BYU-MCL Boot Camp
Optimization Labs
Professor Richard W. Evans

1 Introduction

Well defined systems of equations often do not have analytical solutions. However, numerical solutions can also be difficult to find if the system is highly nonlinear, involves many equations, has singularities, involves inequality constraints, or some combination of these. The objective of these exercises is to familiarize you with some of the different root finding and optimization routines available through Python and specifically in the `scipy.optimize` library. You will learn some of the advantages and limitations of each one. You will come away from this understanding that numerical optimization involves a little bit of artistry and an intimate understanding of the theory behind the equations you are optimizing.

2 Root finding

Let $\mathbf{F}(\mathbf{x})$ be a system of m functions of the vector $\mathbf{x}$ of length n, and the function $\mathbf{F}(\mathbf{x})$ returns a vector of length m. The root of the function is the particular vector $\mathbf{x}$ such that $\mathbf{F}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{0}$ is an m-length vector of zeros. Because many economic models are characterized by systems of equations, the roots of those systems are often the solutions to economic models.

If a system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ is linear, it can be represented as $\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ or $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{x}$ is an $n \times 1$ vector, $\mathbf{b}$ is an $m \times 1$ vector, and $\mathbf{0}$ is an $m \times 1$ vector of zeros. This is the classical linear algebra problem in which there may be many, exactly one, or no solution to the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$.

1. There is exactly one solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ if $\mathbf{A}$ is square ($n \times n$ or $m = n$) and has full rank, $\text{rank}(\mathbf{A}) = n$. This means $\mathbf{A}$ has n linearly independent rows and $\mathbf{A}$ is invertible.

2. There are multiple solutions to $\mathbf{A}\mathbf{x} = \mathbf{b}$ if $\text{rank}(\mathbf{A}) < n$.

3. There is no solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ if $\text{rank}(\mathbf{A}) > n$.

If the system $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ is nonlinear, the process of finding the roots is much less straightforward. It often involves a Newton method and the evaluation of a gradient or matrix of derivatives (Jacobian matrix).

Use the following description of the 3-period-lived agent, perfect foresight OLG model from the first week of the course in order to complete Exercises 1 through 4 below. The decisions of the households in the economy can be summarized by the following equations.

$$(c_{1,t})^{-\gamma} - \beta(1 + r_{t+1} - \delta)(c_{2,t+1})^{-\gamma} = 0 \tag{1}$$
$$(c_{2,t})^{-\gamma} - \beta(1 + r_{t+1} - \delta)(c_{3,t+1})^{-\gamma} = 0 \tag{2}$$
$$c_{1,t} + k_{2,t+1} - w_t = 0 \tag{3}$$
$$c_{2,t} + k_{3,t+1} - w_t - (1 + r_t - \delta)k_{2,t} = 0 \tag{4}$$
$$c_{3,t} - (1 + r_t - \delta)k_{3,t} = 0 \tag{5}$$
$$w_t - (1-\alpha)A\left(\frac{K_t}{L_t}\right)^{\alpha} = 0 \tag{6}$$
$$r_t - \alpha A\left(\frac{L_t}{K_t}\right)^{1-\alpha} = 0 \tag{7}$$
$$K_t - k_{2,t} - k_{3,t} = 0 \tag{8}$$
$$L_t - 2 = 0 \tag{9}$$

If we substitute equations (3) through (9) into (1) and (2) and look only at the steady-state, the steady-state values $\big(\bar{k}_2, \bar{k}_3\big)$ are characterized by the following two equations.

$$\begin{aligned} &u'\Big(w(\bar{k}_2,\bar{k}_3) - \bar{k}_2\Big) - \beta\big(1 + r(\bar{k}_2,\bar{k}_3) - \delta\big) \times ... \\ &\quad u'\Big(w(\bar{k}_2,\bar{k}_3) + [1 + r(\bar{k}_2,\bar{k}_3) - \delta]\bar{k}_2 - \bar{k}_3\Big) = 0 \end{aligned} \tag{10}$$

$$\begin{aligned} &u'\Big(w(\bar{k}_2,\bar{k}_3) + [1 + r(\bar{k}_2,\bar{k}_3) - \delta]\bar{k}_2 - \bar{k}_3\Big) - ... \\ &\quad \beta\big(1 + r(\bar{k}_2,\bar{k}_3) - \delta\big)u'\Big([1 + r(\bar{k}_2,\bar{k}_3) - \delta]\bar{k}_3\Big) = 0 \end{aligned} \tag{11}$$

Let the parameter vector values of the model be given by $\theta = [\beta, \gamma, \alpha, \delta, A] = [0.442, 3, 0.35, 0.6415, 1]$. In Exercises 1 and 2, set the tolerance in the solver to `xtol=1e-10`. Also, because some of the root finding methods in Exercise 2 do not allow you to pass extra arguments into the root finding function, you will need to define an anonymous function in addition to your steady-state distribution of capital function. `ksssolve` is the function that I have defined for my root finding algorithm to call to find the steady-state distribution of capital.

```
def ksssolve(kvec, params):
    ... # Define function to solve for steady-state distribution
    ... # of capital "kvec" given parameters vector "params"
```

For some of the root finders, like `fsolve` in Exercise 1, you can pass in many extra arguments through the `args=(params)` syntax. However, two of the root finders in Exercise 2 do not allow for extra arguments to be passed. So you'll have to write an anonymous function in Python to be an intermediate step between the root finding command and the `ksssolve` function.

```
zero_func = lambda x: ksssolve(x, params)
```

For Exercises 1 and 2, you can write a root finding function syntax in the following form,

```
import scipy.optimize as opt
...
kssvec = opt.[RootFinder](zero_func, kinit, [meth='method'], [x]tol=1e-10)
```

that calls the anonymous function `zero_func`, which passes both the guess for the distribution of capital `kinit` and the parameters `params` to the `ksssolve` function, regardless of whether `fsolve` or the particular method of `root` allows extra argument passing.

The last little examples of Python code I want you to use is the time library for clocking computation speeds of different solution methods as well as some output printing commands.

```
import scipy.optimize as opt
import time
...
# Put all parameter and function definition code outside of timer
...
start_time = time.time()
kssvec = opt.[RootFinder](zero_func, kinit, [meth='method'], [x]tol=1e-10)
elapsed = time.time() - start_time
```

In order to have your script print the output that you want, you can use the following code.

```
print('The SS capital levels and comp. times for fsolve are:')
print("k2ss_f = %.8f" % k2ss_f)
print("k3ss_f = %.8f" % k3ss_f)
print("Time Elapsed: %.6f seconds" % elapsed_f)
```

The `%` operator tells the `print` command that a string is going to be formatted. The `.8f` and `.6f` commands tell the `print` command that the string following the second `%` character is to be displayed as a floating point number with 8 and 6 decimal places, respectively, represented after the decimal.

Exercise 1. Solve for the steady-state distribution of capital savings $(\bar{k}_2, \bar{k}_3)$ from the three-period lived agent perfect foresight model described in (10) and (11) above using the `scipy.optimize.fsolve` command. Report the computed steady-state distribution of capital $(\bar{k}_2, \bar{k}_3)$ and how long (in seconds) it took to compute.

Exercise 2. Solve for the same OLG steady-state distribution of capital savings $(\bar{k}_2, \bar{k}_3)$ using the alternative root finding command `scipy.optimize.root` with each of the following methods options: `hybr`, `broyden1`, and `krylov`. One of these methods will not work. Report the computed steady-state distribution of capital $(\bar{k}_2, \bar{k}_3)$ for each method and how long (in seconds) each one took to compute.

Exercise 3. What is the biggest difference between the solution from Exercise 1 and different solution methods in Exercise 2? That is, what is the biggest $(\bar{k}_2, \bar{k}_3)_{\text{ex1}} - (\bar{k}_2, \bar{k}_3)_{\text{ex2i}}$ where $i = \{\texttt{hybr}, \texttt{broyden1}, \texttt{krylov}\}$ among methods that worked?

Exercise 4. Which method had the minimum computation time and which method had the maximum computation time among the method in Exercise 1 and the two methods that worked in Exercise 2? [NOTE: You will get different computation times each time you run this. But the relative computation speeds ordering will remain the same.]

Each solver uses different methods and different coding approaches and efficiencies to arrive at the solution. So it can sometimes be a significant task to find a solver that works. Once you find a class of solvers that works, you may face a tradeoff between robustness and computational speed. Another library of convex optimization functions for Python that wraps the BLAS, LAPACK, FFTW, UMFPACK, CHOLMOD, GLPK, DSDP5, and MOSEK routines is `cvxopt`.

3 Unconstrained optimization (minimization)

The characterizing equations or data generating process (DGP) in an economic model often come from some set of optimization problems. In the case of the OLG problem from the first chapter of the course and the infinite horizon, representative agent DSGE model from the second chapter, both households and firms are optimizing. Although some economic problems do involve minimization decisions (e.g., cost minimization, loss function minimization), most focus on some type of maximization. This begs the question of why the code libraries of all major programming languages have only minimizers and not maximizers. First, any maximization problem can be rewritten as a minimization problem. Second, a rewritten minimization problem that must converge to a finite number like zero is easier than a problem that can go to ∞ or $-\infty$.

The difference between a root finder and a minimizer is subtle but important. And the take away should be that a minimizer is more complex and less exact than a root finder, but more flexible. Let's use the general system of equations $\mathbf{F}(\mathbf{x}) = \mathbf{0}$ discussed in Section 2. A root finder is finding a solution $\mathbf{x}$ that delivers the zero vector $\mathbf{0}$. But a minimizer is finding the vector $\mathbf{x}$ that minimizes the scalar valued function $\mathbf{g}(\mathbf{x})$. Note that you often want $\mathbf{g}(\mathbf{x})$ to map $\mathbf{x}$ into the nonnegative real line so that the minimizing solution $\mathbf{x}$ sets $\mathbf{g}(\mathbf{x})$ as close to the scalar 0 as possible. One very common example for the nonnegative scalar valued function $\mathbf{g}(\mathbf{x})$ is the sum of the squares of the value of each individual equation in $\mathbf{F}(\mathbf{x})$: $\mathbf{g}(\mathbf{x}) = \sum_{i=1}^m \Big[F_i(\mathbf{x})\Big]^2$. This minimizer is a least squares solution technique.

Exercise 5. Solve for the steady-state distribution of capital savings $\left(\bar{k}_2, \bar{k}_3\right)$ from the three-period lived agent perfect foresight OLG model described in (10) and (11) in Section 2 above using the `scipy.optimize.minimize` minimizer command with the method set to `method='Nelder-Mead'` and the tolerance set to `tol=1e-10`. Report your solution and computation time.

Exercise 6. What do you get if you try to solve for the steady state in Exercise 5 using the initial guess $(\bar{k}_2, \bar{k}_3) = (0.51, 0.51)$? Why does this happen? [HINT: try looking at the values of the other variables when $(\bar{k}_2, \bar{k}_3) = (0.51, 0.51)$ is the steady-state.]

Unconstrained optimization (minimization) computational routines obviously are the right choice when the range of the vector being chosen is unbounded. But unconstrained minimizers can also work well, as in Exercise 5, when the bounds are well known and theoretical conditions (Inada conditions) push the solution away from the boundary. Care must simply be taken to input good starting values.

Now we use the representative infinitely lived DSGE Brock and Mirman (1972) model with inelatically supplied labor of $l_t = 1$ in every period and known closed form solution for the equilibrium policy function from the second chapter to illustrate the value of a minimizer. I have characterized a very simple form of i.i.d. uncertainty for the firm's productivity shock in (19) to make computing the expectation in the household's problem easier, and the Brock and Mirman (1972) policy function is in equation (18). The decisions of the representative household and the representative firm in the economy can be summarized by the following equations.

$$(c_t)^{-1} = \beta E\Big[r_{t+1}(c_{t+1})^{-1}\Big] \tag{12}$$

$$c_t = w_t + r_t k_t - k_{t+1} \tag{13}$$

$$w_t = (1-\alpha)e^{z_t}\left(\frac{K_t}{L_t}\right)^{\alpha} \tag{14}$$

$$r_t = \alpha e^{z_t}\left(\frac{L_t}{K_t}\right)^{1-\alpha} \tag{15}$$

$$K_t = k_t \tag{16}$$

$$L_t = 1 \tag{17}$$

$$k_{t+1} = \psi(k_t, z_t) = \alpha\beta e^{z_t} k_t^{\alpha} \tag{18}$$

$$\Pr(z_t = -0.2) = \frac{1}{2} \quad \text{and} \quad \Pr(z_t = 0.2) = \frac{1}{2} \tag{19}$$

If we substitute equations (13) through (18) into (12), then the equilibrium is characterized by a sequence of Euler equations holding in every period of time in which the expectations on the right-hand-side are formed using knowledge of the distribution of the productivity shock z_{t+1} as described in (19).

$$\begin{aligned} \Bigl[w(k_t) + r(k_t)k_t - k_{t+1}\Bigr]^{-1} &= ... \\ \beta E\left[r(k_{t+1})\Bigl(w(k_{t+1}) + r(k_{t+1})k_{t+1} - \psi(k_{t+1}, z_{t+1})\Bigr)^{-1}\right] \end{aligned} \tag{20}$$

Now suppose that each period is a year, and you have 41 years of data on the capital stock, wage, and interest rate in your economy $\{k_t, w_t, r_t\}_{t=1}^{41}$. Suppose that you thought that the data $\{k_t, w_t, r_t\}_{t=1}^{41}$ were generated by a process described in equations (12) through (19). An equivalent way of saying that is to suppose that you though the data were generated by the intertemporal Euler equation (20). But you do not know the value of the parameters of the model $[\alpha, \beta]$. How might you estimate those parameters $[\alpha, \beta]$ to make the model (20) best match the data $\{k_t, w_t, r_t\}_{t=1}^{41}$? We want to choose parameter values α and β that make the model match up best with the data in some sense.

Define the parameter vector as $\theta = [\alpha, \beta]$. Suppose we can write the characterizing equations of the model, or data generating process (DGP), in a form in which the expected value of the characterizing equations equals zero. Then the generalized method of moments (GMM) approach of estimating the parameter vector θ is to choose θ to minimize some scalar-valued nonnegative function of those equations.[1] Put differently, GMM chooses θ to minimize a scalar function (or function of moments) of the parameters that should equal zero. Each equation can be thought of as a moment or function of the parameters.

A set of functions of the parameters in the Brock and Mirman (1972) model for the 41 years of capital stock data are the 40 intertemporal Euler equations that can be evaluated and must hold over that period. Further, these equations can be rearranged to equal zero, and they hold in expectation.

$$\begin{aligned} &\Bigl[w(k_t) + r(k_t)k_t - k_{t+1}\Bigr]^{-1} - ... \\ &\quad \beta E_{z_{t+1}}\left[r(k_{t+1})\Bigl(w(k_{t+1}) + r(k_{t+1})k_{t+1} - \psi(k_{t+1}, z_{t+1})\Bigr)^{-1}\right] = 0 \\ &\quad \text{for} \quad 1 \leq t \leq 40 \end{aligned} \tag{21}$$

The difference in expected marginal utilities that equals zero as represented in (21) is often referred to as an Euler error. The GMM approach to estimation of the Brock and Mirman (1972) model is to choose $\theta = [\alpha, \beta]$ to minimize the sum of squared Euler errors given the data $\{k_t, w_t, r_t\}_{t=1}^{41}$.

This problem is overidentified in that we have 40 equations or moments and 2 unknowns. As such, no combination of $\theta = [\alpha, \beta]$ is likely to set all 40 equations or moments represented in (21) simultaneously equal to zero.[2] So we just want to choose

[1]See Davidson and MacKinnon (2004, Ch. 9) for a more detailed treatment of GMM. The estimation methods of linear least squares, nonlinear least squares, generalized least squares, and instrumental variables estimation are all specific cases of the more general GMM estimation method.

[2]Note that θ could be chosen to exactly set the moments to zero, exactly solving (21), if we had exactly as many parameters to estimate as we had equations or moments.

θ to minimize the sum of squared Euler errors. To see how this works, we rewrite (21) in its inexact Euler error form.

$$\begin{aligned}&\Big[w_t + r_t k_t - k_{t+1}\Big]^{-1} - ...\\ &\qquad \beta E_{z_{t+1}}\left[r(k_{t+1})\Big(w(k_{t+1}) + r(k_{t+1})k_{t+1} - \psi(k_{t+1}, z_{t+1})\Big)^{-1}\right] = \mu_t(\theta) \\ &\qquad\qquad \text{for} \quad 1 \leq t \leq 40\end{aligned} \tag{22}$$

For given values of the parameters θ and for given data $\{k_t, w_t, r_t\}_{t=1}^{41}$, the Euler errors in (22) evaluate to a sequence of 40 scalars. A GMM estimator of the parameter vector $\hat{\theta} = [\hat{\alpha}, \hat{\beta}]$ can be represented in the following way.

$$\hat{\theta} = \arg\min \sum_{t=1}^{40} \mu_t(\theta)^2 \tag{23}$$

Computing the Euler errors in (22) can be a little tricky because you have to evaluate the expectation in the second term as some type of integral. The timing of the representative household's decision each period and is such that the household only knows the current period wage w_t, interest rate r_t, capital investment from the previous period k_t, and capital investment decision for the current period k_{t+1}. So even though the econometrician has the entire time series of wages, interest rates, and capital stocks $\{k_t, w_t, r_t\}_{t=1}^{41}$, the households only have the current period's data and the point of any decision. So they must calculate an expectation over the next period. Because we have simplified the shock process for z_t in (19) to be i.i.d. and discrete with only two values, evaluating the expected value in the Euler error (22) is fairly straightforward.

$$\frac{1}{w_t + r_t k_t - k_{t+1}} - \beta\frac{1}{2}\left[\frac{\alpha e^{-0.2} k_{t+1}^{\alpha-1}}{(1-\alpha\beta)e^{-0.2}k_{t+1}^{\alpha}}\right] - \beta\frac{1}{2}\left[\frac{\alpha e^{0.2} k_{t+1}^{\alpha-1}}{(1-\alpha\beta)e^{0.2}k_{t+1}^{\alpha}}\right] = \mu_t(\theta) \quad \text{for} \quad 1 \leq t \leq 40 \tag{24}$$

Note how much cancels in the last two terms on the left-hand-side of (24). In fact, the series of Euler errors simplifies down to a zero version of the policy function in (18).

$$\alpha\beta(w_t + r_t k_t) - k_{t+1} = \mu_t(\alpha, \beta) \quad \text{for} \quad 1 \leq t \leq 40 \tag{25}$$

One last characteristic to note from the characterizing equation (25) or DGP is that we cannot identify α and β separately. The two parameters enter every equation as a product $\alpha\beta$, so we will not be able to identify both parameters unless we had another set of equations in which α and β entered in a way different from a product.

Exercise 7. Assume that the capital share of income $\alpha = 0.35$. Use the 41 years of data $\{k_t, w_t, r_t\}_{t=1}^{41}$ in the tab-delimited file `gmmdata.txt` and the simplified representation of the Euler errors in (25) to estimate by GMM the discount factor β that minimizes the sum of squared Euler errors. In the file `gmmdata.txt`, each row is a period, the first column is capital k_t, the second column is the wage w_t, and the third column is the interest rate r_t. Use the `scipy.optimize.minimize` minimizer command with the method set to `method='Nelder-Mead'` and the tolerance set to `tol=1e-10`. Report your solution for $\hat{\beta}$, the vector of Euler errors, the sum of squared Euler errors $\sum_{t=1}^{40}\mu_t(\hat{\beta})^2$, and the computation time. [Try some different initial guesses for β to see how robust this GMM estimation method is.]

$$\hat{\beta} = \arg\min \sum_{t=1}^{40}\mu_t(\beta)^2$$

If you were serious about the econometrics behind estimating $\hat{\beta}$, you would want to include an optimal weighting matrix in the criterion function (sum or squared Euler errors) and also calculate the vector of standard errors associated with the vector of parameter estimates. See Davidson and MacKinnon (2004, Ch. 9) for more details.

As a coding note, Exercise 7 asks you to report not only your GMM estimate for $\hat{\beta}$, but also the vector of Euler errors and the sum of squared Euler errors (criterion function). The sum of squared errors is easy because it is one of the objects automatically saved as output by the `scipy.optimize.minimize` function. Suppose I had executed my `minimize` function with the following line of code.

```
parvec = opt.minimize(crit_func, parinit, method='Nelder-Mead', tol=1e-10)
```

The return object `parvec` is called a result object and automatically stores a number of important results of the minimization function. The first one that is probably most important is the array of parameter estimates in `parvec.x`. But the sum of squared Euler errors will be stored in `parvec.fun`. You can see all the objects stored in `parvec` by typing `parvec` in the command line after running the minimization script.

You can save your entire vector of Euler errors by declaring the array as a vector of zeros before the function being minimized is called. Then you must refer to the vector of Euler errors within the function being minimized as `ArrayName[:]`. This tells Python that the object within the function is the same one declared outside of the function. You only want to do this in the case of objects that you will need for future use in your computations.

```
EulErrVec = np.zeros(data.shape[0]-1)
...
def gmm_crit(paramvec, params, data):
    ...
    EulErrVec[:] = alpha * beta * (wt + rt * kt) - ktp1
    ...
```

4 Constrained optimization (minimization)

The problem with using the unconstrained minimizer in Exercise 7 is that the parameter that we were trying to estimate was constrained $\beta \in (0,1)$. Luckily, our problem was very well behaved, and the constraints on β were not a problem. The GMM estimator returned the same $\hat{\beta}$ for many different initial guesses. However, this is not always the case. In many economic problems, the estimation will go to a place that is outside the constraints of the model unless those constraints are explicitly accounted for in the minimization. That is, constraints are specified in economic models because they are often binding. Your task in Exercise 8 is to do the same operation as in Exercise 7 but with a constrained minimizer.

One note regarding the constraints is important before actually executing a constrained minimization. `scipy.optimize.minimize` has two constrained minimizers—`meth='L-BFGS-B'` and `meth='TNC'`—for simple *lower bound* and *upper bound* constraints. Two other constrained minimizers—`meth='COBYLA'` and `meth='SLSQP'`—allow for more complex forms of constraints. In our example, the discount factor is restricted to $\beta \in (0,1)$. The parenthesis mean that 0 and 1 are not included in the feasible set. When we enter constraints in a python minimizer, the bounds are inclusive. So we have to enter them as something like $(\varepsilon, 1-\varepsilon)$, where ε is some small positive number.

```
parvec = opt.minimize(crit_func, parinit, method='L-BFGS-B', \
  bounds=[(1e-10,1-1e-10)], tol=1e-10)
```

Exercise 8. Assume that the capital share of income $\alpha = 0.35$. Use the 41 years of data $\{k_t, w_t, r_t\}_{t=1}^{41}$ in the tab-delimited file `gmmdata.txt` and the simplified representation of the Euler errors in (25) to estimate by GMM the discount factor β that minimizes the sum of squared Euler errors. Use the `scipy.optimize.minimize` constrained minimizer command with the method set to `method='L-BFGS-B'` and the tolerance set to `tol=1e-10`. Input the bounds to be $\beta \in [\varepsilon, 1-\varepsilon]$, where $\varepsilon = 1e-10$. Report your solution for $\hat{\beta}$, the vector of Euler errors, the sum of squared Euler errors $\sum_{t=1}^{40} \mu_t(\hat{\beta})^2$, and the computation time.

$$\hat{\beta} = \arg\min \sum_{t=1}^{40} \mu_t(\beta)^2 \quad \text{s.t.} \quad \beta \in (0,1)$$

References

BROCK, W. A., AND L. MIRMAN (1972): "Optimal Economic Growth and Uncertainty: the Discounted Case," *Journal of Economic Theory*, 4(3), 479–513.

DAVIDSON, R., AND J. G. MACKINNON (2004): *Econometric Theory and Methods*. Oxford University Press.