Econ 7218 Computational Methods for Econometrics
Linear Equations and Nonlinear Optimization Methods

Chih-Sheng Hsieh

Department of Economics
National Taiwan University

February 19, 2023

Linear Equations

- Solving linear equations is a common practice in computational economic analysis. For example, researchers may need to solve an equilibrium from a demand-supply system or an input-output system.
- In a linear equation, one needs to compute the $n \times 1$ solution vector of x that satisfies

$$Ax = b,$$

  where A is a $n \times n$ matrix and b is a $n \times 1$ vector.
- Although the algebra solution $x = A^{-1}b$ is known, but computing A^{-1} numerically is costly when n is large and therefore how to efficiently compute x is a practical issue, especially when computing such a solution is an elementary task for solving more complicated economic models or need to perform many times.

Linear Equations

Example:

Spatial autoregressive model is used to study spatial (social) interactions and spillover effects between spatial units (network members).

$$Y = \lambda WY + X\beta + \epsilon$$

- Coefficient λ captures the spatial interaction (peer) effect.
- W is a square matrix representing a spatial weights matrix (or a network matrix) – W_{ij} equals one if there is a link between i and j, and zero otherwise.
- The model can be transformed into a reduced form,

$$Y = (I - \lambda W)^{-1}(X\beta + \epsilon).$$

- This reduced form illustrates how to simulate Y given X, ϵ, and coefficients λ and β via solving a linear equation.

LU factorization

- Some linear equations $Ax = b$ are relatively easy to solve. For example, if A is a lower triangular matrix (all non-zero elements are on or below the diagonal),

$$A = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ & & & & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix},$$

then x can be solved recursively using **forward substitution**:

$$x_1 = b_1/a_{11}$$
$$x_2 = (b_2 - a_{21}x_1)/a_{22}$$
$$x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1})/a_{nn}$$

LU factorization

- Similarly, if A is an upper triangular matrix, then the elements of x can be computed recursively by **backward substitution**.
- As most of the linear equations do not have a triangular matrix A, in such cases, **L-U factorization algorithm** utilizes the Gaussian elimination to solve the linear equation with a **non-singular** square matrix A.
- The L-U factorization algorithm involves two phrases:
  1. Factorization: Using Gaussian elimination to factor the matrix A into the product $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix.
  2. Solution: $Ax = (LU)x = L(UX) = b$. First solve y from $Ly = b$ using forward substitution and then solve x from $Ux = y$ using backward substitution.

Gaussian Elimination

Given a linear system expressed in matrix form $Ax = b$, first write down the corresponding augmented matrix: $[A|b]$. Then perform a sequence of row operations, which are any of the following:

a. Interchange any two rows
b. Multiply a row by a nonzero constant
c. Add a multiple of one row to another row

The goal of these operations is to transform the original augmented matrix into the row echelon form $[A'|b']$, where A' is upper triangular so that the system of $A'x = b'$ can be easily solved by backward substitution.

Example: solve the system

$$x - 2y + z = 0$$
$$2x + y - 3z = 5$$
$$4x - 7y + z = -1$$

LU factorization

Some remarks about L-U factorization:

- If any of the diagonal element from the matrix A is zero or small, then one could use the **pivoting** to rearrange the rows (or even columns) of the matrix so that the L-U factorization is more numerically precise, the so-called **LUP** factorization.

$$PAQ = LU$$

  where P is a permutation matrix which, when left-multiplied to A, reorders the rows of A and Q is a permutation matrix that reorders the columns of A.

- In terms of computation cost, the factorization step involves roughly $n^3/3$ multiplications and additions. Solving the two triangular systems use a total of n^2 multiplications and divisions. So the total computation cost is roughly $n^3/3 + n^2$ operations.

QR factorization

QR factorization is an alternative to L-U factorization.

- We say that A is orthogonal if $A^\top A = AA^\top = I$.
- **QR factorization** for an arbitrary **non-singular** square matrix A is $A = QR$ where Q is orthogonal and R is upper triangular.
- There are also **QL**, **RQ**, or **LQ** factorization.
- We can solve $Ax = b$ by

$$Q^\top Ax = Q^\top b \Leftrightarrow Q^\top QRx = Q^\top b \Leftrightarrow Rx = Q^\top b.$$

- Since R is upper triangular, x can be computed by applying back-substitution to $Rx = Q^\top b$.

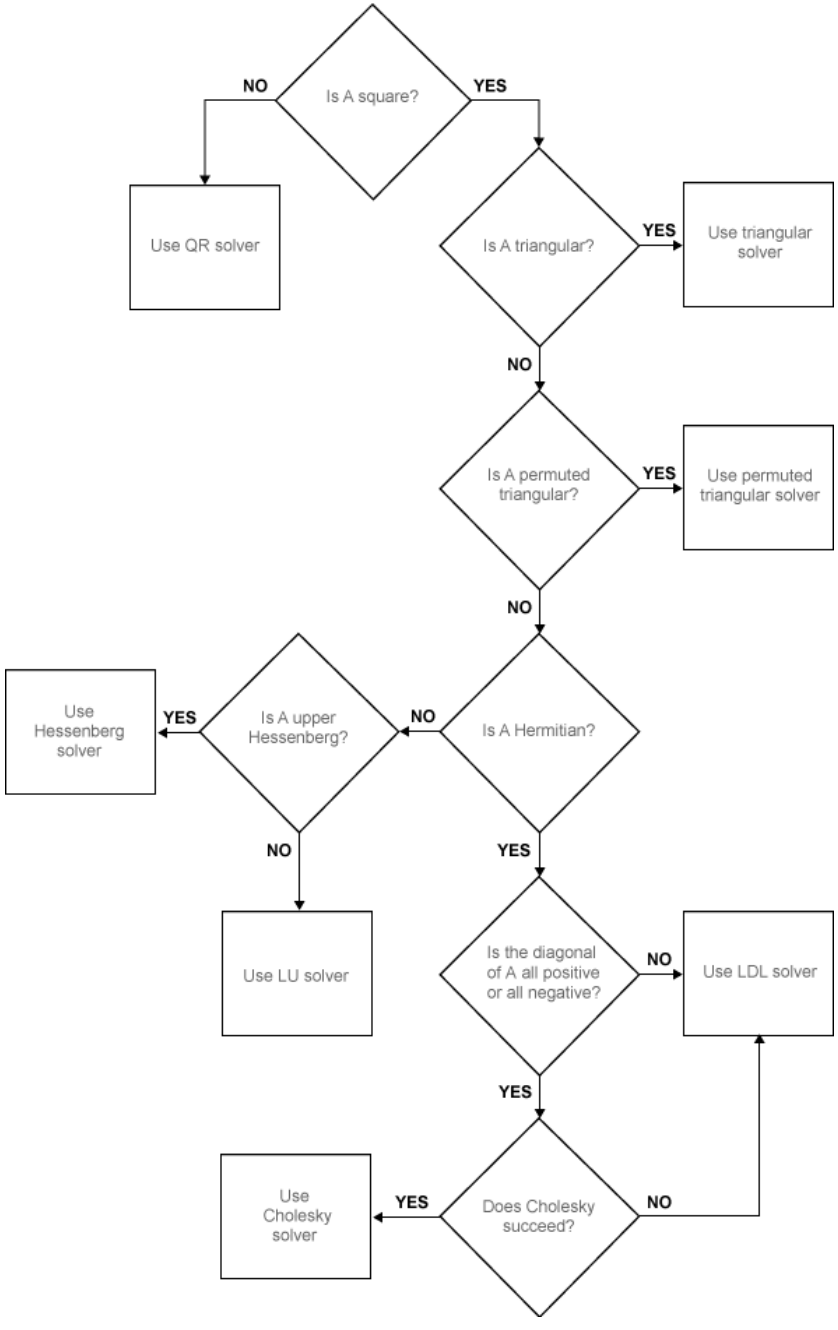
Cholesky factorization

- When the matrix A is **symmetric** and **positive definite**, a special form of factorization, the **Cholesky factorization** algorithm, can be applied.
- Cholesky factorization decomposes the matrix A as $A = U^\top U$, the product of an upper triangular matrix U and its transpose.
- The linear equation $Ax = U^\top Ux = U^\top (Ux) = b$ can be solved efficiently by using forward substitution to solve $U^\top y = b$ and then using backward substitution to solve $Ux = y$.
- Cholesky decomposition only involves $n^3/6$ multiplications and additions, which is about half of the cost of LU decomposition.
- It is also more stable than the LU decomposition because there is no need for pivoting.
- Check the exercise code to compare the computational cost of LU, QR, and Cholesky decomposition methods.

Cholesky factorization

- An important application of Cholesky decomposition occurs in the probability theory.
- If $Y \sim N(\mu, \Sigma)$ is a multivariate normal random vector with mean μ and variance Σ, where Σ is symmetric and positive definite.
- If $\Sigma = \Omega\Omega^\top$ is a Cholesky decomposition of Σ, then we can generate $Y = \mu + \Omega X$, where $X \sim N(0, I)$ is from i.i.d. standard normal.

- To solve x from $Ax = B$ in MATLAB, you simply type $x = A\backslash B$
- MATLAB is built on LAPACK (Linear Algebra Package), a library of routines that provides fast, robust algorithms for numerical linear algebra, matrix computations.



Iterative Methods

- Decomposition methods (LU,QR,Cholesky) for linear equations are direct methods of solution.
- These direct methods can be very costly for large systems, since the time requirements are in the order of $O(n^3)$.
- For large sparse matrices, there are specific modifications of direct methods tailored to the matrix structure to improve computation efficiency. All MATLAB, R, and Python have comprehensive tools to deal with the sparse matrix.
- Besides, there are also **iterative methods** that economize on space and often provide good answers in reasonable time. Let's introduce two basic iterative methods. There is also a (preconditioned) conjugate gradients method suitable for solving **sparse symmetric positive definite** matrix (implemented by pcg in MATLAB).

Iterative Methods: Gauss-Jacobi Algorithm

- Consider the equation from the first row of $Ax = b$:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1.$$

- We can solve for x_1 in terms of $(x_2, \cdots, x_n)$ if $a_{11} \neq 0$, yielding

$$x_1 = a_{11}^{-1}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n).$$

- In general, if $a_{ii} \neq 0$, we can use the i^{th} row of A to solve for x_i,

$$x_i = a_{ii}^{-1}\left\{b_i - \sum_{j \neq i} a_{ij}x_j\right\}.$$

- We can apply this expression in an iterative process. First we guess x^0 and use the single equation solution to compute a new guess of x in $(k+1)^{th}$ iteration:

$$x_i^{k+1} = a_{ii}^{-1}\left\{b_i - \sum_{j \neq i} a_{ij}x_j^k\right\}, \quad i = 1, \cdots, n.$$

Iterative Methods: Gauss-Seidel Algorithm

- In Gauss-Jacobi algorithm, we use a new guess for x_i, x_i^{k+1}, only after we have computed the entire vector of new values, x^{k+1}.
- This delay in using the new information is not sensible and therefore the Gauss-Seidel method uses a new approximation of x_i^* as soon as possible.
- In the Gauss-Seidel method, the sequence $\{x^k\}_{k=1}^{\infty}$ is defined by the iteration,

$$x_i^{k+1} = a_{ii}^{-1} \left\{ b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right\}.$$

- Compared to Gauss-Jacobi, the order in which we solve for the successive components of x in Gauss-Seidel matters. So one can try different orderings in Gauss-Seidel if one ordering does not converge.

Linear Programming

- Linear programming (LP) is a special case of constrained optimization where both the objective and constraint functions are linear.
- LP problems can be expressed in a canonical form as

$$\begin{aligned} \text{minimize}_x \quad & c^\top x \\ \text{subject to} \quad & Ax \leq b \\ \text{and} \quad & x \geq 0 \end{aligned}$$

  where x is a vector of variables.
- This canonical form is very general. The problem constraint $Ax \leq b$ can change to equality $Ax = b$ or another inequality $Ax \geq b$.
- The problem can also be expressed in a *matrix* form:

$$\min_x \{c^\top x | Ax \leq b \wedge x \geq 0\}$$

Linear Programming

Example:

- A firm has the following quantities of factors for production: 400 units of labor hours (L), 300 units of capital (K), and 1000 units of land (S). The firm can produce either commodity x (profit=2) or commodity y (profit=1) with the following processes

	Activity A_1 *for* x	*Activity* A_2 *for* y
Labour	$l_x = 4$	$l_y = 1$
Capital	$k_x = 1$	$k_y = 1$
Land	$s_x = 2$	$s_y = 5$

Linear Programming

- Given the above information, the LP problem can be set up as

$$\begin{aligned} \text{minimize}_{x,y} \quad & -(2x+y) \\ \text{subject to} \quad & 4x+1y \leq 400 \\ & 1x+1y \leq 300 \\ & 2x+5y \leq 1000 \\ & x \geq 0,\ y \geq 0 \end{aligned}$$

- MATLAB function to solve LP problem(`https://www.mathworks.com/help/optim/ug/linprog.html`)
- See solution from the practice code.

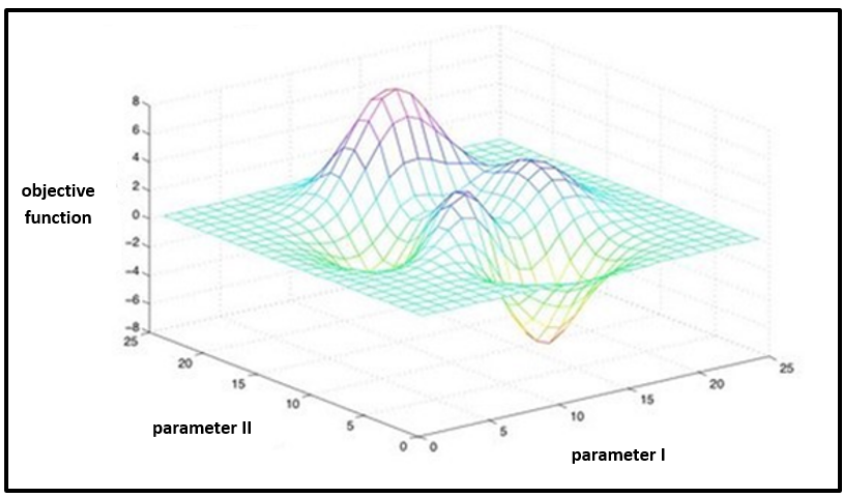
Linear Programming

Simplex Method

- Solutions of simple LP problems can be found by the **graphical method** (`https://www.youtube.com/watch?v=gbL3vYq3cPk`).
  1. identify the coordinates of all corner points of the feasible region.
  2. evaluate the objective function at all of these corner points.
  3. pick the best corner point.
- In general, LP problem should be solved by the **simplex method** (Dantzig, 1990), which is an algebraic procedure for solving LP with any (finite) number of variables and constraints.
- Examples of using the simplex method to solve LP can be found in `https://www.youtube.com/watch?v=rzRZLGD_aeE` and `https://www.youtube.com/watch?v=U41-HLS_oF8`.
- Simplex method is efficient and has polynomial-time (instead of exponential) average complexity in general cases.

Grid Search

- Econometric estimation is often based on an estimator $\hat{\theta}$ that maximizes a stochastic objective function $Q_N(\theta)$ (or minimize $-Q_N(\theta)$), where $\hat{\theta}$ solves the first-order condition (FOC) $\frac{\partial Q_N(\theta)}{\partial \theta} = 0$.
- Instead of using FOC, the grid search method is a brute-force, selecting many values of θ along a grid, compute $Q_N(\theta)$ for each of these values, and choose the estimator $\hat{\theta}$ that provides the largest value of $Q_N(\theta)$.



Grid Search

- Grid search methods may be impractical when the dimension of parameter is high and fine grids are expected. For example, if there are 10 parameters and consider 10 grids for each parameter, there are 10^{10} evaluations.
- Grid search methods are nonetheless useful when using on only a subsect of the parameters.
- Grid search produces the response surface of the objective function, which helps to uncover if there are multiple maxima.
- Grid search is a last resort if none of other numerical methods work.
- Grid search is also used in Machine learning to determine model tuning parameters.

Grid Search

Example of "Grid Search in MATLAB" written by Andrii Pakhomenko (USC):

- problem: $\max_{x \in \mathcal{R}^2} f(x) = x_1 - 0.2x_1^2 + x_2 - 0.3x_2^2$
- construct an equal-space grid:
  $G = \{(x_1, x_2) | x_1 \in \{0, 1, 2, 3, 4, 5\}, x_2 \in \{0, 1, 2, 3, 4, 5\}\}$
- the above grid is too coarse to find the true maximum: the algorithm finds $x = (3, 2)$ and the true maximum happens at $x = (2.5, 1.667)$.
- To get better solutions, one can consider (1) finer grid (at higher time cost) (2) non-equally spaced grid:
  $G = \{h(x_1), h(x_2) | x_1 \in \{0, 1, 2, 3, 4, 5\}, x_2 \in \{0, 1, 2, 3, 4, 5\}\}$, where $h(x_i) = \ln(3x_i + 1)$.
- the new algorithm finds $x = (2.565, 1.946)$, closer to the true maximum.

Grid Search

- A better idea is to use the adaptive grid search: start with a coarse grid, then refine as you approach to the solution.
- start with $G = \{(x_1, x_2) | x_1 \in \{0, 1, 2, 3, 4, 5\}, x_2 \in \{0, 1, 2, 3, 4, 5\}\}$
- obtain the interim solution: $x = (3, 2)$
- refine the grid around the interim solution by halving the length of the search interval:
  $G = \{(x_1, x_2) | x_1 \in \{1.75, 2.25, 2.75, 3.25, 3.75, 4.25\}, x_2 \in \{0.75, 1.25, 1.75, 2.25, 2.75, 3.25\}\}$
- repeat the above steps until convergence (i.e., the desired precision is reached).
- After 9 rounds of iterations, we obtain $x = (2.5005, 1.6666)$.

Gradient Methods

- Gradient methods determine the value $\hat{\theta}$ by iteration.

$$\hat{\theta}_{s+1} = \hat{\theta}_s + \mathbf{A}_s g_s, \quad s = 1, 2, \cdots,$$

  where $\mathbf{A}_s$ is a $k \times k$ matrix that depends on $\hat{\theta}_s$, and $g_s = \frac{\partial Q_N(\hat{\theta}_s)}{\partial \theta}$ is the $k \times 1$ gradient vector evaluated at $\hat{\theta}_s$.
- A leading example of $\mathbf{A}_s$ is $-\mathbf{H}_s^{-1}$, where $\mathbf{H}_s$ is the Hessian matrix evaluated on $\hat{\theta}_s$, which corresponds to Newton-Raphson (or simply Newton) method.
- Two common modifications to gradient methods are to add **step-wise adjustment** and modify $\mathbf{A}_s$ to make it *better* behaved.

$$\hat{\theta}_{s+1} = \hat{\theta}_s + \hat{\alpha}_s \mathbf{A}_s g_s,$$

  where the iteration-specific step-size $\hat{\alpha}_s$ is a scalar chosen to prevent possible overshooting or undershooting.
- We will discuss various choices of $\mathbf{A}_s$ later.

Gradient Methods

- Iterations of a gradient method ideally stop when (i) a small relative change occurs in the objective function $Q_N(\hat{\theta}_s)$; (ii) a small change of the gradient vector g_s occurs relative to the Hessian; (iii) a small relative change occurs in the parameter estimates $\hat{\theta}_s$.
- Gradient methods do not guarantee finding the global maximum; so a range of starting (initial) values should be used.
- As gradient methods use derivatives of the objective function, either numerical derivative or analytical derivatives may be used.

$$\frac{\Delta Q_N(\hat{\theta}_s)}{\Delta \theta_j} = \frac{Q_N(\hat{\theta}_s + h\mathbf{e}_j) - Q_N(\hat{\theta}_s - h\mathbf{e}_j)}{2h}, \quad j = 1, \cdots, k,$$

where h is chosen to be small and $\mathbf{e}_j = (0, \cdots, 0\ 1\ 0, \cdots, 0)'$ is a vector with unity in the j^{th} row and zeros elsewhere.

Gradient Methods

- We present a few commonly used gradient methods (corresponding to different $\mathbf{A}_s$)
  - Method of Steepest Ascent (Descent)
  - Newton-Raphson (or Newton) method
  - Berndt, Hall, Hall, and Hausman (BHHH) method
  - Davidon, Fletcher, and Powell (DFP) method
  - Boyden, Fletcher, Goldfarb, and Shannon (BFGS) method

Method of Steepest Ascent (Descent)

- Given the gradient method, $\hat{\theta}_{s+1} = \hat{\theta}_s + \mathbf{A}_s g_s$, the method of steepest ascent sets $\mathbf{A}_s = -\mathbf{I}_k$, the simplest choice of weighting matrix.
- The method of steepest ascent faces the issue of slow convergence when approaching the local optimum point.
- Thus, the step-size $\hat{\alpha}_s$ is usually used to adjust the convergence rate. The optimal choice of $\hat{\alpha}_s$ can be shown to be $\hat{\alpha}_s = g_s' g_s / g_s' \mathbf{H}_s g_s$, which is somewhat computational intensive.
- Considering that $\mathbf{H}_s$ is needed, one might instead using the Newton method.

Newton-Raphson Method

- The motivation of Newton (Raphson) method is by taking the second-order Taylor series expansion around $\hat{\theta}_s$:

$$Q_N(\theta) = Q_N(\hat{\theta}_s) + \frac{\partial Q_N(\hat{\theta}_s)}{\partial \theta'}(\theta - \hat{\theta}_s) + \frac{1}{2}(\theta - \hat{\theta}_s)' \frac{\partial^2 Q_N(\hat{\theta}_s)}{\partial \theta \partial \theta'}(\theta - \hat{\theta}_s) + R,$$

  where R is the remainder term which can be ignored.

- The Newton method proposes $\hat{\theta}_{s+1} = \hat{\theta}_s - \mathbf{H}_s^{-1} g_s$, where $\mathbf{H}_s = \frac{\partial^2 Q_N(\hat{\theta}_s)}{\partial \theta \partial \theta'}$ is the $k \times k$ Hessian matrix evaluated at $\hat{\theta}_s$.
- The Newton method works particularly well when the objective function is globally concave or globally convex – convergence often occurs within a few iterations.
- One potential problem of the Newton method is that in some cases the Hessian matrix could be singular and thus $\mathbf{H}_s^{-1}$ cannot be computed.

Berndt, Hall, Hall, and Hausman (BHHH) Method

- BHHH Method is a common modification of the Newton method, which replace $-\mathbf{H}_s^{-1}$ by $\mathbf{H}_{BHHH,s}^{-1}$, where $\mathbf{H}_{BHHH,s} = \frac{\partial Q_N(\hat{\theta}_s)}{\partial \theta} \frac{\partial Q_N(\hat{\theta}_s)}{\partial \theta'}$.
- Compared to the Newton method, BHHH method only requires evaluation of first derivatives and thus is less computationally costly.
- Use of the BHHH method is justified by the information matrix equality:

$$\mathrm{E}\left[\frac{\partial^2 Q_N(\theta_0)}{\partial \theta \partial \theta'}\right] = -\mathrm{E}\left[\frac{\partial Q_N(\theta_0)}{\partial \theta} \frac{\partial Q_N(\theta_0)}{\partial \theta'}\right],$$

Davidon, Fletcher, and Powell (DFP) Method

- The DFP algorithm proposes the weighting matrix $\mathbf{A}_s$ which only requires first derivatives and is positive definite.

$$\mathbf{A}_s = \mathbf{A}_{s-1} + \frac{\delta_{s-1}\delta_{s-1}'}{\delta_{s-1}'\gamma_{s-1}} - \frac{\mathbf{A}_{s-1}\gamma_{s-1}\gamma_{s-1}'\mathbf{A}_{s-1}}{\gamma_{s-1}'\mathbf{A}_{s-1}\gamma_{s-1}},$$

  where $\delta_{s-1} = \mathbf{A}_{s-1}g_{s-1}$ and $\gamma_{s-1} = g_s - g_{s-1}$.
- When the initial $\mathbf{A}_0$ is positive definite, e.g., $\mathbf{I}_k$, then $\mathbf{A}_s$ will be positive definite.
- The justification of DFP is that $\mathbf{A}_s$ provides an approximate estimate of the inverse of the Hessian, $\mathbf{H}_s^{-1}$.
- However, it turns out that in the case of larger nonquadratic problems the DFP algorithm has the tendency of sometimes getting stuck. This phenomenon is attributed to $\mathbf{H}_s$ becoming nearly singular.

BFGS Methods

- The BFGS algorithm is a refinement of the DFP algorithm.
- Compared to DFP which applies the Rank One correction, BFGS applies the Rank Two correction.
- The BFGS algorithm proposes the weighting matrix $\mathbf{A}_s$ which only requires first derivatives and is positive definite.

$$\mathbf{A}_s = \mathbf{A}_{s-1} + \frac{\delta_{s-1}\delta'_{s-1}}{\delta'_{s-1}\gamma_{s-1}} + \frac{\mathbf{A}_{s-1}\gamma_{s-1}\gamma'_{s-1}\mathbf{A}_{s-1}}{\gamma'_{s-1}\mathbf{A}_{s-1}\gamma_{s-1}} - (\gamma'_{s-1}\mathbf{A}_{s-1}\gamma_{s-1})\eta_{s-1}\eta'_{s-1},$$

where $\delta_{s-1} = \mathbf{A}_{s-1}g_{s-1}$, $\gamma_{s-1} = g_s - g_{s-1}$, and
$\eta_{s-1} = (\delta_{s-1}/\delta'_{s-1}\gamma_{s-1}) - (\mathbf{A}_{s-1}\gamma_{s-1}/\gamma'_{s-1}\mathbf{A}_{s-1}\gamma_{s-1})$.

Non-gradient Iterative Methods

Nelder-Mead Method

- Other than gradient methods, there are direct search methods to find the minimum or maximum of an objective function.
- For example, the Nelder-Mead (downhill simplex) method is used in Matlab's well-known *fminsearch* commend. ▸ fminsearch
- The Nelder-Mead method uses a geometrical shape called a simplex as its ''vehicle'' of sorts to search the domain. In layman's terms, a simplex is the n-dimensional version of a ''triangle''.
- Nelder-Mead starts off with a randomly-generated simplex, At every iteration, it proceeds to reshape/move this simplex, one vertex at a time, towards an optimal region in the search space. During each step, it basically tries out one or a few modifications to the current simplex, and chooses one that shifts it towards a "better" region of the domain. ▸ illustration1 ▸ illustration2

Non-gradient Iterative Methods

Simulated Annealing Method

- All numerical methods discussed above do not guarantee global optimization. They only search local maximum or local minimum near the starting value.
- Simulated annealing is an algorithm designed to find the global optimal – the name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects.

Non-gradient Iterative Methods

Simulated Annealing Method

- Given $\hat{\theta}_s$ at the s^{th} round of optimization iteration, we perturb the j^{th} component of $\hat{\theta}_s$ to obtain a new trial value of

$$\theta_s^* = \hat{\theta}_s + [0 \cdots 0 \ (\lambda_j r_j) \ 0 \cdots 0]',$$

where λ_j is a prespecified step length and r_j is a draw from a uniform distribution on (-1,1).

- The new trial value is accepted, i.e., $\hat{\theta}_{s+1} = \theta_s^*$, if it increases the objective function.
- In simulated annealing, the new trail value is also accepted if it does not increase the objective function value but passes the Metropolis criterion that

$$\exp((Q_N(\theta_s^*) - Q_N(\hat{\theta}_s))/T_s) > u,$$

where u is a draw from a uniform (0,1) distribution.

Non-gradient Iterative Methods

Simulated Annealing Method

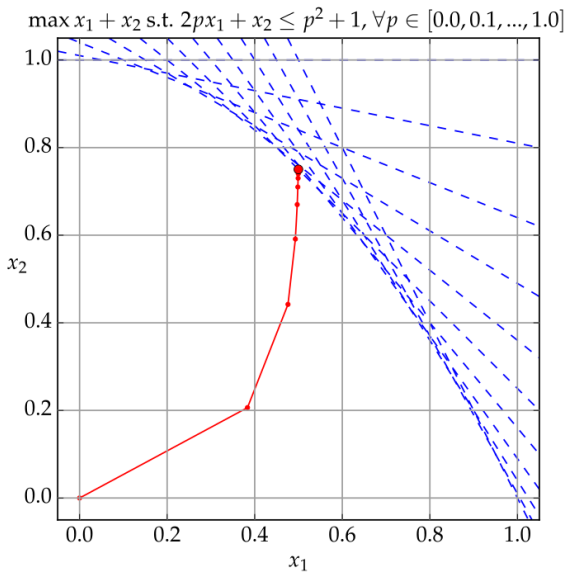
- In the above Metropolis criterion, the function T_s is a scaling parameter called the **temperature**.
- Thus, not only uphill moves are accepted, but downhill moves can also be accepted with a probability that decreases with the difference between $Q_N(\theta_s^*)$ and $Q_N(\hat{\theta}_s)$ and that increases with the temperature.
- The temperature needs to be chosen to reduce during the course of iterations.
- Cooling is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is explored.

  ▸ illustration

- Matlab has the "Global Optimization Toolbox" which includes simulated annealing. ▸ simulannealbnd

Interior Point Method

- An interior point method is a linear or nonlinear programming method for **convex problems** that contain inequalities as constraints.
- It achieves optimization by going through the middle of the solid defined by the problem rather than around its surface (simplex method).



Interior Point Method

- See instructions of interior point method in
  - `https://www.youtube.com/watch?v=oVqpaZB48eM`
  - `https://www.youtube.com/watch?v=zm4mfr-QT1E`
- The MATLAB function *fmincon* solves the nonlinear programming problem using the interior point method (`https://www.mathworks.com/help/optim/ug/fmincon.html`).

Convex Optimization

- General nonlinear optimization problems are difficult to solve (called **NP-Hard**).
- Convex optimization studies the problem of minimizing convex functions over convex sets. Every local minimum is a global minimum.
- With recent advancements in computing and optimization algorithms, convex programming is nearly as straightforward as linear programming, admitting polynomial-time solving algorithms.
- linear programming, quadratic programming, geometric programming, second-order cone programming, semidefinite programming, vector optimization, regularized least-squares can all be recognized as examples of convex problems.

Convex Optimization

- Convex problems can be expressed as

$$\begin{aligned} \text{minimize}_x \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, \cdots, m \\ \text{and} \quad & Ax = b \end{aligned}$$

  where $f_0, f_1, \cdots, f_m$ are convex.
- There is a very good online learning source from Stanford University for convex optimization(`https://web.stanford.edu/class/ee364a/lectures.html`).
- There is a MATLAB compatible package **CVX** (`http://cvxr.com/`) which is easy to use and efficient in handling convex optimization.
- video demonstration of CVX (`https://www.youtube.com/watch?v=N2b_B4TNfUM&feature=youtu.be`)

Convex Optimization

Example: LASSO (least absolute shrinkage and selection operator) problem

$$\min_x \left\{ \frac{1}{N} \| Y - X\beta \|_2^2 + \lambda \|\beta\|_1 \right\},$$

where $Y \in \mathbb{R}^N$, $X \in \mathbb{R}^{N \times K}$, and $\| \cdot \|_p = \left(\sum_{i=1}^N | \cdot |^p \right)^{1/p}$ is the standard ℓ^p norm.

- Here we set $N = 200$ and $K = 90$.
- See the exercise code for demonstration.

Reference books

Some useful reference books are

- Judd, Kenneth (1998): *Numerical Methods in Economics*, the MIT Press
- Miranda, M. and Fackler, P. (2002) *Applied Computational Economics and Finance*. MIT
- Nocedal, Jorge, and Stephen Wright (2006). *Numerical optimization*. Springer Science & Business Media

Dantzig, George B (1990) "Origins of the simplex method," in *A history of scientific computing*, pp. 141–151.