Numerical Optimization in DRIP

Lakshmi Krishnamurthy

v0.34, 3 May 2016

Section I: Fixed Point Finder

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

    d. Open Search Initialization
    e. Search/Bracketing Initializer Customization Heuristics
3. Numerical Challenges in Search
4. Variate Iteration
5. Open Search Methods
    a. Newton's Method
6. Closed Search Methods
    a. Secant
    b. Bracketing Iterative Search
    c. Univariate Iterator Primitive
        i. Bisection
        ii. False Position
        iii. Inverse Quadratic
        iv. Ridder's
    d. Univariate Compound Iterator
        i. Brent's Method
        ii. Zheng's Method
7. Polynomial Root Search
8. References
9. Figures
10. Fixed Point Search Software Components
    a. Execution Initialization
    b. Bracketing
    c. Execution Customization
    d. Fixed Point Search
    e. Variate Iteration
    f. Initialization Heuristics

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 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. Chindris, 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.

Section II: Convex Optimization – Problem Space Specification

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 A $C(A)$.
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

$$x^T Ax \leq c$$

   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

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

   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.

Section III: Numerical Optimization – Approaches and Solutions

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 $(x, \vec{\lambda})$ space one seeks the root of the equation

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

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.

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

where $\mathcal{H}$ denotes the Hessian

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

evaluated at $\left(x_t, \vec{\lambda}_t\right)$, $\vec{g}$ 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

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

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.

Figure #1
Fixed Point Search SKU Flow

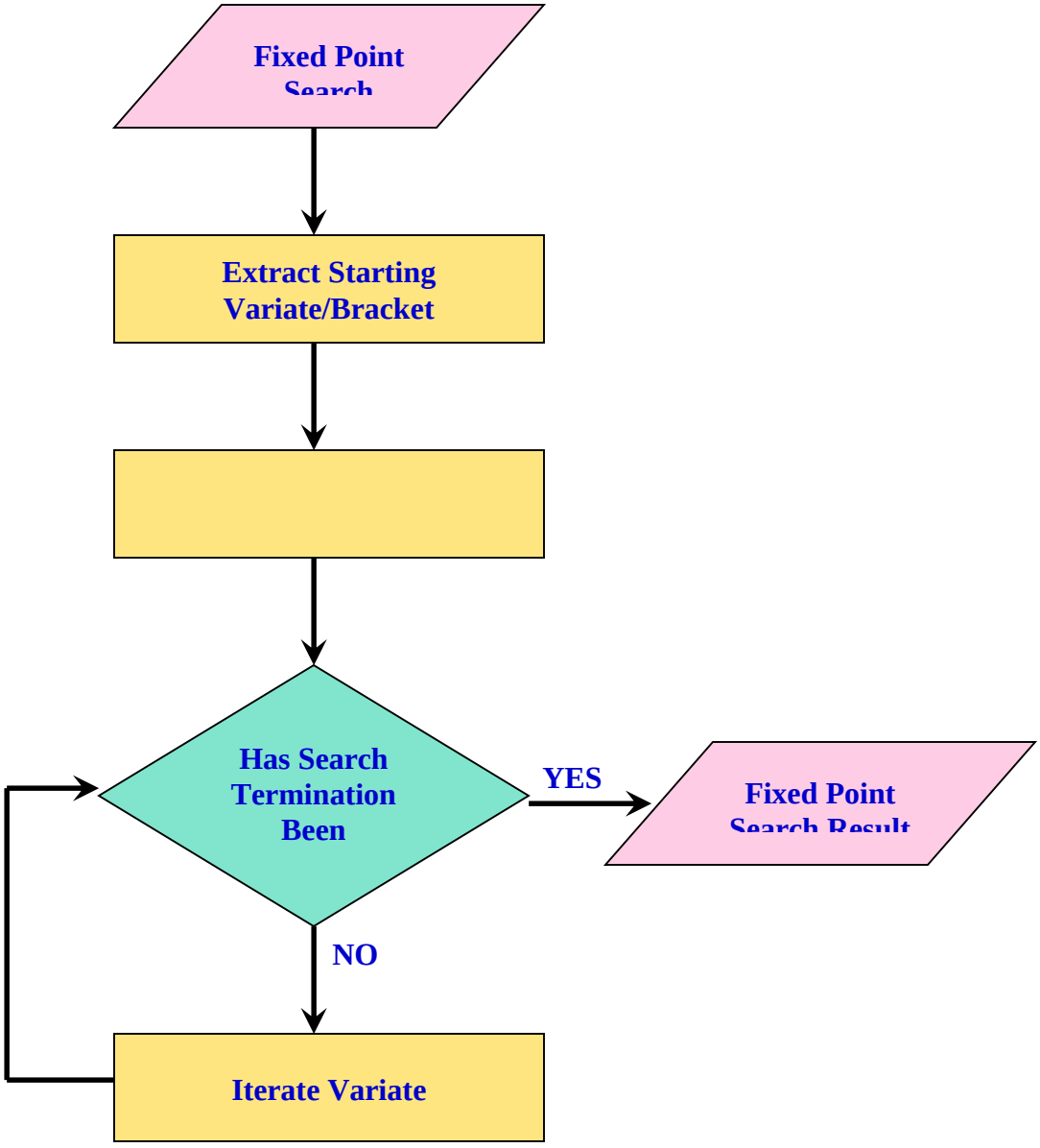

Figure #2
Bracketing SKU Flow

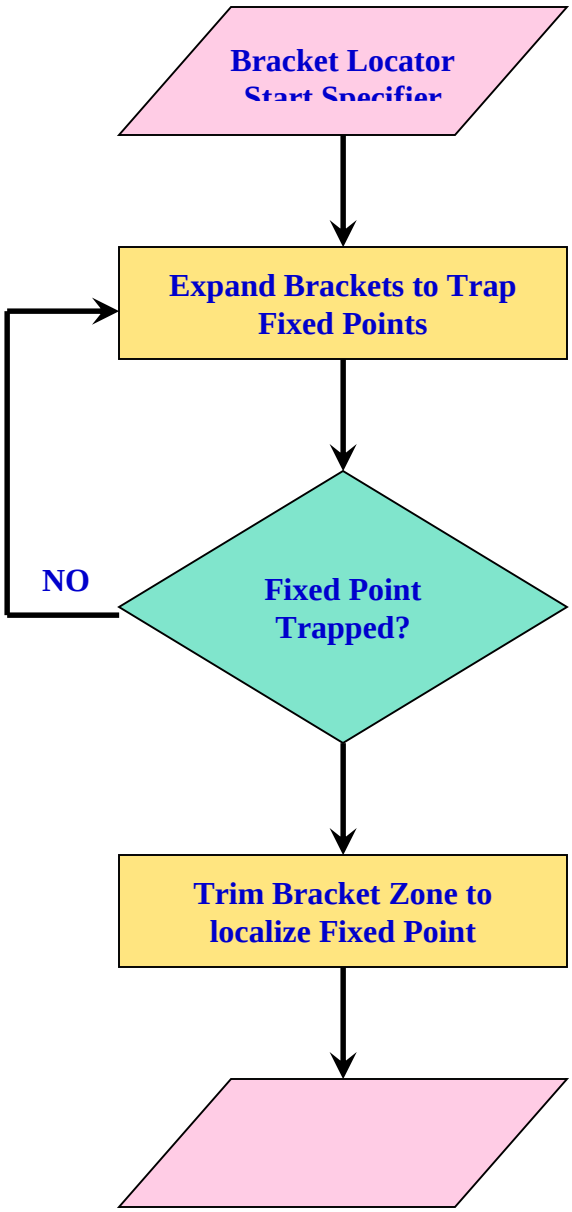

Figure #3
Objective Function Undefined at the Starting Variate

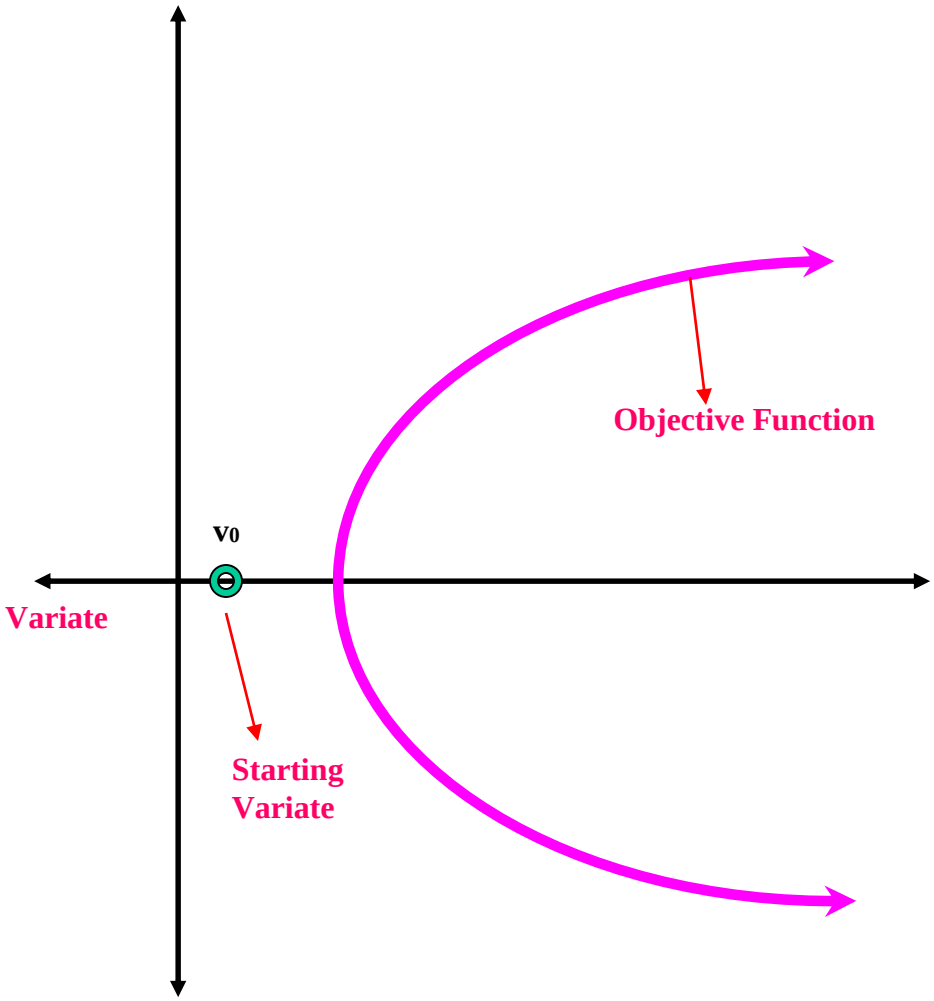

Figure #4
Objective Function Undefined at any of the Candidate Variates

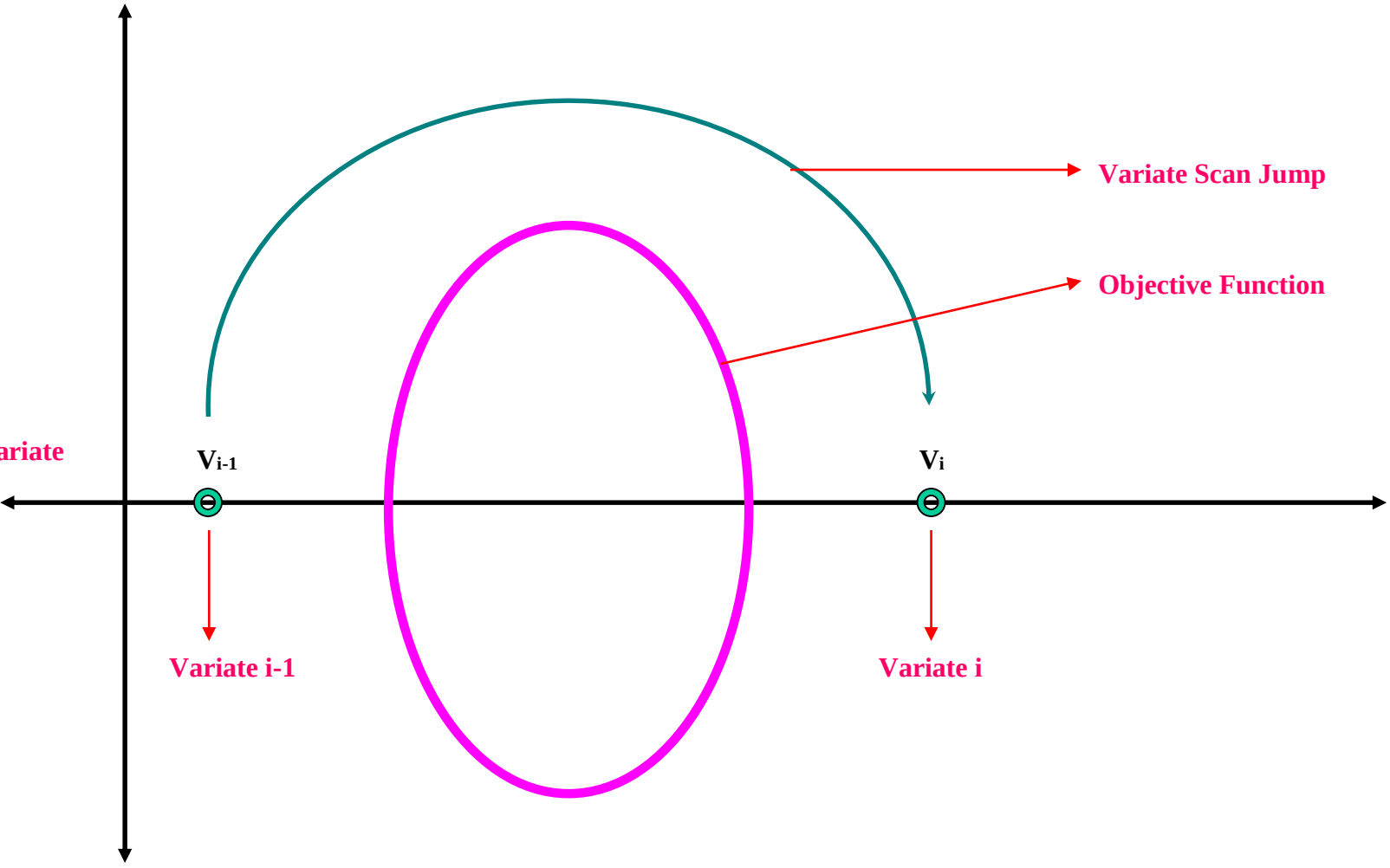

Figure #5
General Purpose Bracket Start Locator

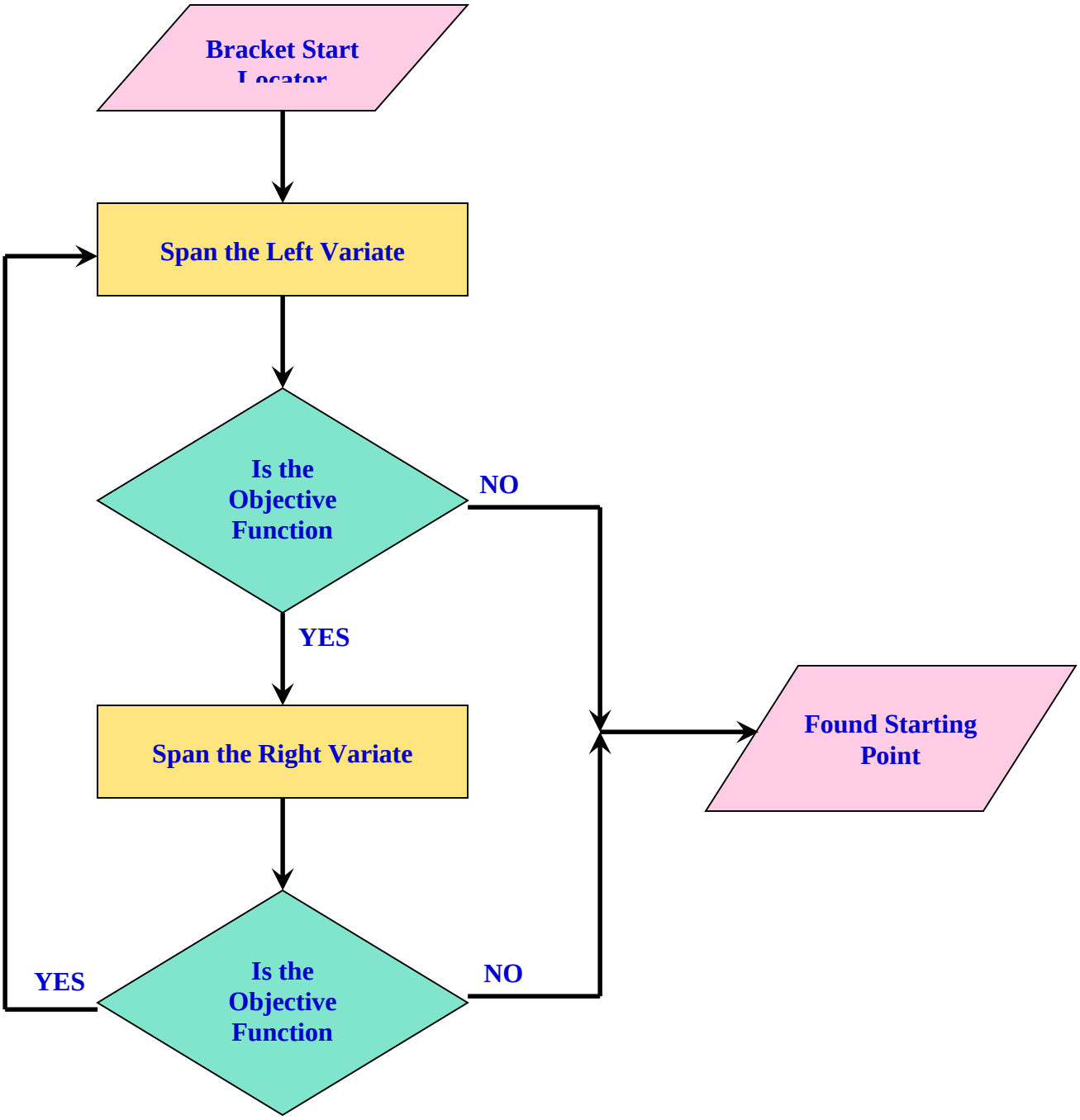

Figure #6
Bracketing when Objective Function Validity is Range-bound

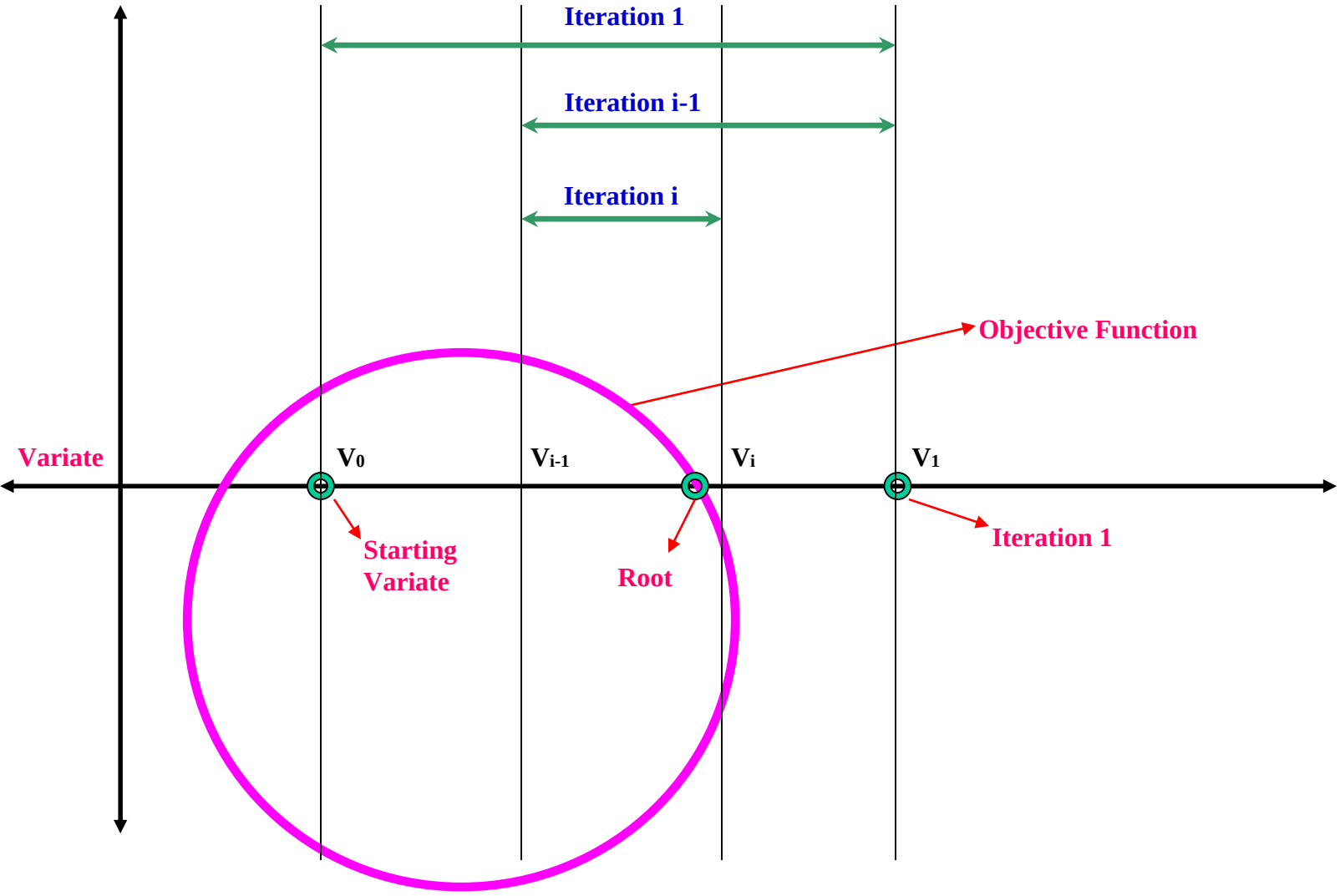

Figure #7
Objective Function Fixed Point Bracketing