ME 575 Final Exam Review

Winter 2016

Review Outline

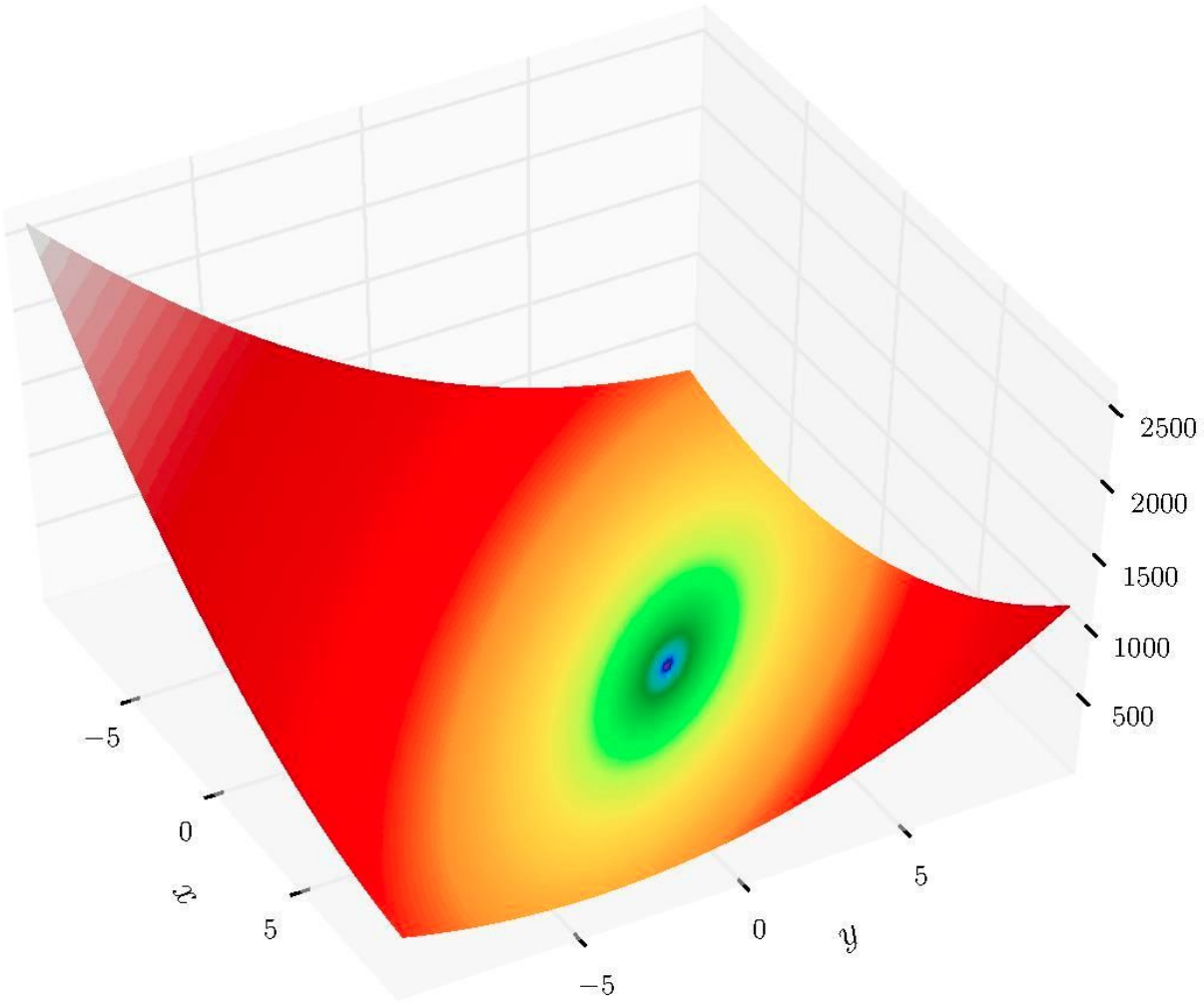
- Optimality conditions
- Exact line search methods (bisection method)
- Inexact line search (backtracking, parameters, Wolfe conditions)
- Gradients and hessians
- Gradient-based optimization (steepest descent, conjugate gradient, and quasi-Newton methods)
- Trust regions
- Obtaining gradients (finite difference, complex step, adjoint, and automatic differentiation)
- Lagrangian and KKT conditions
- Penalty methods
- SQP
- Gradient-free optimization (Nelder-Mead, genetic algorithm, and particle swarm methods)
- Surrogate-based optimization
- Making optimization robust and reliable
- Optimizers examples (fmincon, SciPy, and pyoptsparse)

Booth's Function

$f(x,y) = (x+2y-7)^2+(2x+y-5)^2$

Minimum at:

$f(1,3) = 0$



Optimality Conditions (1D - also need to know ND)

- For f* to be a local minima:

$$\varepsilon f'(x^*) + \frac{1}{2}\varepsilon^2 f''(x^* + \theta\varepsilon) \geq 0$$

- First order optimality condition:

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

  - If satisfied, then x* is a “stationary point”
- Second order optimality term

$$f''(x^*) \geq 0.$$

- Together, these are the “sufficient conditions” for a strong local minimum

Optimality Conditions

The optimality conditions can be used to:

1. Verify that a point is a minimum (sufficient conditions).

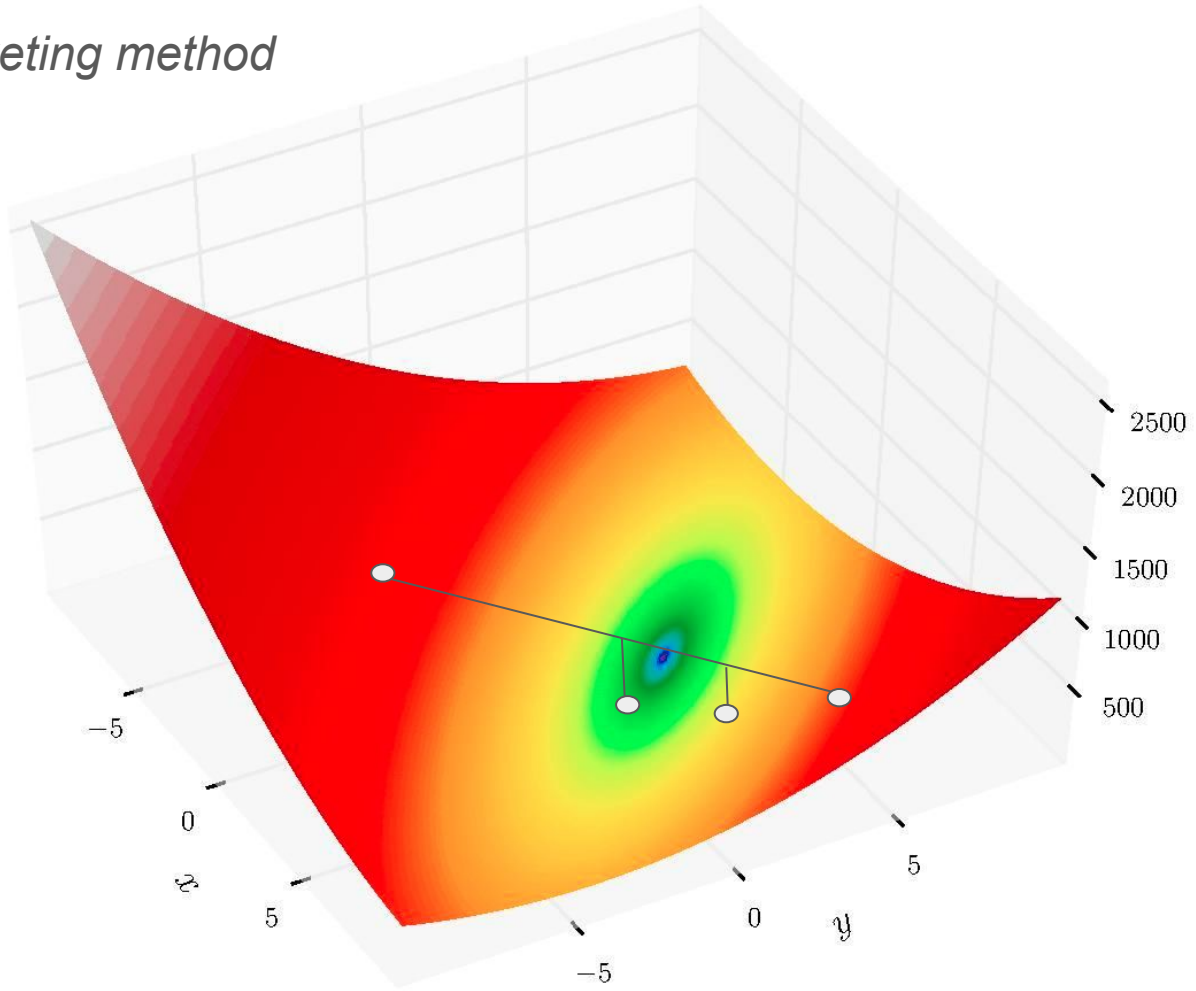
2. Realize that a point is not a minimum (necessary conditions).

3. Define equations that can be solved to find a minimum.

Gradient-based minimization methods find a local minima by finding points that satisfy the optimality conditions.

Exact Line Search (Bisection Example)

- Bisection is a *bracketing method*



Other methods (used for either exact or inexact line search):

- Newton's Method
- Secant Method
  - Newton with FD
- Golden Section Search
- Polynomial Interpretation
- Brent's Method
  - GS + Polynomial Interp.

Inexact Line Search

- Wolfe Conditions
  - Sufficient Decrease

$$f(x_k + \alpha p_k) \leq f(x_k) + \mu_1 \alpha g_k^T p_k$$

  - Curvature Condition

$$g(x_k + \alpha p_k)^T p_k \geq \mu_2 g_k^T p_k$$

  - (you don't need to know the strong Wolfe conditions)
    - Strong Wolfe Conditions ($0 < \mu 1 < \mu 2 < 1$)

$$\begin{aligned} f(x_k + \alpha p_k) &\leq f(x_k) + \mu_1 \alpha g_k^T p_k. \\ \left| g(x_k + \alpha p_k)^T p_k \right| &\leq \mu_2 \left| g_k^T p_k \right|, \end{aligned}$$

The strong cond. essentially just exclude any points that are far from stationary

Strong Wolfe Conditions (you don't need to know these)

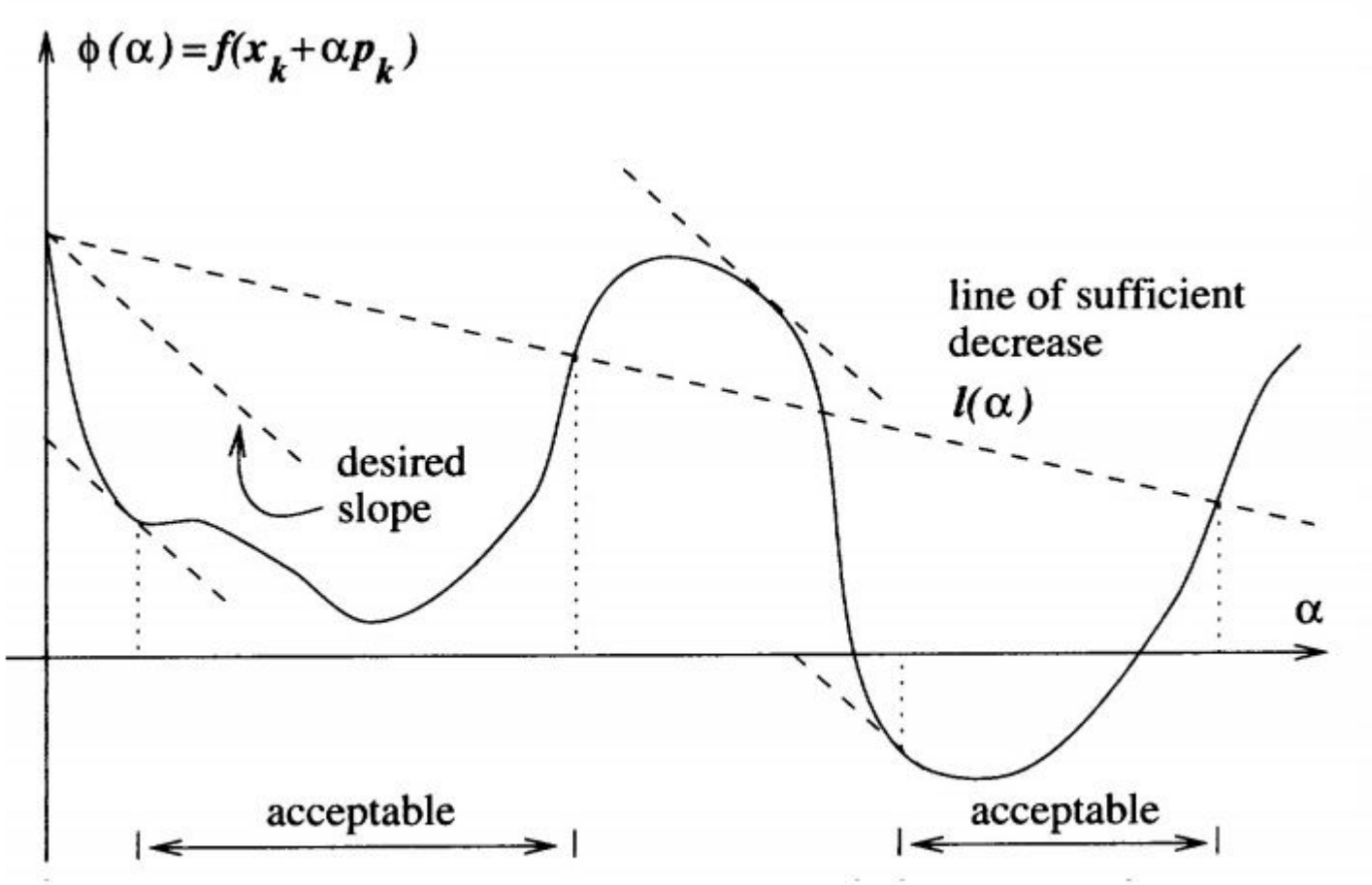


Figure 2.6: Acceptable steps for the strong Wolfe conditions.

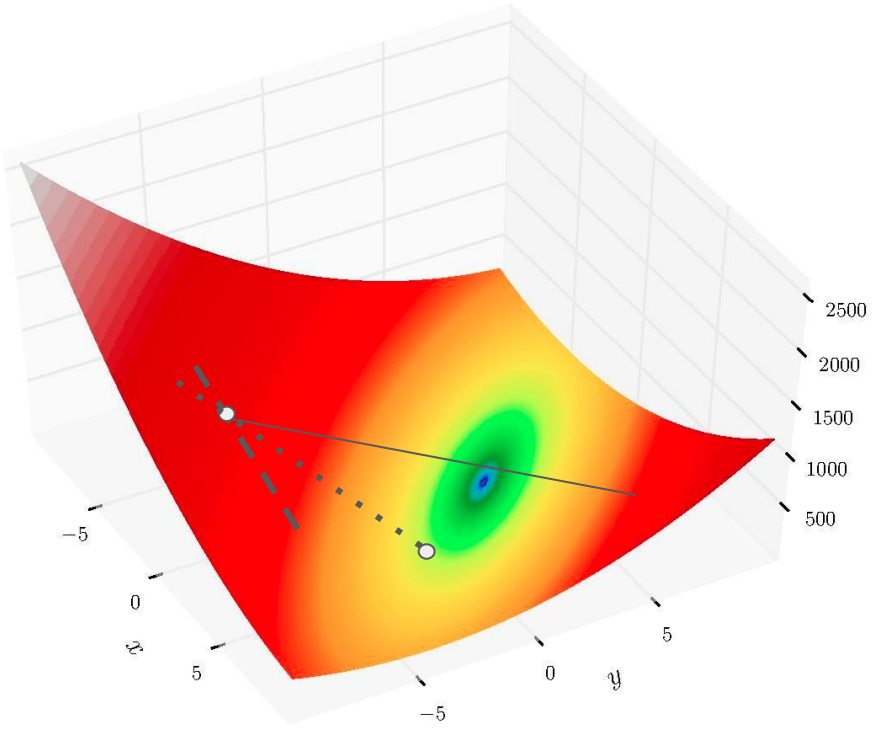
~~Exact~~ line search (backtracking example, this is not the only exact method - see slide 7)

Algorithm 1 Backtracking line search algorithm

Input: $\alpha > 0$ and $0 < \rho < 1$

Output: α_k

1: **repeat**
2: $\quad \alpha \leftarrow \rho\alpha$
3: **until** $f(x_k + \alpha p_k) \leq f(x_k) + \mu_1 \alpha g_k^T p_k$
4: $\alpha_k \leftarrow \alpha$



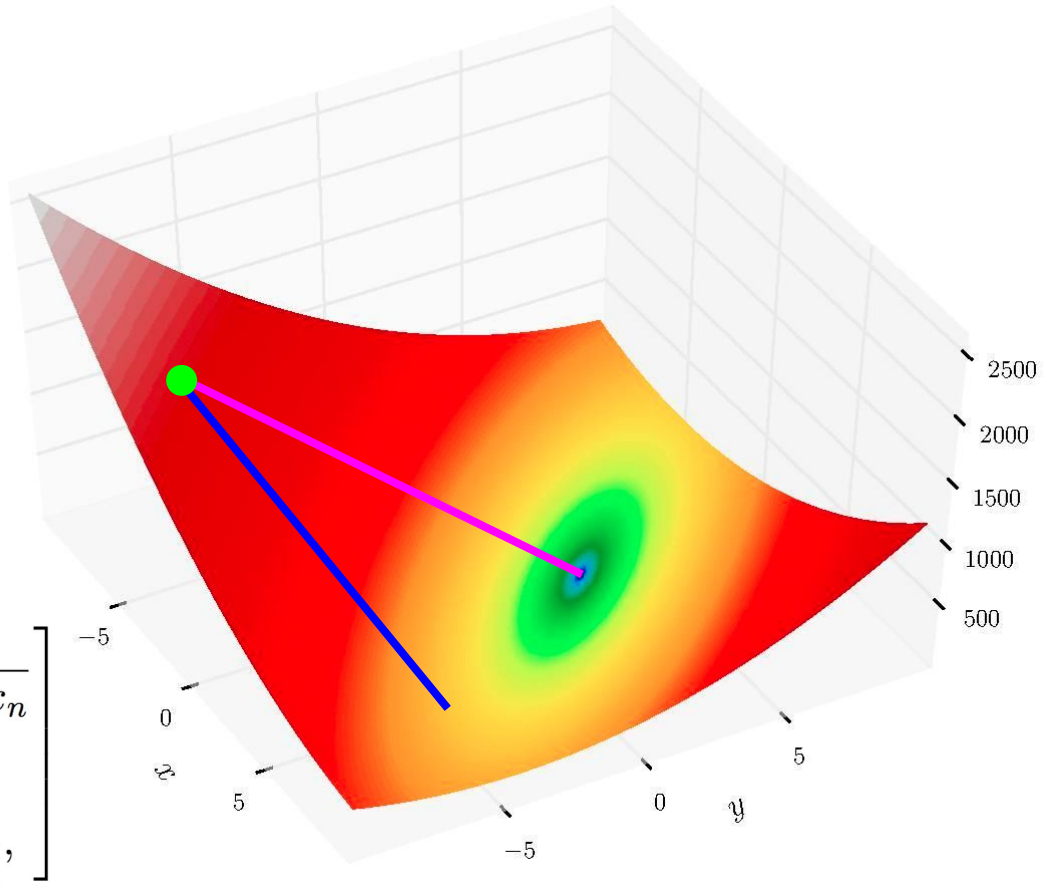
Gradients and Hessians

Gradient- first derivative (slope)

$$\nabla f(x) \equiv g(x) \equiv \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Hessian- second derivative (curve)

$$\nabla^2 f(x) \equiv H(x) \equiv \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_n}, \end{bmatrix}$$



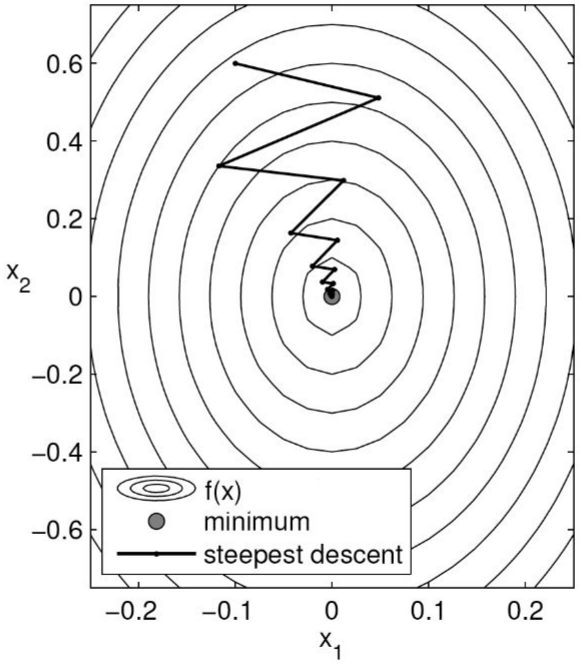
Gradient-Based Optimization- Steepest Descent

Algorithm 5 Steepest descent

Input: Initial guess, x_0, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and ε_r.
Output: Optimum, x^*

$k \leftarrow 0$
repeat
  Compute the gradient of the objective function, $g(x_k) \equiv \nabla f(x_k)$
  Compute the normalized search direction, $p_k \leftarrow -g(x_k)/\|g(x_k)\|$
  Perform line search to find step length α_k
  Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $k \leftarrow k + 1$
until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$

GO DOWN THE STEEPEST ROUTE



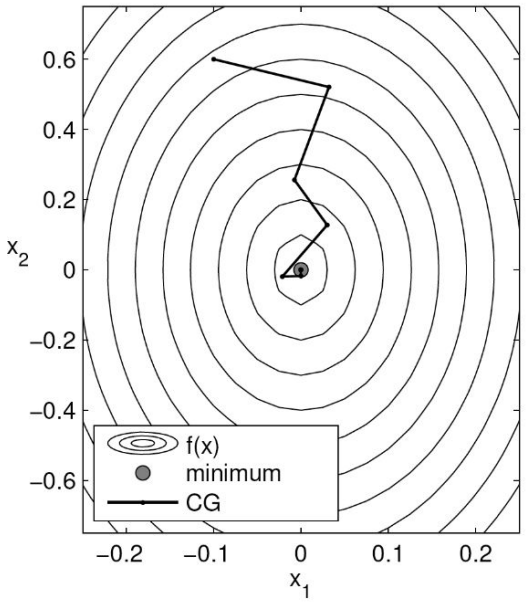
Gradient-Based Optimization- Conjugate Gradient

Algorithm 6 Nonlinear conjugate gradient method

Input: Initial guess, x_0, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and ε_r.
Output: Optimum, x^*

$k \leftarrow 0$
repeat
  Compute the gradient of the objective function, $g(x_k)$
  if k=0 **then**
    Compute the normalized steepest descent direction, $p_k \leftarrow -g(x_k)/\|g(x_k)\|$
  else
    Compute $\beta \leftarrow \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$
    Compute the conjugate gradient direction $p_k = -g_k/\|g(x_k)\| + \beta_k p_{k-1}$
  end if
  Perform line search to find step length α_k
  Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $k \leftarrow k + 1$
until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$

GO DOWN STEEPEST ROUTE WITH MEMORY



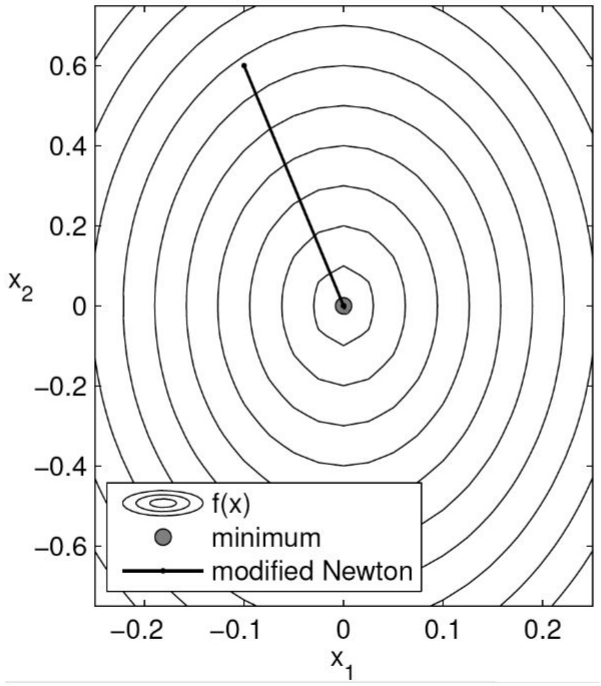
Gradient-Based Optimization- Newton

Algorithm 7 Modified Newton's method

Input: Initial guess, x_0, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and ε_r.
Output: Optimum, x^*

$k \leftarrow 0$
repeat
  Compute the gradient of the objective function, $g(x_k)$
  Compute the Hessian of the objective function, $H(x_k)$
  Compute the search direction, $p_k = -H^{-1} g_k$
  Perform line search to find step length α_k, starting with $\alpha = 1$
  Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  $k \leftarrow k + 1$
until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$

NEED A HESSIAN!!!! DIFFICULT!!!

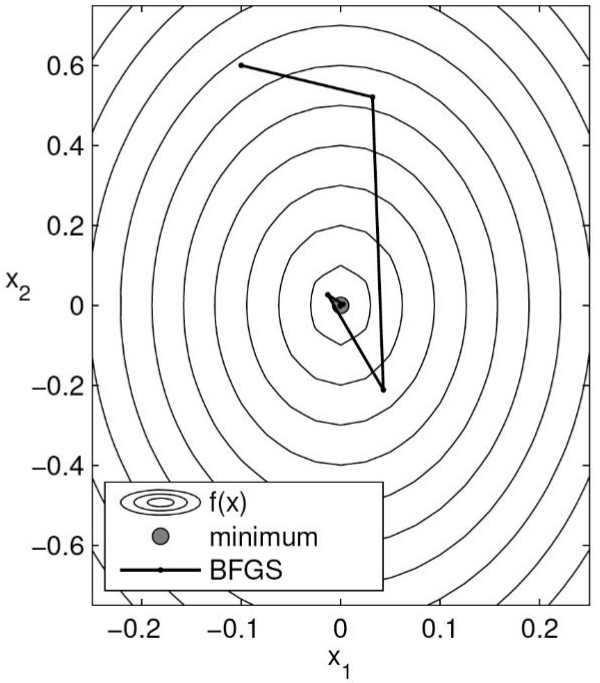


Gradient-Based Optimization- Quasi-Newton

Algorithm 8 Quasi-Newton method with DFP update

Input: Initial guess, x_0, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and ε_r.
Output: Optimum, x^*

$k \leftarrow 0$
$V_0 \leftarrow I$
repeat
  Compute the gradient of the objective function, $g(x_k)$
  Compute the search direction, $p_k \leftarrow -V_k g_k$
  Perform line search to find step length α_k, starting with $\alpha \leftarrow 1$
  Update the current point, $x_{k+1} \leftarrow x_k + \alpha_k p_k$
  Set the step length, $s_k \leftarrow \alpha_k p_k$
  Compute the change in the gradient, $y_k \leftarrow g_{k+1} - g_k$
  $A_k \leftarrow \frac{V_k y_k y_k^T V_k}{y_k^T V_k y_k}$
  $B_k \leftarrow \frac{s_k s_k^T}{s_k^T y_k}$
  Compute the updated approximation to the inverse of the Hessian, $V_{k+1} \leftarrow V_k - A_k + B_k$
until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and $\|g(x_{k-1})\| \leq \varepsilon_g$



$$V_{k+1} = \left[I - \frac{s_k y_k^T}{s_k^T y_k}\right] V_k \left[I - \frac{y_k s_k^T}{s_k^T y_k}\right] + \frac{s_k s_k^T}{s_k^T y_k}$$

ESTIMATE HESSIAN

Trust Regions

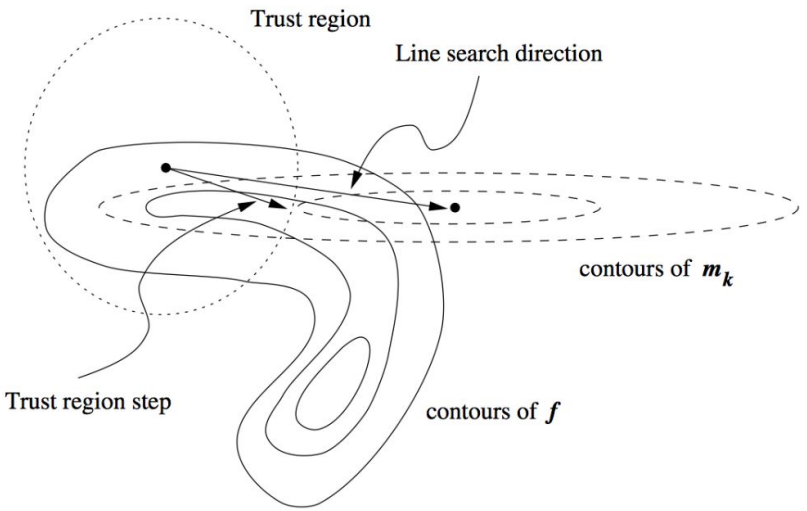
Algorithm 9 Trust region algorithm

Input: Initial guess x_0, convergence tolerances, $\varepsilon_g, \varepsilon_a$ and ε_r, initial size of the trust region, Δ_0
Output: Optimum, x^*

$k \leftarrow 0$
repeat
  Compute the Hessian of the objective function $H(x_k)$, and (approximately) solve the quadratic subproblem (Eq. (3.50))
  Evaluate the reliability index r_k (Eq. (3.51))
  if $r_k < 0.25$ **then**
    $\Delta_{k+1} \leftarrow \Delta_k/4$ {Model is poor; shrink the trust region}
  else if $r_k > 0.75$ and $\|s_k\| = \Delta_k$ **then**
    $\Delta_{k+1} \leftarrow \min(2\Delta_k, \Delta_{max})$ {Model is good and new point on edge; expand trust region}
  else
    $\Delta_{k+1} \leftarrow \Delta_k$ {New point is within trust region and/or the model is reasonable; keep trust region the same size}
  end if
  if $r_k \leq 0$ **then**
    $x_{k+1} \leftarrow x_k$ {Reject step; keep trust region centered about the same point}
  else
    $x_{k+1} \leftarrow x_k + s_k$ {Move center of trust region to new point}
  end if
  $k \leftarrow k + 1$
until $|f(x_k) - f(x_{k-1})| \leq \varepsilon_a + \varepsilon_r |f(x_{k-1})|$ and/or $\|g(x_{k-1})\| \leq \varepsilon_g$

$$\begin{aligned} \text{minimize} \quad & m(s) = f_k + g_k^T s + \frac{1}{2} s^T B_k s \\ \text{subject to} \quad & \|s\|_2 \leq \Delta_k \end{aligned}$$

$$r_k = \frac{f(x_k) - f(x_k + s_k)}{m(0) - m(s_k)}.$$



FIND STEP SIZE AND THEN PERFORM APPROXIMATE MINIMIZATION IN REGION

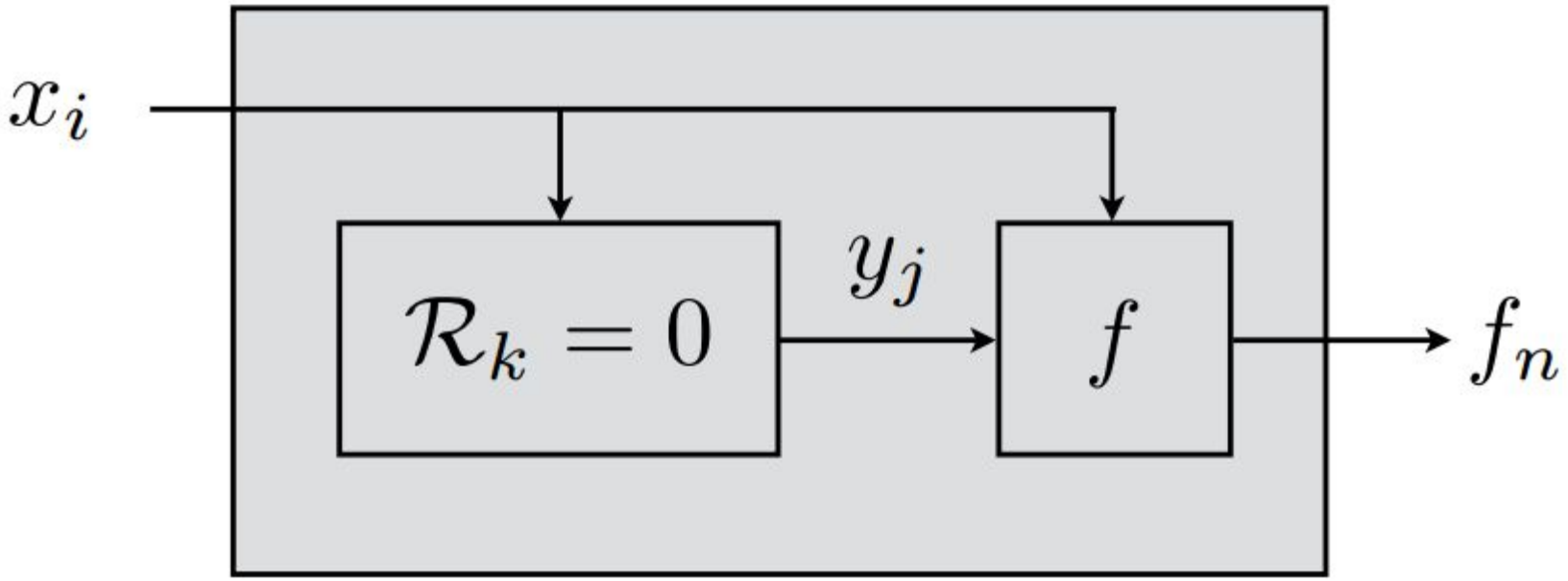
Obtaining Gradients

- Finite Difference
- Complex Step
- Symbolic Gradients
- Adjoint
- Automatic Differentiation
- Advantages and Disadvantages of each

Symbolic gradients and Adjoint gradients

Reverse / Forward (Direct/Adjoint)

$$\frac{df_n}{dx_i} = \frac{\partial f_n}{\partial x_i} - \underbrace{\frac{\partial f_n}{\partial y_j}}_{\Psi} \overbrace{\left[\frac{\partial \mathcal{R}_k}{\partial y_j}\right]^{-1}}^{\Phi} \frac{\partial \mathcal{R}_k}{\partial x_i}$$



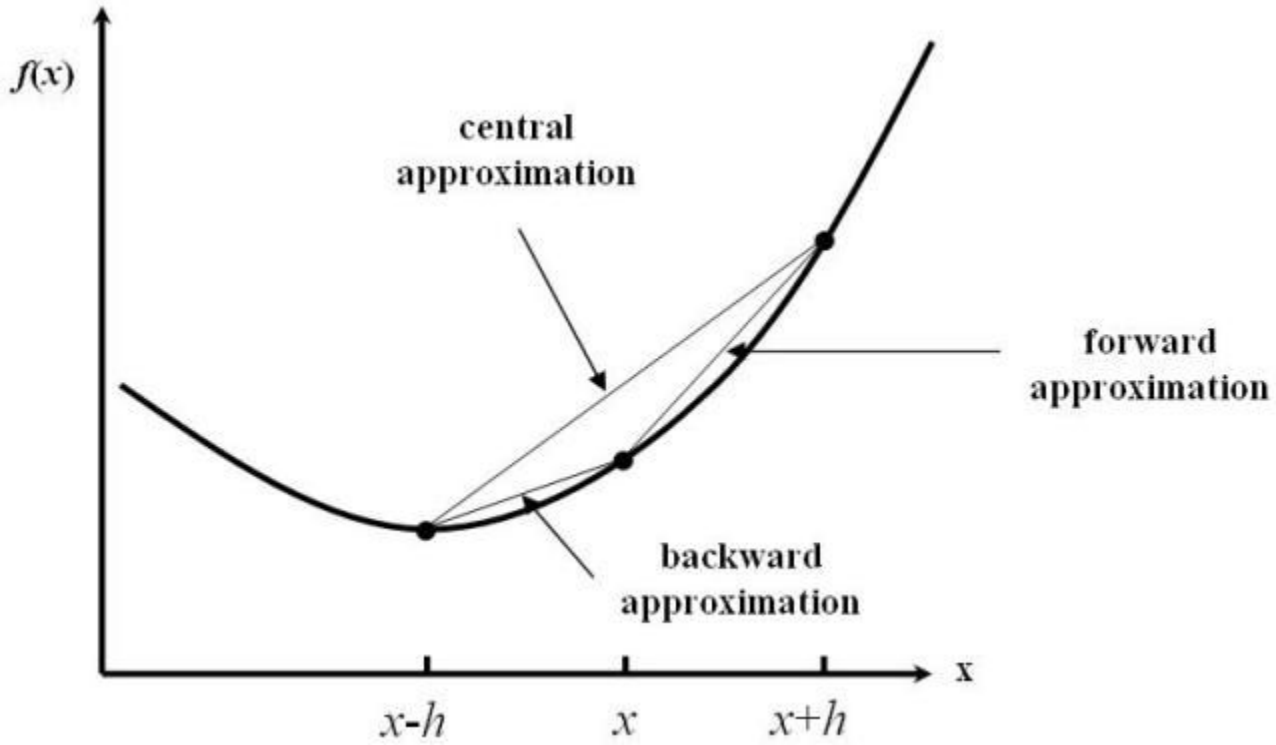
Finite Difference and Complex Step

Finite Difference

- Step size? Types?

Complex Step

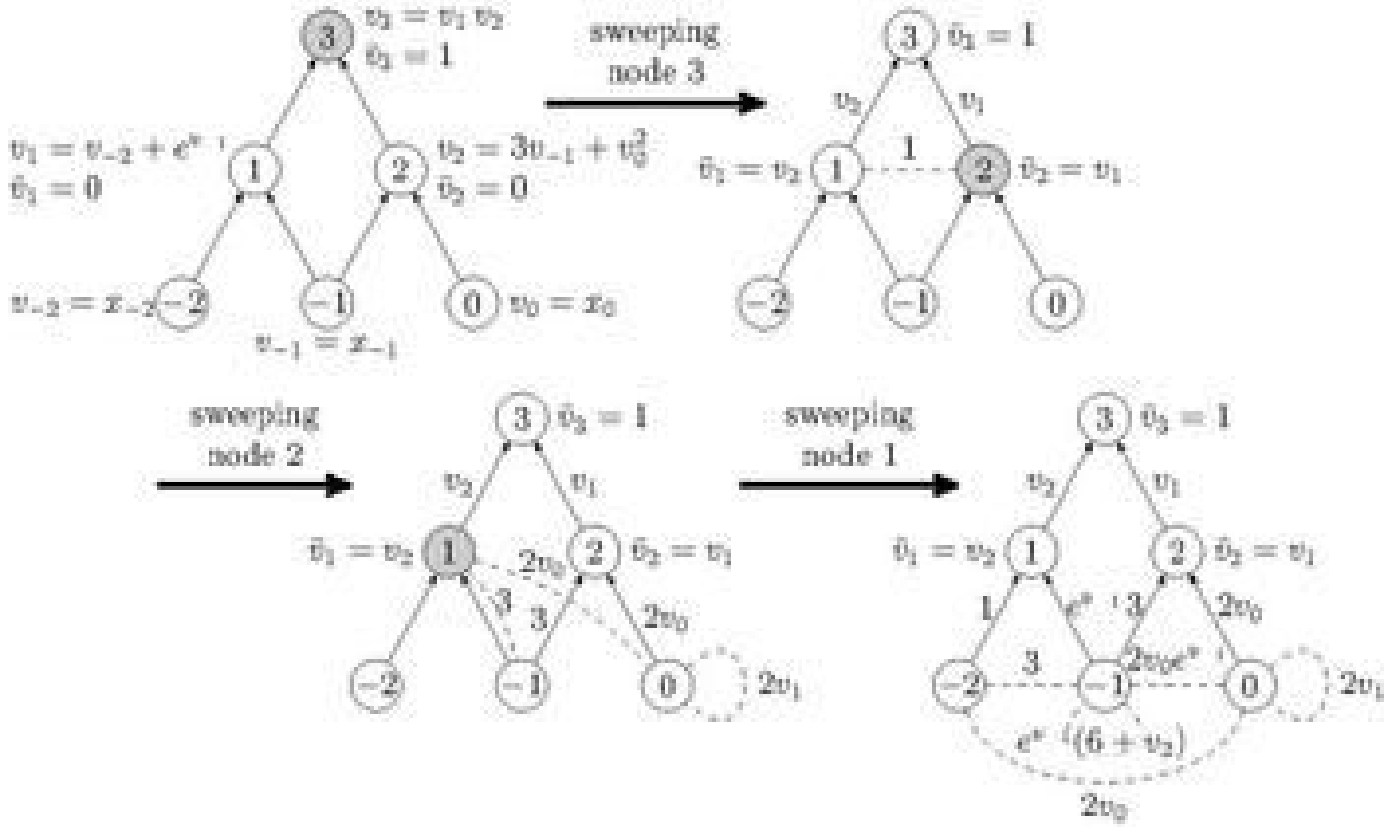
- Step size? Subtraction cancellation error



Automatic Differentiation

Overloading

Source code transformation



Lagrangian and KKT conditions

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{with respect to} \quad & x \in \mathbb{R}^n \\ \text{subject to} \quad & \hat{c}_j(x) = 0, \quad j = 1, \ldots, \hat{m} \end{aligned}$$

$$\mathrm{d}f = \frac{\partial f}{\partial x_1}\,\mathrm{d}x_1 + \frac{\partial f}{\partial x_2}\,\mathrm{d}x_2 + \cdots + \frac{\partial f}{\partial x_n}\,\mathrm{d}x_n = \nabla f^T\,\mathrm{d}x = 0.$$

$$\mathrm{d}\hat{c}_j = \frac{\partial \hat{c}_j}{\partial x_1}\,\mathrm{d}x_1 + \cdots + \frac{\partial \hat{c}_j}{\partial x_n}\,\mathrm{d}x_n = \nabla \hat{c}_j^T\,\mathrm{d}x = 0, \quad j = 1, \ldots, \hat{m}$$

$$\mathrm{d}f - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j\,\mathrm{d}\hat{c}_j = 0 \Rightarrow \sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i} - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \frac{\partial \hat{c}_j}{\partial x_i} \right) \mathrm{d}x_i = 0.$$

$$\frac{\partial f}{\partial x_i} - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \frac{\partial \hat{c}_j}{\partial x_i} = 0, \qquad (i = 1, 2, \ldots, n)$$

Lagrangian and KKT conditions

$$\mathcal{L}(x, \hat{\lambda}) = f(x) - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \hat{c}_j(x) \Rightarrow$$

$$\boxed{\mathcal{L}(x, \hat{\lambda}) = f(x) - \hat{\lambda}^T \hat{c}(x)}$$

Lagrange Multipliers

Lagrangian

KKT (Karush, Kahn, Tucker) Conditions (necessary):

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial f}{\partial x_i} - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \frac{\partial \hat{c}_j}{\partial x_i} = 0, \quad (i = 1, \ldots, n)$$

$$\frac{\partial \mathcal{L}}{\partial \hat{\lambda}_j} = \hat{c}_j = 0, \quad (j = 1, \ldots, \hat{m}).$$

second-order sufficient conditions (w is direction):

$$w^T \nabla^2_{xx} \mathcal{L}(x^*, \hat{\lambda}^*) w > 0, \quad \text{for all } w \in \mathbb{R}^n \text{ such that}$$

$$\nabla \hat{c}_j(x^*)^T w = 0, \qquad j = 1, \ldots, \hat{m}.$$

Slack variables (used for inequality constraints)

$$
\begin{aligned}
\text{minimize} \quad & f(x) \\
\text{w.r.t} \quad & x \in \mathbb{R}^n \\
\text{subject to} \quad & \hat{c}_j(x) = 0, \quad j = 1, \ldots, \hat{m} \\
& c_k(x) \geq 0, \quad k = 1, \ldots, m
\end{aligned}
$$

$$
\begin{aligned}
c(x) - s^2 &= 0 \\
c(x) &= s^2 \\
\Rightarrow c(x) &\geq 0
\end{aligned}
$$

$$
\mathcal{L}(x, \hat{\lambda}, \lambda, s) = f(x) - \hat{\lambda}^T \hat{c}(x) - \lambda^T \left(c(x) - s^2 \right)
$$

Slack variables (used for inequality constraints)

First order KKT conditions:

$$
\begin{aligned}
\nabla_x \mathcal{L} = 0 \Rightarrow \frac{\partial \mathcal{L}}{\partial x_i} &= \frac{\partial f}{\partial x_i} - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \frac{\partial \hat{c}_j}{\partial x_i} - \sum_{k=1}^{m} \lambda_k \frac{\partial c_k}{\partial x_i} = 0, \quad i = 1, \ldots, n \\
\nabla_{\hat{\lambda}} \mathcal{L} = 0 \Rightarrow \frac{\partial \mathcal{L}}{\partial \hat{\lambda}_j} &= \hat{c}_j = 0, \quad j = 1, \ldots, \hat{m} \\
\nabla_{\lambda} \mathcal{L} = 0 \Rightarrow \frac{\partial \mathcal{L}}{\partial \lambda_k} &= c_k - s_k^2 = 0 \quad k = 1, \ldots, m \\
\nabla_{s} \mathcal{L} = 0 \Rightarrow \frac{\partial \mathcal{L}}{\partial s_k} &= \lambda_k s_k = 0, \quad k = 1, \ldots, m \\
\lambda_k &\geq 0, \quad k = 1, \ldots, m.
\end{aligned}
$$

complementarity conditions

$s_k > 0$: which means that the k-th constraint is inactive, and thus the corresponding Lagrange multiplier must be zero ($\lambda_k = 0$).

$s_k = 0$: the k-th constraint is active. λ_k must then be non-negative, otherwise from the first equations, the gradient of objective and gradient of constraint point in the same direction.

Slack variables (used for inequality constraints)

Sufficient conditions:

1. KKT necessary conditions must be satisfied at x^*.
2. The Hessian matrix of the Lagrangian, Is positive definite in the feasible directions

$$\nabla^2 \mathcal{L} = \nabla^2 f(x^*) - \sum_{j=1}^{\hat{m}} \hat{\lambda}_j \nabla^2 \hat{c}_j - \sum_{k=1}^{m} \lambda_k \nabla^2 c_k \tag{5.24}$$

$$\rightleftarrows \; y^T \nabla^2 \mathcal{L}(x^*) y > 0.$$

Penalty methods

Penalty functions suffer from problems of ill conditioning. The solution of the modified problem approaches the true solution as $\lim_{\rho \to +\infty} x*(\rho) = x*$, but, as the penalty parameter increases, the condition number of the Hessian matrix of $\pi(x, \rho)$ increases and tends to ∞. This makes the problem increasingly difficult to solve numerically. The other challenge with penalty methods is that the starting value of the penalty parameter must also be chosen sufficiently large, or else the penalty function could be unbounded from below even though the original constrained problem was not

- Minimize an unconstrained problem where the **infeasibility is minimized** along with the objective
- Interior and exterior

Algorithm 11 General algorithm

Check termination conditions. if x_k satisfies the optimality conditions, the algorithm terminates successfully.

Minimize the penalty function. With x_k as the starting point, execute an algorithm to solve the unconstrained subproblem

$$\begin{aligned} \text{minimize} \quad & \pi(x; \rho_k) \\ \text{w.r.t.} \quad & x \end{aligned}$$

and let the solution of this subproblem be x_{k+1}.

Increase the penalty parameter. Set ρ_{k+1} to a larger value than ρ_k, set $k = k+1$ and return to 1.

Interior

Logarithmic Barrier Method

Used to use:

$$\pi(x,\mu) = f(x) - \mu \sum_{j=1}^{m} \log\left(c_j(x)\right),$$

Now use:

$$\begin{aligned} \text{minimize} \quad & f(x) - \mu \sum_{j=1}^{m} \log\left(s_j\right) \\ \text{subject to} \quad & c_j(x) - s_j = 0 \\ & s_j \geq 0 \quad \text{automatically enforced through the logarithm} \end{aligned}$$

Interior

Inverse Barrier Method

Barrier parameter

$$\pi(x,\mu) = f(x) + \mu \sum_{j=1}^{m} \frac{1}{c_j(x)},$$

$$\lim_{\mu \to 0} x^*(\mu) = x^*$$

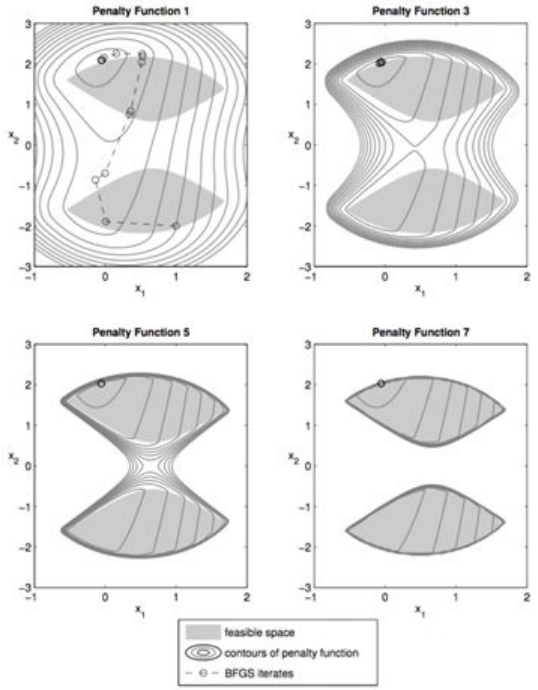
A value of 0.1 for the ratio μk+1/μk is usually considered ambitious

Exterior

Quadratic Penalty Method (QP)

$$\pi(x;\rho) = f(x) + \frac{\rho}{2}\sum_{i=1}^{\hat{m}} \hat{c}_i(x)^2 = f(x) + \frac{\rho}{2}\hat{c}(x)^T\hat{c}(x).$$

$$\phi(x;\rho) = \frac{\rho}{2}\sum_{i=1}^{m} \left(\max\left[0, -c_i(x)\right]\right)^2.$$



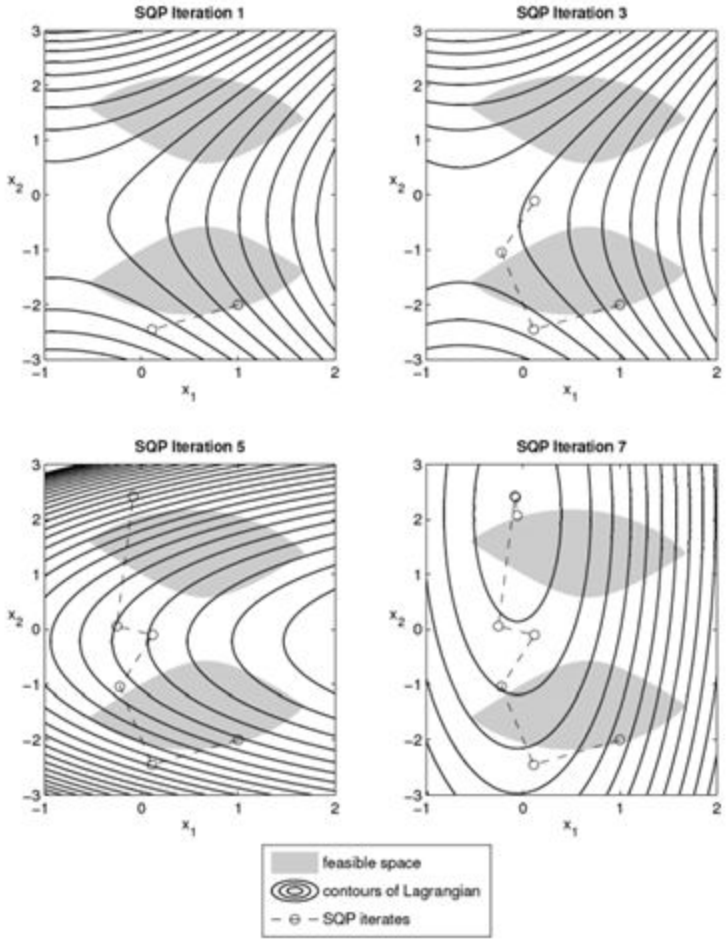
Exterior

Augmented Lagrangian

$$\mathcal{L}_A(x, \lambda; \rho) = f(x) - \sum_i \lambda_i c_i(x) + \frac{\rho}{2} \sum_i c_i(x)^2$$

$$c_i(x^k) \approx -\frac{1}{\rho^k}(\lambda_i^* - \lambda_i^k)$$

Sequential Quadratic Programming (SQP)



Gradient-Free Optimization- Nelder-Mead

Creating Simplex:

$$b = \frac{c}{n\sqrt{2}}\left(\sqrt{n+1}-1\right)$$

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

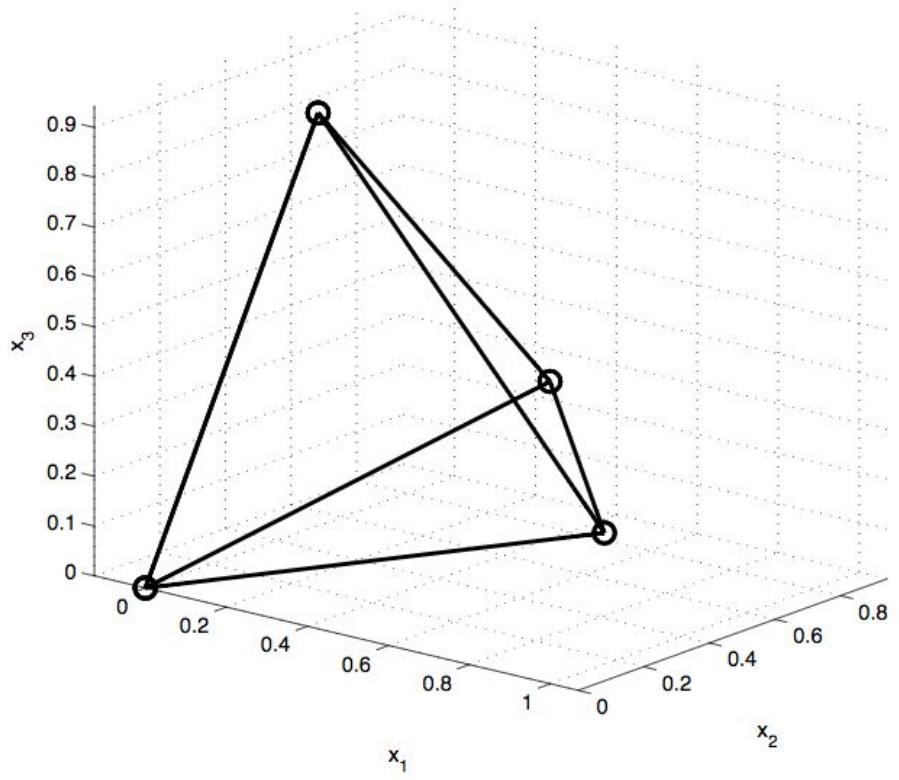
x0 = [0,0,0,0,...]

x1 = [a,b,b,b,...]

x2 = [b,a,b,b,...]

x3 = [b,b,a,b,...]

x4 = [b,b,b,a,...]

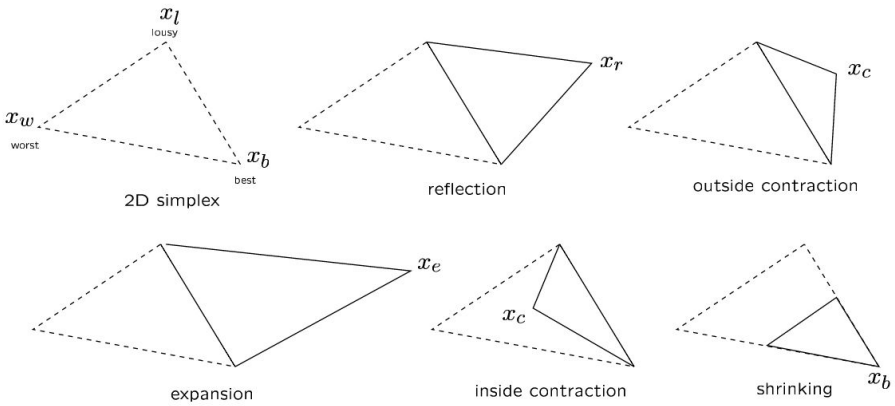


Gradient-Free Optimization- Nelder-Mead

Algorithm 13 Nelder–Mead Algorithm

Input: Initial guess, x_0
Output: Optimum, $x^\star$

$k \leftarrow 0$
Create a simplex with edge length c
repeat
  Identify the highest (x_w: worst), second highest (x_l, lousy) and lowest (x_b: best) value points with function values f_w, f_l, and f_b, respectively
  Evaluate x_a, the average of the point in simplex excluding x_w
  Perform *reflection* to obtain $x_r = x_a + \alpha_r (x_a - x_w)$, evaluate f_r
  if $f_r < f_b$ **then**
    Perform *expansion* to obtain $x_e = x_r + \alpha_e (x_r - x_a)$, evaluate f_e.
    if $f_e < f_b$ **then**
      $x_w \leftarrow x_e$, $f_w \leftarrow f_e$ (accept expansion)
    else
      $x_w \leftarrow x_r$, $f_w \leftarrow f_r$ (accept reflection)
    end if
  else if $f_r \leq f_l$ **then**
    $x_w \leftarrow x_r$, $f_w \leftarrow f_r$ (accept reflected point)
  else
    if $f_r > f_w$ **then**
      Perform an *inside contraction* to obtain $x_c = x_a - \beta (x_a - x_w)$, and evaluate f_c
      if $f_c < f_w$ **then**
        $x_w \leftarrow x_c$ (accept contraction)
      else
        Shrink the simplex
      end if
    else
      Perform an *outside contraction* to obtain $x_o = x_a + \beta (x_a - x_w)$, and evaluate f_o
      if $f_o \leq f_r$ **then**
        $x_w \leftarrow x_o$ (accept contraction)
      else
        Shrink the simplex
      end if
    end if
  end if
  $k \leftarrow k + 1$
until $(f_w - f_b) < (\varepsilon_1 + \varepsilon_2 |f_b|)$



Gradient-Free Optimization- Genetic Algorithm

1. Initialize population- 15 to 20 times the number of design variables
2. **Determine mating pool**
   a. Tournament- prefer feasible options; 2 feasible- better objective; 2 infeasible- smaller violation
   b. Roulette wheel- create better probabilities for better fitness
3. **Generate offspring**- crossover; interpolation, extrapolation, pass on bits, etc.
4. **Mutation**- add randomness to the offspring
5. Compute offspring fitness- based on objective function
6. Identify best member- best of population hasn't changed for 10 generations
7. Return to Step 2 or STOP

Selection (survival of the fittest), Crossover (reproduction), Mutation (variation)

Gradient-Free Optimization- Particle Swarm

1. Initialize positions (x) and velocities (v)
2. Evaluate function values f(x)
3. Update best position for particles (p_i) and for the whole swarm (p_g)
4. Update velocities and positions- enforce bounds
5. Repeat Steps 2-4 until convergence
   a. Best fitness hasn't changed for many iterations
   b. Mean of the particle positions (p_i) hasn't changed for several iterations
   c. Velocity of particles falls below tolerance for several iterations
   d. Distance between particles and best position (p_g) is below tolerance
   e. Distance between best and worst fitness is below tolerance

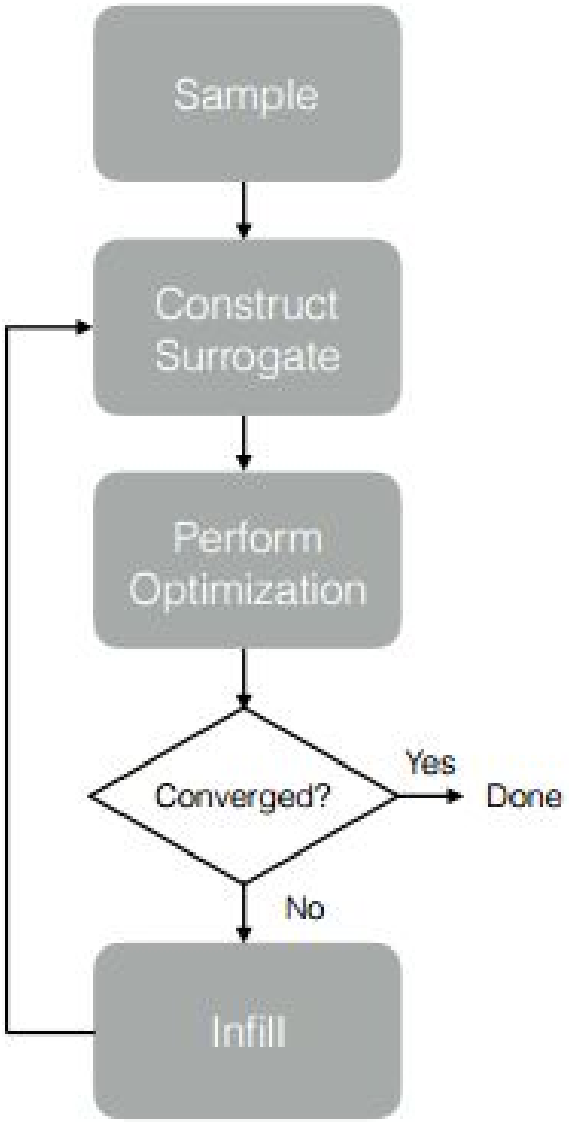
$$\bar{v}_{k+1}^i = \bar{w}\bar{v}_k^i + c_1 r_1 \left(p_k^i - x_k^i\right) + c_2 r_2 \left(p_k^g - x_k^i\right)$$

$$x_{k+1}^i = x_k^i + \bar{v}_{k+1}^i$$

c_1- [0,2] self-confidence, c_2- [0,2] swarm-confidence

w- [0,1.2] inertial parameter, r_1, r_2- [0,1] randomness

Surrogate-Based Optimization (SBO)



- Simulation is expensive (takes a long time to run).
- Simulation output is noisy.
- Experimental data.
- Understand functional relationships
- Multifidelity optimization.

Making optimization robust and reliable

Robust

Minimize both mean

and standard deviation

Reliable

Deterministic Optimum

Worst Case

Transmitted Variance

Finding K

Monte Carlo

- Robust: performance is less sensitive to inherent variability (objective function)
- Reliable: less prone to failure under inherent variability (constraints)

$$\Delta c = \sum_{i=1}^{n} \left| \frac{\partial c}{\partial x_i} \Delta x_i \right| + \sum_{j=1}^{m} \left| \frac{\partial c}{\partial p_j} \Delta p_j \right|$$

$$\sigma_c^2 = \sum_{i=1}^{n} \left(\frac{\partial c}{\partial x_i} \sigma_{x_i} \right)^2 + \sum_{j=1}^{m} \left(\frac{\partial c}{\partial p_j} \sigma_{p_j} \right)^2$$

Optimizers

- MATLAB
  - fmincon
  - fminunc
  - ga
- PYTHON
  - Scipy
    - SLSQP
    - Nelder-Mead
  - Pyoptsparse
    - SNOPT
    - NSGA