CHEMICAL ENGINEERING 4G03

Module 09

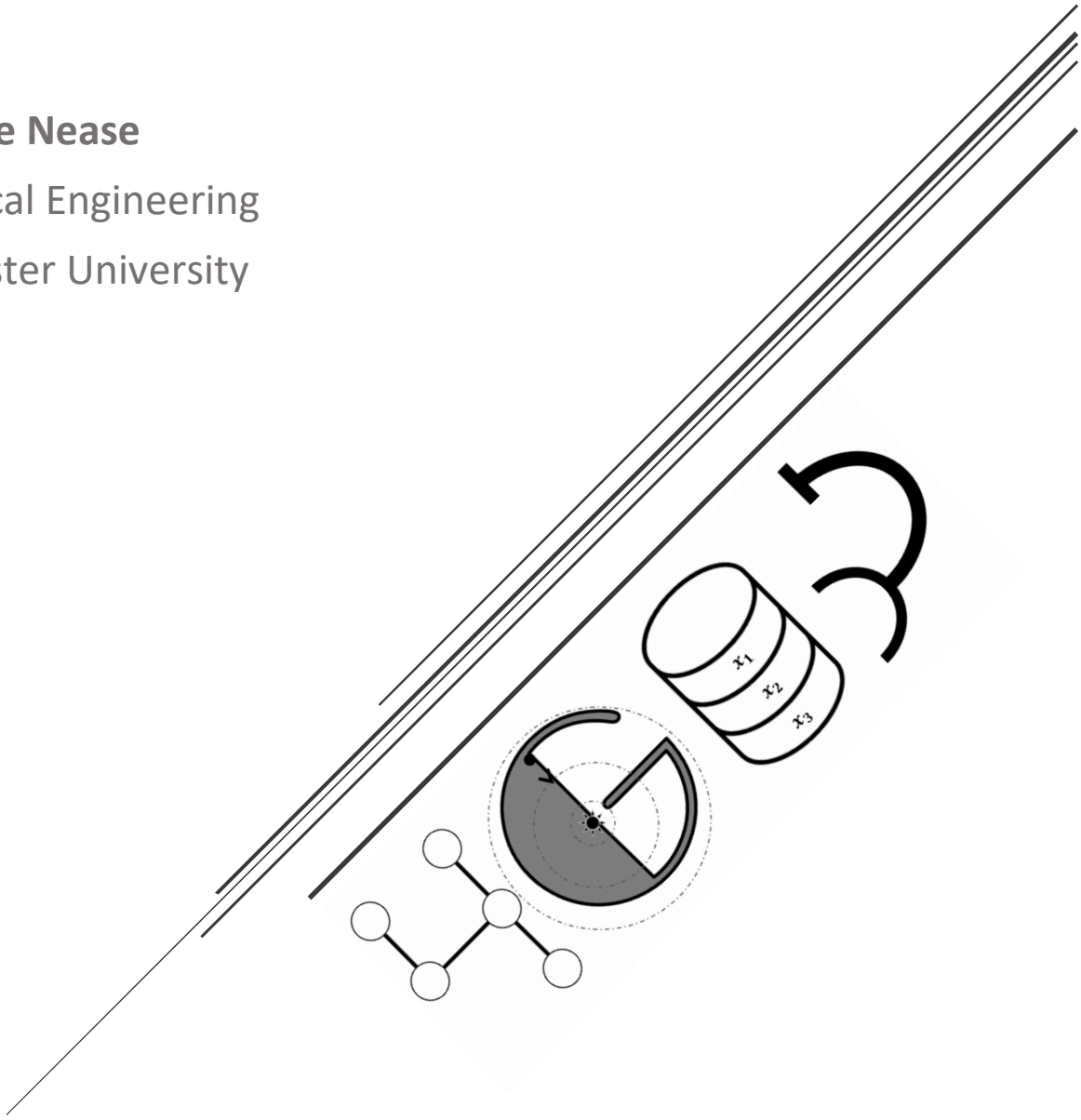
Nonlinear Programming (II)

Derivative-Based Search Methods

Dr. Jake Nease

Chemical Engineering

McMaster University



Updated March 29, 2023

Outline of Module

Suggested Readings

Rardin (1st edition): Chapter 13.5-13.7

Rardin (2nd edition): Chapter 16.5-16.7

Introduction and Perspective

We now know how to formulate (since it is really no different than the LP or MILP programming sections) and what the necessary and sufficient conditions for unconstrained optimality are. The next question is: **How do we actually solve nonlinear programs**? This is analogous to using the Simplex Search for LPs and the Branch and Bound for MILPs (recall that the B&B actually uses the Simplex Search to solve all candidate problems at each node!).

However, we have a bit of a problem with NLPs, as many of you have already experienced. Their troublesome nature of having multiple local optima, or perhaps even saddle points that APPEAR to be optima, result in some frustrating complications. Such complications include:

- Multiple optima mean that we might find any one of them for a certain **initial guess**.
- Different approximations, calculations, and algorithms can find different optima.
- Many algorithms will tend to fail if they get caught in a flat region of the objective function because **all derivative-based methods** require local information of the objective function's curvature.

Either way, we still want to investigate some methods for finding (at least local) optima. This is not necessarily a "something is better than nothing" mentality, but we have to start somewhere. I would also like to take the chance to emphasize that **solving NLPs and MINLPs to global optimality is still a cutting-edge problem that is the topic of many graduate and industry projects**. Our standard search idea, as we described all the way back at the beginning of the course, follows the **improving search** methodology:

Algorithm: Continuous Improving Search

1. **INITIALIZATION**
   Select any feasible starting point $x^{(0)}$ with counter $k = 0$.

2. **MOVE DIRECTION**
   If no improving feasible direction Δx can be found, STOP.
   OTHERWISE, determine an improving feasible direction Δx.

3. **STEP SIZE**
   If there is *no limit* to the improvement when going in the direction Δx while also maintaining feasibility, STOP: Model is *unbounded*.
   OTHERWISE, choose the largest allowable step size α that improves the objective while remaining feasible.

4. **UPDATE**
   $x^{(k+1)} = x^{(k)} + \alpha \Delta x$
   $k = k + 1$
   Return to step (2)

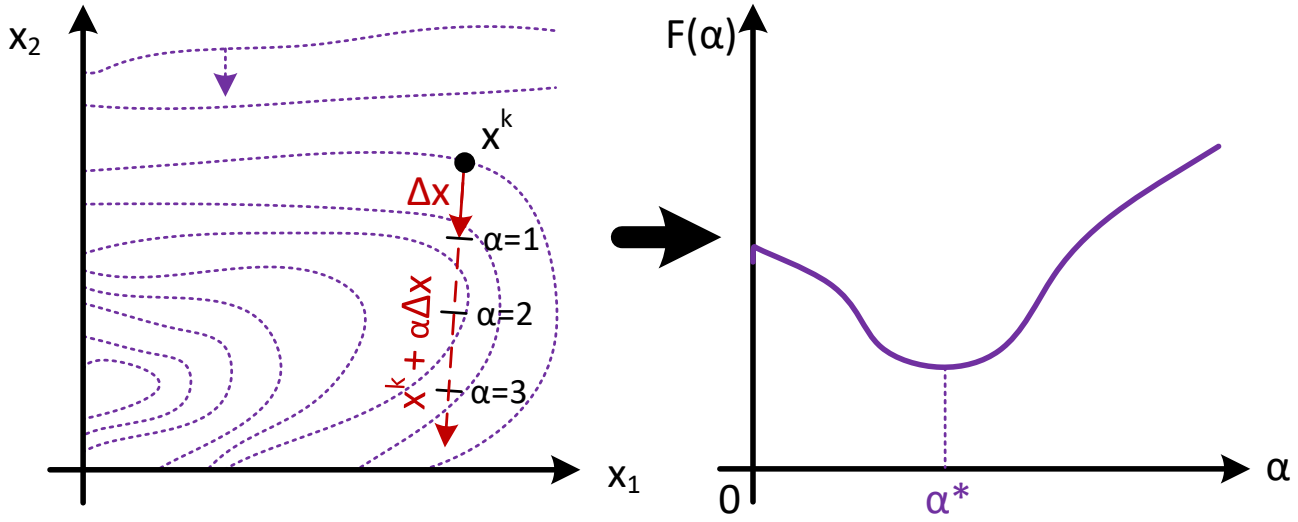
In this section we will focus on three main methods of finding a **search direction Δx**, each building on each other by adding additional derivative information or approximating that information to aid convergence:

1. Steepest Descent.
2. Newton Search.
3. Quasi-Newton Search.

That being said, each of the methods above only finds us a **direction**. We have done this before using the Simplex Search, although we were able to avoid using derivatives since we were dealing only with LPs. The OTHER thing we need is a step size α. For this I would like to introduce a concept known as the **Line Search**.

One-Dimensional Line Search

Let's say for the sake of argument that we have identified a search direction Δx from our current iterate during x^k. The figure below shows such a scenario. The new question is: **For how far do I follow the direction Δx before determining a new direction**? In the scenario below we could use $\alpha = 1$ or even $\alpha = 2$. As a matter of fact, $\alpha = 3$ still provides us with an *improvement from our current iterate*, but less so than the other two α values.



HOWEVER, if I follow the projection of the objective contours onto the search direction $x^k + \alpha\Delta x$, I end up with the figure on the right, which is a **one-dimensional problem in which I am trying to locate the optimum $\alpha^\star$**. In this case, α is actually my *search variable*. Here is the general idea:

- Once I have a search direction, I can technically compute the objective function $\phi(x)$ for any values of $x^k + \alpha\Delta x$.
- I am looking to follow the direction Δx until it no longer results in an improving objective function. This corresponds to the *step length* $\alpha^\star$.
- **In other words**, I am looking for the α in which the function $\phi(x^k + \alpha\Delta x)$ is *MINIMIZED* (for a minimization program)

What does this all mean? Well, it means that the "perfect" step size $\alpha^\star$ is precisely the solution to the **line search optimization program**:

$$\boxed{\min_{\alpha} F(\alpha) \triangleq \phi(x^k + \alpha\Delta x)}$$

Linear Search Problem

The step length $\alpha^\star$ that results in the best possible improvement in the objective function $\phi(x)$ in some search direction Δx is **exactly equal to the global optimum of the unconstrained problem**:

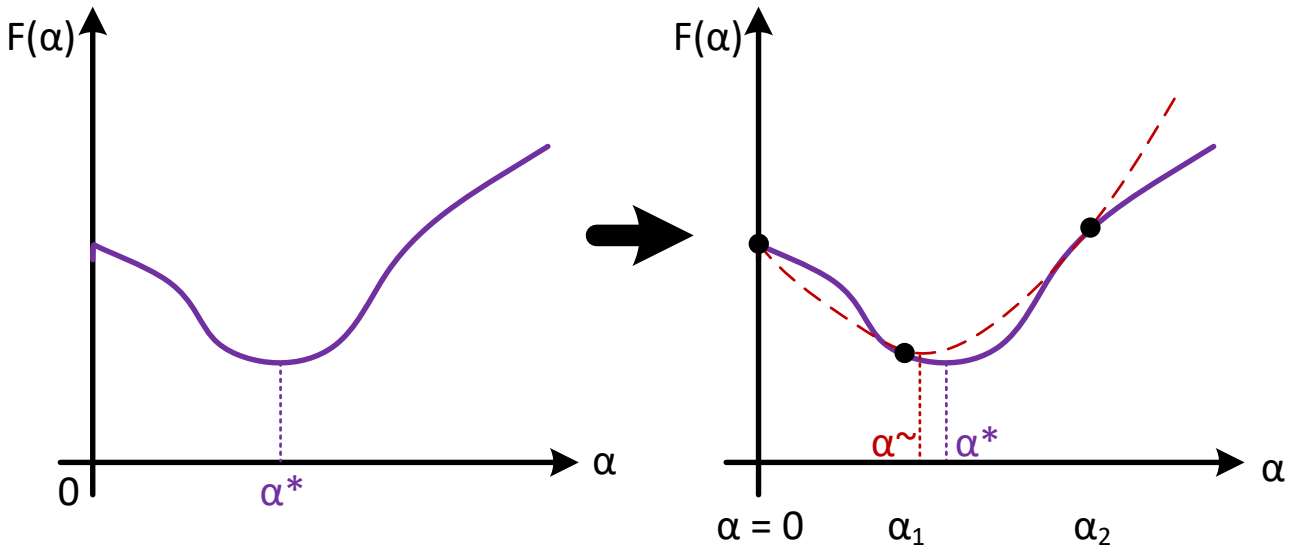
$$\min_{\alpha} F(\alpha) \triangleq \phi(x^k + \alpha \Delta x)$$

Quadratic Fit Lagrange Polynomials

The idea that we have to solve the line search problem is a good one, but one question remains: How do we actually find the optimum of $\min_{\alpha} F(\alpha) \triangleq \phi(x^k + \alpha \Delta x)$ if we are unable to compute an analytical expression for $F(\alpha)$? You should notice that it is actually not possible to explicitly define this equation, especially considering we could be at any location x^k. Time to put on our 3E04 thinking hats and do something clever! Since the objective function $F(\alpha)$ is one-dimensional, I can approximate it using a **best fit approach!** Consider the following proposal:

> Instead of trying to find the minimum of the function $F(\alpha)$, instead lets find the minimum of a one-dimensional *approximation* of $F(\alpha)$. If we choose this approximation to be *quadratic*, we are guaranteed to find an approximate value for $\alpha^\star$.

For example, we might want to fit the curve in the figure below. In this figure, the red dashed line represents the quadratic fit by taking three sample points along the actual function $F(\alpha)$. The minimum of the approximation is $\alpha^\sim$, which is (ideally) close to $\alpha^\star$.



The only question left is **how do we determine the fit**? Well, we can use Lagrange Polynomials! The idea is as follows:

- I evaluate three points of the function $F(\alpha)$ for three different values of α (usually 0, 1, and 2).
- I fit a quadratic polynomial p_2 that passes through those three points *exactly*.
- I can then determine the **analytical** derivative of the polynomial as the point where $\frac{dp_2}{d\alpha} = 0$.
- The value of α at which p_2 is minimized corresponds to $\alpha^\sim$, which approximates $\alpha^\star$.

Lagrange Polynomials

Given $N+1$ data points on a function $f(x)$, there is **one and only one** polynomial of order N that pass through **all** of those points exactly:

$$\boldsymbol{p_N(x) \triangleq a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N}$$

One such way of computing this interpolation is through the use of **Lagrange Polynomials**:

$$p_N(x) \triangleq \sum_{k=0}^{N} f(x_k)\mathcal{L}_x(x)$$

Where the Lagrange Multiplier $\mathcal{L}_k(x)$ is defined as:

$$\mathcal{L}_k(x) \triangleq \prod_{\substack{i=1 \\ i\neq k}}^{N} \frac{x - x_i}{x_k - x_i}$$

For a **quadratic polynomial p_2** that passes through the points $(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))$ we may explicitly define the polynomial equation as:

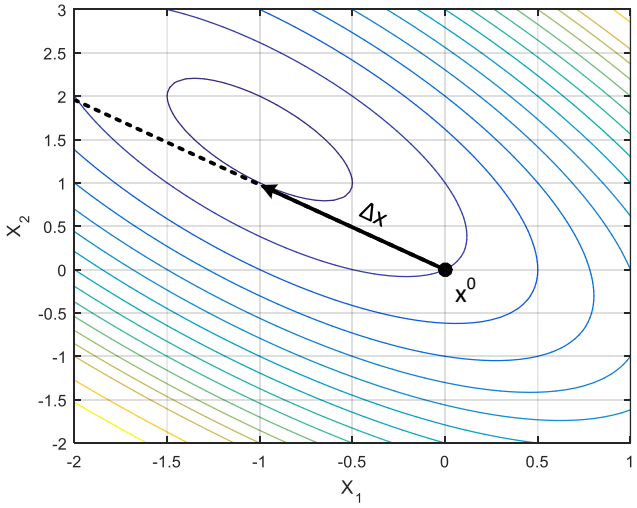
$$p_2(x) = f(x_1)\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + f(x_2)\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

The polynomial $p_2(x)$ has the **unique global optimum** $x^{\sim}$ corresponding to the analytical solution:

$$\boxed{\tilde{\boldsymbol{x}} \triangleq \frac{1}{2}\frac{\boldsymbol{f(x_1)(x_2^2 - x_3^2) + f(x_2)(x_3^2 - x_1^2) + f(x_3)(x_1^2 - x_2^2)}}{\boldsymbol{f(x_1)(x_2 - x_3) + f(x_2)(x_3 - x_1) + f(x_3)(x_1 - x_2)}}}$$

Performing the Line Search

OK, here is the deal: if given a search direction, we want to determine the best possible value for α that yields the optimal improvement in $\phi(x)$, which is denoted as α^*. **However**, we are typically satisfied with choosing a value for α that is *good enough*. That is, it yields any amount of improvement at all. For this reason we can safely use $\alpha^{\sim}$ as an analog to α^*. Consider the figure below at point x^0 and a certain search direction Δx. Let's perform an interpolating line search for this scenario.



Class Workshop – Line Search

Consider the unconstrained NLP and perform a line search in the direction Δx from the point x^0 using the interpolative values of α as $\alpha = 0.5$, $\alpha = 1$, $\alpha = 2$.

$$\min_x \phi(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2 \qquad \Delta x = [-1\ 1]^T \qquad x^0 = (0,0)$$

Workshop Solution – Line Search

α	$x_1^0 + \alpha\Delta x_1$	$x_2^0 + \alpha\Delta x_2$	$F(\alpha) = \phi(x^0 + \alpha\Delta x)$
0			
0.5			
1			
2			

$$\alpha^\star \approx \tilde{\alpha} = \frac{1}{2}\frac{F(\alpha_1)(\alpha_2^2 - \alpha_3^2) + F(\alpha_2)(\alpha_3^2 - \alpha_1^2) + F(\alpha_3)(\alpha_1^2 - \alpha_2^2)}{F(\alpha_1)(\alpha_2 - \alpha_3) + F(\alpha_2)(\alpha_3 - \alpha_1) + F(\alpha_3)(\alpha_1 - \alpha_2)} = \_\_\_\_\_$$

$$f(x^0 + \alpha\Delta x) = \_\_\_\_\_$$

As long as $f(x^k + \alpha\Delta x) < f(x^k)$, the line search should be accepted.

Algorithm Statement: One-Dimensional Line Search

When provided with a direction Δx and a current point x, below is the rudimentary line search algorithm. It should be noted that additional statements may be included (such as extending α or adding a minimum acceptable improvement), but they are optional and omitted for simplicity.

1. **Compute $F(\alpha)$**
   Select 3 values of α (default: 0.5, 1, 2).
   Compute $F(\alpha)$ for all values of α.

2. **Determine Line Search Optimum**
   Compute $\tilde{\alpha} = \frac{1}{2}\frac{F(\alpha_1)(\alpha_2^2-\alpha_3^2)+F(\alpha_2)(\alpha_3^2-\alpha_1^2)+F(\alpha_3)(\alpha_1^2-\alpha_2^2)}{F(\alpha_1)(\alpha_2-\alpha_3)+F(\alpha_2)(\alpha_3-\alpha_1)+F(\alpha_3)(\alpha_1-\alpha_2)}$.

3. **Assess Improvement**
   If $f(x + \tilde{\alpha}\Delta x) < f(x)$ **STOP** → Assign improving search direction $\alpha \leftarrow \tilde{\alpha}$.
   Else **UPDATE** → Replace $\alpha_i \leftarrow 0.5\alpha_i$
   Return to step (1).

Optimization Algorithm: Steepest Descent

We now move on to the second part of the problem: **determining the search direction**. We are going to cover three ways of doing this, the first of which is the steepest descent method.

Steepest Descent Direction

The steepest descent method is grounded in the most basic of geometric interpretations, in which we know the **gradient** $\nabla\phi(x)$ of an objective function $\phi(x)$ is in fact the direction of steepest descent. We may therefore compute the search direction Δx^{k+1} as:

Steepest Descent Direction

At any x^k with an objective function gradient $\nabla\phi(x^k) \neq 0$, the **steepest** rate of improvement is obtained by selecting a search direction Δx^{k+1} equal to the gradient:

$$\boldsymbol{\Delta x^{k+1} \triangleq \pm \nabla \phi(x^k)}$$

NB – use (+) for a maximization, (-) for a minimization.

Once the direction Δx^{k+1} has been determined, we may use a line search to yield the next iteration for the steepest descent improving search.

Algorithm Statement: Steepest Descent

1. **Initialization**
   Choose an *initial guess* x^0 and a stopping tolerance $\epsilon > 0$.
   Set the iterate counter $k = 0$.

2. **Determine Gradient Direction**
   Compute the objective function gradient $\nabla\phi(x^k)$.

3. **Stopping Criteria**
   If $\left|\left|\nabla\phi(x^k)\right|\right| < \epsilon$: **STOP** → x^k is a stationary point and satisfies the 1st order NCO.
   Report x^k as an **approximate local solution**.

4. **Steepest Descent Step**
   Direction: Set the move direction $\Delta x^{k+1} = \pm\nabla\phi(x^k)$.

   Line Search: Solve (approximately) 1D Line Search: $\min_{\alpha} F(\alpha) = \phi(x^k + \alpha\Delta x^{k+1})$.
   Record step length as Line Search solution: $\alpha^{k+1} = \alpha^\star$.

   Update: Record $x^{k+1} = x^k + \alpha^{k+1}\Delta x^{k+1}$.

   Increment: $k = k + 1$.
   Return to step (2).

Of course, this would be **terribly tedious** to do by hand, especially since it might take many iterations. As with all numerical methods, computer implementations are the way to go.

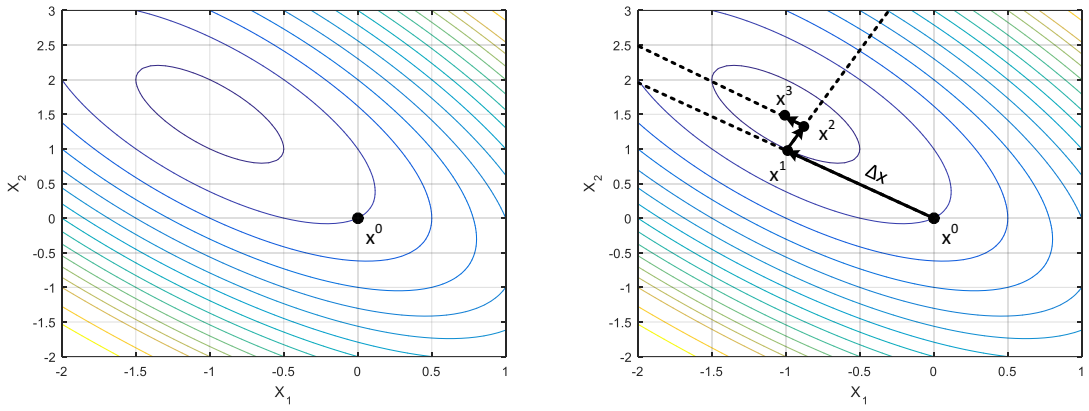
Class Workshop – Steepest Descent

Consider the unconstrained NLP from before with the initial point x^0:

$$\min_x \phi(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2 \qquad x^0 = (0,0)$$

1. Use the steepest descent method to find a search direction Δx.
2. Apply a line search to obtain the step size α (*hint – we have already done it!*).
3. Sketch one iteration of the steepest descent method.
4. Sketch the remaining iteration procedure for the steepest descent method.

Workshop Solution – Steepest Descent



Pros and Cons of Steepest Descent

PROS

- Approach is appealingly straightforward.
- Approach is robust (will always give an improving direction, if one exists).

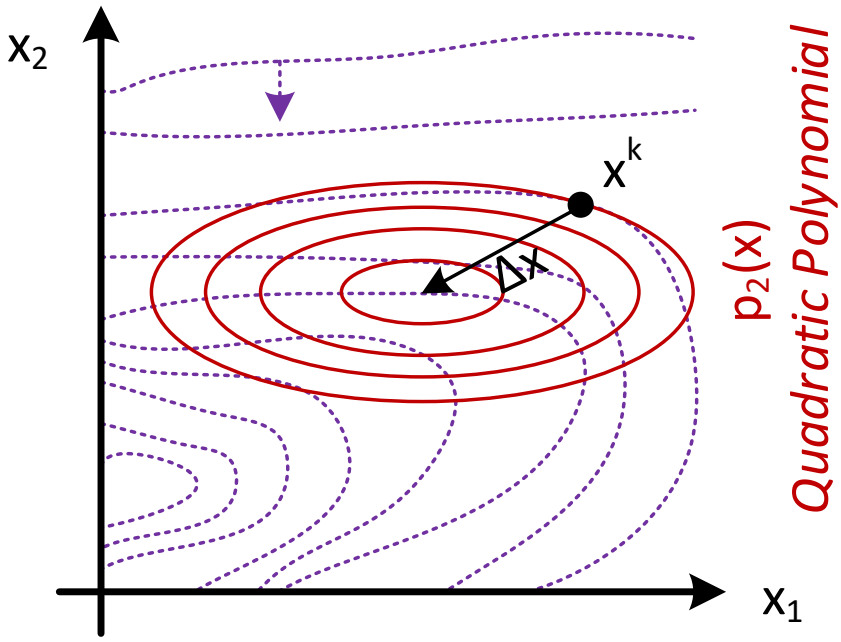
CONS

- Can (and won't hesitate to) converge to a saddle point.
- Convergence is slow due to "zig-zagging" near the optimum.
- Very sensitive to numerical errors (gradient usually has to be taken numerically).

Can we do better if we introduce new information, such as the **second derivative**?

Optimization Algorithm: Newton Search

Perhaps we can speed convergence of our algorithm by including **second-order** curvature information. With this in mind, consider the figure below in which the red ellipses correspond to a *quadratic approximation* of the function $\phi(x)$.



We desire to orient the direction Δx so that we find the *minimum* of the quadratic approximation $p_2(x)$ with respect to Δx. In other words, we want to **find the value of Δx such that**:

$$\nabla_{\Delta x}[p_2(x+\Delta x)] = 0$$

In words, it means we are looking for the value of Δx that makes our polynomial p_2 have a gradient of zero at the coordinate point $(x+\Delta x)$. Your first question (should) be "OK, what is $p_2(x)$?" I am glad you asked, it comes from the **Taylor Series Expansion of $\phi(x)$**:

$$p_2(x+\Delta x) \triangleq \phi(x) + \nabla\phi(x)\Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x$$

The equation above looks just like our second-order Taylor Series from the previous module, except we have replaced the candidate direction δ with the *desired search direction* Δx. What fun! This lets us define the Newton Direction as follows.

Newton Direction (Minimization)
At any x^k with an objective function $\phi(x^k)$, the **Newton Direction** Δx^{k+1} is obtained by solving for the value of Δx such that $\nabla_{\Delta x}[p_2(x+\Delta x)] = 0$. This is equivalent to:

$$\boldsymbol{\Delta x^{k+1} \triangleq H(x^k)^{-1}[-\nabla\phi(x^k)]}$$

***OR** the solution to the system of linear equations*:

$$\boxed{\boldsymbol{H(x^k)\Delta x^{k+1} = -\nabla\phi(x^k)}}$$

Algorithm Statement: Newton Search

The differences between the Newton Search and the Steepest Descent Search are given in **purple**.

1. **Initialization**
   Choose an *initial guess* x^0 and a stopping tolerance $\epsilon > 0$.
   Set the iterate counter $k = 0$.

2. **Determine Gradient Direction**
   Compute the objective function gradient $\nabla\phi(x^k)$ and Hessian $H(x^k)$.

3. **Stopping Criteria**
   If $\left|\left|\nabla\phi(x^k)\right|\right| < \epsilon$: **STOP** → x^k is a stationary point and satisfies the 1st order NCO.
   Report x^k as an **approximate local solution**.

4. **Newton Step**
   Direction: Set the move direction Δx^{k+1} as the **solution** to $\boldsymbol{H(x^k)\Delta x^{k+1} = \pm\nabla\phi(x^k)}$.

   Line Search: Solve (approximately) 1D Line Search: $\min_{\alpha} F(\alpha) = \phi(x^k + \alpha\Delta x^{k+1})$.
   Record step length as Line Search solution: $\alpha^{k+1} = \alpha^\star$.

   Update: Record $x^{k+1} = x^k + \alpha^{k+1}\Delta x^{k+1}$.

   Increment: $k = k + 1$.
   Return to step (2).

Pros and Cons of Newton Search

PROS

- Excellent performance when close to the optimum (why?).
- Uses higher-order information and is thus less sensitive to numerical errors than Steepest Descent.

CONS

- Results are very sensitive to the starting point x^0 – can still find saddle points.
- Could diverge if starting location is too far from optimum.
- Hessian matrix must be computed (usually numerically) at every iteration! Furthermore, we need to compute Δx as a solution to a system of equations. **Can be burdensome** for large-scale optimization programs (lots of variables).

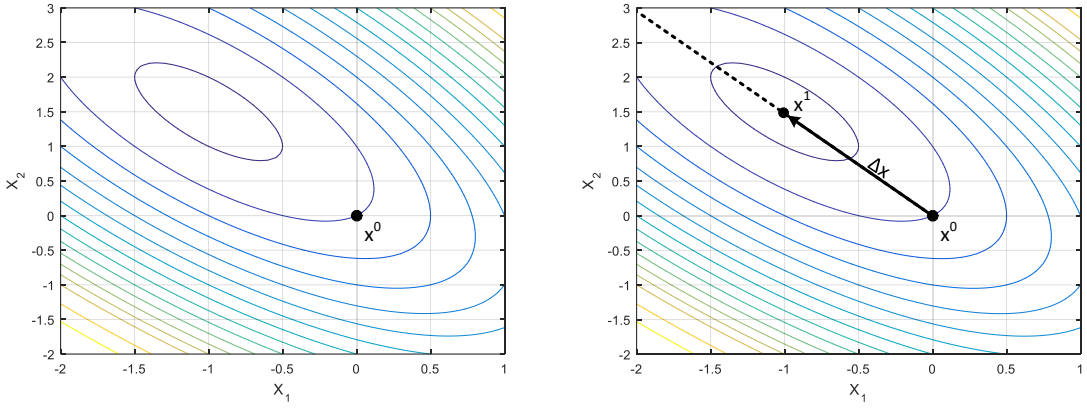
Class Workshop – Newton Search

Consider the unconstrained NLP from before with the initial point x^0:

$$\min_x \phi(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2 \qquad x^0 = (0,0)$$

1. Use the Newton method to find a search direction Δx.
2. Argue why the step size α found via a line search will be exactly $\alpha = 1$.
3. Sketch one iteration of the Newton Search method.

Workshop Solution – Newton Search



Wait… What?

Well, the only thing left to do now is try to mitigate the computational annoyances of computing the inverse Hessian (or solving a system of linear equations with the Hessian).

Blending First- and Second-Order Methods: Deflection Matrices

The **idea** behind a deflection matrix is actually fairly clever. We wish to *combine* first- and second-order methods together to exploit their advantages (robustness and computational simplicity for first-order, convergence rate and iterative performance of second-order) and mitigate their weaknesses. How? We will use a so-called **Deflection Matrix** D.

Deflection Matrix

A deflection matrix D produces a **modified gradient search direction**:

$$\boxed{\Delta \boldsymbol{x}^{\boldsymbol{k+1}} \triangleq \pm \boldsymbol{D}^{\boldsymbol{k}} \nabla \boldsymbol{\phi}\left(\boldsymbol{x}^{\boldsymbol{k}}\right)}$$

This might seem familiar, because we have seen two deflection matrices already!

- For **Steepest Descent**: $D^k \triangleq I$
- For **Newton Search**: $D^k \triangleq H(x^k)^{-1}$

So, if we want to improve convergence speed, is it possible to **approximate the Hessian inverse** by using a deflection matrix? Since the Hessian reflects the *rate of change* of the gradient we can write:

$$\nabla\phi(x^{k+1}) - \nabla\phi(x^k) \approx H(x^k)\underbrace{(x^{k+1} - x^k)}_{\Delta x}$$

Therefore if we want to represent the **Hessian inverse** with some symmetric matrix D^k, we desire a matrix that obeys the so-called quasi-Newton condition:

$$\boldsymbol{H(x^k)^{-1}}[\nabla\phi(x^{k+1}) - \nabla\phi(x^k)] \approx \cancel{\boldsymbol{H(x^k)^{-1}}\, H(x^k)}\underbrace{(x^{k+1} - x^k)}_{\Delta x}$$

$$\boxed{\boldsymbol{D^k}[\nabla\phi(x^{k+1}) - \nabla\phi(x^k)] = x^{k+1} - x^k}$$

$$\boxed{D^k = (D^k)^T}$$

As one more condition, we also require to preserve that the search direction Δx always **improves**. This was analogous to selecting the positive or negative search direction in our Newton Search and Steepest Descent. If we want to make sure the deflection matrix **does not change the sign of our search direction**, we require only one thing: D^k must be POSITIVE DEFINITE!

$$\boxed{D^k \succ 0}$$

The BFGS Formula

There are several ways to compute a deflection matrix D^k that obeys the conditions above, but the best-known version used these days is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula:

BFGS Formula for a Deflection Matrix

A symmetric, positive definite Hessian approximation deflection matrix D^k can be computed as:

$$\boxed{\boldsymbol{D^{k+1} = D^k + \left(1 + \frac{g^T D^k g}{d^T g}\right)\frac{dd^T}{d^T g} - \frac{D^k g d^T + d g^T D^k}{d^T g}}}$$

In the above equation, the vectors d and g are:

- $d \triangleq x^{k+1} - x^k$.
- $g \triangleq \nabla\phi(x^{k+1}) - \nabla\phi(x^k)$.

The deflection matrix must be initialized with a symmetric positive definite matrix, usually I.

$$D^0 \triangleq I$$

- **Every** subsequent D^k is *guaranteed to be symmetric*.
- **Every** subsequent D^k is *guaranteed to be positive definite*.
- **Upon convergence** to a local optimum $x^\star$, D^k approaches $H(x^k)^{-1}$ **exactly**.

Optimization Algorithm: Quasi-Newton Search

The Quasi-Newton Search is the same as the Newton Search, but eschews the computation of the Hessian inverse in favour of approximating it via the BFGS deflection matrix!

Algorithm Statement: Quasi-Newton Search

The differences between the Quasi-Newton Search and the Newton Search are given in **purple**.

1. **Initialization**
   Choose an *initial guess* x^0, a stopping tolerance $\epsilon > 0$ and an initial deflection $D^0 = I$.
   Set the iterate counter $k = 0$.

2. **Determine Gradient and Stopping**
   Compute the objective function gradient $\nabla\phi(x^k)$
   If $\left|\left|\nabla\phi(x^k)\right|\right| < \epsilon$: **STOP** → x^k is a stationary point and satisfies the 1st order NCO.
   Report x^k as an **approximate local solution**.

3. **Quasi-Newton Step**
   Direction: Set the move direction Δx^{k+1} as $\boldsymbol{\Delta x^{k+1} = \pm D^k \nabla\phi(x^k)}$

   Line Search: Solve (approximately) 1D Line Search: $\min_\alpha F(\alpha) = \phi\left(x^k + \alpha\Delta x^{k+1}\right)$.
   Record step length as Line Search solution: $\alpha^{k+1} = \alpha^\star$.

4. **Update**
   Iterate: Record $x^{k+1} = x^k + \alpha^{k+1}\Delta x^{k+1}$.

   Deflection Matrix: $D^{k+1} = D^k + \left(1 + \frac{g^T D^k g}{d^T g}\right)\frac{dd^T}{d^T g} - \frac{D^k g d^T + d g^T D^k}{d^T g}$
   $d = x^{k+1} - x^k$
   $g = \nabla\phi\left(x^{k+1}\right) - \nabla\phi(x^k)$

   Increment $k = k + 1$
   Return to step (2).

This may seem like a pain, but again I want to emphasize that this kind of thing is actually done rather **routinely** using computer software. It doesn't hurt to make sure we understand the math, though!

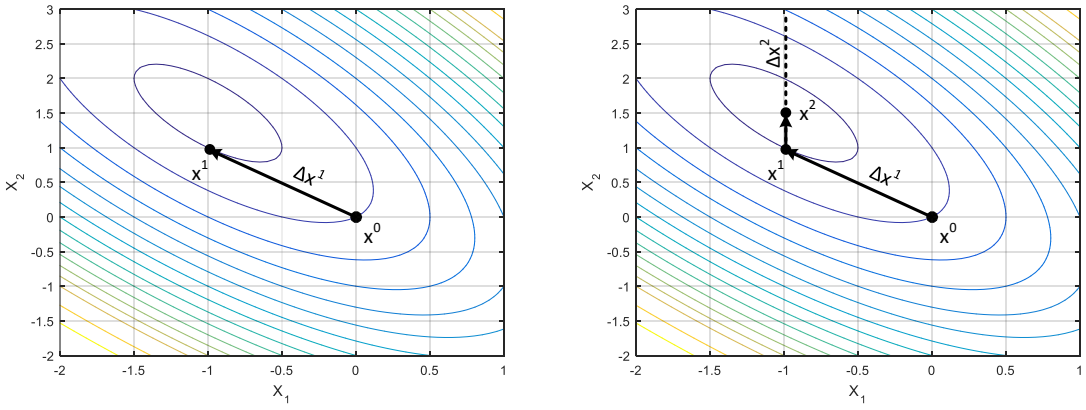
Class Workshop – Quasi-Newton Search

Consider the unconstrained NLP from before with the initial point x^0, second point x^1 and D^0:

$$\min_x \phi(x) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2 \qquad x^0 = (0,0) \qquad x^1 = (-1,1) \qquad D^0 = I$$

1. Compute D^1 using the BFGS formula.
2. Compute the Quasi-Newton direction Δx^2 at the point x^1.
3. Sketch the second iteration of the Quasi-Newton Search.

Workshop Solution – Quasi-Newton Search



Conclusions

YIKES! That was heavy. However, let's go over the main points.

- Search directions Δx can be obtained by using the gradient, gradient/hessian, or gradient/deflection matrix to point us toward a local optimum.
- A 1D line search is used to determine the step size α for which to follow a proposed direction Δx.
- All algorithms involve the repetitive use of the **same** gradient/hessian/BFGS formula evaluated at different locations x^k.
- Gradients and Hessians are typically evaluated **numerically** using computer software.

And this, ladies and gentlemen, is how GAMS (or any optimization package) may choose to solve your NLP. However, note the limitation of this section, which is that the NLP is **unconstrained**. Constrained

NLPs are often solved using constraint elevation or relaxation techniques, which we will touch on briefly in the last section.

Our final wrinkles to sort out are: what if I do not have access to derivative information? Alternatively, what if the derivative is particularly cumbersome to calculate, or I can't compute the Hessian (or BFGS) in a reasonable amount of time? What if there are multiple local optima and I want to find the global optimum? We will look at derivative-free and scattered methods in the next section.

~~ END OF MODULE ~~

\|T|/