Nonlinear-Equation Solving & Numerical Optimization

Shunsuke Tsuda

ECON 2020 Computing for Economists
Spring 2023

Introduction

- Many economic or econometric problems do not have closed-form solutions.
- Two types of problems to numerically solve:
  (1) Solving a set of equations
  (2) Maximizing or minimizing an objective function

Introduction

- Many economic or econometric problems do not have closed-form solutions.
- Two types of problems to numerically solve:
  - (1) Solving a set of equations
  - (2) Maximizing or minimizing an objective function
- Prioritize (1) and avoid (2) whenever possible, because numerical optimization is costly in terms of:
  - (i) Computational time
  - (ii) Risk of missing the exact solution
- However, there are also many situations where numerical optimization is necessary for solving problems.

Motivating Eg. from Applied Microeconomics

- BLP random coefficient logit demand models
- "BLP": Berry, Levinsohn, and Pakes (1995 ECTA) "Automobile prices in market equilibrium"
- Influential research in industrial organization and widely imported by other fields (trade, urban, education, health, development, etc)
- Illustrate the estimation of nonlinear models, where the objective funciton may not be globally concave or convex
- Illustrate the practicality of a wide range of numerical methods (nonlinear equation solving; numerical optimization; numerical differentiation; numerical integration)
- To study BLP, let's first review the homogeneous logit (white board)
- Recommended: Train "Discrete Choice Methods with Simulation"

BLP RC-logit Demand Model: Setup

- Consumer i's utility from product j in market t:

$$u_{ijt} = x_j\beta_i - \alpha_i p_{jt} + \xi_{jt} + \varepsilon_{ijt}$$

  where

  - p_{jt}: price of product j in market t
  - x_j: row vec. of non-price characteristics of j
  - ξ_{jt}: **product- & market-specific demand shock**
    $\mathbb{E}[\xi_{jt}|p_{jt}, x_j] \neq 0$: **endogeneity** of prices

- **Discrete choice model**: Each consumer chooses one product $j \in 1, \ldots, J$ in the market or an outside option $j = 0$ with $u_{i0t} = \varepsilon_{i0t}$, which maximizes his/her utility

BLP RC-logit Demand Model: Setup

- Consumer i's utility from product j in market t:

$$u_{ijt} = x_j \beta_i - \alpha_i p_{jt} + \xi_{jt} + \varepsilon_{ijt}$$

  where

  - p_{jt}: price of product j in market t
  - x_j: row vec. of non-price characteristics of j
  - ξ_{jt}: **product- & market-specific demand shock**
    $\mathbb{E}[\xi_{jt} | p_{jt}, x_j] \neq 0$: **endogeneity** of prices

- **Discrete choice model**: Each consumer chooses one product $j \in 1, \ldots, J$ in the market or an outside option $j = 0$ with $u_{i0t} = \varepsilon_{i0t}$, which maximizes his/her utility

- Sources of **consumer heterogeneity**:

  - $\varepsilon_{ijt} \sim_{i.i.d}$ Type I extreme value distribution
  - $(\alpha_i, \beta_i) \sim N(\mu, \Sigma)$

BLP RC-logit Demand Model: Intuition

- Consumer heterogeneity in preferences over product characteristics
  $\Rightarrow$ Flexible substitution patterns across products (compared to the simple homogeneous logit)
- Product- & market-specific demand shock unobserved by the econometrician
  $\Rightarrow$ Address endogeneity caused by this, employing IVs
- Estimate consumer demand of differentiated products (e.g., cars, cereals) using only widely available aggregate data at the market level

BLP RC-logit Demand Model: Estimation

Step 0 Guess parameter values

Step 1 Define a function that, given ξ_{jt}, predicts market share:

$$\tilde{S}_{jt} = \int \frac{\exp(x_j \beta_i - \alpha_i p_{jt} + \xi_{jt})}{\sum_k \exp(x_k \beta_k - \alpha_k p_{kt} + \xi_{kt})} dF(\alpha, \beta)$$

by **numerical integration**

Step 2 Solve for $\{\xi_{jt}\}$
s.t. Predicted market share = Observed market share
by **nonlinear equation solving**

Step 3 Construct the GMM objective function using instruments

Step 4 Repeat Step 0. – Step 3. for minimizing the GMM objective function by **numerical optimization** to estimate parameter values (& SE by **numerical differentiation**)

Nonlinear Equations

Motivation

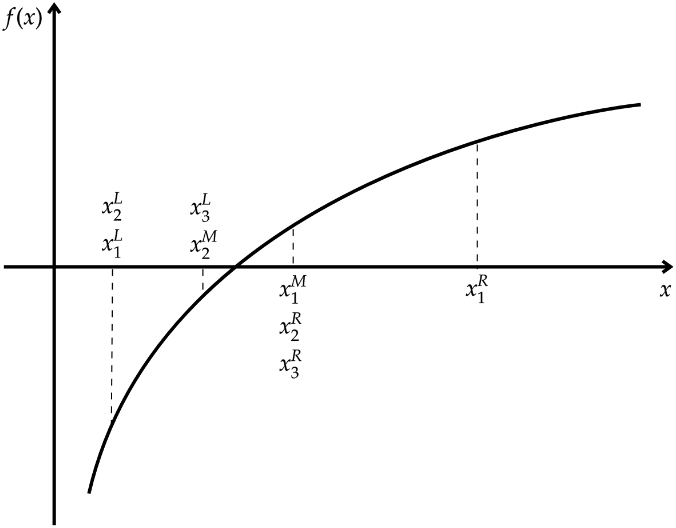
Economic equilibrium is often characterized by systems of nonlinear equations. For example,

- $z(\mathbf{p}) = 0$: System of excess demands with the price vector **p** in general equilibrium
- Set of FOCs as conditions for Nash equilibria of games with continuous strategies
  (e.g., Cournot competition)
- $k^* = f(k^*)$: Finding a fixed point
  (e.g., Steady state of dynamic models; Value function iteration)

Overview

- Bisection method
- Newton's method
- Secant & Broyden's methods
- As a fixed-point problem
- As a minimization problem
- SciPy optimization package (tutorial)
- Further reading: *Judd, Chapter 5*

Bisection Method



Bisection Method: Algorithm

- Based on the Intermediate Value Theorem:
  $f : \mathbb{R} \to \mathbb{R}$ continuous and $f(a) < 0 < f(b)$
  Then, $\exists\, c \in (a, b)$ s.t. $f(c) = 0$

Bisection Method: Algorithm

- Based on the Intermediate Value Theorem:
  $f : \mathbb{R} \to \mathbb{R}$ continuous and $f(a) < 0 < f(b)$
  Then, $\exists\, c \in (a, b)$ s.t. $f(c) = 0$

Step 0 Find x^L & x^R s.t. $f(x^L)f(x^R) < 0$
Choose stopping criterion[1]: ϵ or δ

Step 1 Compute midpoint: $x^M = (x^L + x^R)/2$

Step 2 Update $x^L = x^L$ & $x^R = x^M$ if $f(x^L)f(x^M) < 0$
Update $x^L = x^M$ & $x^R = x^R$ if $f(x^L)f(x^M) > 0$

Step 3 Check stopping criterion:
If $x^R - x^L \leq \epsilon(1 + |x^L| + |x^R|)$ or $|f(x^M)| \leq \delta$,
stop and report the solution at x^M
Otherwise, go back to Step 1 again

Bisection Method: Pros and Cons

Pros:

- The simplest and most robust method

Bisection Method: Pros and Cons

Pros:

- The simplest and most robust method
- Simplicity: only requires continuity of function
- Robustness: as long as $f(x^L)f(x^R) < 0$, it achieves global convergence with any initial guesses (in case there is only a single root)

Bisection Method: Pros and Cons

Pros:

- The simplest and most robust method
- Simplicity: only requires continuity of function
- Robustness: as long as $f(x^L)f(x^R) < 0$, it achieves global convergence with any initial guesses (in case there is only a single root)

Cons:

- Applicable only to 1-dim rootfinding problems
- Slow: require more iterations than other methods, because it ignores information on the function's curvature

Newton's Method

- Take advantage of information on derivatives of a function (Recall: Bisection method used only continuity of a function)

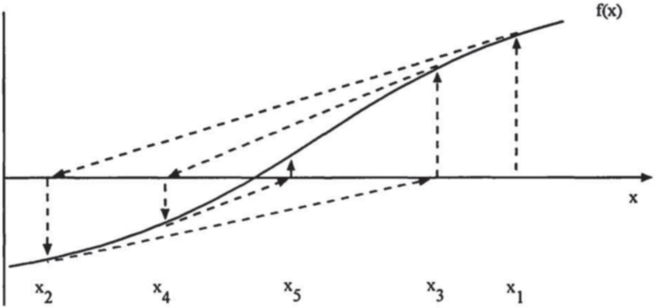
Newton's Method

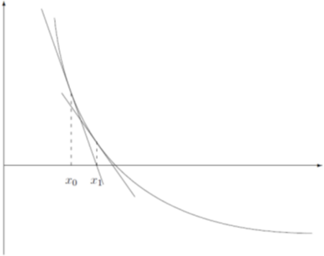
- Take advantage of information on derivatives of a function (Recall: Bisection method used only continuity of a function)
- Reduce a nonlinear problem to a sequence of linear problems, where zeros are easy to compute
- $0 = f(x^*) \approx f(x_k) + (x^* - x_k) f'(x_k) \equiv g(x_k)$
- Instead of solving $f(x) = 0$, solve $g(x) = 0$

Newton's Method

- Take advantage of information on derivatives of a function (Recall: Bisection method used only continuity of a function)
- Reduce a nonlinear problem to a sequence of linear problems, where zeros are easy to compute
- $0 = f(x^*) \approx f(x_k) + (x^* - x_k)f'(x_k) \equiv g(x_k)$
- Instead of solving $f(x) = 0$, solve $g(x) = 0$
- Trade-off: Compared to bisection, faster when it works, but may not always converge

Newton's Method





(*Judd, Chapter 5 & Collard's lecture note*)

Newton's Method: Algorithm

Step 0 Set x_k with $k = 0$
Choose stopping criteria: ϵ & δ

Step 1 Compute $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

Step 2 Check stopping criterion:
If $|x_k - x_{k+1}| \leq \epsilon(1 + |x_{k+1}|)$, go to Step 3.
Otherwise, go back to Step 1.

Step 3 If $|f(x_{k+1})| \leq \delta$, report $x^* = x_{k+1}$ as a solution.
Otherwise, report failure.

Newton's Method: Algorithm

Step 0 Set x_k with $k = 0$
Choose stopping criteria: ϵ & δ

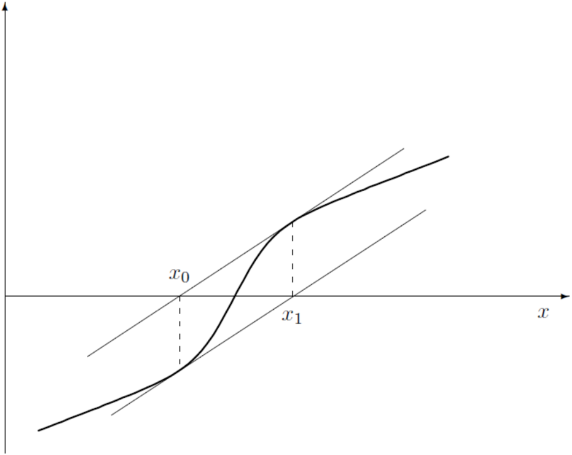
Step 1 Compute $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

Step 2 Check stopping criterion:
If $|x_k - x_{k+1}| \leq \epsilon(1 + |x_{k+1}|)$, go to Step 3.
Otherwise, go back to Step 1.

Step 3 If $|f(x_{k+1})| \leq \delta$, report $x^* = x_{k+1}$ as a solution.
Otherwise, report failure.

- Sufficient condition for convergence: x_0 is sufficiently close to x^*, $f'(x^*) \neq 0$, and $|f''(x^*)/f'(x^*)| < \infty$.
(See Theorem 2.1. in *Judd, Chapter 5*)

Remark. x_0 should be *sufficiently* close to x^*



One practical way: First use bisection to obtain a crude approximation for the root, and then shift to Newton.

Newton's Method (Multidimensional)

- $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$; $J_f()$: Jacobian of $f()$
  $\mathbf{0} = f(\mathbf{x}^*) \approx f(\mathbf{x}_k) + J_f(\mathbf{x}_k)(\mathbf{x}^* - \mathbf{x}_k)$

Newton's Method (Multidimensional)

- $f : \mathbb{R}^n \to \mathbb{R}^n$; $J_f()$: Jacobian of $f()$
  $\mathbf{0} = f(\mathbf{x}^*) \approx f(\mathbf{x}_k) + J_f(\mathbf{x}_k)(\mathbf{x}^* - \mathbf{x}_k)$

Step 0 Set $\mathbf{x}_k$ with $k = 0$
Choose stopping criteria: ϵ & δ

Step 1 Compute $\mathbf{x}_{k+1} = \mathbf{x}_k - J_f(\mathbf{x}_k)^{-1} f(\mathbf{x}_k)$

Step 2 Check stopping criterion:
If $\|\mathbf{x}_k - \mathbf{x}_{k+1}\| \leq \epsilon(1 + \|\mathbf{x}_{k+1}\|)$, go to Step 3.
Otherwise, go back to Step 1.

Step 3 If $\|f(\mathbf{x}_{k+1})\| \leq \delta$, report $\mathbf{x}^* = \mathbf{x}_{k+1}$ as a solution.
Otherwise, report failure.

Remark. Require $\det(J(x^*)) \neq 0$

Secant Method

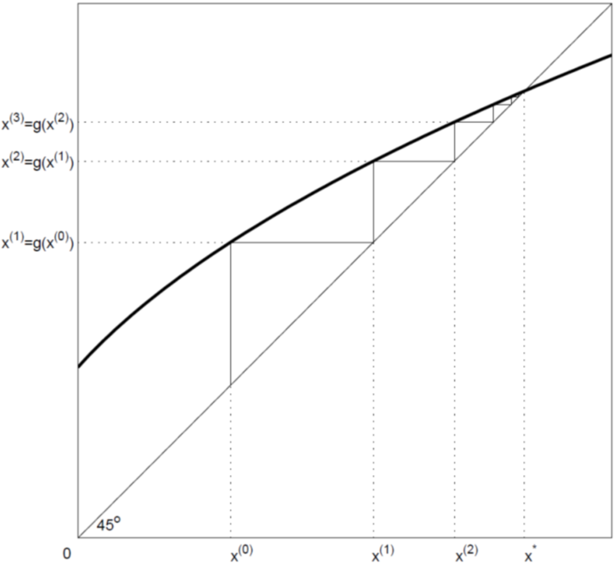
- Obtaining $f'(x_k)$ or Jacobian is costly to compute and code
- Secant method instead uses the simplest approximations of $f'(x_k)$ & $J_f(\mathbf{x}_k)$ by m_k & A_k s.t.
  $f(x_k) - f(x_{k-1}) = m_k(x_k - x_{k-1})$ for one-dimensional

  $f(\mathbf{x}_k) - f(\mathbf{x}_{k-1}) = A_k(\mathbf{x}_k - \mathbf{x}_{k-1})$ for multi-dimensional
- Called Broyden's method in a multidimensional case
- Set (x_0, x_1) at the beginning
- Other processes and properties are similar to Newton's method
- See *Judd, Chapter 5* in detail

As a Fixed-Point Problem

- Fixed-point problems arise frequently in economic problems.
- Any fixed point problems: $x = g(x)$
  can be cast as
  a nonlinear-eq solving: $f(x) \equiv x - g(x) = 0$
- Some (not all) nonlinear-eq solving problems can be cast as a fixed point problem

Fixed-Point Iteration



(*Miranda and Fackler, Chapter 3*)

Fixed-Point Iteration

- Compute a fixed point of $x = g(x)$ $(g : \mathbb{R} \rightarrow \mathbb{R})$
- Construct a sequence $\{x_k\}$ s.t. $x_{k+1} = g(x_k)$
- If it converges $(\lim_{k\rightarrow\infty} x_k = x^*)$, then $x^* = g(x^*)$

Fixed-Point Iteration

- Compute a fixed point of $x = g(x)$ ($g : \mathbb{R} \to \mathbb{R}$)
- Construct a sequence $\{x_k\}$ s.t. $x_{k+1} = g(x_k)$
- If it converges ($\lim_{k\to\infty} x_k = x^*$), then $x^* = g(x^*)$
- *Contraction mapping theorem*: If there exists $T \in [0, 1)$ such that $|g(x) - g(x')| \le T|x - x'|\ \forall x, x' \in D \subset \mathbb{R}$, then x^* is a unique fixed point in D.

Fixed-Point Iteration

- Compute a fixed point of $x = g(x)$ $(g : \mathbb{R} \to \mathbb{R})$
- Construct a sequence $\{x_k\}$ s.t. $x_{k+1} = g(x_k)$
- If it converges $(\lim_{k\to\infty} x_k = x^*)$, then $x^* = g(x^*)$
- *Contraction mapping theorem*: If there exists $T \in [0,1)$ such that $|g(x) - g(x')| \le T|x - x'|\ \forall x, x' \in D \subset \mathbb{R}$, then x^* is a unique fixed point in D.
- Finding such D is hard. In that case, a modified updating scheme with *extrapolation* is preferable to stabilize:

$$x_{k+1} = \lambda_k x_k + (1 - \lambda_k) g(x_k)$$

where $\lambda_k \in [0,1]$ and $\lim_{k\to\infty} \lambda_k = 0$

Fixed-Point Iteration

- Compute a fixed point of $x = g(x)$ $(g : \mathbb{R} \rightarrow \mathbb{R})$
- Construct a sequence $\{x_k\}$ s.t. $x_{k+1} = g(x_k)$
- If it converges $(\lim_{k\rightarrow\infty} x_k = x^*)$, then $x^* = g(x^*)$
- *Contraction mapping theorem*: If there exists $T \in [0, 1)$ such that $|g(x) - g(x')| \leq T|x - x'|\ \forall x, x' \in D \subset \mathbb{R}$, then x^* is a unique fixed point in D.
- Finding such D is hard. In that case, a modified updating scheme with *extrapolation* is preferable to stabilize:

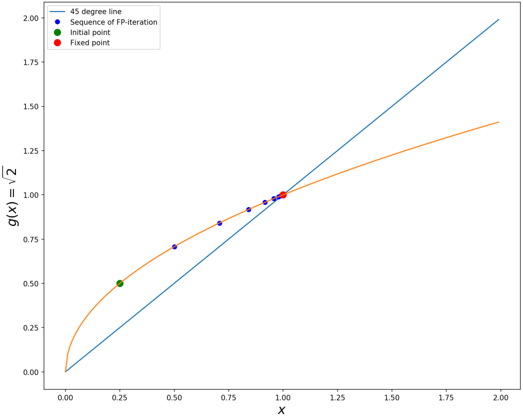
  $$x_{k+1} = \lambda_k x_k + (1 - \lambda_k) g(x_k)$$

  where $\lambda_k \in [0, 1]$ and $\lim_{k\rightarrow\infty} \lambda_k = 0$
- Trade-off between accuracy and speed exists
- On the other hand, if the original system is converging too slowly, $\lambda_k < 0$ could be a way to accelerate convergence

Fixed-Point Interation

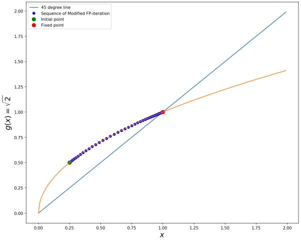
```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import norm
from scipy.optimize import bisect, newton

def fp_iter(g,x0,tol=10e-8,maxiter=100):
 """ Fixed point iteration """
 e = 1  # error
 iter = 0 # number of iteration
 x_seq = [] # store the sequence
 while(e > tol and iter < maxiter):
  ###You will code by yourself in the lab session###
return x,x_seq

def fp_iter_rev(g,x0,lambda_k,tol=10e-8,maxiter=100):
 """ Fixed point iteration with alternative updating scheme"""
  ###You will code by yourself in the lab session###
return x,x_seq

g1 = lambda x : np.sqrt(x)     # E.g.1)
x_init = .25
x1_fp, x1_seq = fp_iter(g1,x_init)

g2 = lambda x : x**2 - 1      # E.g.2)
x_init = -0.5
lambda_init = 0.99
x2_fp,x2_seq = fp_iter(g2,x_init)
x2_fp_rev,x2_seq_rev = fp_iter_rev(g2,x_init,lambda_init)
```

E.g.1) Convergence by the Benchmark Scheme



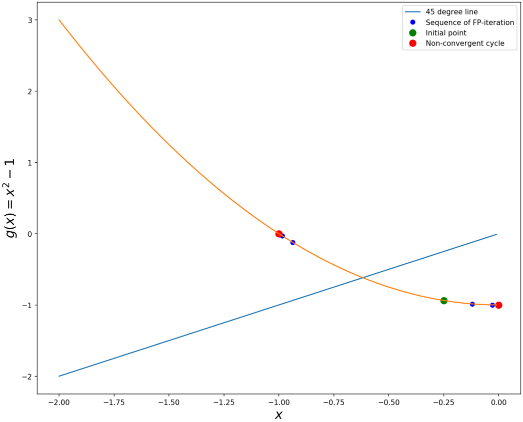

```
In [1]: x1_seq
Out[1]:
array([0.5       , 0.70710678, 0.84089642, 0.91700404, 0.95760328,
       0.97857206, 0.98922801, 0.99459942, 0.99729606, 0.99864711,
       0.99932333, 0.99966161, 0.99983079, 0.99991539, 0.99995769,
       0.99997885, 0.99998942, 0.99999471, 0.99999736, 0.99999868,
       0.99999934, 0.99999967, 0.99999983, 0.99999992])
```

E.g.1) Slower Convergence by the Modified Scheme



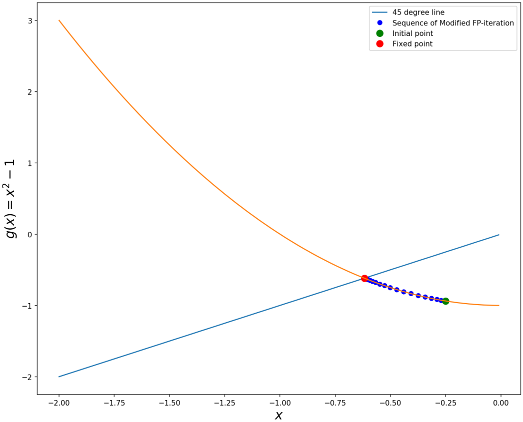

```
In [1]: x1_seq_rev
Out[1]:
array([0.2525    , 0.25747488, 0.26489849, 0.27474099, 0.28696488,
       0.3015203 , 0.31834012, 0.33733537, 0.35839136, 0.38136516,
       0.40608443, 0.43234806, 0.45992833, 0.4885748 , 0.51801943,
       0.54798281, 0.57818103, 0.60833281, 0.63816647, 0.66742641,
       0.69587881, 0.72331621, 0.749561  , 0.77446754, 0.79792312,
       0.81984774, 0.84019279, 0.85893899, 0.87609352, 0.89168681,
       0.90576896, 0.91840611, 0.92967684, 0.93966874, 0.94847527,
       0.95619295, 0.96291894, 0.96874895, 0.97377564, 0.9780873 ,
       0.98176696, 0.98489171, 0.98753235, 0.98975326, 0.99161242,
       0.99316159, 0.99444663, 0.99550781, 0.9963803 , 0.99709453,
       0.99767672, 0.99814927, 0.99853123, 0.9988387 , 0.9990852 ,
       0.99928202, 0.99943855, 0.99956254, 0.99966037, 0.99973726,
       0.99979747, 0.99984442, 0.99988091, 0.99990916, 0.99993095,
       0.99994769, 0.9999605 , 0.99997028, 0.99997771, 0.99998334,
       0.99998759, 0.99999079, 0.99999318, 0.99999497, 0.9999963 ,
       0.99999729, 0.99999802, 0.99999856, 0.99999895, 0.99999924,
       0.99999945, 0.99999961, 0.99999972, 0.9999998 , 0.99999986,
       0.9999999 ])
```

E.g.2) Convergence Failure by the Benchmark Scheme




```
In [1]: x2_seq
Out[2]:
array([-9.37500000e-01, -1.21093750e-01, -9.85336304e-01, -2.91123686e-02,
       -9.99152470e-01, -1.69434170e-03, -9.99997129e-01, -5.74157937e-06,
       -1.00000000e+00, -6.59314825e-11, -1.00000000e+00,  0.00000000e+00,
       -1.00000000e+00,  0.00000000e+00, -1.00000000e+00,  0.00000000e+00,
       -1.00000000e+00,  0.00000000e+00, -1.00000000e+00,  0.00000000e+00,
       -1.00000000e+00,  0.00000000e+00, -1.00000000e+00,  0.00000000e+00,
       -1.00000000e+00,  0.00000000e+00, -1.00000000e+00,  0.00000000e+00,
       -1.00000000e+00,  0.00000000e+00, -1.00000000e+00,  0.00000000e+00,
```

E.g.2) Convergence by the Modified Scheme




```
In [1]: x2_seq_rev
Out[1]:
array([-0.256875  , -0.27035009, -0.2898506 , -0.31452286, -0.34326977,
       -0.37480589, -0.40773481, -0.44064692, -0.47222899, -0.50137054,
       -0.52724889, -0.54937654, -0.5676025 , -0.58206998, -0.5931428 ,
       -0.60131839, -0.60714485, -0.61115463, -0.6138204 , -0.61553283,
       -0.61659578, -0.61723331, -0.61760269, -0.61780934, -0.61792094,
       -0.61797907, -0.61800825, -0.61802237, -0.61802894, -0.61803188,
       -0.61803314, -0.61803366, -0.61803387, -0.61803395, -0.61803397])
```

Fixed-Point Iteration (Multidimensional)

- Fixed point problem in a multidimensional system with $g : \mathbb{R}^n \longrightarrow \mathbb{R}^n$

$$\begin{aligned} x_1 &= g_1(x_1, \ldots, x_n) \\ &\vdots \\ x_n &= g_n(x_1, \ldots, x_n) \end{aligned}$$

- Construct a sequence $\{\mathbf{x}_k\}$ s.t. $\mathbf{x}_{k+1} = g(\mathbf{x}_k)$
- Contraction mapping theorem applies.
  (Check the properties related to Jacobian.)
- The modified updating scheme can also be used similarly to the single-dimensional case

Anderson Acceleration for Fixed-Point Iterations

- Alleviates the potential concern of slow convergence or divergence associated with the standard FP iteration
- This nests the standard FP iteration, i.e., equivalent under some parameter values
- See Walker and Ni (2011) for detail
- Both Python (Scipy) and Julia (NLsolve.jl) contain the package to implement the Anderson Acceleration

Other Elementary Methods for a Multidimensional System

Instead of solving n equations for n unknowns, repeatedly solve each one of n equations with one unknown in turn:

- Gauss-Jacobi Algorithm
- Gauss-Seidel Algorithm

Toward Convergence

- The convergence is not guaranteed in most methods for a multi-dimensional system
- Sensitive to initial guesses, but often difficult to find good ones

Toward Convergence

- The convergence is not guaranteed in most methods for a multi-dimensional system
- Sensitive to initial guesses, but often difficult to find good ones
- Obtain better initial guesses from **optimization** ideas:
  - Optimization problems are less sensitive to initial guesses
  - One simple way is to obtain a rough guess of $f(\mathbf{x}) = 0$ by solving $\min_x \sum_{i=1}^{n} f^i(\mathbf{x})^2$ with a loose stopping rule
  - (We cover optimization algorithms in the next section)

Toward Convergence

- The convergence is not guaranteed in most methods for a multi-dimensional system
- Sensitive to initial guesses, but often difficult to find good ones
- Obtain better initial guesses from **optimization** ideas:
  - Optimization problems are less sensitive to initial guesses
  - One simple way is to obtain a rough guess of $f(\mathbf{x}) = 0$ by solving $\min_x \sum_{i=1}^{n} f^i(\mathbf{x})^2$ with a loose stopping rule
  - (We cover optimization algorithms in the next section)
- **Continuation methods**: Construct a sequence of problems (each of which is reasonably solvable) that ultimately leads to the problem of interest

Lab

Exercise

Aim. Get accustomed to basic one-dimensional methods for solving a non-linear equation

- $f(x) \equiv \exp((x-2)^2) - 2 - x$
  $g(x) \equiv \exp((x-2)^2) - 2$
- Solve $f(x) = 0$ or $x = g(x)$ over the domain $x \in [0, 2]$

Tasks.

1. Solve $f(x) = 0$ by the bisection method with initial $x^L = 0.5$ & $x^R = 1.5$. You can use the SciPy root-finding package.

2.1. Solve $x = g(x)$ by the fixed-point iteration with the updating rule $x_{k+1} = g(x_k)$. Code the algorithm by yourself.
Does this work? Explain why or why not.

2.2. Solve $x = g(x)$ by the fixed-point iteration with the updating rule $x_{k+1} = \lambda_k x_k + (1-\lambda_k) g(x_k)$ where $\lambda_0 = 1$ & $\lambda_k = 0.99\lambda_{k-1}$

- Using `matplotlib.pyplot`, plot the convergent sequence (like what we saw in the lecture).

(Adapted from *Collard, Lecture Notes 5.*)

Assignment 3

Assignment: Cournot Duopoly Model

Aim. Get accustomed to basic multi-dimensional methods for solving a system of non-linear equations. Vectorize the system.

- Quantity competition by 2 firms $i = 1, 2$
- Inverse demand of a good: $P(q) = q^{-1/\alpha}$
- Cost function: $C_i(q_i) = \frac{1}{2} c_i q_i^2$
- Profit for each firm i: $\pi_i(q_1, q_2) = P(q_1 + q_2) q_i - C_i(q_i)$
- Assume $c_1 = 0.6$ & $c_2 = 0.8$

Task 1. Solve for equilibrium q_i^* for $i = 1, 2$ and $P(q_1^* + q_2^*)$ with $\alpha = 1.5$ by

(1) Newton's method: Code the algorithm by yourself. That is, analytically derive the Jacobian of the set of FOCs.
(2) Broyden's method: You can use the SciPy root-finding package.
(3) Fixed-point iteration: Code the algorithm by yourself.

Task 2. Solve the model for all $\alpha \in [1, 3]$ (construct grids) by Broyden's method. Using `matplotlib.pyplot`, produce two plots with x-axis α & y-axes (i) q_1^*, q_2^* (ii) $P(q_1^* + q_2^*)$.

Note for Non-Economics Students

- **Imperfect competition**: There are only two firms in the market. Each firm's quantity decision impacts the market price via consumer demand structure.

- The first order condition of firm i's profit maximization problem (given the other firm's decision):

$$\frac{\partial \pi_i}{\partial q_i} = P(q_1 + q_2) + P'(q_1 + q_2)q_i - C_i'(q_i) = 0$$

- The equilibrium condition is that both the following equalities hold simultaneously:

$$\begin{aligned} f_1(q_1, q_2) &\equiv (q_1 + q_2)^{-\frac{1}{\alpha}} - \frac{1}{\alpha}(q_1 + q_2)^{-\frac{1}{\alpha}-1} q_1 - c_1 q_1 = 0 \\ f_2(q_1, q_2) &\equiv (q_1 + q_2)^{-\frac{1}{\alpha}} - \frac{1}{\alpha}(q_1 + q_2)^{-\frac{1}{\alpha}-1} q_2 - c_2 q_2 = 0 \end{aligned}$$

- Numerically solve for the equilibrium q_1^* & q_2^*.

Optimization

Numerical Optimization: Motivation

- Optimization is ubiquitous in economics and econometrics
- **Economic problems**: Consumer's utility maximization, Firm's profit maximization and cost minimization, Social planner's total surplus maximization, etc
- **Econometric problems**: Minimizing the sum of squared errors, Minimizing the GMM objective function, Maximizing the likelihood function, etc

Numerical Optimization: Remarks

- Numerical optimization is costly in terms of:
  (i) Computation time
  (ii) Risk of missing the exact solution

  Hence, avoid it whenever possible

Numerical Optimization: Remarks

- Numerical optimization is costly in terms of:
  (i) Computation time
  (ii) Risk of missing the exact solution

  Hence, avoid it whenever possible

(i): Check if solving a set of equations suffices to solve the whole problem, which saves comuputation time. E.g.:

- Convex optimization problems in which KKT conditions are sufficient for optimization
- Exactly identified case in GMM

Numerical Optimization: Remarks

- Numerical optimization is costly in terms of:
  (i) Computation time
  (ii) Risk of missing the exact solution

  Hence, avoid it whenever possible

(i): Check if solving a set of equations suffices to solve the whole problem, which saves comuputation time. E.g.:
  - Convex optimization problems in which KKT conditions are sufficient for optimization
  - Exactly identified case in GMM

- Choosing just a faster optimization algorithm is also dangerous due to (ii).
- Important to understand the trade-off between accuracy and speed across different optimization algorithms.

Remarks

- Optimization is costly in terms of
  (i) Computation time
  (ii) Risk of missing the exact solution

  Hence, avoid it whenever possible

(ii): Any optimization algorithm finds *local* optimum, but there is no guarantee that *global* optimum is found (unless the objective function is globally concave or convex).

Obtained solutions are susceptible to:

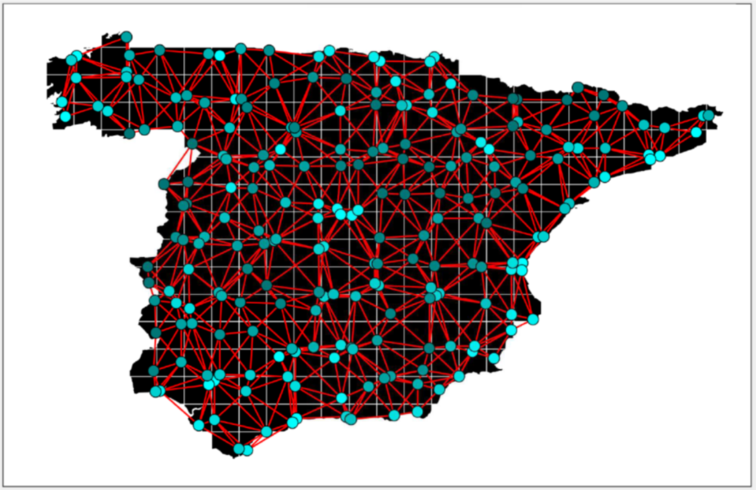
- Search algorithms
- Initial guesses
- Stopping rules

Example of (i): Fajgelbaum & Schaal (2020 ECTA)

- Fajgelbaum & Schaal (2020 ECTA) "Optimal Transport Networks in Spatial Equilibrium"

Q. How large are the gains from expansion and the losses from misallocation of current road networks in Europe?

- An example of an economically simple, but a computationally hard problem
- Also, an example of research in which quantification is of first order importance
- Illustrate the importance of caring about computational burden in a large-scale problem



The problem of optimally designing the road network is determining how much to invest in each link.

Fajgelbaum & Schaal (2020): Environment

Geography:

- $\mathcal{J} = \{1, \dots, J\}$: locations (nodes)
- $\mathcal{N}(j)$: set of connected location of j
  Goods can be directly shipped only through connected locations
- L_j: Number of workers in j, immobile across locations
  L: Total number of workers

Fajgelbaum & Schaal (2020): Environment

Geography:

- $\mathcal{J} = \{1, \dots, J\}$: locations (nodes)
- $\mathcal{N}(j)$: set of connected location of j
  Goods can be directly shipped only through connected locations
- L_j: Number of workers in j, immobile across locations
  L: Total number of workers

Commodities:

- $Y_j^n = z_j^n L_j^n$: tradable good production for sector n
- H_j: the non-tradable good endowment (constant)
  $h_j = H_j / L_j$: per-capita consumption of the non-traded good
- $C_j = \left(\sum_{n=1}^{N} (C_j^n)^{\frac{\sigma-1}{\sigma}} \right)^{\frac{\sigma}{\sigma-1}}$
  $c_j = C_j / L_j$: per-capita consumption of traded goods bundle

Fajgelbaum & Schaal (2020): Environment

Geography:

- $\mathcal{J} = \{1, \dots, J\}$: locations (nodes)
- $\mathcal{N}(j)$: set of connected location of j
  Goods can be directly shipped only through connected locations
- L_j: Number of workers in j, immobile across locations
  L: Total number of workers

Commodities:

- $Y_j^n = z_j^n L_j^n$: tradable good production for sector n
- H_j: the non-tradable good endowment (constant)
  $h_j = H_j / L_j$: per-capita consumption of the non-traded good
- $C_j = \left(\sum_{n=1}^{N} (C_j^n)^{\frac{\sigma-1}{\sigma}} \right)^{\frac{\sigma}{\sigma-1}}$
  $c_j = C_j / L_j$: per-capita consumption of traded goods bundle

Preference:

- $U(c, h)$: worker's utility (homothetic and concave)

Network Building & Transport Technologies

- I_{jk}: Infrastructure (roads) along the link jk
  Def "Transport network" $=$ Distribution of $\{I_{jk}\}_{j\in\mathcal{J},k\in\mathcal{N}(j)}$
- K: total resource for building infrastructure
  Building I_{jk} requires investment $\delta^{I}_{jk}I_{jk}$ units of K

Network Building & Transport Technologies

- I_{jk}: Infrastructure (roads) along the link jk
  <u>Def</u> "Transport network" = Distribution of $\{I_{jk}\}_{j\in\mathcal{J},k\in\mathcal{N}(j)}$
- K: total resource for building infrastructure
  Building I_{jk} requires investment $\delta^I_{jk} I_{jk}$ units of K
- Q^n_{jk}: Quantity of good n shipped from j to $k \in \mathcal{N}(j)$
- Transporting one unit of good n from j to $k \in \mathcal{N}(j)$ requires τ^n_{jk} units of good n , i.e., "Iceberg cost" = $1 + \tau^n_{jk}$, where

$$\tau^n_{jk} = \tau_{jk}(Q^n_{jk}, I_{jk}) = \delta^\tau_{jk} \frac{(Q^n_{jk})^\beta}{(I_{jk})^\gamma}$$

δ^τ_{jk}: geographic frictions
(e.g., distance, elevation, ruggedness, river, etc)
$\beta, \gamma > 0$: decreasing returns to transport (congestion force) & positive returns to infrastructure

Network Building & Transport Technologies

- I_{jk}: Infrastructure (roads) along the link jk
  Def "Transport network" = Distribution of $\{I_{jk}\}_{j\in\mathcal{J},k\in\mathcal{N}(j)}$
- K: total resource for building infrastructure
  Building I_{jk} requires investment $\delta^{I}_{jk}I_{jk}$ units of K
- Q^{n}_{jk}: Quantity of good n shipped from j to $k \in \mathcal{N}(j)$
- Transporting one unit of good n from j to $k \in \mathcal{N}(j)$ requires τ^{n}_{jk} units of good n , i.e., "Iceberg cost" = $1 + \tau^{n}_{jk}$, where

$$\tau^{n}_{jk} = \tau_{jk}(Q^{n}_{jk}, I_{jk}) = \delta^{\tau}_{jk}\frac{(Q^{n}_{jk})^{\beta}}{(I_{jk})^{\gamma}}$$

  δ^{τ}_{jk}: geographic frictions
  (e.g., distance, elevation, ruggedness, river, etc)
  $\beta, \gamma > 0$: decreasing returns to transport (congestion force) & positive returns to infrastructure
- Total transport costs $Q^{n}_{jk}\tau_{jk}(Q^{n}_{jk}, I_{jk})$ are jointly convex over Q^{n}_{jk} and I_{jk} iff $\beta \geq \gamma$

Fajgelbaum and Schaal (2020): Social Planner's Problem

SP's problem consists of three subproblems:

- Optimal Allocation (given infrastructure and goods flow):
- Optimal Transport (given infrastructure)
- Optimal Infrastructure Network Design
  ($=$ the full problem)

$$
\begin{aligned}
W &= \max_{\{c_j, h_j, \{I_{jk}\}_{k\in\mathcal{N}(j)}, C_j^n, L_j^n, \{Q_{jk}^n\}_{k\in\mathcal{N}(j)}\}} \sum_j \omega_j L_j U(c_j, h_j) \\
&= \underbrace{\max_{\{I_{jk}\}_{k\in\mathcal{N}(j)}} \underbrace{\max_{\{Q_{jk}^n\}_{k\in\mathcal{N}(j)}} \underbrace{\max_{\{c_j, h_j, C_j^n, L_j^n\}} \sum_j \omega_j L_j U(c_j, h_j)}_{\text{Optimal allocation subproblem}}}_{\text{Optimal transport subproblem}}}_{\text{Optimal infrastructure network design problem}}
\end{aligned}
$$

s.t.

(i) Availability of tradable & non-tradable goods:

$$c_j L_j \le C_j \;\&\; h_j L_j \le H_j \quad \forall j$$

(ii) Balanced-flows constraint:

$$\underbrace{C_j^n}_{\text{Consumption}} + \underbrace{\sum_{k\in\mathcal{N}(j)} (1 + \tau_{jk}(Q_{jk}^n, I_{jk}))Q_{jk}^n}_{\text{Exports}} \le \underbrace{Y_j^n}_{\text{Production}} + \underbrace{\sum_{i\in\mathcal{N}(j)} Q_{ij}^n}_{\text{Imports}} \quad \forall n, j$$

(iii) Network-building constraint:

$$\sum_j \sum_{k\in\mathcal{N}(j)} \delta_{jk}^I I_{jk} \le K$$

(iv) Local labor market clearing

(v) Non-negativity constraints on consumption, flows, and factor use

Fajgelbaum & Schaal (2020): Social Planner's Problem

SP's problem consists of three subproblems:

- Optimal Allocation (given infrastructure and goods flow)
- Optimal Transport (given infrastructure)
- Optimal Infrastructure Network Design
  (= the full problem)

The full problem is **globally convex** if the transport costs are jointly convex, i.e., if $\beta \geq \gamma$
(review the 1st year math!)

Fajgelbaum & Schaal (2020): Numerical Implementation

Convex cases ($\beta \geq \gamma$):

- KKT conditions are both necessary and sufficient
- Numerically tractable

Fajgelbaum & Schaal (2020): Numerical Implementation

Convex cases ($\beta \geq \gamma$):

- KKT conditions are both necessary and sufficient
- Numerically tractable

Non-Convex cases ($\beta < \gamma$):

- The above approach is not guaranteed to find the global optimum
- Optimal transport and allocation subproblems are convex if $Q\tau_{jk}(Q, I_{jk})$ is convex in Q, i.e., if $\beta \geq 0$
- Iterative procedure over the infrastructure investments:
  - Guess on the network investment I_{jk}
  - Solve for the optimum over $\{c_j, C_j^n, h_j, L_j^n, Q_{jk}^n\}$
  - Obtain a new guess over I_{jk}
  - Repeat until convergence...

Example of (ii): BLP Again

- Consumer i's utility from product j in market t:

$$u_{ijt} = x_j\beta_i - \alpha_i p_{jt} + \xi_{jt} + \epsilon_{ijt}$$

  where

  - p_{jt}: price of product j in market t
  - x_j: row vec. of non-price characteristics of j
  - ξ_{jt}: **product- & market-specific demand shock** $\mathbb{E}(\xi_{jt}|p_{jt}, x_j) \neq 0$: **endogeneity** of prices

- **Discrete choice model**: Each consumer chooses one product $j \in 1, \ldots, J$ in the market or an outside option $j = 0$ with $u_{i0t} = \epsilon_{i0t}$, which maximizes his/her utilit

- Sources of **consumer heterogeneity**:

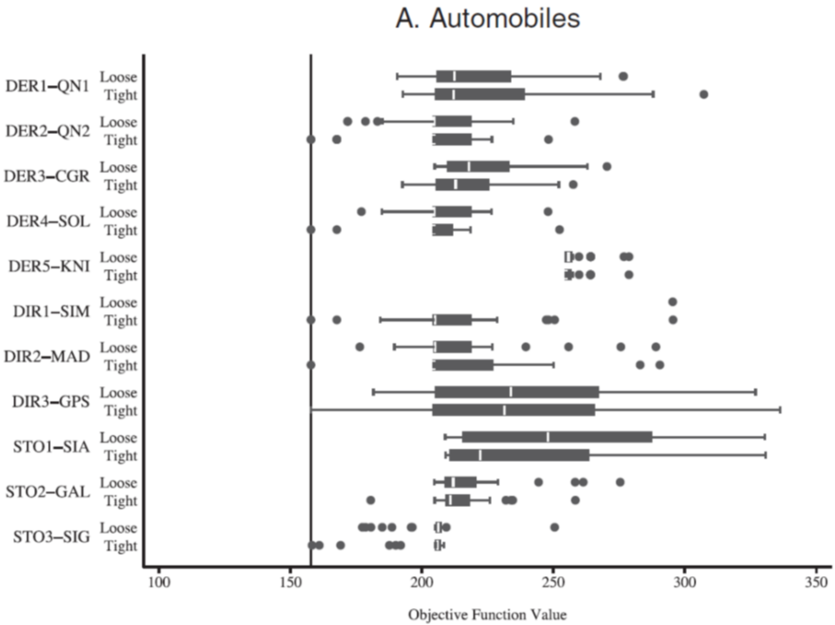
  - $\epsilon_{ijt} \sim_{i.i.d}$ Type I extreme value distribution
  - $(\alpha_i, \beta_i) \sim N(\mu, \Sigma)$

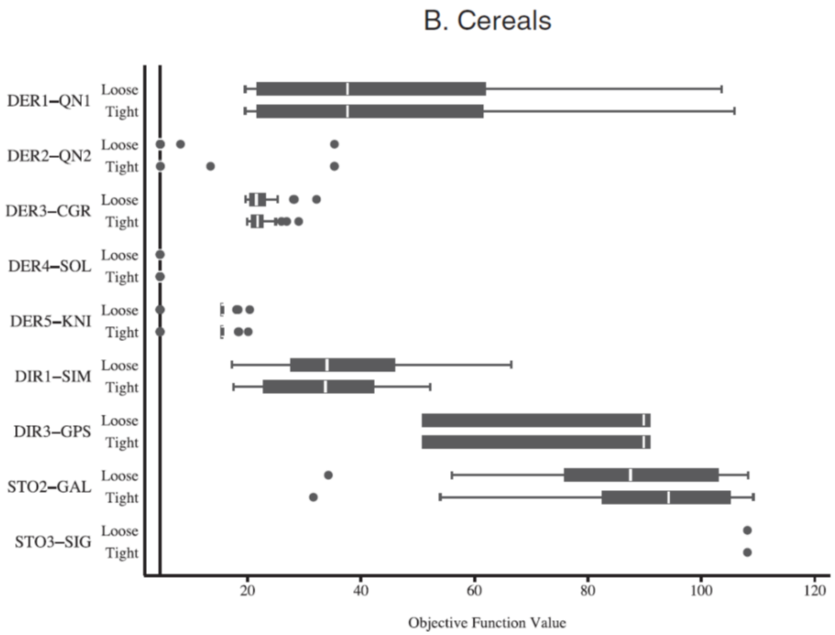
Example of (ii): BLP Again

- Here I follow Knittel and Metaxoglou (2014 REStat) which emphasize numerical challenges of BLP RC-logit demand models
- Illustrates with the BLP RC-logit demand model that different combinations of search algorithms, initial guesses, and stopping rules lead to convergences at different optima
- Observed convergences at:
  - Points where 1st- and 2nd- order conditions fail!
  - Local optima
- Also, observed convergence failure in some instances

TABLE 2.—OPTIMIZATION ALGORITHMS

Class	Description	Source	Acronym
Derivative-based	Quasi-Newton 1	MathWorks	DER1-QN1
	Quasi-Newton 2	Publicly available	DER2-QN2
	Conjugate gradient	Publicly available	DER3-CGR
	SOLVOPT	Publicly available	DER4-SOL
	KNITRO	Ziena Optimization	DER5-KNI
Deterministic direct search	Simplex	MathWorks	DIR1-SIM
	Mesh adaptive direct search	MathWorks	DIR2-MAD
	Generalized pattern search	MathWorks	DIR3-GPS
Stochastic direct search	Simulated annealing	Publicly Available	STO1-SIA
	Genetic algorithm GADS	MathWorks	STO2-GAL
	Simulated annealing GADS	MathWorks	STO3-SIG


A. Automobiles
DER1–QN1
DER2–QN2
DER3–CGR
DER4–SOL
DER5–KNI
DIR1–SIM
DIR2–MAD
DIR3–GPS
STO1–SIA
STO2–GAL
STO3–SIG
Loose
Tight
100
150
200
250
300
350
Objective Function Value


B. Cereals
DER1–QN1
DER2–QN2
DER3–CGR
DER4–SOL
DER5–KNI
DIR1–SIM
DIR3–GPS
STO2–GAL
STO3–SIG
Loose
Tight
20
40
60
80
100
120
Objective Function Value

Summary

- Optimization is costly in terms of
  (i) Computation time
  (ii) Risk of missing the exact solution

  Hence, avoid it whenever possible

(i): Check if solving a set of equations suffices to solve the whole problem, which saves comuputation time.

(ii): Any optimization algorithm finds *local* optimum, but there is no guarantee that *global* optimum is found (unless the objective function is globally concave or convex).

- Understand the trade-off between accuracy and speed across different optimization algorithms.

3 Types of Methods

- Derivative-based methods (1st order):
  - Require differentiability
  - Uses information about gradients
- Derivative-based methods (2nd order):
  - Uses information about gradients and curvature
  - Converges more rapidly to the solution
  - High cost of computing and storing Hessians
- Derivative-free methods (0th order):
  - Slow
  - Suitable for problems with kinks and discontinuities

3 Types of Methods

- Derivative-based methods (1st order):
  - Require differentiability
  - Uses information about gradients
- Derivative-based methods (2nd order):
  - Uses information about gradients and curvature
  - Converges more rapidly to the solution
  - High cost of computing and storing Hessians
- Derivative-free methods (0th order):
  - Slow
  - Suitable for problems with kinks and discontinuities
- Trade-off: Higher order methods are speedier, while lower order methods give more accurate solutions

Several Common Methods

- Derivative-based methods
  - Bisection method (1st order)
  - Newton's method (2nd order)
  - Quasi-Newton method (2nd order)
- Derivative-free methods
  - Grid search method
  - Bracket method
  - Golden section search method
  - Nelder-Mead method
- Constrained optimization
  - Penalty function method
  - Sequential least squares programming (Derivative-based)
  - By linear approximation (Derivative-free)
- SciPy optimization package (tutorial)

Derivative-Based Methods

Bisection Method (1st order)

- Similar logic to the bisection in non-linear equation solving, based on the Intermediate Value Theorem

Step 0 Find x^L & x^R s.t. $f'(x^L)f'(x^R) < 0$
Choose stopping criterion[2]: ϵ or δ

Step 1 Compute midpoint: $x^M = (x^L + x^R)/2$

Step 2 Update $x^L = x^L$ & $x^R = x^M$ if $f'(x^L)f'(x^M) < 0$
Update $x^L = x^M$ & $x^R = x^R$ if $f'(x^L)f'(x^M) > 0$

Step 3 Check stopping criterion:
If $x^R - x^L \le \epsilon(1 + |x^L| + |x^R|)$ or $|f'(x^M)| \le \delta$,
stop and report the solution at x^M
Otherwise, go back to Step 1 again

Newton's Method (2nd order)

- Similar logic to Newton's method in non-linear equation solving
- $f(x) \approx f(x_k) + f'(x_k)(x - x_k) + \frac{f''(x_k)}{2}(x - x_k)^2$
- FOC: $0 = f'(x^*) \approx f'(x_k) + (x^* - x_k)f''(x_k)$

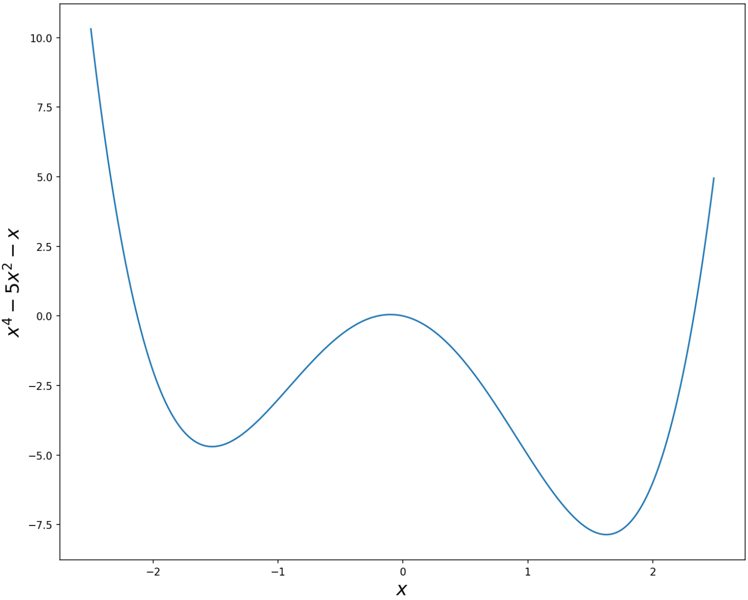
Step 0 Set x_k with $k = 0$
Choose stopping criteria: ϵ & δ

Step 1 Compute $x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$

Step 2 Check stopping criterion:
If $|x_k - x_{k+1}| \leq \epsilon(1 + |x_{k+1}|)$, go to Step 3.
Otherwise, go back to Step 1.

Step 3 If $|f'(x_{k+1})| \leq \delta$, report $x^* = x_{k+1}$ as a solution.
Otherwise, report failure.

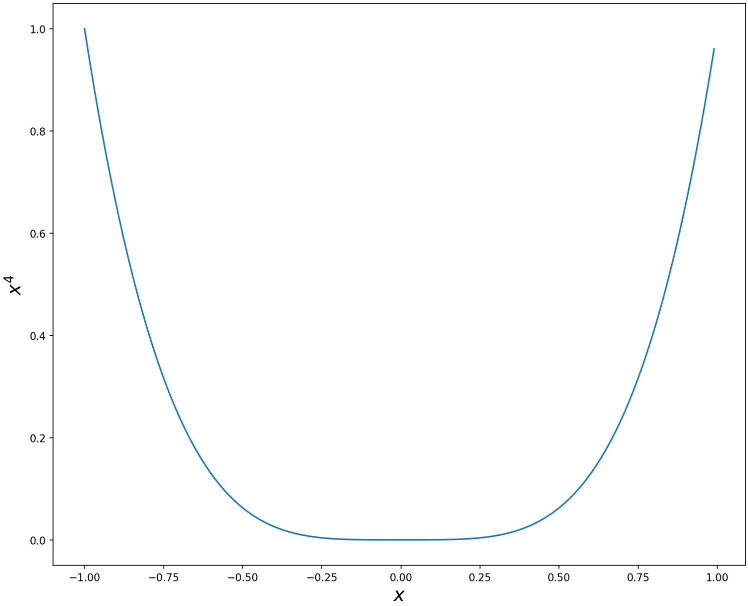
Eg.1) Local & Global Optimum



Eg.1) Initial Guess Matters

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy.optimize import minimize

obj_1 = lambda x: x**4-5*(x**2)-x
argmin_1_1 = minimize(obj_1, x0=-0.5, method='BFGS')   # a quasi-newton's method
argmin_1_2 = minimize(obj_1, x0=0.5, method='BFGS')

In [12]: argmin_1_1
Out[12]:
      fun: -4.694706337665813
 hess_inv: array([[ 0.05546626]])
      jac: array([ -5.96046448e-08])
  message: 'Optimization terminated successfully.'
     nfev: 18
      nit: 4
     njev: 6
   status: 0
  success: True
        x: array([-1.52854364])

In [13]: argmin_1_2
Out[13]:
      fun: -7.855394472077334
 hess_inv: array([[ 0.0457116]])
      jac: array([ -8.34465027e-07])
  message: 'Optimization terminated successfully.'
     nfev: 21
      nit: 5
     njev: 7
   status: 0
  success: True
        x: array([ 1.62894846])
```

Eg.2) Too Small Changes in Gradients



Eg.2) Scaling Matters

```
import matplotlib.pyplot as plt
import numpy as np
import scipy
from scipy.optimize import minimize

obj_2 = lambda x: x**4
obj_2_transform = lambda x: 10000*x**4

In [12]: minimize(obj_2, 10, method='BFGS')
Out[12]:
      fun: 1.1274113375014166e-08
 hess_inv: array([[ 328.87754176]])
      jac: array([  4.37645767e-06])
  message: 'Optimization terminated successfully.'
     nfev: 78
      nit: 25
     njev: 26
   status: 0
  success: True
        x: array([ 0.01030435])

In [13]: minimize(obj_2_transform, 10, method='BFGS')
Out[13]:
      fun: 1.4693518450033662e-09
 hess_inv: array([[ 9.10967096]])
      jac: array([  9.49336157e-06])
  message: 'Optimization terminated successfully.'
     nfev: 108
      nit: 35
     njev: 36
   status: 0
  success: True
        x: array([ 0.00061913])
```

Newton's Method (Multidimensional)

- $f : \mathbb{R}^n \longrightarrow \mathbb{R}$; $H_f()$: Hessian of $f()$
  $\mathbf{0} = \nabla f(\mathbf{x}^*) \approx \nabla f(\mathbf{x}_k) + H_f(\mathbf{x}_k)(\mathbf{x}^* - \mathbf{x}_k)$

Step 0 Set $\mathbf{x}_k$ with $k = 0$
Choose stopping criteria: ϵ & δ

Step 1 Compute $\mathbf{x}_{k+1} = \mathbf{x}_k - H_f(\mathbf{x}_k)^{-1}\nabla f(\mathbf{x}_k)$

Step 2 Check stopping criterion:
If $\|\mathbf{x}_k - \mathbf{x}_{k+1}\| \leq \epsilon(1 + \|\mathbf{x}_{k+1}\|)$, go to Step 3.
Otherwise, go back to Step 1.

Step 3 If $\|\nabla f(\mathbf{x}_k)\| \leq \delta(1 + |f(x_k)|)$, report $\mathbf{x}^* = \mathbf{x}_{k+1}$ as an optimum.
Otherwise, report failure.

- Converge quadratically to a local optimum.

Newton's Methods (Multidimensional): Caveats

- Calculating Hessian and its inverse is costly

Newton's Methods (Multidimensional): Caveats

- Calculating Hessian and its inverse is costly
- Also, Hessian must be well-conditioned:
  - Invertible
  - Positive semi-definiteness (in a minimization problem) around the solution, so that the objective function value is approached toward the solution in each Newton step

Newton's Methods (Multidimensional): Caveats

- Calculating Hessian and its inverse is costly
- Also, Hessian must be well-conditioned:
  - Invertible
  - Positive semi-definiteness (in a minimization problem) around the solution, so that the objective function value is approached toward the solution in each Newton step
- However, no guarantee that the above conditions hold, especially at points far from the solution

Practical Methods for Multidimensional Cases

- **Quasi-Newton methods:** approximate a Hessian (or its inverse) by a positive definite H_k, guaranteeing that function value can be decreased in the direction of the Newton step (in a minimization problem), s.t.

$$H_k^{-1}(\nabla f(\mathbf{x}_{k+1})' - \nabla f(\mathbf{x}_k)') = \mathbf{x}_{k+1} - \mathbf{x}_k$$

  Check several quasi-Newton methods with different updating rules of $\{H_k\}$ by yourself:

  - Broyden-Fletcher-Goldfarb-Shanno (BFGS) method
  - Davidson-Flecther-Powerl (DFP) method

- **Conjugate Gradient Method (CGM):** Store only a gradient, while implicitly keeping track of curvature information in a useful way without storing a Hessian

- See *Judd, Chapter 4* in detail

Derivative-Free Methods

Grid Search Method

- Simplest and most primitive

Grid Search Method

- Simplest and most primitive
- Credible global solution (if parameter space is sufficiently wide and grids are not too coarse)
- Slow (especially for a high demensional case)

Grid Search Method

- Simplest and most primitive
- Credible global solution (if parameter space is sufficiently wide and grids are not too coarse)
- Slow (especially for a high demensional case)
- Useful for understanding the shape of objective function:
    - Unless you are aware of the functional form clearly (which rarely happens), for whatever method you will finally adopt, begin by plotting with this method
    - That helps you to select which type of more sophisticated method to adopt if necessary

Nelder-Mead (Downhill Simplex) Method

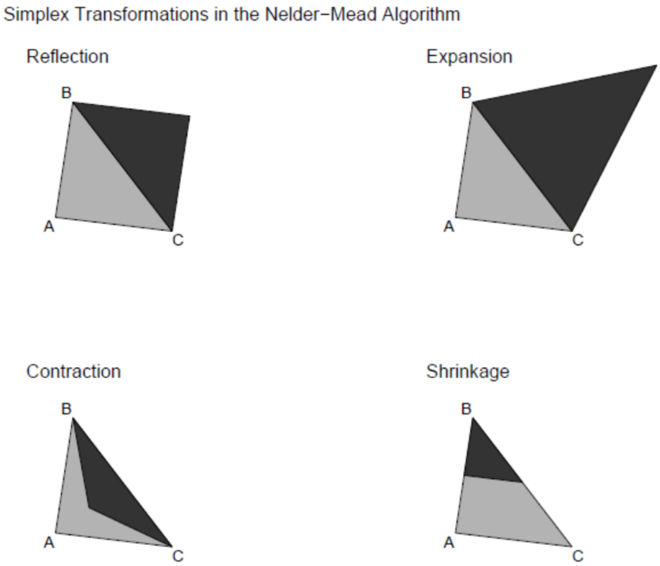
- A widely-used derivative-free optimization method for multi-dimensional functions
- $f : \mathbb{R}^n \to \mathbb{R}$
- Begin by evaluating the objective function at $n + 1$ points
- These points form a *simplex* in $\mathbb{R}$
- Directly search for the optimum by moving this simplex with several steps
  (reflection; expansion; contraction; shirinkage)

Nelder-Mead (Downhill Simplex) Method

- A widely-used derivative-free optimization method for multi-dimensional functions
- $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- Begin by evaluating the objective function at $n + 1$ points
- These points form a *simplex* in $\mathbb{R}$
- Directly search for the optimum by moving this simplex with several steps
  (reflection; expansion; contraction; shirinkage)

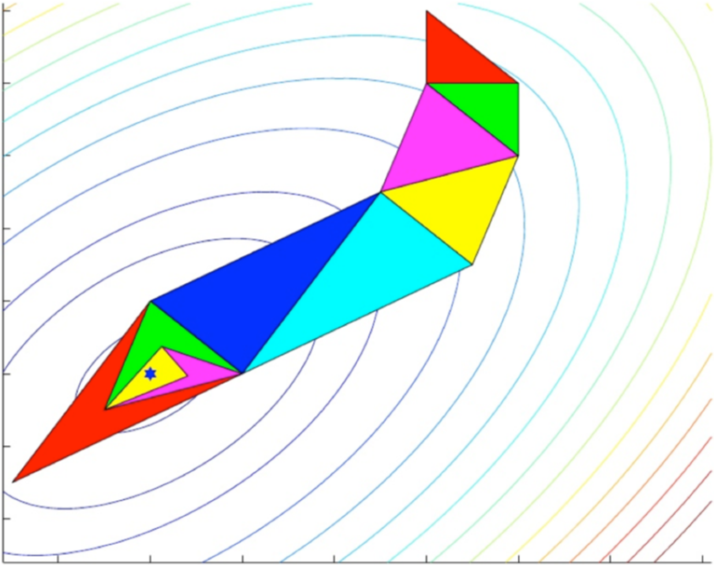
Remark. Speedier than the grid search method (and much more accurate than derivative-based methods), but not still perfect for converging to a global solution. Try with several initial guesses.

Illustration with $n = 2$



(*Miranda and Fackler, Chapter 4*)

Illustration with $n = 2$



(*Lecture note by Fernández-Villaverde and Guerrón*)

Derivative-Based vs. Derivative-Free Methods

- Plot the objective function with some coarse grids to get a sense.
- If you are sure that the function is globally concave/convex, use a derivative-based method.
  (Check if different initial guesses actually globally converge.)
- Otherwise, go for a derivative-free method like Nelder-Mead.
  (Again, check with different initial guesses.)
- Whatever method you finally adopt, check the robustness with the grid search method in some reasonable range of parameters.

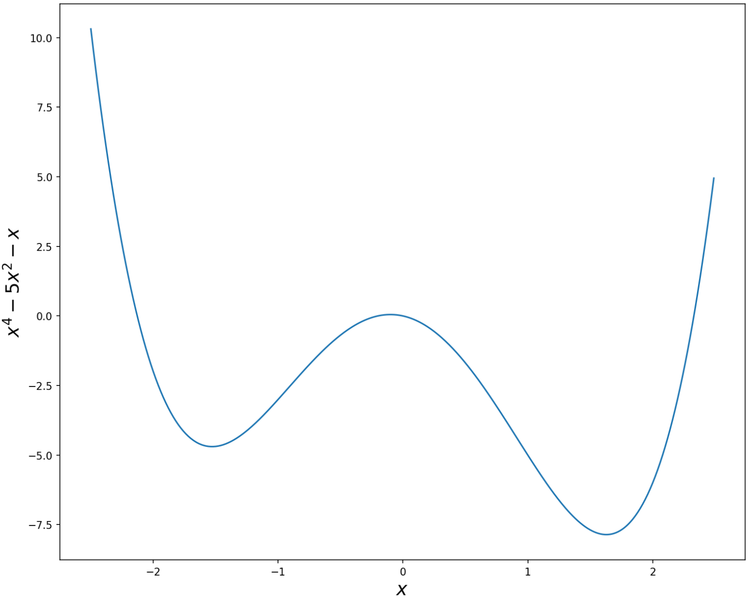
Derivative-Based vs. Derivative-Free Methods

- Plot the objective function with some coarse grids to get a sense.
- If you are sure that the function is globally concave/convex, use a derivative-based method.
  (Check if different initial guesses actually globally converge.)
- Otherwise, go for a derivative-free method like Nelder-Mead.
  (Again, check with different initial guesses.)
- Whatever method you finally adopt, check the robustness with the grid search method in some reasonable range of parameters.
- Trade-off: In general, higher order methods (derivative-based) are speedier, while lower order methods (derivative-free) methods tend to give more accurate solutions

Derivative-Based vs. Derivative-Free Methods

- Plot the objective function with some coarse grids to get a sense.
- If you are sure that the function is globally concave/convex, use a derivative-based method.
  (Check if different initial guesses actually globally converge.)
- Otherwise, go for a derivative-free method like Nelder-Mead.
  (Again, check with different initial guesses.)
- Whatever method you finally adopt, check the robustness with the grid search method in some reasonable range of parameters.
- Trade-off: In general, higher order methods (derivative-based) are speedier, while lower order methods (derivative-free) methods tend to give more accurate solutions
- Accuracy of derivative-free methods is not still perfect. (Eg. 1)

Eg.1) (Again) Local and global optimum



Eg.1) Newton's vs. Nelder-Mead

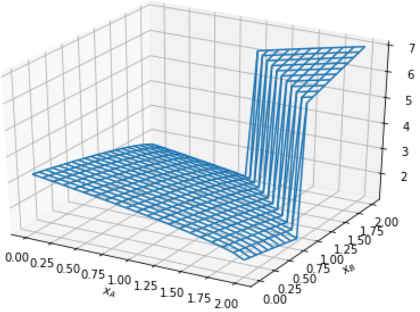
```
In [12]: minimize(obj_1, -0.1, method='BFGS')
Out[12]:
      fun: -7.855394472077359
 hess_inv: array([[ 419.3338208]])
      jac: array([  1.19209290e-07])
  message: 'Optimization terminated successfully.'
     nfev: 36
      nit: 2
     njev: 12
   status: 0
  success: True
        x: array([ 1.6289485])

In [13]: minimize(obj_1, -0.1, method='Nelder-Mead')
Out[13]:
 final_simplex: (array([[-1.52851563],
       [-1.52859375]]), array([-4.69470633, -4.69470632]))
           fun: -4.6947063305936503
       message: 'Optimization terminated successfully.'
          nfev: 42
           nit: 21
        status: 0
       success: True
             x: array([-1.52851563])
```

Derivative-Based vs. Derivative-Free Methods

- Plot the objective function with some coarse grids to get a sense.
- If you are sure that the function is globally concave/convex, use a derivative-based method.
  (Check if different initial guesses actually globally converge.)
- Otherwise, go for a derivative-free medhod like Nelder-Mead.
  (Again, check with different initial guesses.)
- Whatever method you finally adopt, check the robustness with the grid search method in some reasonable range of parameters.
- Trade-off: In general, higher order methods (derivative-based) are speedier, while lower order methods (derivative-free) methods tend to give more accurate solutions
- Accuracy of derivative-free methods is NOT still perfect. (Eg. 1)
- Derivative-free methods are superior especially when there are discontinuities in the objective function. In such cases, derivative-based methods work very poorly. (Eg. 3)

Eg.3) Discontinuities in the Objective Function

- 2-player public investment game
- Utility:
  $u_i(x_A, x_B) = \left(2 - x_i + \mathbb{1}_{x_A + x_B \geq 3} \frac{3(x_A + x_B)}{2}\right)^\sigma$, $i = A, B$
- Welfare: $W(x_A, x_B) \equiv u_A(x_A, x_B) + u_B(x_A, x_B)$
- Welfare-maximizer: $(x_A^*, x_B^*) = (2, 2)$



Eg.3) Discontinuities in the Objective Function

- Utility:
  $$u_i(x_A, x_B) = \left(2 - x_i + \mathbb{1}_{x_A+x_B\geq 3}\frac{3(x_A+x_B)}{2}\right)^{\sigma},\ i = A, B$$
- Welfare: $W(x_A, x_B) \equiv u_A(x_A, x_B) + u_B(x_A, x_B)$

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def Inv_W(x):
    x_a = x[0]
    x_b = x[1]
    sigma = 0.7
    inv_return = np.where(x_a+x_b>=3, 3*(x_a+x_b), 0)
    return -(((2-x_a) + inv_return/2)**sigma + ((2-x_b) + inv_return/2)**sigma)

x_a = np.arange(0, 2.0+0.1, 0.1)
x_b = np.arange(0, 2.0+0.1, 0.1)
X_a, X_b = np.meshgrid(x_a, x_b)
PG_welfare = -Inv_W(np.array([X_a, X_b]))

ax = Axes3D(plt.figure())
ax.set_xlabel("$x_A$")
ax.set_ylabel("$x_B$")
ax.set_zlabel("$W(x_A, x_B)$")
ax.plot_wireframe(X_a, X_b, PG_welfare)
plt.show()
```

Set up for a constrained optimization problem:

```
constraint1 = lambda x: x-0.0
constraint2 = lambda x: 2.0-x
const1 = ({'type': 'ineq', 'fun' : constraint1})
const2 = ({'type': 'ineq', 'fun' : constraint2})
const = [const1, const2]
```

Derivative-based (Sequential Least Squares Programming):

```
In[12]: inv_init = [1.49, 1.49]
   ...: minimize(Inv_W, inv_init,  method='SLSQP', constraints=const)
Out[12]:
     fun: -3.2490095854249414
     jac: array([0.56857666, 0.56857666])
 message: 'Optimization terminated successfully.'
    nfev: 12
     nit: 3
    njev: 3
  status: 0
 success: True
       x: array([4.4408921e-16, 0.0000000e+00])
```

Derivative-free (Constrained Optimization By Linear Approximation):

```
In[13]: inv_init = [1.1, 1.1]
   ...: minimize(Inv_W, inv_init,  method='COBYLA', constraints=const)
Out[13]:
     fun: -7.010346004639045
   maxcv: 3.535533905951738e-05
 message: 'Optimization terminated successfully.'
    nfev: 13
  status: 1
 success: True
       x: array([2.00003536, 2.00003536])
```

Practical Advice: Solving

- First of all, judge if numerical optimization is really necessary

Practical Advice: Solving

- First of all, judge if numerical optimization is really necessary
- Estimate the objective function with multiple optimization algorithms, ideally from difference orders
- Try multiple stopping rules based on tight tolerances
- Try multiple initial guesses for parameter values:
  - Based on prior information from economic theory
  - Without such information, use random draws

Practical Advice: Solving

- First of all, judge if numerical optimization is really necessary
- Estimate the objective function with multiple optimization algorithms, ideally from difference orders
- Try multiple stopping rules based on tight tolerances
- Try multiple initial guesses for parameter values:
    - Based on prior information from economic theory
    - Without such information, use random draws
- (For a very high dimensional problem,) divide the optimization problem in hand in smaller dimensions:
    - Split the paramter vector θ into θ_1 & θ_2
    - Fix θ_1 and estimate θ_2
    - With estimates of θ_2, estimate θ_1
    - Iterate this process until the objective function value changes little
    - Finally, simultaneously estimate θ_1 & θ_2

Practical Advice: Solving

- First of all, judge if numerical optimization is really necessary
- Estimate the objective function with multiple optimization algorithms, ideally from difference orders
- Try multiple stopping rules based on tight tolerances
- Try multiple initial guesses for parameter values:
  - Based on prior information from economic theory
  - Without such information, use random draws
- (For a very high dimensional problem,) divide the optimization problem in hand in smaller dimensions:
  - Split the paramter vector θ into θ_1 & θ_2
  - Fix θ_1 and estimate θ_2
  - With estimates of θ_2, estimate θ_1
  - Iterate this process until the objective function value changes little
  - Finally, simultaneously estimate θ_1 & θ_2
- Consider combining optimization algorithms:
  - Grid search with coarse grids for guessing a reasonable range of parameter values
  - Then, use other faster methods to estimate the parameters

Practical Advice: Validating

- Collect the sets of parameter estimates & objective function values
- Pick the parameter estimates which achieve the lowest objective function value among the collected sets
- Make sure that the algorithm converges to the picked parameter estimates

Practical Advice: Validating

- Collect the sets of parameter estimates & objective function values
- Pick the parameter estimates which achieve the lowest objective function value among the collected sets
- Make sure that the algorithm converges to the picked parameter estimates
- Plot the objective function value against each of the parameters over an interval around the estimated value, keeping the other parameters fixed at the estimated values

$\Rightarrow$ Make sure it passes the "ocular" identification test

Practical Advice: Validating

- Collect the sets of parameter estimates & objective function values
- Pick the parameter estimates which achieve the lowest objective function value among the collected sets
- Make sure that the algorithm converges to the picked parameter estimates
- Plot the objective function value against each of the parameters over an interval around the estimated value, keeping the other parameters fixed at the estimated values

$\Rightarrow$ Make sure it passes the "ocular" identification test

- Simulate a dataset given the estimated parameter values

$\Rightarrow$ Estimate the parameters given the simulated dataset and make sure that the newly estimated parameters have the same values as the ones obtained first

Practical Advice: Validating

- Collect the sets of parameter estimates & objective function values
- Pick the parameter estimates which achieve the lowest objective function value among the collected sets
- Make sure that the algorithm converges to the picked parameter estimates
- Plot the objective function value against each of the parameters over an interval around the estimated value, keeping the other parameters fixed at the estimated values

$\Rightarrow$ Make sure it passes the "ocular" identification test

- Simulate a dataset given the estimated parameter values

$\Rightarrow$ Estimate the parameters given the simulated dataset and make sure that the newly estimated parameters have the same values as the ones obtained first

- **Again, no perfect procedure for identifying global optima exists!** But think wisely for reducing the possibilities of failures!

TABLE 4.—OPTIMIZATION-DESIGN DETAILS AND DIAGNOSTICS CHECKLIST

Step	Details and Diagnostics
1. Optimization design	Optimization algorithm Starting values Objective function value tolerance Parameter vector tolerance Other optimization settings Fixed-point iteration settings Market share evaluation draws
2. Convergence and local optima	Multiple optima (Y/N) Number of runs converged Algorithm exit code Objective function value Parameter estimates Gradient-based FOC diagnostics Hessian-based SOC diagnostics
3. Implications for economic variables of interest	Variation due to multiple optima, if any, for: • Objective function value • Parameter estimates • Own- and cross-price elasticities: statistics • Other economic variables of interest: statistics

Source: Nittel & Metaxoglou (2014)

Further Readings

There are still much more methods that this lecture has not covered. Please see, for example:

- *Judd, Chapter 4*
- Note by Fernández-Villaverde & Guerrón
- Note by Todd Munson
- Knittel, Christopher R., and Konstantinos Metaxoglou. "Estimation of random-coefficient demand models: two empiricists' perspective." *Review of Economics and Statistics* 96.1 (2014): 34-59.
- Also, read SciPy documentations carefully

Lab

Exercise

Aim. Experience the imperfection of numerical optimization for finding a global solution

Task. Minimize $f(x) = 3x^4 - 5x^3 + 2x^2$ by

1. (Quasi-)Newton's and Nelder-Mead methods with initial guesses
   1.1. -0.25
   1.2. 0
   1.3. 0.25
   1.4. 0.5
   1.5. 0.75
   1.6. 1
2. Grid search

Assignment 4

Setting: Quasi-hyperbolic discounting structure

- An application to a simple structural model
- Time preferences play important roles for various dynamic decisions.
- Not only discount factor, but also present biasness matters.
  e.g.) I do not want to do my homework just now, so I allocate much more study time to tomorrow. The ratio of study time between today and a future day can differ from the planned ratio between two future days with an equal interval.
- An individual at period t maximizes lifetime utility:

$$U(c) = u(c_t) + \beta \sum_{k=1}^{\infty} \delta^k u(c_{t+k})$$

- Read Andreoni and Sprenger (2012 AER), Augenblick et al. (2015 QJE), and Casaburi and Macchiavello (2019 AER) if you are interested, but not necessary for this assignment.

Experiment & Data Generating Process

- A researcher conducts a lab experiment in India for obtaining time preference parameters.
- Subjects are asked to choose two-period intertemporal allocations of money (Rs. 4000) within a convex budget set, with various t (earlier date), k (time interval between th earlier and later dates), and several interest rates $P = (1 + r)$.
- Assume that the subjects solve:

$$U(c_t, c_{t+k}) = c_t^\alpha + \beta^{\mathbb{1}_{t=0}} \delta^k c_{t+k}^\alpha$$
$$\text{s.t. } Pc_t + c_{t+k} = 4000$$

- Solving this,

$$c_t = \frac{4000(\beta^{\mathbb{1}_{t=0}} \delta^k P)^{1/(\alpha-1)}}{1 + P(\beta^{\mathbb{1}_{t=0}} \delta^k P)^{1/(\alpha-1)}} \equiv g(\mathbb{1}_{t=0}, k, P; \beta, \delta, \alpha)$$

- Parameters: β: present biasness, δ: discount factor, α: curvature $(IES = 1/(1-\alpha))$.

Convext Time Budget (CTB) Experiment

Section C: निर्देश

- Test C-1 के answer sheet का एक उदाहरण

ऑप्षन (option) A-E में कोई एक चुने

	Option A	Option B	Option C	Option D	Option E
Today	3800 Rs.	2850 Rs.	1900 Rs.	950 Rs.	0 Rs.
	&	&	&	&	&
5 weeks from today	0 Rs.	1000 Rs.	2000 Rs.	3000 Rs.	4000 Rs.

Convext Time Budget (CTB) Experiment (cont'd)

Answer sheet for Test C-19

30

	Option A	Option B	Option C	Option D	Option E
	☐	☐	☐	☐	☐
9 weeks later→	4000 Rs.	3000 Rs.	2000 Rs.	1000 Rs.	0 Rs.
	&	&	&	&	&
18 weeks later→	0 Rs.	1000 Rs.	2000 Rs.	3000 Rs.	4000 Rs.

Assignment: Non-Linear Least Square Estimation

Aim. Validate the numerical solution to a nonlinear problem

- Distributed data: $w_{i,q} \equiv \{c_{i,t_q}, c_{i,t_q+k_q}, t_q, k_q, P_q\}$
  (i: individual, q: question)

Task. Estimate the parameters $\hat{\theta} = (\hat{\beta}, \hat{\delta}, \hat{\alpha})$ by non-linear least squares (NLLS):

$$\min_{\beta,\delta,\alpha} \sum_{i,q} [c_{i,t_q} - g(\mathbb{1}_{t_q=0}, k_q, P_q; \beta, \delta, \alpha)]^2$$

- Try both derivative-based and detivative-free methods.
- Feel free to use the SciPy optimization package.
- Follow the practical advice to validate your result.

  E.g.) Plot each parameter value and the objective function, fixing the other parameter values at the estimated values.