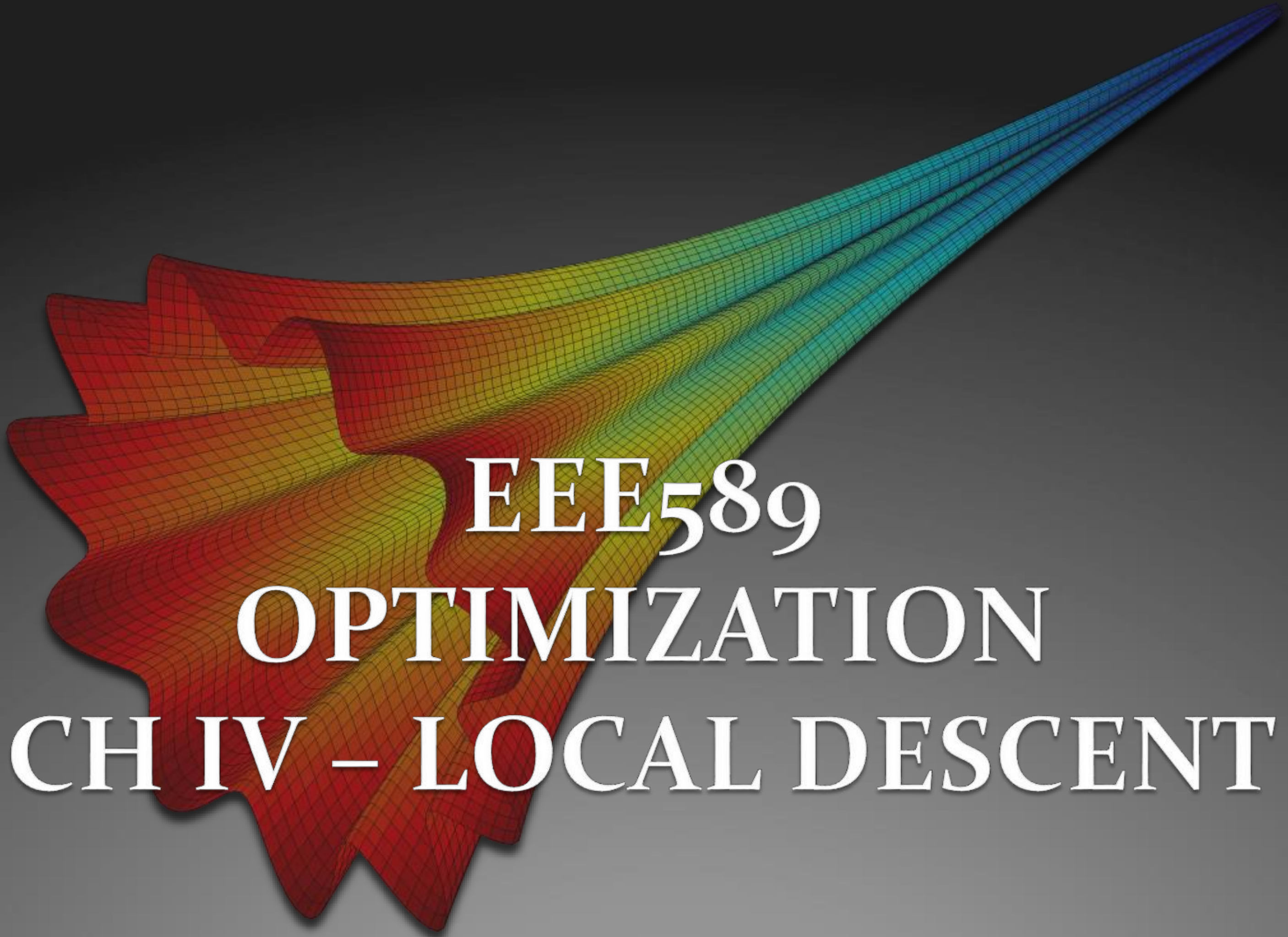


EEE589 OPTIMIZATION CH IV – LOCAL DESCENT

Introduction

- Up to this point, we have focused on optimization involving a single design variable.
- This lecture introduces a general approach to optimization involving *multivariate* functions, or functions with more than one variable.
- The focus is on how to use *local models* to incrementally improve a design point until some convergence criterion is met.
- We begin by discussing methods that, at each iteration, choose a descent direction based on a local model and then choose a step size.
- We then discuss methods that restrict the step to be within a region where the local model is believed to be valid.
- We conclude with a discussion of convergence conditions.

Descent Direction Iteration

- A common approach to optimization is to incrementally improve a design point **x** by taking a step that minimizes the objective value based on a local model.
- The local model may be obtained, for example, from a first- or second-order Taylor approximation.
- Optimization algorithms that follow this general approach are referred to as *descent direction methods*.
- They start with a design point $\boldsymbol{x}^{(\mathbf{1})}$ and then generate a sequence of points, sometimes called *iterates*, to converge to a local minimum.

Descent Direction Iteration

- The iterative descent direction procedure involves the following steps:

1. Check termination conditions at $\mathbf{x}^{(k)}$; if not met, continue.
2. Decide descent direction $\mathbf{d}^{(k)}$ using local information such as the gradient or Hessian.
3. Decide step size or learning rate $\alpha^{(k)}$
4. Compute next design point $\mathbf{x}^{(k+1)}$

$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$$

Line Search

- For the moment, assume we have chosen a descent direction **d**, perhaps using one of the methods discussed in one of the subsequent chapters.
- We need to choose the step factor α to obtain our next design point.
- One approach is to use *line search*, which selects the step factor that minimizes the one-dimensional function:

$$\underset{\alpha}{\text{minimize}}\, f(\mathbf{x} + \alpha \mathbf{d})$$

Line Search

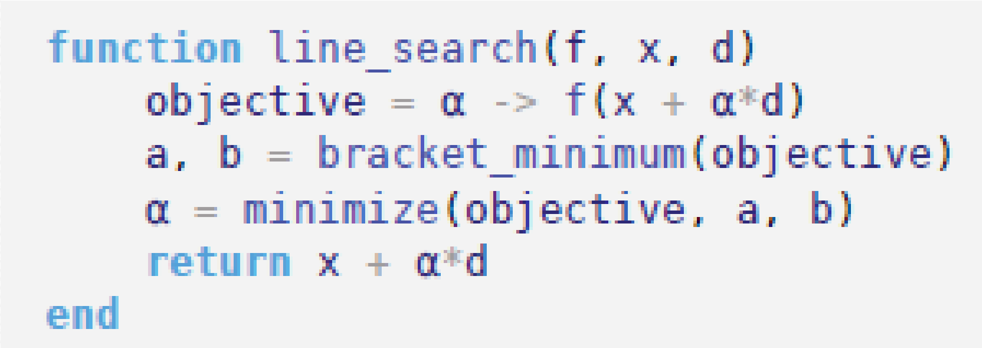
- Used to compute α
- Using the techniques discussed in Chapter 3,

$$\underset{\alpha}{\text{minimize}}\, f(\mathbf{x} + \alpha\mathbf{d})$$

- Often this is computed approximately to reduce cost

Line Search

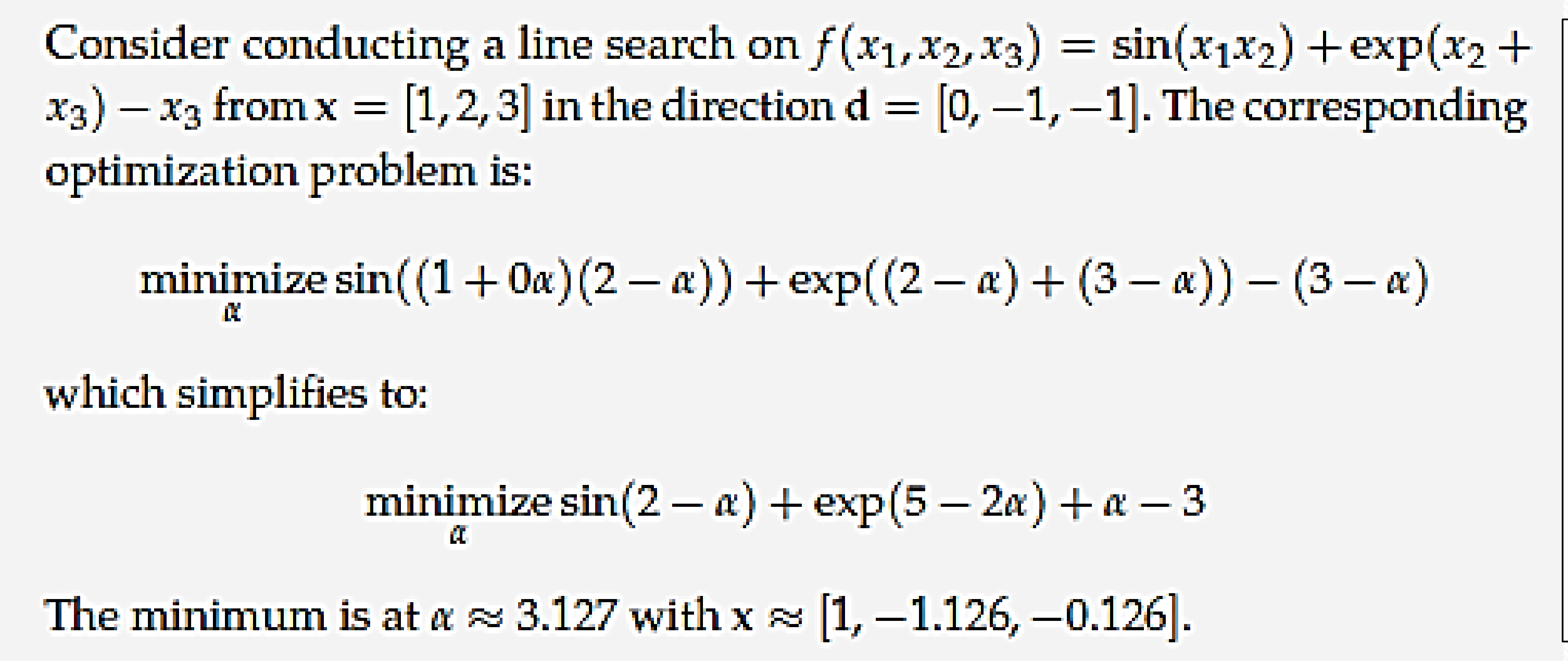
- Line search is a univariate optimization problem, which was covered in chapter 3.
- We can apply the univariate optimization method (The Brent-Dekker method) of our choice.
- To inform the search, we can use the derivative of the line search objective, which is simply the directional derivative along **d** at **x**+α**d**.
- Line search is demonstrated in next slide and implemented in algorithm.



```
function line_search(f, x, d)
    objective = α -> f(x + α*d)
    a, b = bracket_minimum(objective)
    α = minimize(objective, a, b)
    return x + α*d
end
```

Line Search

$$\boxed{\underset{\alpha}{\text{minimize}}\; f(\mathbf{x} + \alpha\mathbf{d})}$$



Consider conducting a line search on $f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_2 + x_3) - x_3$ from $\mathbf{x} = [1, 2, 3]$ in the direction $\mathbf{d} = [0, -1, -1]$. The corresponding optimization problem is:

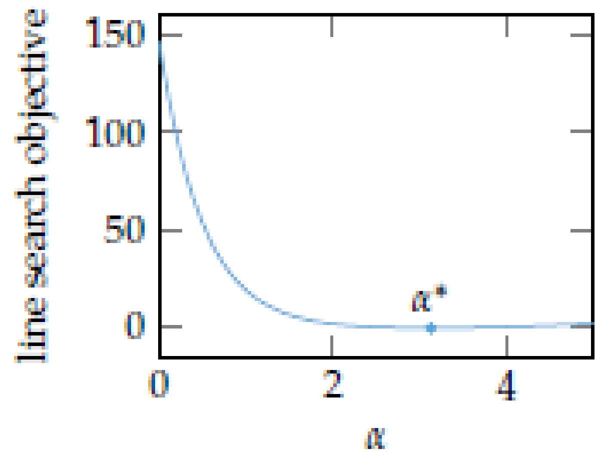
$$\underset{\alpha}{\text{minimize}}\; \sin((1 + 0\alpha)(2 - \alpha)) + \exp((2 - \alpha) + (3 - \alpha)) - (3 - \alpha)$$

which simplifies to:

$$\underset{\alpha}{\text{minimize}}\; \sin(2 - \alpha) + \exp(5 - 2\alpha) + \alpha - 3$$

The minimum is at $\alpha \approx 3.127$ with $\mathbf{x} \approx [1, -1.126, -0.126]$.

Line search used to minimize a function along a descent direction.



Line Search

- One disadvantage of conducting a line search at each step is the computational cost of optimizing α to a high degree of precision.
- Instead, it is common to quickly find a reasonable value and then move on, selecting $\boldsymbol{x}^{(k+1)}$, and then picking a new direction $\boldsymbol{d}^{(k+1)}$.
- Some algorithms use a fixed step factor.
- Large steps will tend to result in faster convergence but risk overshooting the minimum.
- Smaller steps tend to be more stable but can result in slower convergence. A fixed step factor α is sometimes referred to as a *learning rate*.

Line Search: Alternatives

- Another method is to use a decaying step factor:

$$\alpha^{(k)} = \alpha^{(1)}\gamma^{k-1} \quad \text{for } \gamma \in (0, 1]$$

- Decaying step factors are especially popular when minimizing noisy objective functions, and are commonly used in machine learning applications.

Approximate Line Search

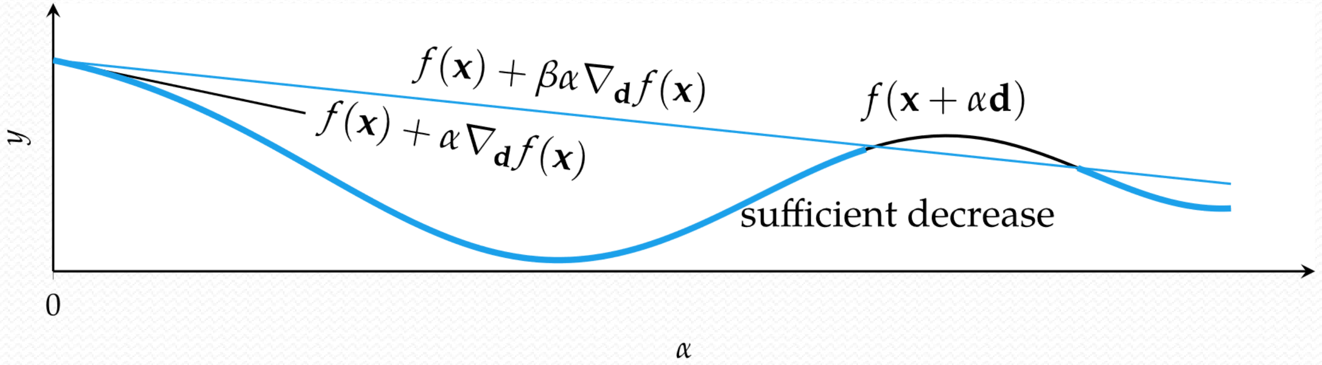
- It is often more computationally efficient to perform more iterations of a descent method than to do exact line search at each iteration, especially if the function and derivative calculations are expensive.
- Many of the methods discussed so far can benefit from using *approximate line search* to find a suitable step size with a small number of evaluations.
- Since descent methods must descend, a step size α may be suitable if it causes a decrease in the objective function value.
- However, a variety of other conditions may be enforced to encourage faster convergence.

Approximate Line Search

- If function calls are expensive, rather than finding the minimum along a search direction, find a point of sufficient decrease

$$f(\mathbf{x}^{(k+1)}) \leq f(\mathbf{x}^{(k)}) + \beta\alpha\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

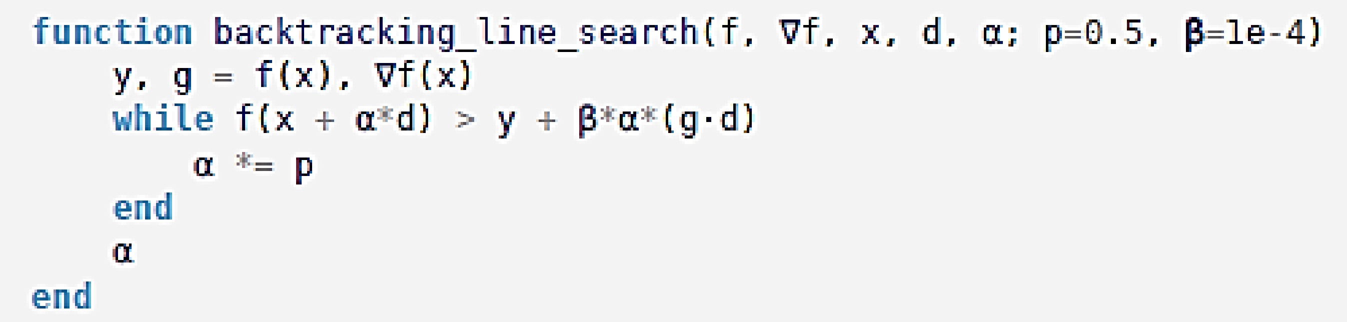
$\beta \in [0,1]$, usually $\beta = 1\times10^{-4}$



- If β = 0, then any decrease is acceptable. If β = 1, then the decrease has to be at least as much as what would be predicted by a first-order approximation.
- Backtracking line search starts with a large step and then backs off

Approximate Line Search

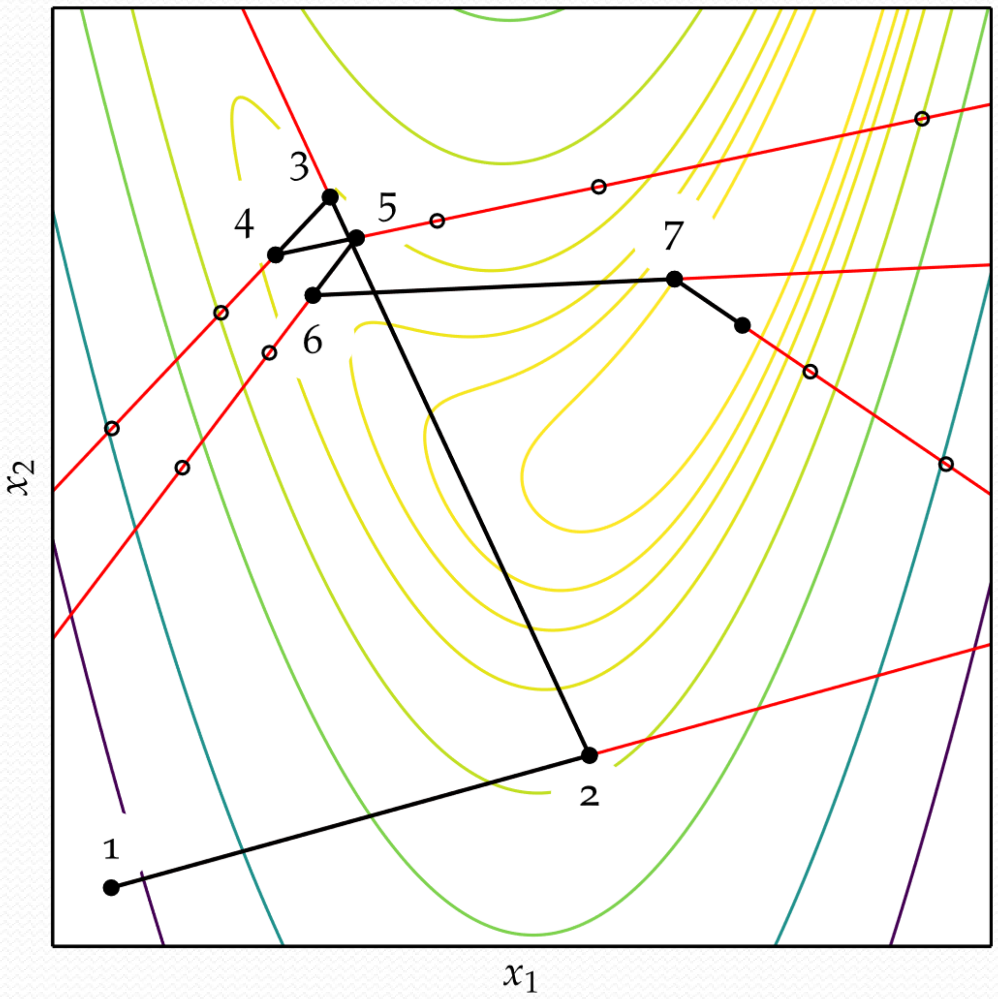
- If **d** is a valid descent direction, then there must exist a sufficiently small step size that satisfies the sufficient decrease condition.
- We can thus start with a large step size and decrease it by a constant reduction factor until the sufficient decrease condition is satisfied.
- This algorithm is known as *backtracking line search* because of how it backtracks along the descent direction.
- Backtracking line search is shown in next figure and implemented in algorithm.
- We walk through the procedure in example.



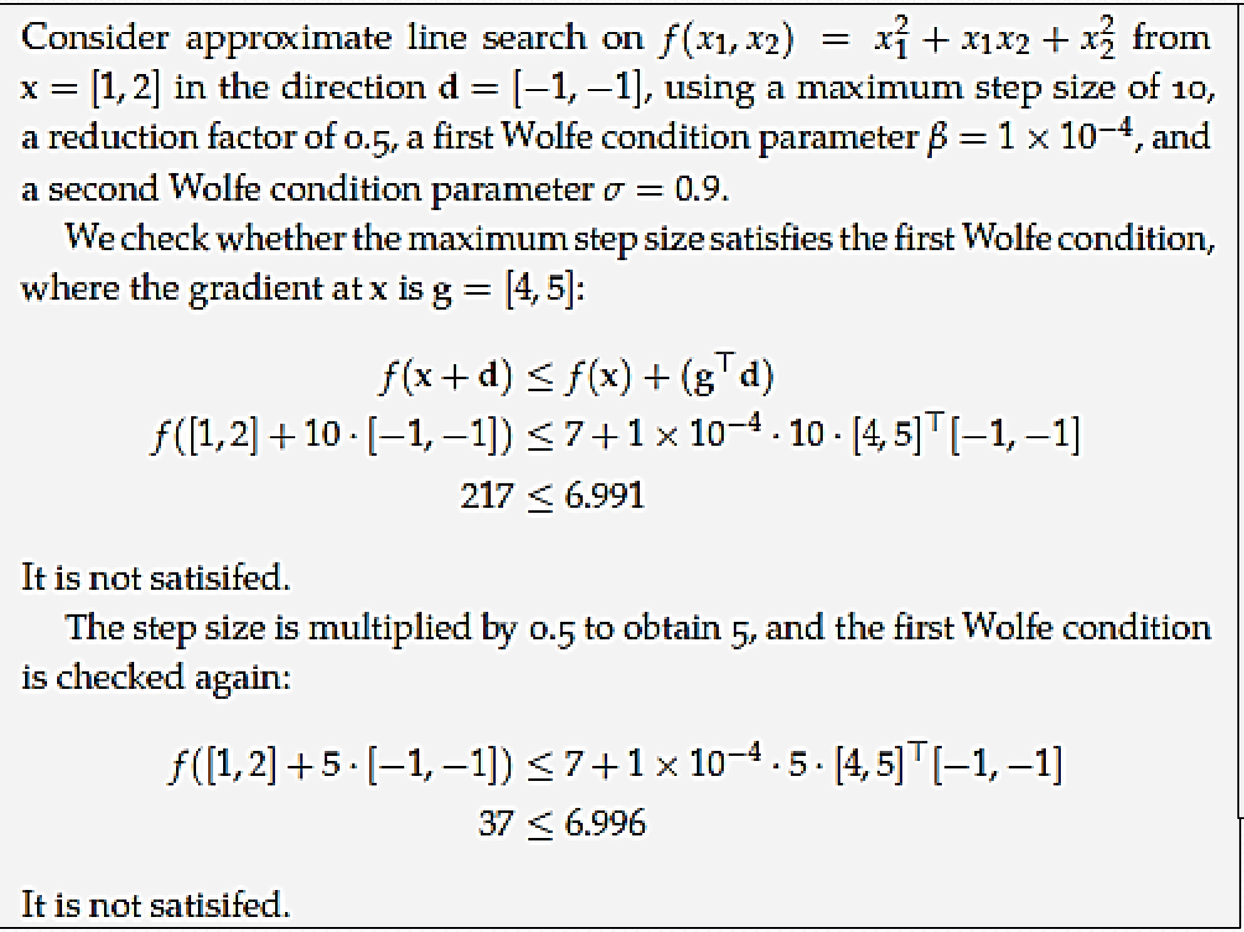
```
function backtracking_line_search(f, ∇f, x, d, α; p=0.5, β=1e-4)
    y, g = f(x), ∇f(x)
    while f(x + α*d) > y + β*α*(g⋅d)
        α *= p
    end
    α
end
```

Approximate Line Search

Backtracking line search
on Rosenbrock function



Approximate Line Search



Consider approximate line search on $f(x_1, x_2) = x_1^2 + x_1x_2 + x_2^2$ from $\mathbf{x} = [1, 2]$ in the direction $\mathbf{d} = [-1, -1]$, using a maximum step size of 10, a reduction factor of 0.5, a first Wolfe condition parameter $\beta = 1 \times 10^{-4}$, and a second Wolfe condition parameter $\sigma = 0.9$.

We check whether the maximum step size satisfies the first Wolfe condition, where the gradient at $\mathbf{x}$ is $\mathbf{g} = [4, 5]$:

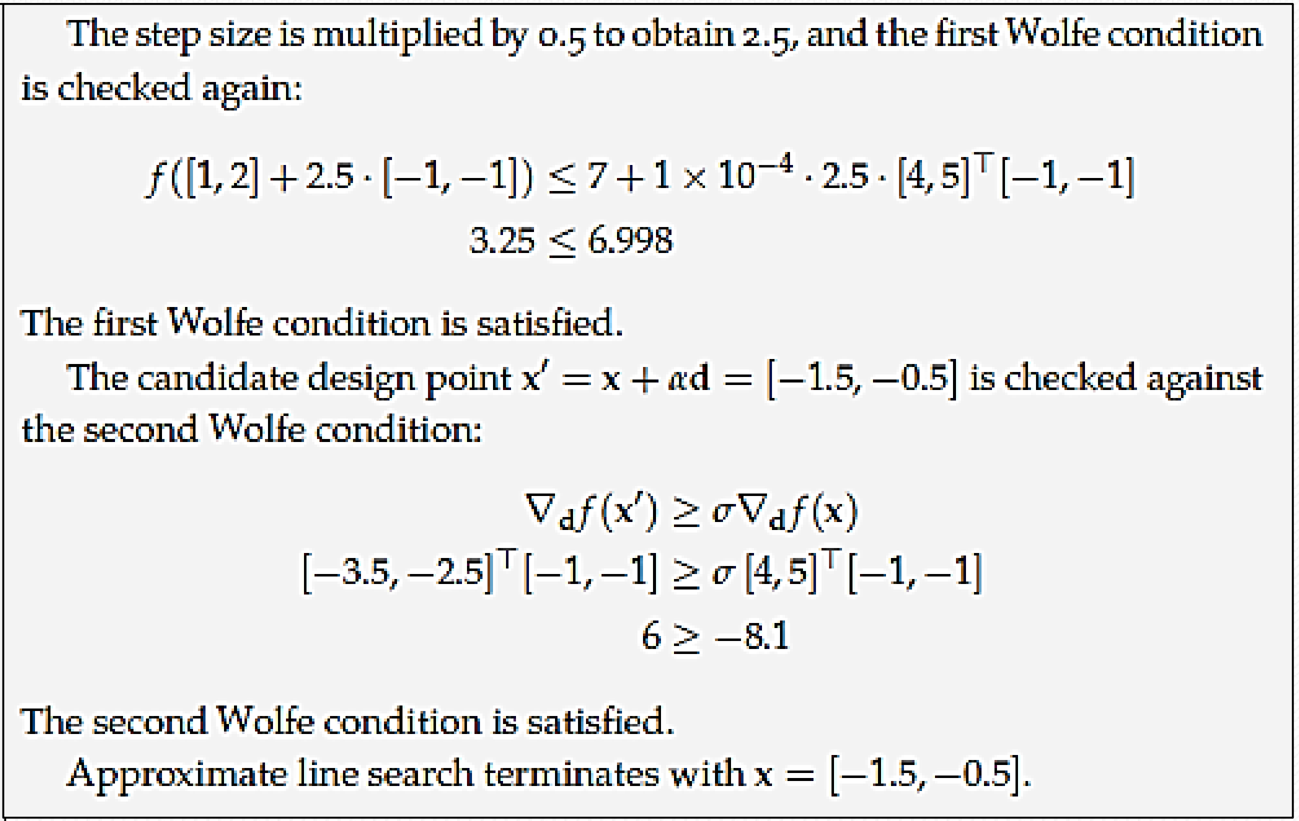
$$\begin{gathered} f(\mathbf{x} + \mathbf{d}) \le f(\mathbf{x}) + (\mathbf{g}^\top \mathbf{d}) \\ f([1, 2] + 10 \cdot [-1, -1]) \le 7 + 1 \times 10^{-4} \cdot 10 \cdot [4, 5]^\top [-1, -1] \\ 217 \le 6.991 \end{gathered}$$

It is not satisifed.

The step size is multiplied by 0.5 to obtain 5, and the first Wolfe condition is checked again:

$$\begin{gathered} f([1, 2] + 5 \cdot [-1, -1]) \le 7 + 1 \times 10^{-4} \cdot 5 \cdot [4, 5]^\top [-1, -1] \\ 37 \le 6.996 \end{gathered}$$

It is not satisifed.



The step size is multiplied by 0.5 to obtain 2.5, and the first Wolfe condition is checked again:

$$\begin{gathered} f([1, 2] + 2.5 \cdot [-1, -1]) \le 7 + 1 \times 10^{-4} \cdot 2.5 \cdot [4, 5]^\top [-1, -1] \\ 3.25 \le 6.998 \end{gathered}$$

The first Wolfe condition is satisfied.

The candidate design point $\mathbf{x}' = \mathbf{x} + \alpha \mathbf{d} = [-1.5, -0.5]$ is checked against the second Wolfe condition:

$$\begin{gathered} \nabla_{\mathbf{d}} f(\mathbf{x}') \ge \sigma \nabla_{\mathbf{d}} f(\mathbf{x}) \\ [-3.5, -2.5]^\top [-1, -1] \ge \sigma \, [4, 5]^\top [-1, -1] \\ 6 \ge -8.1 \end{gathered}$$

The second Wolfe condition is satisfied.

Approximate line search terminates with $\mathbf{x} = [-1.5, -0.5]$.

Approximate Line Search

- The first condition is insufficient to guarantee convergence to a local minimum.
- Very small step sizes will satisfy the first condition but can prematurely converge.
- Backtracking line search avoids premature convergence by accepting the largest satisfactory step size obtained by sequential downscaling and is guaranteed to converge to a local minimum.
- Another condition, called the *curvature condition*, requires the directional derivative at the next iterate to be shallower:

$$\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k+1)}) \geq \sigma \nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

- where σ controls how shallow the next directional derivative must be.

Approximate Line Search

- **Building on backtracking line search are the Wolfe Conditions**

1. First Wolfe Condition: Sufficient Decrease

$$f(\mathbf{x}^{(k+1)}) \leq f(\mathbf{x}^{(k)}) + \beta\alpha\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

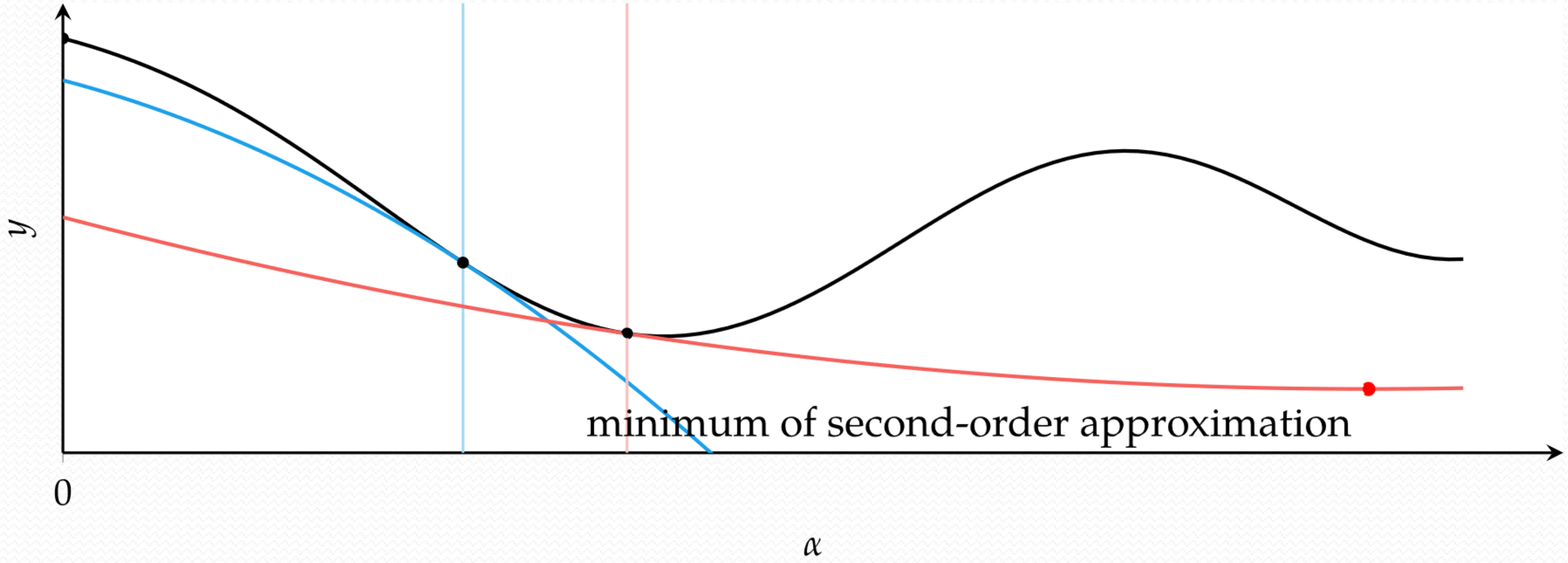
2. Second Wolfe Condition: Curvature Condition

$$\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k+1)}) \geq \sigma\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

- Figures illustrate this condition.
- It is common to set $\beta < \sigma < 1$ with $\sigma = 0.1$ when approximate linear search is used with the conjugate gradient method and to 0.9 when used with Newton's method.

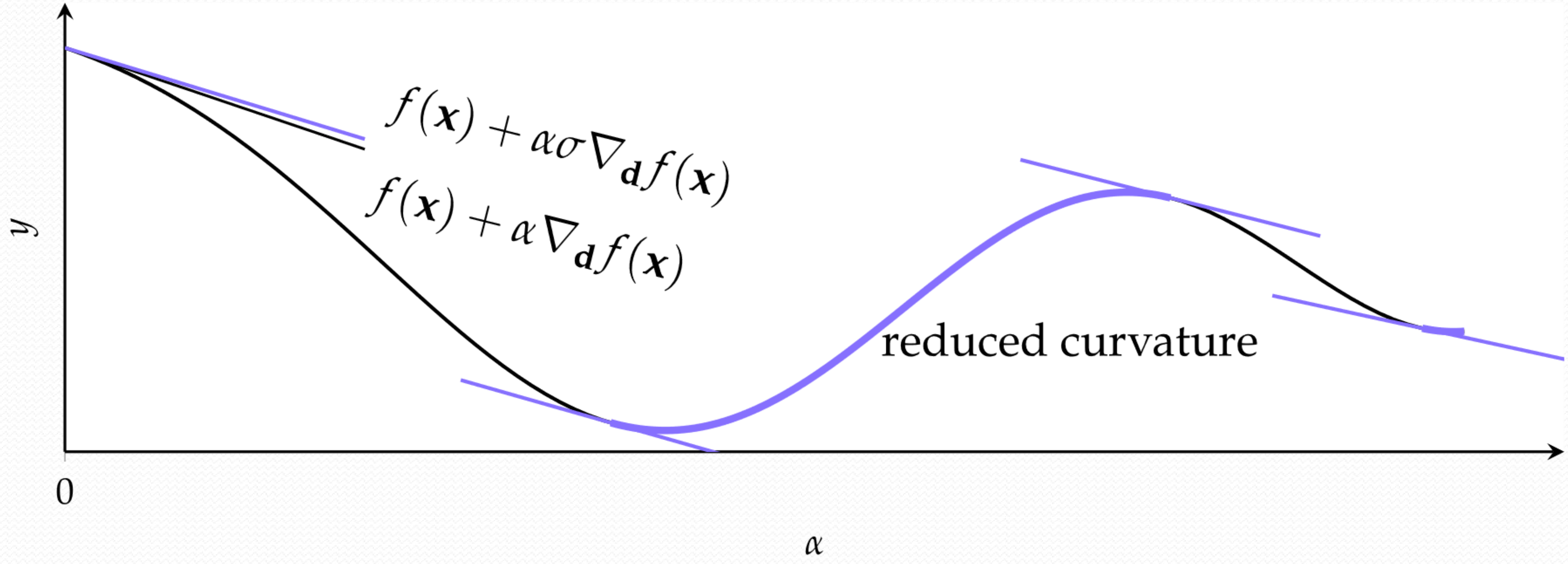
Approximate Line Search

- The curvature condition ensures the second-order function approximations have positive curvature



Approximate Line Search

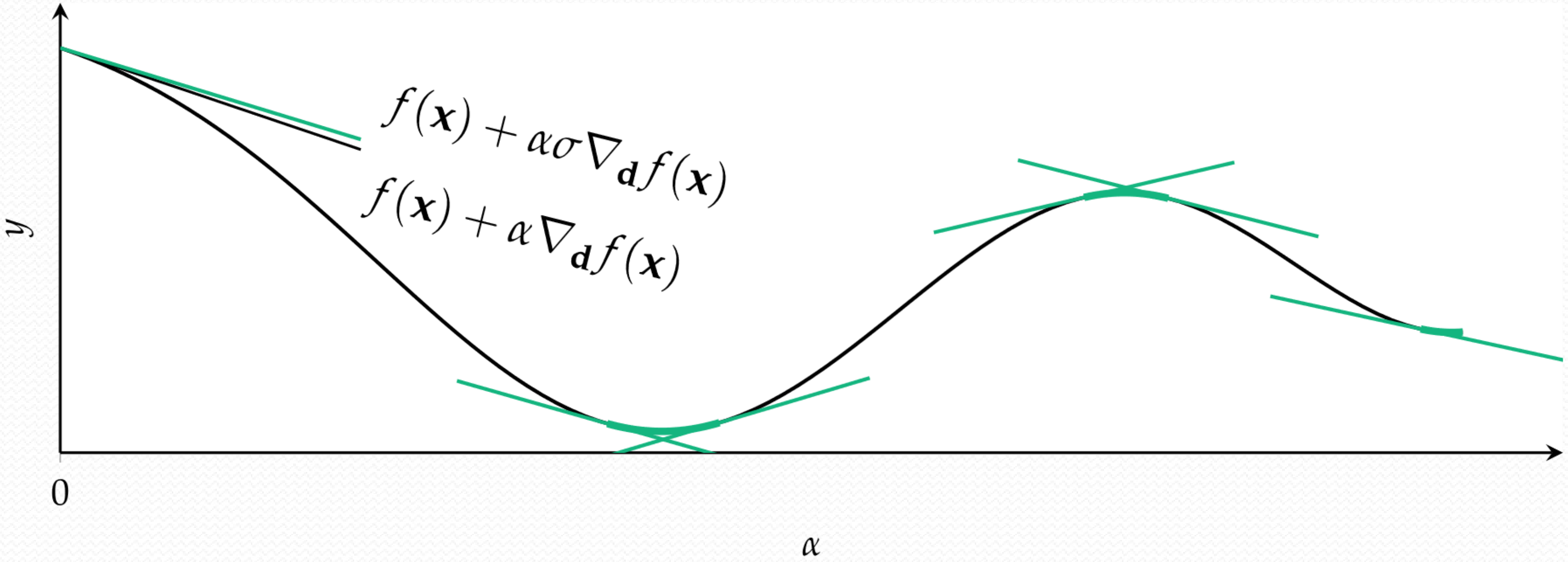
- Regions satisfying the curvature condition


$f(\mathbf{x}) + \alpha\sigma\nabla_{\mathbf{d}} f(\mathbf{x})$
$f(\mathbf{x}) + \alpha\nabla_{\mathbf{d}} f(\mathbf{x})$
reduced curvature
y
0
α

Approximate Line Search

- An alternative to the curvature condition is the *strong Wolfe condition*, which is a more restrictive criterion in that the slope is also required not to be too positive:

$$|\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k+1)})| \leq -\sigma \nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$



Approximate Line Search

- Together, the sufficient decrease condition and the first curvature condition form the *Wolfe conditions*.
- The sufficient decrease condition is often called the *first Wolfe condition* and the curvature condition is called the *second Wolfe condition*.
- The sufficient decrease condition with the second curvature condition form the *strong Wolfe conditions*.
- Satisfying the strong Wolfe conditions requires a more complicated algorithm, *strong backtracking line search* (algorithm).
- The method operates in two phases.
- The first phase, the *bracketing phase*, tests successively larger step sizes to bracket an interval $[\alpha^{(k-1)}, \alpha^{(k)}]$ guaranteed to contain step lengths satisfying the Wolfe conditions.

Approximate Line Search

1. Bracketing Phase: test successively larger step sizes to find interval guaranteed to contain step lengths satisfying Wolfe conditions
2. Zoom Phase: shrink the interval using bisection to find point satisfying Wolfe conditions

Approximate Line Search

- An interval guaranteed to contain step lengths satisfying the Wolfe conditions

is found when one of the following conditions hold:

$$f(\mathbf{x} + \alpha^{(k)}\mathbf{d}) \geq f(\mathbf{x})$$

$$f(\mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}) > f(\mathbf{x}^{(k)}) + \beta\alpha^{(k)}\nabla_{\mathbf{d}^{(k)}} f(\mathbf{x}^{(k)})$$

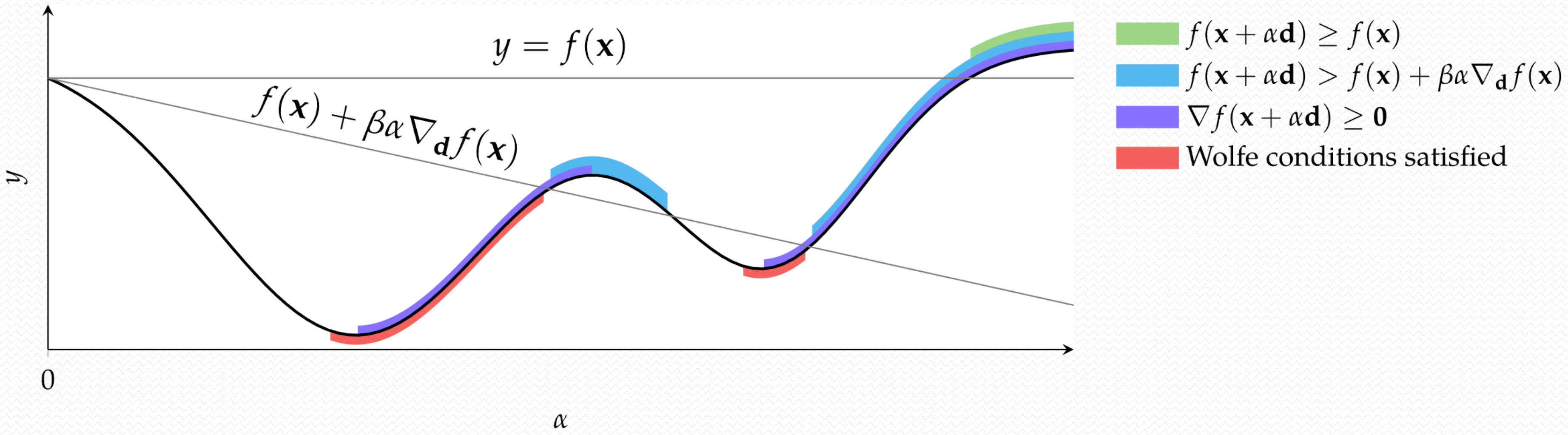
$$\nabla f(\mathbf{x} + \alpha^{(k)}\mathbf{d}) \geq \mathbf{0}$$

- Satisfying equation(2) is equivalent to violating the first Wolfe condition, thereby ensuring that shrinking the step length will guarantee a satisfactory step length. Similarly, equation (1) and equation (3) guarantee that the descent step has overshot a local minimum, and the region between must therefore contain a satisfactory step length.

Approximate Line Search

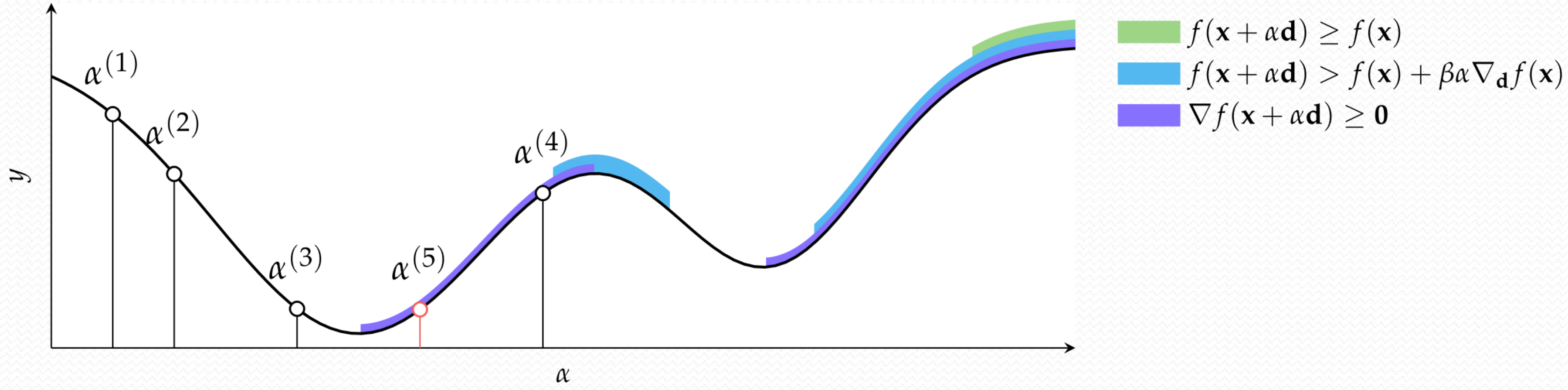
- Figure shows where each bracketing condition is true for an example line search. The figure shows bracket intervals $[0, \alpha]$, whereas advanced backtracking line search successively increases the step length to obtain a bracketing interval $[\alpha^{(k-1)}, \alpha^{(k)}]$.

Bracketing Phase:



Approximate Line Search

- In the *zoom phase*, we shrink the interval to find a step size satisfying the strong Wolfe conditions.
- The shrinking can be done using the bisection method (section 3), updating the interval boundaries according to the same interval conditions.
- This process is shown in figure.
- **Zoom Phase:**



$\alpha^{(1)}$
$\alpha^{(2)}$
$\alpha^{(3)}$
$\alpha^{(4)}$
$\alpha^{(5)}$
y
α
🟩 $f(\mathbf{x} + \alpha\mathbf{d}) \geq f(\mathbf{x})$
🟦 $f(\mathbf{x} + \alpha\mathbf{d}) > f(\mathbf{x}) + \beta\alpha\nabla_{\mathbf{d}} f(\mathbf{x})$
🟪 $\nabla f(\mathbf{x} + \alpha\mathbf{d}) \geq \mathbf{0}$

Trust Region Methods

- Descent methods can place too much trust in their first- or second-order information, which can result in excessively large steps or premature convergence.
- A *trust region* is the local area of the design space where the local model is believed to be reliable.
- A trust region method, or *restricted step method*, maintains a local model of the trust region that both limits the step taken by traditional line search and predicts the improvement associated with taking the step.
- If the improvement closely matches the predicted value, the trust region is expanded.
- If the improvement deviates from the predicted value, the trust region is contracted.

Trust Region Methods

- Trust region methods first choose the maximum step size and then the step direction, which is in contrast with line search methods that first choose a step direction and then optimize the step size.
- A trust region approach finds the next step by minimizing a model of the objective function $\hat{f}$ over a trust region centered on the current design point $\mathbf{x}$.
- An example of $\hat{f}$ is a second-order Taylor approximation.
- The radius of the trust region, δ, is expanded and contracted based on how well the model predicts function evaluations.

Trust Region Methods

- The next design point $\mathbf{x}'$ is obtained by solving:

$$\begin{aligned} \underset{\mathbf{x}'}{\text{minimize}} \quad & \hat{f}(\mathbf{x}') \\ \text{subject to} \quad & \|\mathbf{x} - \mathbf{x}'\| \leq \delta \end{aligned}$$

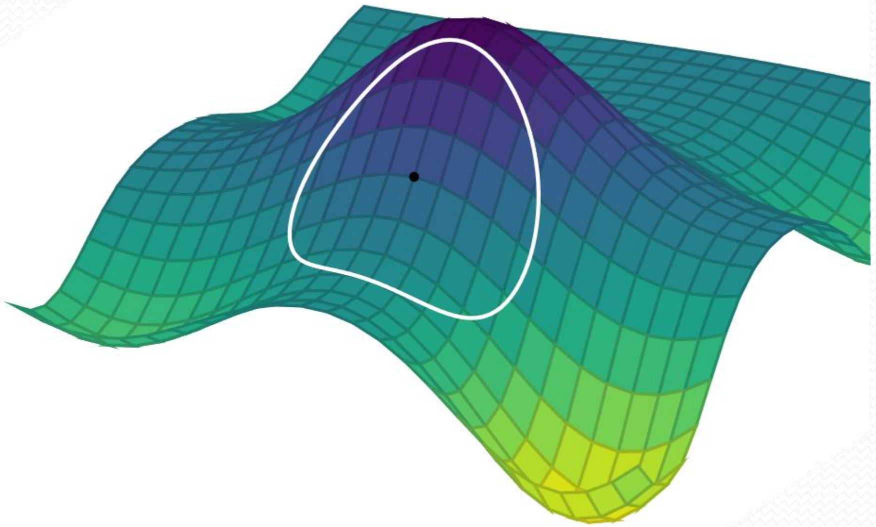
where the trust region is defined by the positive radius δ and a vector norm.

- The equation above is a constrained optimization problem, which is covered in later.

Trust Region Methods

- Figure shows a design point centered within a circular trust region.
- $\hat{f}(x')$ is local function approximation at new design point
- δ is trust region radius
- x' is new design point

$$\begin{aligned} \underset{\mathbf{x}'}{\text{minimize}} \quad & \hat{f}(\mathbf{x}') \\ \text{subject to} \quad & \|\mathbf{x} - \mathbf{x}'\| \leq \delta \end{aligned}$$



Trust Region Methods

- The trust region radius δ is expanded or contracted based on the local model's predictive performance.
- Trust region methods compare the predicted improvement $\Delta y_{\text{pred}} = f(\mathbf{x}) - \hat{f}(\mathbf{x}')$

to the actual improvement $\Delta y_{\text{act}} = f(\mathbf{x}) - f(\mathbf{x}')$:

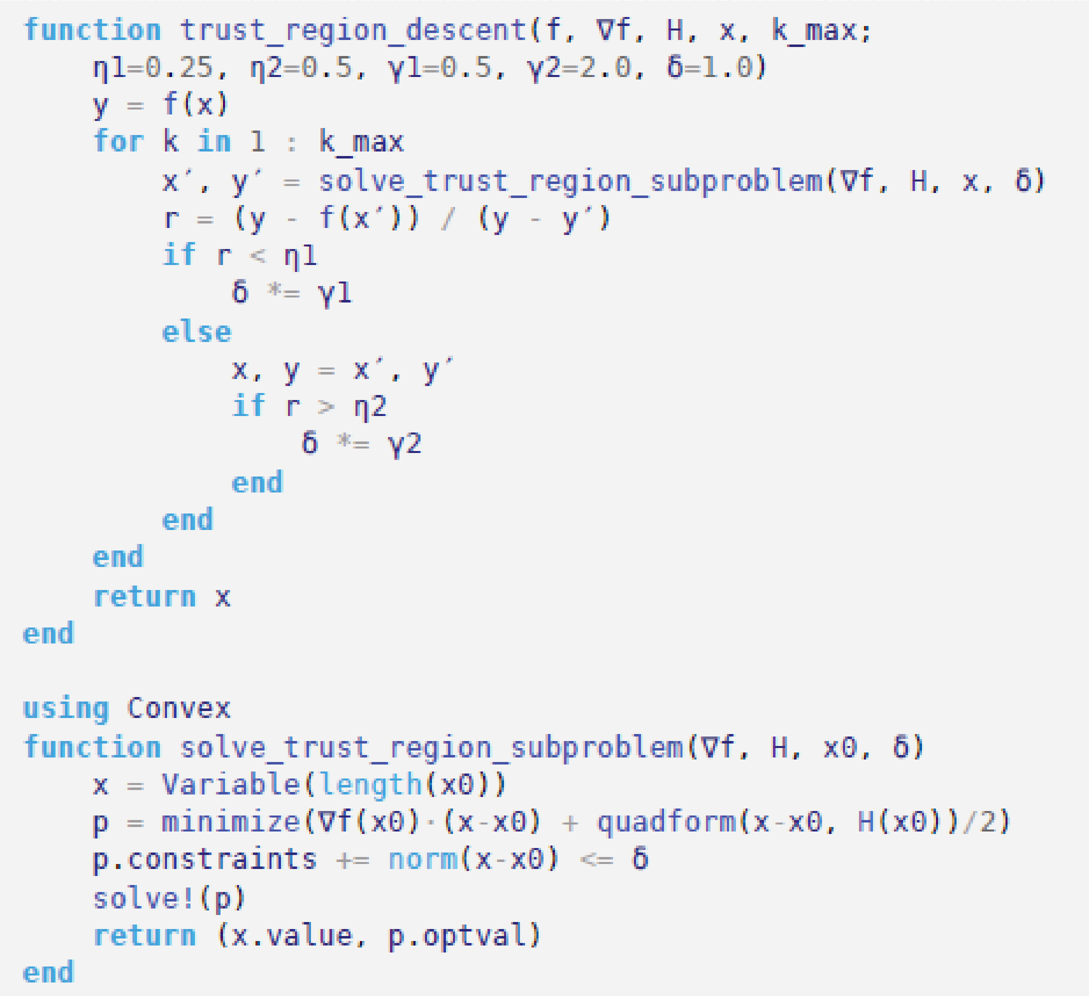
$$\eta = \frac{\text{actual improvement}}{\text{predicted improvement}} = \frac{f(\mathbf{x}) - f(\mathbf{x}')}{f(\mathbf{x}) - \hat{f}(\mathbf{x}')}$$

Trust Region Methods

- The ratio η is close to 1 when the predicted step size matches the actual step size.
- If the ratio is too small, such as below a threshold $\eta 1$, then the improvement is considered sufficiently less than expected, and the trust region radius is scaled down by a factor $\gamma 1 < 1$.
- If the ratio is sufficiently large, such as above a threshold $\eta 2$, then our prediction is considered accurate, and the trust region radius is scaled up by a factor $\gamma 2 > 1$.

Trust Region Methods

- Algorithm provides an implementation.

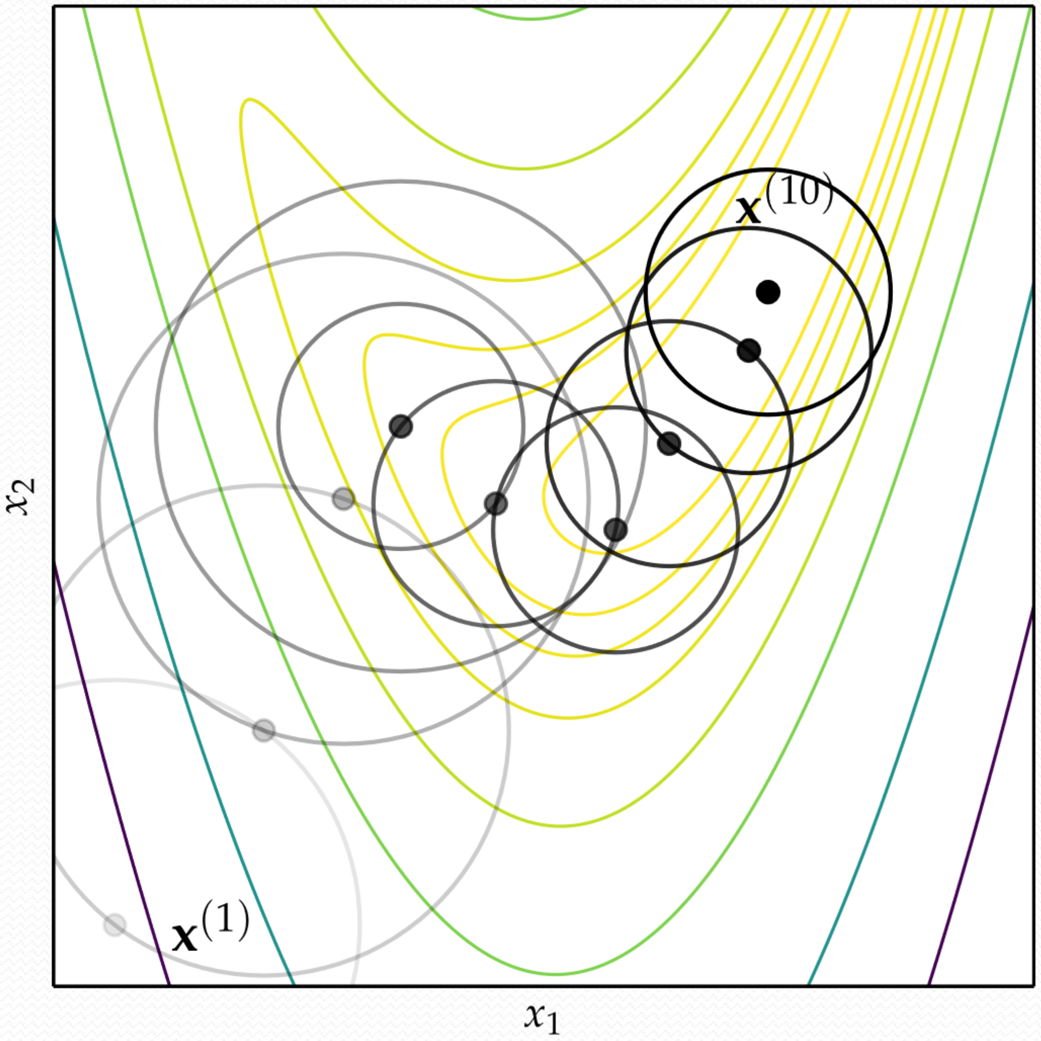


```
function trust_region_descent(f, ∇f, H, x, k_max;
    η1=0.25, η2=0.5, γ1=0.5, γ2=2.0, δ=1.0)
    y = f(x)
    for k in 1 : k_max
        x′, y′ = solve_trust_region_subproblem(∇f, H, x, δ)
        r = (y - f(x′)) / (y - y′)
        if r < η1
            δ *= γ1
        else
            x, y = x′, y′
            if r > η2
                δ *= γ2
            end
        end
    end
    return x
end

using Convex
function solve_trust_region_subproblem(∇f, H, x0, δ)
    x = Variable(length(x0))
    p = minimize(∇f(x0)⋅(x-x0) + quadform(x-x0, H(x0))/2)
    p.constraints += norm(x-x0) <= δ
    solve!(p)
    return (x.value, p.optval)
end
```

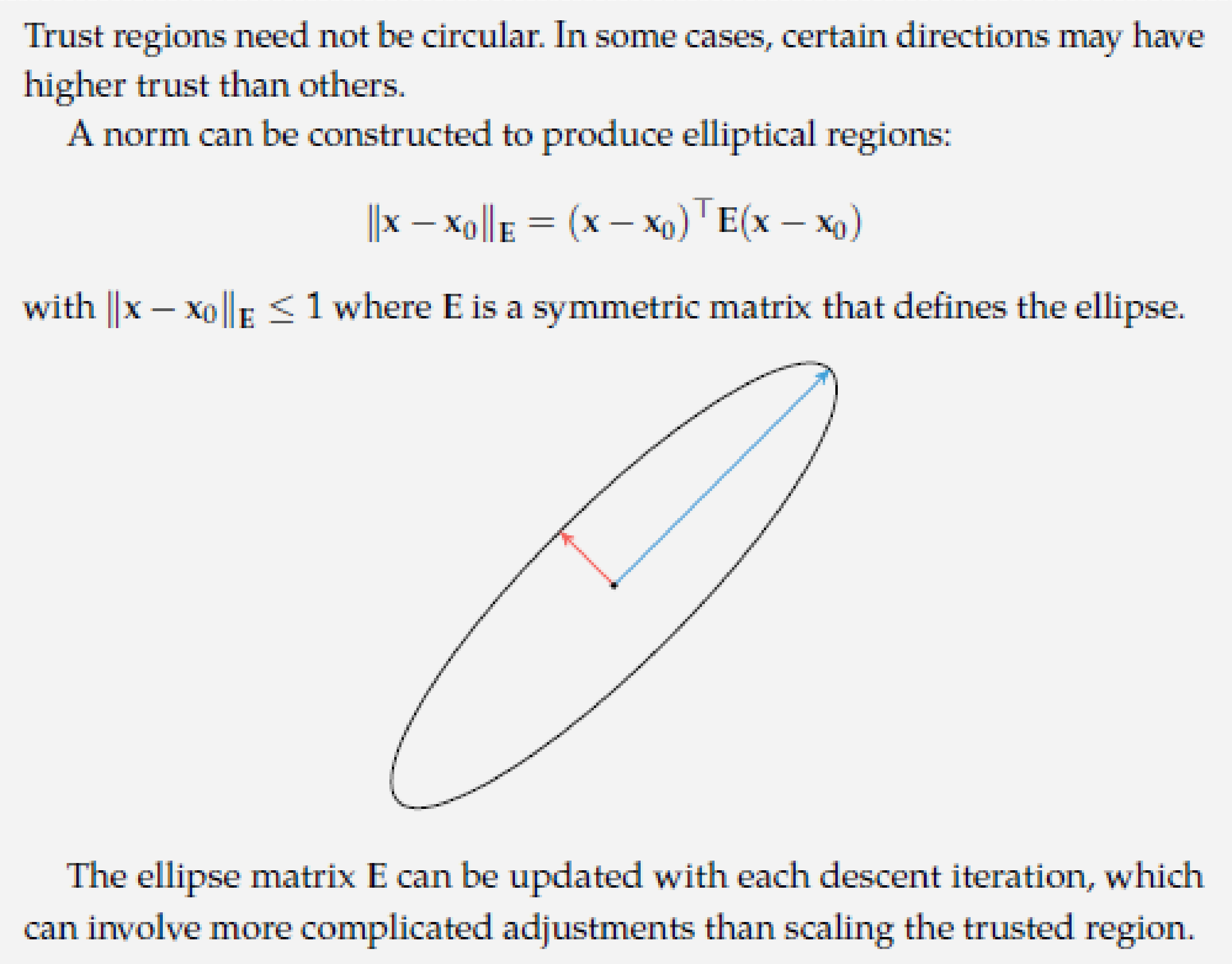
Trust Region Methods

- Figure visualizes the optimization procedure.



Trust Region Methods

- Example shows how to construct noncircular trust regions.



Trust regions need not be circular. In some cases, certain directions may have higher trust than others.

A norm can be constructed to produce elliptical regions:

$$\|\mathbf{x} - \mathbf{x}_0\|_{\mathbf{E}} = (\mathbf{x} - \mathbf{x}_0)^\top \mathbf{E}(\mathbf{x} - \mathbf{x}_0)$$

with $\|\mathbf{x} - \mathbf{x}_0\|_{\mathbf{E}} \leq 1$ where $\mathbf{E}$ is a symmetric matrix that defines the ellipse.

The ellipse matrix $\mathbf{E}$ can be updated with each descent iteration, which can involve more complicated adjustments than scaling the trusted region.

Termination Conditions

- There are four common termination conditions for descent direction methods:

- ***Maximum iterations.***

We may want to terminate when the number of iterations k exceeds some threshold k_{max}. Alternatively, we might want to terminate once a maximum amount of elapsed time is exceeded.

$$k > k_{\max}$$

Termination Conditions

- ***Absolute improvement.***

This termination condition looks at the change in the function value over subsequent steps. If the change is smaller than a given threshold, it will terminate:

$$f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_a$$

- ***Relative improvement.***

This termination condition also looks at the change in function value but uses the step factor relative to the current function value:

$$f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)}) < \epsilon_r |f(\mathbf{x}^{(k)})|$$

Termination Conditions

- ***Gradient magnitude.***

We can also terminate based on the magnitude of the gradient:

$$\|\nabla f(\mathbf{x}^{(k+1)})\| < \epsilon_g$$

- In cases where multiple local minima are likely to exist, it can be beneficial to incorporate *random restarts* after our termination conditions are met where we restart our local descent method from randomly selected initial points.

Summary

- Descent direction methods incrementally descend toward a local optimum.
- Univariate optimization can be applied during line search.
- Approximate line search can be used to identify appropriate descent step sizes.
- Trust region methods constrain the step to lie within a local region that expands or contracts based on predictive accuracy.
- Termination conditions for descent methods can be based on criteria such as the change in the objective function value or magnitude of the gradient.