Lecture 4

Finite-Dimensional Unconstrained Optimization

Randall Romero Aguilar, PhD

This draft: September 19, 2018

Universidad de Costa Rica
SP6534 - Economía Computacional

Table of contents

Basic Theory

Applications

Unconstrained optimization problems are ubiquitous in economics:

- Government maximizes social welfare
- Competitive equilibrium maximizes total surplus
- Ordinary least squares estimator minimizes sum of squares
- Maximum likelihood estimator maximizes likelihood function

Definitions

- In the finite-dimensional unconstrained optimization problem, one is given a function $f : \Re^n \mapsto \Re$ and asked to find an x^* such that $f(x^*) \geq f(x)$ for all x.
- We call f the objective function and x^*, if it exists, the global maximum of f.
- We focus on maximization - to solve a minimization problem, simply maximize the negative of the objective.

We say that $x^* \in \Re^n$ is a ...

- strict global maximum of f if $f(x^*) > f(x)$ for all $x \neq x^*$.
- local maximum of f if $f(x^*) \geq f(x)$ for all x in some neighborhood of x^*.
- strict local maximum of f if $f(x^*) > f(x)$ for all $x \neq x^*$ in some neighborhood of x^*.

Optimality Conditions

- Let $f : \Re^n \mapsto \Re$ be twice continuously differentiable.
- First Order Necessary Condition: If x^* is a local maximum of f, then $f'(x^*) = 0$.
- Second Order Necessary Condition: If x^* is a local maximum of f, then $f''(x^*)$ is negative semidefinite.
- We say x is a critical point of f if it satisfies the first-order necessary condition.

- Sufficient Condition: If $f'(x^*) = 0$ and $f''(x^*)$ is negative definite, then x^* is a strict local maximum of f.
- Local-Global Theorem: If f is concave, and x^* is a local maximum of f, then x^* is a global maximum of f.

Example 1:

Maximizing

$$f(x) = x^3 - 12x^2 + 36x + 8$$

- Consider maximizing

$$f(x) = x^3 - 12x^2 + 36x + 8.$$

- The first-order necessary condition

$$f'(x) = 3x^2 - 24x + 36 = 3(x-6)(x-2) = 0$$

- … is satisfied at the critical points $x = 2$ and $x = 6$.

- Since
$$f''(x) = 6x - 24$$
it follows that
$$f''(2) = -12 < 0 \quad \text{and} \quad f''(6) = 12 > 0$$
- Thus,
  - $x = 2$ satisfies the sufficient condition for a strict local maximum, but
  - $x = 6$ fails the second-order necessary condition for a local maximum.

Example 2:

Maximizing

$$f(x) = 3 - x_1^2 - x_2^2 - x_1 x_2 + 2x_1 + x_2$$

- Consider maximizing

$$f(x) = 3 - x_1^2 - x_2^2 - x_1 x_2 + 2x_1 + x_2.$$

- The first-order necessary condition

$$f'(x) = \begin{bmatrix} -2x_1 - x_2 + 2 \\ -x_1 - 2x_2 + 1 \end{bmatrix} = 0$$

- … is satisfied at the critical point $x_1 = 1$ and $x_2 = 0$.

- The Hessian at the critical point

$$f''(x) = \begin{bmatrix} -2 & -1 \\ -1 & -2 \end{bmatrix}$$

has characteristic equation

$$\det \begin{bmatrix} -2-\lambda & -1 \\ -1 & -2-\lambda \end{bmatrix} = \lambda^2 + 4\lambda + 3 = (\lambda+3)(\lambda+1) = 0.$$

- The Hessian has negative eigenvalues, -3 and -1, and thus is negative definite.
- Thus, $x = (1, 0)$ satisfies the sufficient condition for a strict local maximum.

Envelope Theorem

- The Envelope Theorem tells us how the maximum value of a function varies with respect to a parameter.
- Let $f : \Re^{n+1} \mapsto \Re$ be a real-valued continuously differentiable function. If

$$V(\alpha) = \max_{x \in \Re^n} f(x, \alpha)$$

is well-defined and $x(\alpha)$ solves the maximization problem, then

$$V'(\alpha) = \frac{\partial f(x(\alpha), \alpha)}{\partial \alpha}.$$

Example 3:

Envelope theorem

- If $f(x,\alpha) = \alpha x - 0.5x^2$, then

$$V(\alpha) \equiv \max_x f(x,\alpha) = 0.5\alpha^2$$

- Thus

$$V'(\alpha) = \alpha.$$

- For each α, the maximum is $x(\alpha) = \alpha$, so that, by the Envelope Theorem,

$$V'(\alpha) = \frac{\partial f(x(\alpha),\alpha)}{\partial \alpha} = x(\alpha) = \alpha.$$

as expected.

Numerical Algorithms

Newton-Raphson Method

- The Newton-Raphson method maximizes an objective f using successive quadratic approximations.
- Given the k^{th} iterate x_k, the subsequent iterate x_{k+1} is computed by maximizing the quadratic approximation to f about x_k:

$$f(x) \approx f\left(x_k\right)+f'\left(x_k\right)\left(x-x_k\right)+\frac{1}{2}\left(x-x_k\right)' f''\left(x_k\right)\left(x-x_k\right).$$

- Solving the first-order condition

$$f'\left(x_k\right)+f''\left(x_k\right)\left(x-x_k\right)=0,$$

yields the iteration rule

$$x_{k+1}=x_k-\left[f''\left(x_k\right)\right]^{-1} f'\left(x_k\right).$$

- The Newton-Raphson method is identical to using Newton's method to compute the root of the gradient of the objective.
- In theory, it will converge if the initial value is "close" to a critical point of f at which the Hessian is non-singular.
- In practice, it will diverge if the initial value is "far" from a critical point or the Hessian becomes ill-conditioned.
- Moreover, it may converge to a critical point that is not a local maximum, so the second-order necessary condition should always be checked.
- Newton-Raphson can be robust to the starting value if f is globally concave, but sensitive otherwise.

- Newton-Raphson has two drawbacks.
- First, it requires computation of both the first and second derivatives.
- Second, it may not be possible to increase the objective in the direction of the Newton step … this is guaranteed only if $f''(x_k)$ is negative definite.
- For this reason, the Newton-Raphson is rarely used in practice, and then only if the objective is globally concave.

Quasi-Newton Methods

- In analogy with the Newton-Raphson method, quasi-Newton methods update iterates in the direction of the vector

$$d_k = -A_k f'(x_k)$$

  where A_k is an approximation to the inverse Hessian of f at the k^{th} iterate x_k.
- The vector d_k is called the Newton or quasi-Newton step.

- Just as with rootfinding problems, it is not always best to take a full Newton step at each iteration.
- Efficient quasi-Newton methods shorten or lengthen the Newton step to increase gains in the objective.
- This is accomplished by performing a line search in which the Newton step is re-scaled by a factor $s > 0$ that maximizes or nearly maximizes $f\left(x_k + s d_k\right)$.
- Given the computed scaling factor s_k, one updates the iterate as follows:

$$x_{k+1} = x_k + s_k d_k.$$

- In practice, a thorough line search is not necessary.
- Typically, it suffices to assure that the objective increases with each iteration.
- A number of different line search methods are used in practice.
- Line search methods are beyond the scope of the course, but are discussed in most books on applied optimization.
- The CompEcon Toolbox offers four line search methods.

- Quasi-Newton algorithms differ in how the inverse Hessian approximation A_k is constructed and updated.
- Efficient algorithms use negative definite inverse Hessian approximations, guaranteeing the objective can be increased in the direction of the Newton step.
- Efficient quasi-Newton algorithms also employ updating rules that do not require computing second derivatives.
- The CompEcon Toolbox offers three update methods.

- The simplest quasi-Newton method sets $A_k = -I$, where I is the identity matrix, leading to a Newton step that is identical to the gradient of the objective:

$$d_k = f'(x_k).$$

- This is called the method of steepest ascent because the gradient, to a first order, promises the greatest increase in f.
- The steepest ascent method is simple, but numerically less efficient in practice than quasi-Newton methods that employ curvature information.

- The most widely-used quasi-Newton methods employ inverse Hessian update rules that satisfy two conditions.
- First, the inverse Hessian update A_{k+1} is required to satisfy the quasi-Newton condition:

$$x_{k+1} - x_k = A_{k+1}\left(f'(x_{k+1}) - f'(x_k)\right).$$

- Second, the inverse Hessian update is required to be symmetric negative definite to assure the objective can be increased in the direction of the Newton step.
- Two updating methods that satisfy the quasi-Newton and negative definiteness conditions are widely used in practice.

- The Davidson-Fletcher-Powell (DFP) method uses the updating scheme

$$A_{k+1} = A_k + \frac{v_k v_k'}{u_k' v_k} - \frac{A_k u_k u_k' A_k'}{u_k' A_k u_k},$$

where

$$v_k = x_{k+1} - x_k$$

and

$$u_k = f'(x_{k+1}) - f'(x_k).$$

- The Broyden-Fletcher-Goldfarb-Shano (BFGS) method uses the update scheme

$$A_{k+1} = A_k + \frac{1}{v_k' u_k}\left(w_k v_k' + v_k w_k' - \frac{u_k' w_k}{u_k' v_k} v_k v_k'\right),$$

  where

$$w_k = v_k - A_k u_k.$$

- BFGS typically outperforms DFP, although there are problems for which DFP outperforms BFGS.

- Quasi-Newton methods are susceptible to certain problems.
- In both update formulae there is a division by $v_k' u_k$.
- If this value becomes very small in absolute value, numerical instabilities will result.
- Thus, it is best to skip updating A_k or replace it with a scaled negative identity matrix if the value becomes too small.

Numerical Examples

The OP class

- The CompEcon package provides class `OP` (optimization problem) for computing the maximum of function $f : \Re^n \mapsto \Re$.
- A optimization problem is created as follows:

```
from compecon import OP

def f(x):  #objective function
    return ... #function value

problem = OP(f)
x0 = ... #initial guess
x = problem.qnewton(x0) #local maximum of f
```

- Users may use chose different inverse Hessian update and line search methods.

Example 4:

Local maximum of

$$x^3 - 12x^2 + 36x + 8$$

- To maximize

$$f(x) = x^3 - 12x^2 + 36x + 8$$

starting from $x = 4$, execute the script

```
F = OP(lambda x: x**3 - 12*x**2 + 36*x + 8)
x = F.qnewton(x0=4.0)
```

- After 9 iterations, this produces

```
x =  [2.]
```

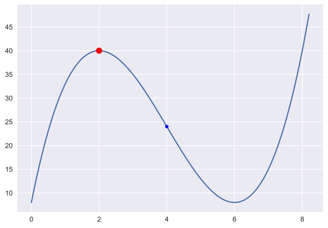

Figure 1: Function $f(x) = x^3 - 12x^2 + 36x + 8$

- To check the first and second derivatives, execute the script

```
J = F.jacobian(x)
H = F.hessian(x)
E = np.linalg.eig(H)[0]
```

- This produces

```
J =  [-0.]
E =  [-12.]
```

- Thus, $x = 2$ is a strict local maximum.

Example 5:

Maximum of

$$g(x, y) = 5 - 4x^2 - 2y^2 - 4xy - 2y$$

- To maximize

$$g(x, y) = 5 - 4x^2 - 2y^2 - 4xy - 2y$$

starting from $x = (0, 0)$, execute the script

```
def g(z):
    x, y = z
    return 5 - 4*x**2 - 2*y**2 - 4*x*y - 2*y

G = OP(g)
x = G.qnewton(x0=[-1, 1])
```

- After 3 iterations, this produces

```
x =  [ 0.5 -1. ]
```

- To check the Jacobian and the eigenvalues of the Hessian, execute the script

```
J = G.jacobian(x)
E = np.linalg.eig(G.hessian(x))[0]
```

- This produces

```
J =  [0. 0.]
E =  [-10.4721  -1.5279]
```

- Thus, $x = (0.5, -1.0)$ is a strict local maximum.

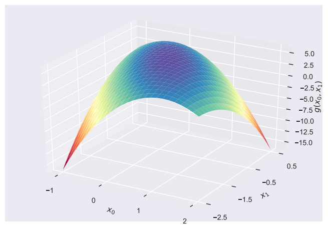

Figure 2: Function $g(x, y) = 5 - 4x^2 - 2y^2 - 4xy - 2y$

Example 6:

Maximize the Rosencrantz function

- To maximize the Rosencrantz or “banana” function

$$f(x,y) = -100(y - x^2)^2 - (1-x)^2$$

starting from $x_0 = (1,0)$, execute the script

```
def f(z):
    x, y = z
    return -100 * (y - x**2)**2 - (1 - x)**2

x0 = [1, 0]
banana = OP(f)
x = banana.qnewton(x0)
```

- After 27 iterations, this produces

```
x = [1. 1.]
```

- To check the Jacobian and the eigenvalues of the Hessian, execute the script

```
J = banana.jacobian(x)
E = np.linalg.eig(banana.hessian(x))[0]
```

- This produces

```
J =  [-0.  0.]
E =  [-1001.6006    -0.3994]
```

- Thus, $x = (1, 1)$ is a strict local maximum.

- To maximize the function using other method, one may override the default update method as follows:

```
banana.qnewton(x0, SearchMeth='steepest')
banana.qnewton(x0, SearchMeth='bfgs')
banana.qnewton(x0, SearchMeth='dfp')
```

- ’steepest’ fails to find the optimum after 250 iterations, the default maximum allowable. The search paths are:

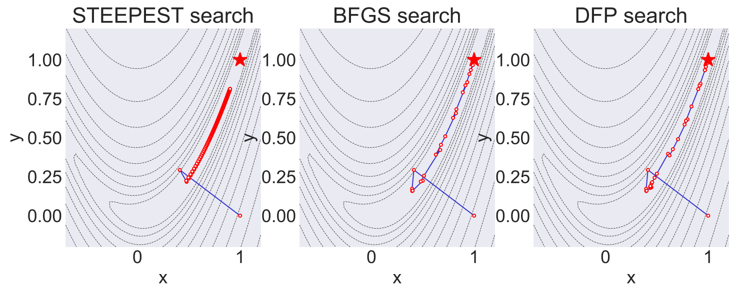

Figure 3: Maximization of Rosencrantz Function

Special Cases

- Two special classes of optimization problems arise often in econometrics and warrant additional discussion.
- Nonlinear least squares and maximum likelihood have special structures that give rise to efficient quasi-Newton methods that use different inverse Hessian approximations.
- Because these problems generally arise in statistical applications, we alter our notation to conform with the conventions for those applications.
- Optimization takes place with respect to a k-dimensional parameter vector θ and n will refer to the number of observations.

- The nonlinear least squares problem takes the form

$$\min_{\theta} \tfrac{1}{2} f(\theta)^\top f(\theta) \quad = \quad \min_{\theta} \sum_{i=i}^{n} \tfrac{1}{2} f_i^2(\theta)$$

  where $f : \Re^k \to \Re^n$.
- This objective has gradient

$$\sum_{i=1}^{n} f_i'(\theta) f_i(\theta) = f'(\theta)^\top f(\theta)$$

  and Hessian

$$f'(\theta)^\top f'(\theta) + \sum_{i=1}^{n} f_i(\theta) \frac{\partial^2 f(\theta)}{\partial \theta \partial \theta^\top}.$$

- Ignoring the second term in the Hessian yields a positive definite matrix with which to determine the search direction:

$$d = -\left[f'(\theta)^\top f'(\theta) \right]^{-1} f'(\theta)^\top f(\theta).$$

Example 7:

Nonlinear least squares estimation

Greene (2012, p.191) considers the following nonlinear consumption function

$$C = \alpha + \beta * Y^{\gamma} + \epsilon$$

which is estimated with quarterly data on real consumption and disposable income for the U.S. economy for 1950 to 2000.

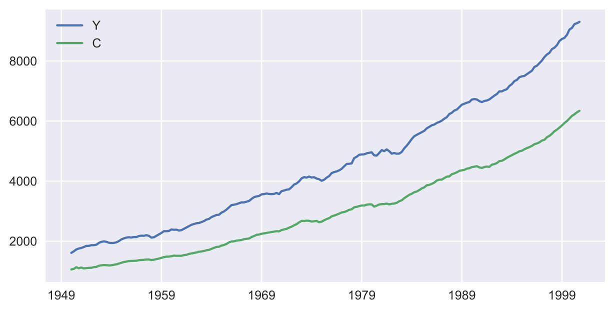

Figure 4: Income and consumption in the U.S.

To get the data

```
import pandas as pd
data = pd.read_table('TableF5-2.txt',sep='\s+')
Y, C = data[['realgdp','realcons']].values.T
```

The objective function is the (negative) of the sum of squared-residuals

```
def ssr(θ):
    α, β, γ = θ
    residuals = C - α - β*Y**γ
    return -(residuals**2).sum()
```

We find the nonlinear least squares estimator, starting from guess $(\alpha, \beta, \gamma) = (0, 0, 1)$

```python
from compecon import OP
θnlls = OP(ssr).qnewton([0.0, 0.0, 1.0])
```

This returns $(\alpha, \beta, \gamma) =$

```
[-91.1965   0.5691   1.0204]
```

This result is not the same found in Greene's textbook, but it can be reproduced with Stata:

```
import delimited TableF5-2.txt, delimiter(space, collapse)
nl (realcons = {alpha=0.0} + {beta=0.0}*realgdp ^{gamma=1.0})
```

- Maximum likelihood problems are specified by a choice of a distribution function f for the data y that depends on a parameter vector θ.
- The log-likelihood function is the sum of the logs of the likelihoods of each of the data points:

$$l(\theta; y) = \sum_{i=1}^{n} \ln f(\theta; y_i).$$

- The score function is defined as the $n \times k$ matrix of derivatives of the log-likelihood function evaluated at each observation:

$$s_i(\theta; y) = \frac{\partial l(\theta; y_i)}{\partial \theta}.$$

- A well-known result in statistical theory is that the expectation of the inner product of the score function is the negative of the expectation of the Hessian of the likelihood function.
- The sample average of the inner product of the score function thus provides a reasonable positive definite approximation of the Hessian that can be used to determine a search direction:

$$d = -\left[s(\theta; y)^\top s(\theta, y)\right]^{-1} s(\theta, y)' 1_n,$$

  where 1_n is an n-vector of ones.
- This approach is known as the modified method of scoring.

Example 8:

Maximum likelihood estimation

Greene (2012, p.590) considers the following binary choice model

$$\mathbb{P}[\text{GRADE} = 1] = F(\beta_0 + \beta_1 \text{GPA} + \beta_2 \text{TUCE} + \beta_3 \text{PSI})$$

where F is cumulative distribution function for either the normal distribution (probit) or the logistic distibution (logit).

To get the data, as well as the cdf for the normal and logistic distributions:

```
from scipy.stats import norm, logistic

data = pd.read_table('TableF14-1.txt',sep='\s+')
data['intercept'] = 1
regressors = ['intercept', 'GPA','TUCE','PSI']

X = data[regressors]
y = data['GRADE']
```

The log-likelihood function for a binary model is given by

$$\ln L = \sum_{i=1}^{n} \left\{ y_i \ln F(x_i'\beta) + (1 - y_i) \ln[1 - F(x_i'\beta)] \right\}$$

which we code as

```python
def binary_model(β,distribution):
    F = distribution.cdf(X @ β)
    return (y*np.log(F) + (1-y)*np.log(1-F)).sum()

def logL_logit(β):
    return binary_model(β,logistic)

def logL_probit(β):
    return binary_model(β,norm)
```

We then estimate the model

```
β0 = np.zeros(4) # initial guess

β_logit = OP(logL_logit).qnewton(β0, SearchMeth='bfgs')
β_probit = OP(logL_probit).qnewton(β_logit/2, SearchMeth='bfgs')

pd.DataFrame({'logit':β_logit,'probit':β_probit},
             index=regressors)
```

which returns

```
              logit  probit
intercept -13.021  -7.452
GPA          2.826   1.626
TUCE         0.095   0.052
PSI          2.379   1.426
```

These results can be reproduced with Stata:

```
infix obs 1-3 gpa 10-14 tuce 19-23 psi 28 grade 37...
             using TableF14-1.txt in 2/33

logit grade gpa tuce psi
probit grade gpa tuce psi
```

References

Greene, William H. (2012). *Econometric Analysis*. 7th ed. Prentice Hall. ISBN: 978-0-13-139538-1.

Miranda, Mario J. and Paul L. Fackler (2002). *Applied Computational Economics and Finance*. MIT Press. ISBN: 0-262-13420-9.