LINE SEARCH METHODS

What is it?

- methods in which the search is conducted along successive lines in the design space

- typically use derivatives in the search ← but not always
  ↳ Sometimes we don't use derivatives directly, but we do a finite difference approximation to determine a derivative

What do we need to know? (assumptions)

- When doing a line search, we don't have to exactly minimize the function along that line → we approximate the minimum
  ↳ Why? It may not be worth while to find the exact minimum. We ultimately don't care what the minimum of the function is along a line. We care that, after a certain number of iterations, we have a very good approximation of the minimum of the function in the entire space

  => we can accept some error (or give up some precision/accuracy) in any one line search because the algorithm will repetitively/iteratively search the design space → search along one line, build a new search direction, search along new line
  => we can develop simpler procedures for the line search itself

- Each algorithm has a different approach to select a new search direction at each iteration ← we will always search in lines in some direction

- Some algorithms require particular types of line search methods
- Some algorithms can be implemented with any line search method

- Line search is conducted in a descent direction ← requirement for most line search algorithms

Advantages?

- more systematic than direct search
- performing a search in 1-D is much easier than in n-D

Disadvantages?

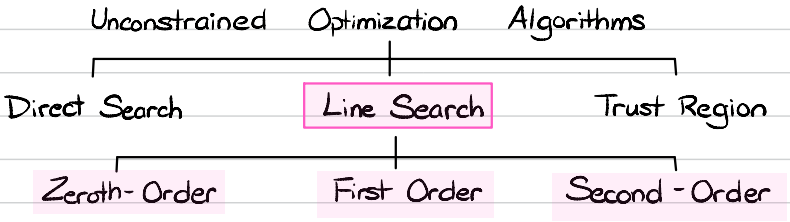
Comments:

Numerical optimization algorithm = systematic approach for improving estimate of minimum/maximum
→ Keep trying to iteratively improve estimate until we try and stop at some point
- formally converge problem according to necessary or sufficient conditions
- or we run out of resources/computing power/time and so we decide to stop

LINE SEARCH METHODS

Types of Line Search Methods

Line search methods are characterized by the quality of the local approximation of the function that they use to determine the search direction s_k ← or p_k
↳ want to determine the quality of the line search



Zeroth-Order Methods ← Some consider zeroth-order methods to be direct search methods

- use only values of the function $f(x)$

ex. Univariate Search, Powell's Method

We test a few particular points sampled along the line and evaluate the function at each of these points.

In certain cases, we could develop derivative approximations using 0^{th} order methods, but no actual gradient information is provided

First-Order Methods

- use values of $f(x)$ and $\nabla f(x)$ ← we have the benefit of using not only not only function values, but also the gradient
- may use approximations of $H(x)$, but are still of 1^{st} order accuracy
  ↳ can develop an approximation of the Hessian with multiple approximations of the gradient over subsequent steps in space
  ↳ never have direct knowledge of $H(x)$ - only have direct knowledge of $f(x)$ and $\nabla f(x)$

ex. Steepest Descent, Conjugate Gradient, BFGS Quasi-Newton Method

Main Points:

Line search methods are characterized by how they determine the search direction (quality of approximation of function to determine s_k)

0^{th}: use $f(x)$ 1^{st}: use $f(x)$, $\nabla f(x)$ 2^{nd}: use $f(x)$, $\nabla f(x)$, $H(x)$

LINE SEARCH METHODS

Types of Line Search Methods

Second-Order Methods

- use values of $f(x)$, $\nabla f(x)$, and $H(x)$ ← quickly converge to the optimum (in terms of overall number of iterations/steps)

ex. Newton's Method

Computing $H(x)$ in higher dimensional problems is difficult
=> rarely have 2nd order in higher dimensions

Most numerical methods for line searches are 1st order numerical methods

How does it work?

We look at a 1-D slice of a function along a line and we solve the optimization problem along that line.

Basic Idea:

Line search methods are based on an iterative procedure of the form:

$$\underline{x}_k = \underline{x}_{k-1} + \alpha^* \underline{s}_k \quad \leftarrow \text{same thing} \rightarrow \quad \underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k$$

fundamental equation that defines line search

1. Select a starting point $\underline{x}_{k-1}$
2. Select a search direction $\underline{s}_k$
3. Select a distance to move in that direction α^*
4. Decide when to stop iterating
   ↳ decide when the process has converged to an acceptable solution

$\underline{x}_k$: new design variable vector
$\underline{x}_{k-1}$: previous design variable vector
$\underline{s}_k$: direction vector in a descent direction that defines the line being searched
α^* : Scalar indicating distance we should (approximately) move to minimize the function along the line ← finding α^* is the task of the line search itself

Comments:

$$f(\underline{x}_k) \equiv f(\underline{x}_{k-1} + \alpha^* \underline{s}_k) \equiv \varphi(\alpha) \equiv f(\underline{x}_{k+1}) \equiv f(\underline{x}_k + \alpha \underline{p}_k)$$

↑ for unconstrained optimization, these are all saying the same thing

$= f(\underline{x}_{k-1} + \alpha^* \underline{s}_k)$ = line search equation
$= \varphi(\alpha)$ = objective function
$= f(\underline{x}_{k+1})$ = merit function

} we want $\min_{\alpha} \varphi(\alpha)$

LINE SEARCH METHODS

How does it work?

1. Select starting point $\underline{x}_{k-1}$ (or $\underline{x}_k$)
   - usually just our choice
   - this is a vector => list of all the design variables
     => point in multidimensional space
   - successively increase (increment) K as we iterate
   - old $\underline{x}$ ($\underline{x}_{k-1}$ or $\underline{x}_k$) informs our new $\underline{x}$ ($\underline{x}_k$ or $\underline{x}_{k+1}$)

2. Select a search direction $\underline{s}_k$ (or $\underline{p}_k$)
   - can have a line in 2-D, 3-D, or 4-D
   - we can unequivically define this direction in our n-D space
     ↳ if we restrict ourselves to move along this direction, we sweep out a line
   - in general we choose our search direction in a direction that decreases the function
     ↳ descent direction
   - in most problems there are an infinite number of descent directions that decrease the objective function
     ↳ steepest descent decreases the function the fastest
     => unique direction

$$\varphi'(0) = \left.\frac{d\varphi}{d\alpha}\right|_{\alpha=0} = \nabla f^T(\underline{x}_k)\,\underline{p} < 0$$

$$\left.\frac{df}{d\alpha}\right|_{\underline{x}_{k-1}} = \underline{s}_k^T \nabla f(\underline{x}_{k-1}) < 0$$

↑ dot product

both of these are saying the same thing: descent direction

If you do not have a descent direction, you cannot be guaranteed to find a minimum

$\underline{s}_k$ (or $\underline{p}_k$) – direction vector that defines the line along which we're searching
↳ like an arrow drawn in the design space in the direction in which we're searching

descent direction – direction in multidimensional space such that, if we move in that direction, we decrease the objective function

Comments:

Selecting a descent direction is not trivial
→ Steepest descent is not always the best option
(there are techniques that can have better performance)

LINE SEARCH METHODS

How does it work?

3. Select a distance α^* (or α_k) to move in the search direction
   - estimating α^* requires a search along the line, defined by the direction s_k, to find an estimate of the minimum along the line
   - α^* is a scalar that (approximately) minimizes the slice of the function along the line ← does not minimize in the entire space

   examples of approaches to find α^*: Polynomial Approximations
   Golden Section Method

4. Decide when to stop iterating
   ↳ decide when the process has converged to an acceptable solution

   examples of knowing when to stop: Wolfe Conditions

LINE SEARCH METHODS

Example:

Presume that a line search algorithm conducts a single line search starting from a point $[3,2]$ in the direction $[1, 0.5]$. The algorithm chooses a distance of 0.5. At what point does the algorithm stop?

Note that the search direction is not normalized to a unit vector.

Solution:

Equation of line search: $\underline{x}_k = \underline{x}_{k-1} + \alpha \underline{s}_k$

Given: $\underline{x}_{k-1} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$, $\underline{s}_k = \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$, $\alpha = 0.5$, $\underline{s}_k$ is not normalized

$$\underline{x}_k = \begin{bmatrix} 3 \\ 2 \end{bmatrix} + 0.5 \begin{bmatrix} 1 \\ 0.5 \end{bmatrix} \Rightarrow \begin{aligned} x_1 &= 3 + 0.5(1) = 3.5 \\ x_2 &= 2 + 0.5(0.5) = 2.25 \end{aligned}$$

Algorithm stops at $[3.5, 2.25]$

Example:

Consider the function $f(\underline{x}) = x_1^2 - 2x_1 + 4x_2^2 + 1$. Do a line search in the $[1, 0]$ direction begining at an initial point of $\underline{x} = [2, 0]$. Use calculus methods to perform the line search analytically.

Solution:

Equation of line search: $\underline{x}_k = \underline{x}_{k-1} + \alpha \underline{s}_k$

Given: $\underline{x}_{k-1} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $\underline{s}_k = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $f(\underline{x}) = x_1^2 - 2x_1 + 4x_2^2 + 1$

$$\underline{x}_k = \begin{bmatrix} 2 \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \begin{aligned} x_1 &= 2 + \alpha \\ x_2 &= 0 \end{aligned}$$

Plug in to $f(\underline{x})$: $f(\alpha) = (2+\alpha)^2 - 2(2+\alpha) + 4(0)^2 + 1$

$$= 4 + 4\alpha + \alpha^2 - 4 - 2\alpha + 1$$
$$= \alpha^2 + 2\alpha + 1 = Q(\alpha)$$

Set $\frac{df}{d\alpha} = 0$: $\frac{df}{d\alpha} = 2\alpha + 2 = 0 \Rightarrow \alpha = -1$

Plug back in to line search equation: $\underline{x}_k = \begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1 \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow \begin{aligned} x_1 &= 1 \\ x_2 &= 0 \end{aligned}$

After the first step, $\underline{x} = [1, 0]$

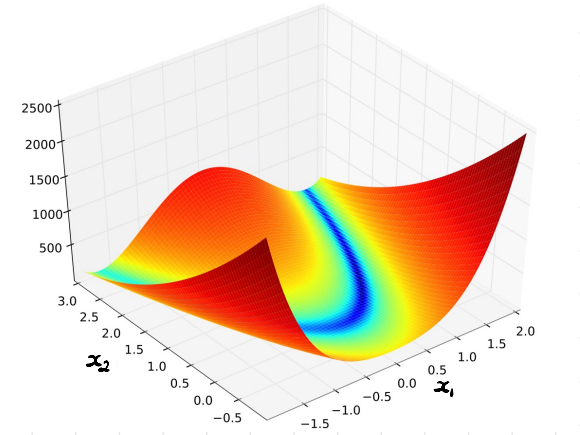
Example:

Consider the Rosenbrock function:

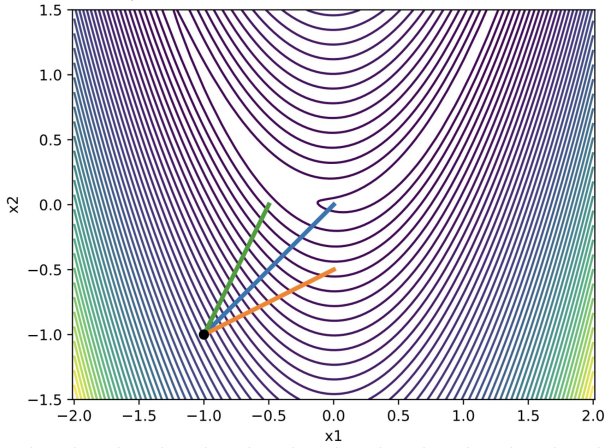
$$f(\underline{x}) = (1-x_1)^2 + 100(x_2 - x_1^2)^2$$

$\underline{x}_{k-1} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$, consider $\underline{s}_k = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $\underline{s}_k = \begin{bmatrix} 1/2 \\ 1 \end{bmatrix}$, and $\underline{s}_k = \begin{bmatrix} 1 \\ 1/2 \end{bmatrix}$

3-D plot:

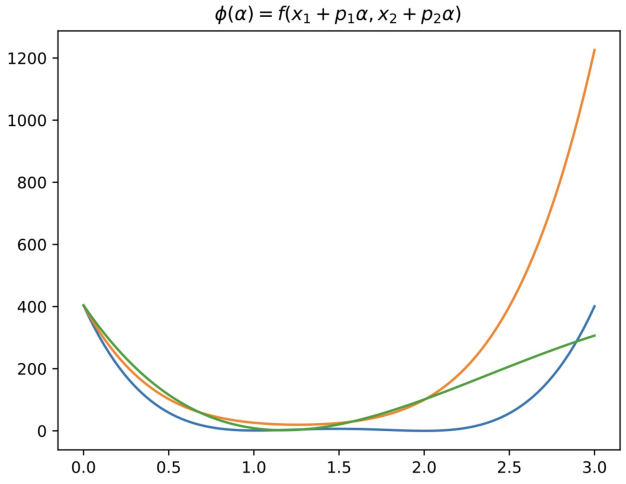


Contour plot:



$\underline{s}_k = [1, 1]$ $\underline{s}_k = [1, 1/2]$ $\underline{s}_k = [1/2, 1]$

Graph of $\varphi(\alpha)$ for each $\underline{s}_k$



← $\varphi(\alpha)$ is the equation of f parameterized in terms of α ($f(\alpha)$)

Using $\underline{x}_{k-1} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$ and $\underline{s}_k = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$:

$$\underline{x}_k = \underline{x}_{k-1} + \alpha \underline{s}_k = \begin{bmatrix} -1 \\ -1 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow \begin{aligned} x_1 &= -1+\alpha \\ x_2 &= -1+\alpha \end{aligned}$$

plug in to $f(\underline{x})$: $(1-(\alpha-1))^2 + 100((\alpha-1) - (\alpha-1)^2)^2$

$= (2-\alpha)^2 + 100(\alpha - 1 - (\alpha^2 - 2\alpha + 1))^2$

Set $\varphi'(\alpha) = 0$: $\varphi'(\alpha) = 2(2-\alpha)(-1) + 200(\alpha - 1 - (\alpha^2 - 2\alpha + 1))(1 - 2\alpha + 2) = 0$

$= -4 + 2\alpha + 200(-\alpha^2 + 3\alpha - 2)(1 - 2\alpha + 2) = 0$

$= -4 + 2\alpha + 200(-\alpha^2 + 2\alpha^3 - 2\alpha^2 + 3\alpha - 6\alpha^2 + 6\alpha - 2 + 4\alpha - 4) = 0$

$= -4 + 2\alpha + 200(2\alpha^3 - 9\alpha^2 + 13\alpha - 6) = 0$

$= -4 + 2\alpha + 400\alpha^3 - 1800\alpha^2 + 2600\alpha - 1200 = 0$

$= 400\alpha^3 - 1800\alpha^2 + 2602\alpha - 1204 = 0$

$$\Rightarrow \alpha = \frac{20 - \sqrt{23}}{20}, \quad \frac{20 + \sqrt{23}}{20}, \quad 2$$

↑ local minima ↑ local maxima ↑ global minima