module 09
unconstrained optimization

ahmad f. taha

ce 2989 — numerical methods in civil & env. engineering

email: ahmad.taha@vanderbilt.edu

webpage: http://lab.vanderbilt.edu/taha

April 14, 2025

optimization algorithm

- objective is to solve minimize $f(x)$ for $x \in \mathbb{R}^n$
- **rule to provide a sequence** $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$

$$x^{(k+1)} = F^{(k)}(x^{(k)})$$

  and a termination criterion
- if the algorithm is started at the problem solution, then it should not move
- convergence can be expressed in several ways, e.g.,

$$\lim_{k \to \infty} f(x^{(k)}) = f(x^*)$$

$$\lim_{k \to \infty} x^{(k)} = x^*$$

- **convergence speed**: how fast does the algorithm converge?
- how many iterations does it take to guarantee that $|f(x^{(k)}) - f^*| \leq \varepsilon$?
- **complexity analysis**: how many operations (additions, multiplications, ...) does it take to converge, as a function of the problem size (number of variables, number of constraints)?

unconstrained minimization algorithms

- Algorithms to solve

$$\text{minimize} \quad f(x)$$

- Interpreted as methods to solve $\nabla f(x) = 0$
- If the problem is convex, convergence to the global minimum
- If the problem is not convex, convergence to a point satisfying $\nabla f(x) = 0$ (which may or may not be local minimum)
- Points satisfying $\nabla f(x) = 0$ are called stationary points

Descent algorithms

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)} \text{ with } f(x^{(k+1)}) < f(x^{(k)})$$

- $\Delta x^{(k)}$: Search direction (vector in $\mathbb{R}^n$)
- $t^{(k)} > 0$: Stepsize

General descent algorithm:

Require: Starting point $x \in \mathrm{dom}(f)$

repeat

Compute search direction Δx

Line search: Compute step size t

Update: $x^+ := x + t\Delta x$

until convergence criterion is satisfied

Search directions for descent

- To guarantee descent, it is necessary to choose $\Delta x^{(k)}$ so that $\nabla f(x^{(k)})^T \Delta x^{(k)} < 0$
- You can see this geometrically for a simple function $f(x) = x^2$
- For a nonconvex function, use Taylor's 1st-order approximation

$$f(x^{(k+1)}) = f(x^{(k)} + t^{(k)} \Delta x^{(k)}) \approx f(x^{(k)}) + t^{(k)} \nabla f(x^{(k)})^T \Delta x^{(k)}$$

- **Bottomline:** If $\nabla f(x^{(k)})^T \Delta x^{(k)} < 0$ and $t^{(k)}$ is sufficiently small, descent can be guaranteed (even if the function is nonconvex)
- $t^{(k)}$ **sufficiently small is not enough for convergence**
- Proper value of $t^{(k)}$ must be selected through *line search*

Example

Solve this using steepest descent: $\min f(x) = 4x_1^2 - 4x_1x_2 + 2x_2^2$

1. Start with initial guess: $x^{(0)} = [2, 3]^T$
2. Compute gradient at this guess: $\nabla f(x^{(0)}) = \begin{bmatrix} 8x_1 - 4x_2 \\ -4x_1 + 4x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$
3. Find a t that minimizes $f(x^{(0)} - t\nabla f(x^{(0)})) = f\left(\begin{bmatrix} 2 - 4t \\ 3 - 4t \end{bmatrix}\right)$ or

$$t = \arg\min_{t>0} \underbrace{\left(4(2-4t)^2 - 4(2-4t)(3-4t) + 2(2-4t)^2\right)}_{g(t)}$$

4. To compute an optimal t, take derivative of $g'(t) = 0$ to obtain

$$g'(t) = 64t - 32 = 0 \Rightarrow t^* = 0.5, \quad g''(t) > 0$$

5. Compute next iterate: $x^{(1)} = x^{(0)} - t\nabla f(x^{(0)}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
6. Go to Step 2 and repeat to get

$$x^{(0)} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \; x^{(1)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; x^{(2)} = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}, \; x^{(3)} = \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}, \; x^{(4)} = \begin{bmatrix} 0.08 \\ 0.12 \end{bmatrix}$$

Search directions

1. **Steepest descent:** $\Delta x^{(k)} = -\nabla f(x^{(k)})$
2. **Newton's method:** $\Delta x^{(k)} = -(\nabla^2 f(x^{(k)}))^{-1} \nabla f(x^{(k)})$
3. **DFP**, **BFGS**, etc..

DFP Method

The **Davidon–Fletcher–Powell formula** (or **DFP**; named after William C. Davidon, Roger Fletcher, and Michael J. D. Powell) finds the solution to the secant equation that is closest to the current estimate and satisfies the curvature condition. It was the first quasi-Newton method to generalize the secant method to a multidimensional problem. This update maintains the symmetry and positive definiteness of the Hessian matrix.

Given a function $f(x)$, its gradient (∇f), and positive-definite Hessian matrix B, the Taylor series is

$$f(x_k + s_k) = f(x_k) + \nabla f(x_k)^T s_k + \frac{1}{2} s_k^T B s_k + \dots,$$

and the Taylor series of the gradient itself (secant equation)

$$\nabla f(x_k + s_k) = \nabla f(x_k) + B s_k + \dots$$

is used to update B.

The DFP formula finds a solution that is symmetric, positive-definite and closest to the current approximate value of B_k:

$$B_{k+1} = (I - \gamma_k y_k s_k^T) B_k (I - \gamma_k s_k y_k^T) + \gamma_k y_k y_k^T,$$

where

$$y_k = \nabla f(x_k + s_k) - \nabla f(x_k),$$
$$\gamma_k = \frac{1}{y_k^T s_k},$$

and B_k is a symmetric and positive-definite matrix.

The corresponding update to the inverse Hessian approximation $H_k = B_k^{-1}$ is given by

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \frac{s_k s_k^T}{y_k^T s_k}.$$

B is assumed to be positive-definite, and the vectors s_k^T and y must satisfy the curvature condition

$$s_k^T y_k = s_k^T B s_k > 0.$$

The DFP formula is quite effective, but it was soon superseded by the Broyden–Fletcher–Goldfarb–Shanno formula, which is its dual (interchanging the roles of *y* and *s*).[1]

Improved DFP: The BFGS Method

Consider the following unconstrained optimization problem

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} \quad f(\mathbf{x}),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear objective function.

From an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$ and an initial guess of the Hessian matrix $B_0 \in \mathbb{R}^{n\times n}$ the following steps are repeated as $\mathbf{x}_k$ converges to the solution:

1. Obtain a direction $\mathbf{p}_k$ by solving $B_k\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$.
2. Perform a one-dimensional optimization (line search) to find an acceptable stepsize α_k in the direction found in the first step. If an exact line search is performed, then $\alpha_k = \arg\min f(\mathbf{x}_k + \alpha\mathbf{p}_k)$. In practice, an inexact line search usually suffices, with an acceptable α_k satisfying Wolfe conditions.
3. Set $\mathbf{s}_k = \alpha_k\mathbf{p}_k$ and update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
4. $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.
5. $B_{k+1} = B_k + \dfrac{\mathbf{y}_k\mathbf{y}_k^{\mathrm{T}}}{\mathbf{y}_k^{\mathrm{T}}\mathbf{s}_k} - \dfrac{B_k\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}}B_k^{\mathrm{T}}}{\mathbf{s}_k^{\mathrm{T}}B_k\mathbf{s}_k}$.

Convergence can be determined by observing the norm of the gradient; given some $\epsilon > 0$, one may stop the algorithm when $||\nabla f(\mathbf{x}_k)|| \leq \epsilon$. If B_0 is initialized with $B_0 = I$, the first step will be equivalent to a gradient descent, but further steps are more and more refined by B_k, the approximation to the Hessian.

The first step of the algorithm is carried out using the inverse of the matrix B_k, which can be obtained efficiently by applying the Sherman–Morrison formula to the step 5 of the algorithm, giving

$$B_{k+1}^{-1} = \left(I - \frac{\mathbf{s}_k\mathbf{y}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}\right)B_k^{-1}\left(I - \frac{\mathbf{y}_k\mathbf{s}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}\right) + \frac{\mathbf{s}_k\mathbf{s}_k^T}{\mathbf{y}_k^T\mathbf{s}_k}.$$

This can be computed efficiently without temporary matrices, recognizing that B_k^{-1} is symmetric, and that $\mathbf{y}_k^{\mathrm{T}}B_k^{-1}\mathbf{y}_k$ and $\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k$ are scalars, using an expansion such as

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k + \mathbf{y}_k^{\mathrm{T}}B_k^{-1}\mathbf{y}_k)(\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}})}{(\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k)^2} - \frac{B_k^{-1}\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}} + \mathbf{s}_k\mathbf{y}_k^{\mathrm{T}}B_k^{-1}}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k}.$$

Therefore, in order to avoid any matrix inversion, the **inverse** of the Hessian can be approximated instead of the Hessian itself: $H_k \overset{\text{def}}{=} B_k^{-1}$.[9]

From an initial guess $\mathbf{x}_0$ and an approximate **inverted** Hessian matrix H_0 the following steps are repeated as $\mathbf{x}_k$ converges to the solution:

1. Obtain a direction $\mathbf{p}_k$ by solving $\mathbf{p}_k = -H_k\nabla f(\mathbf{x}_k)$.
2. Perform a one-dimensional optimization (line search) to find an acceptable stepsize α_k in the direction found in the first step. If an exact line search is performed, then $\alpha_k = \arg\min f(\mathbf{x}_k + \alpha\mathbf{p}_k)$. In practice, an inexact line search usually suffices, with an acceptable α_k satisfying Wolfe conditions.
3. Set $\mathbf{s}_k = \alpha_k\mathbf{p}_k$ and update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{s}_k$.
4. $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.
5. $H_{k+1} = H_k + \dfrac{(\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k + \mathbf{y}_k^{\mathrm{T}}H_k\mathbf{y}_k)(\mathbf{s}_k\mathbf{s}_k^{\mathrm{T}})}{(\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k)^2} - \dfrac{H_k\mathbf{y}_k\mathbf{s}_k^{\mathrm{T}} + \mathbf{s}_k\mathbf{y}_k^{\mathrm{T}}H_k}{\mathbf{s}_k^{\mathrm{T}}\mathbf{y}_k}$.

Stepsize selection methods

1. Exact line search: t solves $\min_{t>0} f(x + t\Delta x)$
   - Inexact line search: Minimize a quadratic or cubic interpolation of $g(t) = f(x + t\Delta x)$ with respect to t
2. **Backtracking line search (Armijo rule)**
   - **In contrast to exact line search, backtracking only needs function evaluations**
3. Constant stepsize: $t^{(k)} = t > 0$ for all $k = 0, 1, 2, \ldots$
   - Simple choice: no need for minimization as with the exact line search, no need for function evaluations as with backtracking
   - Convergence to stationary point under certain conditions
   - In practice, set t to a small value; in case of divergence, reduce t

Backtracking line search + convergence

Parameters: $0 < \beta < 1$, $0 < \alpha < 1/2$

- Start with $t = 1$
- If $f(x + t\Delta x) < f(x) + \alpha t \nabla f(x)^T \Delta x$, accept t and exit
- Otherwise, set $t := \beta t$, and check the condition again
- As long as $\nabla f(x)^T \Delta x < 0$, the set of acceptable stepsizes t will always contain an interval of the form $[0, \delta]$, and the line search will finish in a finite number of steps
- Generally, convergence to a *stationary point* is guaranteed when
  1. exact/inexact line minimization or backtracking line search is used, and
  2. the search direction is chosen so that $\nabla f(x^{(k)})^T \Delta x^{(k)} < 0$
- Additional conditions on f can guarantee convergence under the (simpler) constant or dimishing stepsize rules.

Convergence (termination) criteria

1. Scale-free
   - $\|\nabla f(x^{(k)})\| \leq \epsilon \|\nabla f(x^{(0)})\|$
   - $\|x^{(k)} - x^{(k-1)}\| \leq \epsilon \|x^{(k)}\|$
2. Scale-dependent
   - $\|\nabla f(x^{(k)})\| \leq \epsilon$
   - $\|x^{(k)} - x^{(k-1)}\| \leq \epsilon$

- Scale-free are preferable to scale-dependent
- Setting of ϵ may require trial and error
- Algorithm specific criteria also exist

Newton's method

- Newton direction

$$\Delta x = -(\nabla^2 f(x))^{-1} \nabla f(x)$$

- If $\nabla^2 f(x) \succ 0$, then Δx is a descent direction

$$-\nabla f(x)^T (\nabla^2 f(x))^{-1} \nabla f(x) < 0$$

unless $\nabla f(x) = 0$, which implies that x is a stationary point

Newton's method:

Require: Starting point $x \in \mathrm{dom}(f)$

repeat

Compute search direction: Solve the linear system $\nabla^2 f(x) \Delta x = -\nabla f(x)$

Line search: Compute step size t by line search

Update: $x^+ := x + t\Delta x$

until convergence criterion is satisfied

Example: minimizing the Powell function

$$f(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

$$x^0 = [3, -1, 0, 1]^\top$$

Solution: We are going to do three iterations with fixed step size $t = 1$. Note that $f(x^0) = 215$, and we have

$$\nabla f(x) = \begin{bmatrix} 2(x_1 + 10x_2) + 40(x_1 - x_4)^3 \\ 20(x_1 + 10x_2) + 4(x_2 - 2x_3)^3 \\ 10(x_3 - x_4) - 8(x_2 - 2x_3)^3 \\ -10(x_3 - x_4) - 40(x_1 - x_4)^3 \end{bmatrix}$$

$$\nabla^2 f(x) = \begin{bmatrix} 2 + 120(x_1 - x_4)^2 & 20 & 0 & -120(x_1 - x_4)^2 \\ 20 & 200 + 12(x_2 - 2x_3)^2 & -24(x_2 - 2x_3)^2 & 0 \\ 0 & -24(x_2 - 2x_3)^2 & 10 + 48(x_2 - 2x_3)^2 & -10 \\ -120(x_1 - x_4)^2 & 0 & -10 & 10 + 120(x_1 - x_4) \end{bmatrix}$$

Iteration 1

$$\nabla f(x^0) = [306, -144, -2, -310]^\top$$

$$\nabla^2 f(x^0) = \begin{bmatrix} 482 & 20 & 0 & -480 \\ 20 & 212 & -24 & 0 \\ 0 & -24 & 58 & -10 \\ -480 & 0 & -10 & 490 \end{bmatrix},$$

$$(\nabla^2 f(x^0))^{-1} = \begin{bmatrix} 1126 & -0.0089 & 0.0154 & 0.1106 \\ -0.0089 & 0.0057 & 0.008 & -0.0087 \\ 0.0154 & 0.008 & 0.0203 & 0.0155 \\ 0.1106 & -0.0087 & 0.0155 & 0.1107 \end{bmatrix} \Rightarrow \textbf{MATLAB}$$

$$\Delta x^0 = [1.4127, -0.8413, -0.254, -0.746]^\top, \text{ hence}$$

$$x^1 = x^0 - \Delta x^0 = [1.5873, -0.1587, 0.254, 0.254]^\top,$$

$$f(x^1) = 31.8$$

Iteration 2

Calculate $\nabla f(x^1), \nabla^2 f(x^1), (\nabla^2 f(x^1))^{-1}$

$$\Delta x^1 = [0.5291, -0.0529, 0.0846, 0.0846]^\top, \text{ Hence,}$$

$$x^2 = x^1 - \Delta x^1 = [1.0582, -0.1058, 0.1694, 0.1694]^\top,$$

$$f(x^2) = 6.28$$

Iteration 3

Calculate $\nabla f(x^{(2)}), \nabla^2 f(x^{(2)}), (\nabla^2 f(x^{(2)}))^{-1}, \Delta x^{(2)}$

$$x^{(3)} = x^{(2)} - \Delta x^{(2)} = [0.7037, -0.0704, 0.1121, 0.1111]^\top,$$

$$f(x^{(3)}) = 1.24$$

Levenberg-Marquardt Modification of Newton's method

- Newton's method can fail if $\nabla^2 f(x)$ is singular (Newton direction is not defined) or $\nabla^2 f(x) \prec 0$ (Newton direction is not a descent direction)
- Newton's method: $x^{(k+1)} = x^{(k)} - t^{(k)}[\nabla^2 f(x^{(k)})]^{-1}\nabla f(x^{(k)})$
- Levenberg-Marquardt modification: $x^{(k+1)} = x^{(k)} - t^{(k)}[\mu_k I + \nabla^2 f(x^{(k)})]^{-1}\nabla f(x^{(k)})$ with $\mu_k > 0$
- Search direction $\Delta x^{(k)} = -[\mu_k I + \nabla^2 f(x^{(k)})]^{-1}\nabla f(x^{(k)})$ can always be made to be a descent direction for sufficiently large μ_k
- Method approaches the behavior of Newton's method for $\mu_k \to 0$ and of a gradient method for $\mu_k \to \infty$
- Implementation aspects
  - Start with a small value of μ_k and increase it slowly until the iteration is descent: $f(x^{(k+1)}) < f(x^{(k)})$
  - Search direction computed as solution to $(\mu I + \nabla^2 f(x))\Delta x = -\nabla f(x)$
  - Combine with line search

Some observations

- Newton's method has several advantages over gradient and steepest descent
  - Convergence of Newton's method is rapid in general and quadratic near x^*
  - Once quadratic convergence phase is reached, at most six or so iterations are required to produce a solution
  - Newton's method is *affine invariant*: it is insensitive to the choice of coordinates, or the condition number of the sublevel sets of the objective
  - Newton's method scales well with problem size: its performance on problems in $\mathbb{R}^{10000}$ is similar to its performance on problems in $\mathbb{R}^{10}$, with only a small increase in the number of steps required
  - The good performance of Newton's method is not dependent on the choice of algorithm parameters
  - In contrast, the choice of norm for steepest descent plays a critical role in its performance
- Main disadvantage: the cost of forming and storing the Hessian, cost of computing the Newton step, which requires solving a set of linear equations.
- But you can exploit problem structure

Conjugate Gradient Algorithms

- How about an algorithm that is an intermediate between the method of steepest descent and Newton's method?
  - Solves quadratics of n variables in n steps
  - Requires no Hessian matrix evaluations
  - No matrix inversion and no storage of an $n \times n$ matrix required
- This method performs better than steepest descent, but not as well as Newton's
- The crucial factor in the efficiency of an iterative search method is the direction of search
- Won't be covered here in this class

class summary (can you believe that we are done?)

what did we learn?