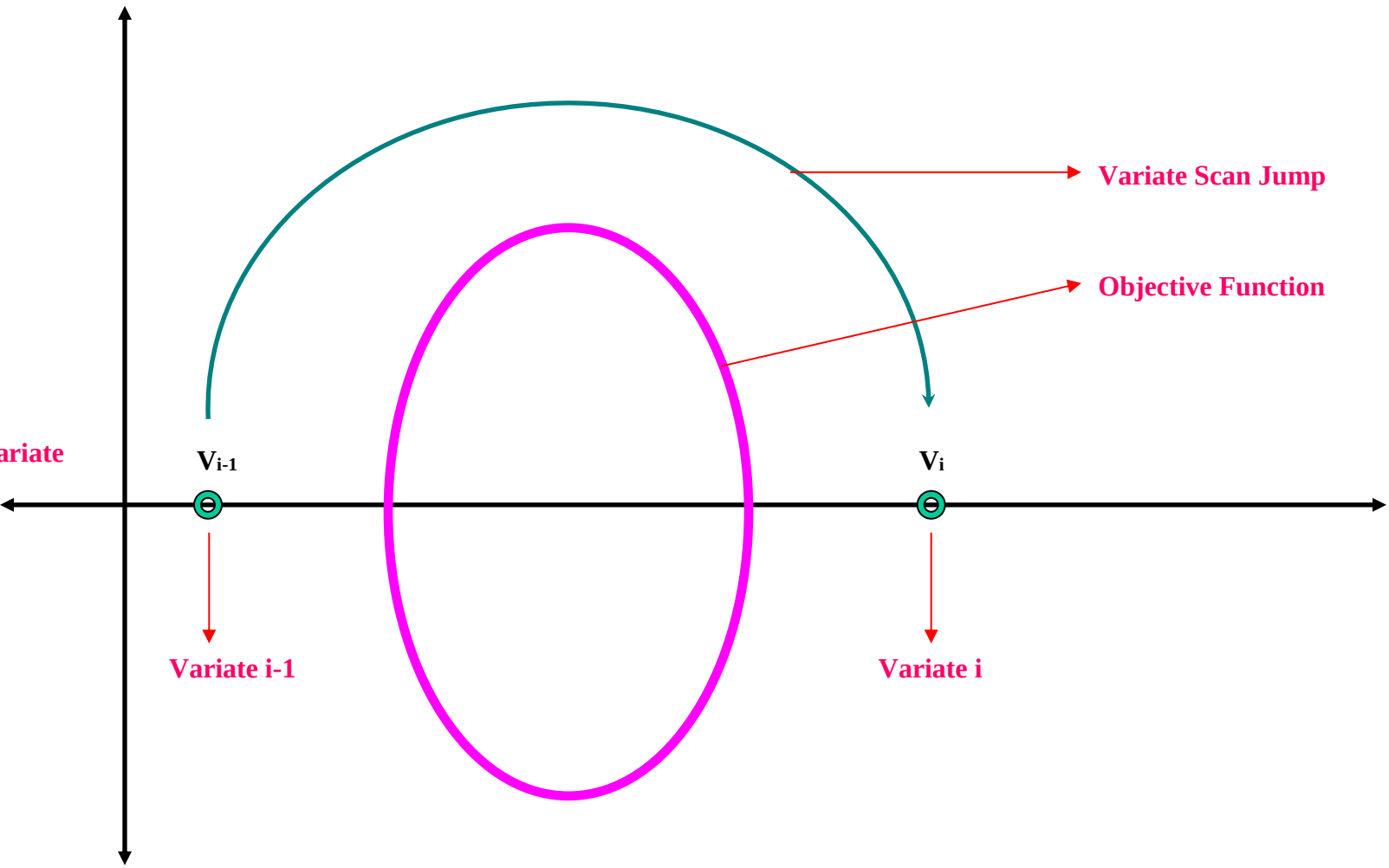

Figure #5
General Purpose Bracket Start Locator

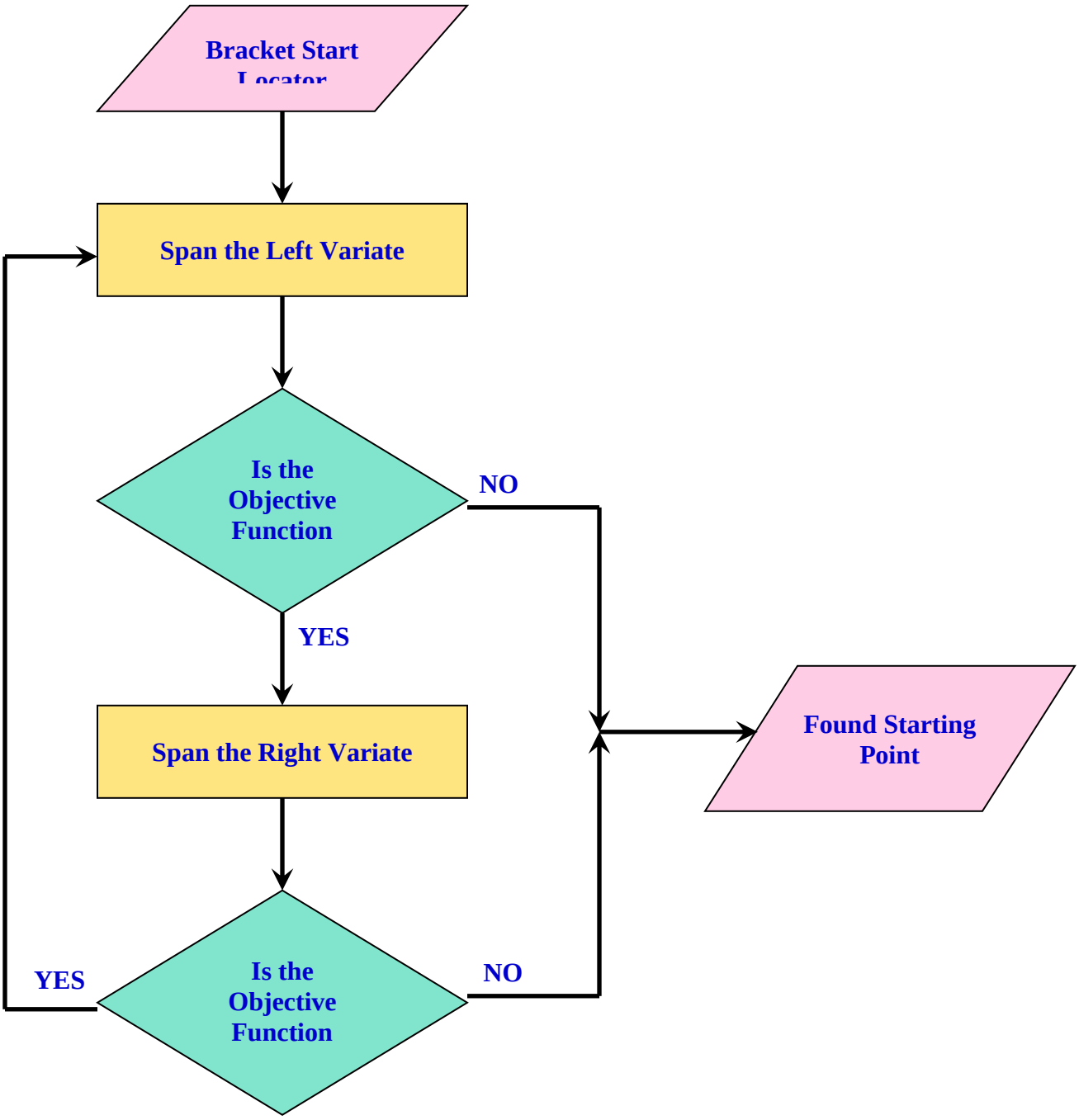

Figure #6
Bracketing when Objective Function Validity is Range-bound

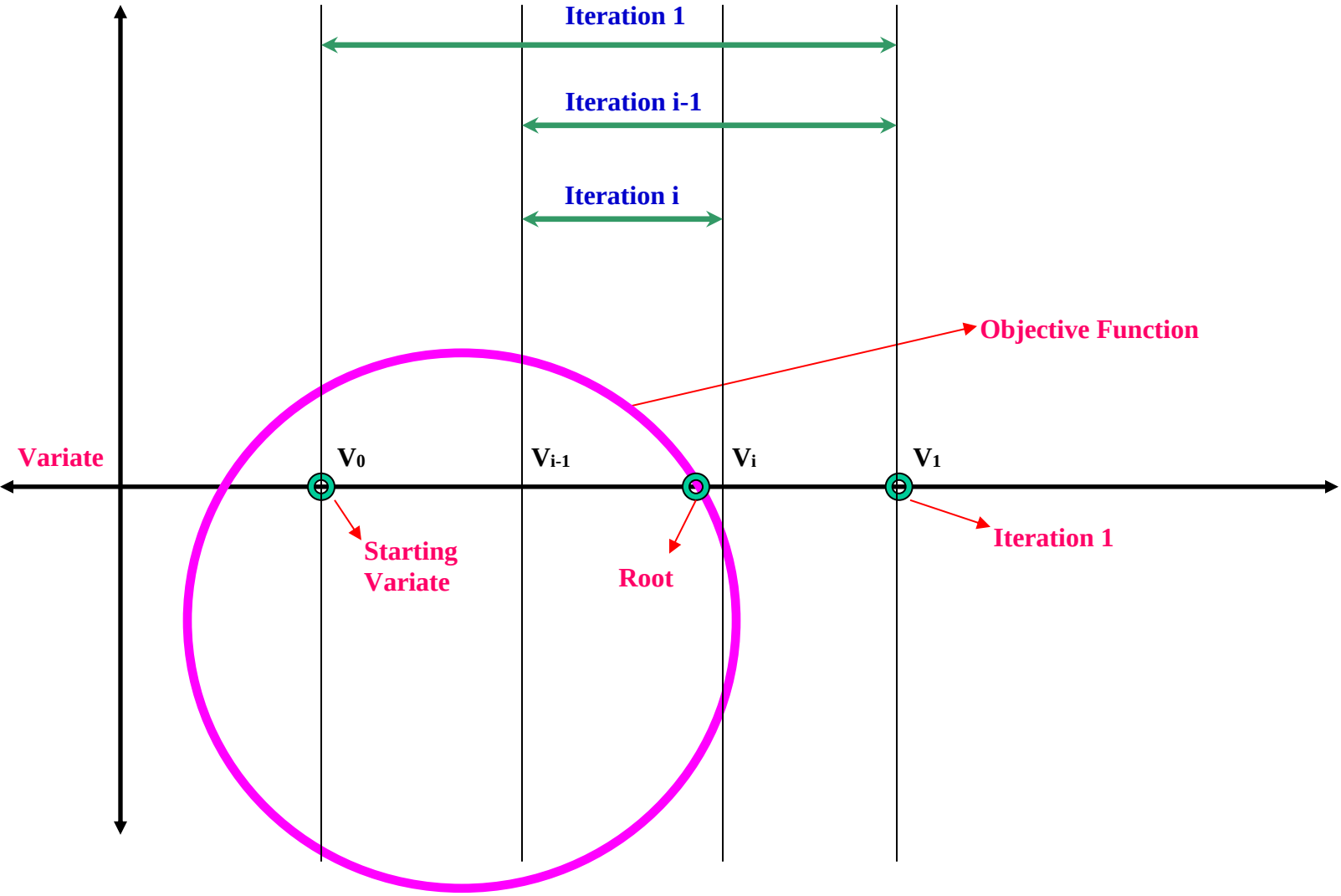

Figure #7
Objective Function Fixed Point Bracketing