BEPP 931 - Solution to Problem Set 3

Shasha Wang, Cung Truong Hoang, Jose Sidaoui

March 13, 2020

Question 1 & Question 2 - fminsearch VS fminunc

Question 1 and Question 2 are exploited to compare between two matlab optimization functions - fminsearch and fminunc. The conclusion is that **although neither fminsearch nor fminunc yields satisfactory solution with unreasonable initial guesses, fminsearch is much more sensitive to initial guesses than fminunc**. It is natural that when solving for optimizers of large dimensionality, fminunc performs well whereas fminsearch hardly solves the problem unless the initial guess is good enough to approach the final solution.

Before explaining the questions' settings, it is necessary to briefly discuss the two function's algorithms. Fminsearch uses comparison method (i.e., the Nelder-Mead simplex algorithm) without making use of derivative information, and fminunc uses direction set method which takes into account the objective function's gradient and hessian. Hence fminsearch is more often used for rough problems whereas fminunc for smooth ones. Since the objective funtions of both Question 1 and Question 2 are smooth, we apply the two functions to both problems to evaluate their performance.

Question 1

Question 1 is a two-period portfolio choice problem, where we are asked to make decisions today on the portfolio of both one risk-free and two risky assets. With asset decisions being made, consumption is naturally set.

Hence we are to solve for a three-dimensional optimizer that maximizes the two-period utility. The investor solves

$$\max_{c,\omega_1,\ldots,\omega_n} u(c) + \sum_{s=1}^{S} \pi^s u\left(\sum_{i=1}^{n} \omega_i z_i^s\right)$$

subject to

$$c + \sum_{i=1}^{n} p_i \left(\omega_i - e_i\right) = 0$$

Note that two considerations have to be taken care of in the code. First, it is economically sensible to assume that consuming a 0 or negative amount would lead to infinitely positive marginal utility or infinitely negative utility. Second, we also have to assume no borrowing, otherwise the objective function would explode without limit. Because of these binding constraints, we have to be very careful with the initial guesses, otherwise it would probably only return the initial guess as the local minimum.

Assume a $\in\{0.5,1,5\}$, p=(1,1,1), and e=(2,0,0).

The returns to asset 2 are uniformly distributed over $(0.72, 0.92, 1.12, 1.32)$. The returns to asset 3 are uniformly distributed over $(0.86, 0.96, 1.06, 1.16)$. This defines 16 equally probable states of the world.

First we use an initial guess of $\omega = (0, 0, 0)$, then $\omega = (1/3, 1/3, 1/3)$, then $\omega = (1/2, 1/2, 1/2)$, and finally $\omega = (2/3, 2/3, 2/3)$. Note that both the first and the last cases are to some extent "unreasonable", since the allocation $\omega = (0, 0, 0)$ consumes up all the resources today without leaving anything to consume tomorrow, and the allocation $\omega = (2/3, 2/3, 2/3)$ doesn't allow any consumption today and thus postpone enjoyment as much as possible till tomorrow.

$\omega = (0, 0, 0)$

By fminsearch, we have

a	0.5	1	5
ω_1	0.2295	0.2284	0.7588
ω_2	0.5102	0.5078	0.0801
ω_3	0.2614	0.2603	0.1603

By fminunc, we have

a	0.5	1	5
ω_1	−1.4124	−0.2062	0.7588
ω_2	0.8015	0.4007	0.0801
ω_3	1.6029	0.8015	0.1603

We can see that fminunc gives correct solution, while fminsearch only does well when $a = 5$.

$\omega = (1/3, 1/3, 1/3)$

By fminsearch, we have

a	0.5	1	5
ω_1	−1.4123	−0.2062	0.7588
ω_2	0.8014	0.4007	0.0801
ω_3	1.6029	0.8014	0.1603

By fminunc, we have

a	0.5	1	5
ω_1	−1.4124	−0.2062	0.7588
ω_2	0.8015	0.4007	0.0801
ω_3	1.6029	0.8015	0.1603

We can see the results are rough the same, and the fminunc yields the same solution as before, indicating its insensitivity to initial guesses.

$\omega = (1/2, 1/2, 1/2)$

Both fminsearch and fminunc give the correct solution.

a	0.5	1	5
ω_1	−1.4124	−0.2062	0.7588
ω_2	0.8015	0.4007	0.0801
ω_3	1.6029	0.8015	0.1603

$\omega = (2/3, 2/3, 2/3)$

Both fminsearch and fminunc yield the solution nowhere than the initial guess.

Fminsearch gives the following result.

a	0.5	1	5
ω_1	0.6667	0.6667	0.6667
ω_2	0.6667	0.6667	0.6667
ω_3	0.6667	0.6667	0.6667

By fminunc, we have

a	0.5	1	5
ω_1	0.6666	0.6666	0.6666
ω_2	0.6666	0.6666	0.6666
ω_3	0.6666	0.6666	0.6666

To sum up, fminunc is more stable given reasonable initial guesses than fminsearch.

Question 2

Now we have a life-cycle consumption problem. The agent solves

$$\max_{S_0,\ldots,S_T,c_1,\ldots,c_T} \sum_{t=1}^{T} \beta^t u\left(c_t\right)$$

subject to

$$\begin{aligned} S_t =& (1+r)S_{t-1} + w_t - c_t \\ S_0 &= S_T = 0 \end{aligned}$$

Wages $w_t = 1 + 0.2t - 0.003t^2$.

Utility function $u(c) = -\frac{1}{c}$. Discount factor $\beta = 0.96$. Interest rate $r = 0.06$. $T = 50$ periods.

This problem requires guessing a 49 by 1 vector of savings. With this relatively high-dimensional problem, we can see more clearly how fminsearch is more sensitive to initial guesses than fminunc.

Also note that this utility function $u(c) = -\frac{1}{c}$ strictly abandons nonpositive consumption amount, and hence the code has to take care of this caveat.

Furthermore, we need to revise the optimization option to set the maximum number of iteration to 800000, otherwise the solver would stop anyway once it reaches the maximum iteration times. But this makes the solving process very slow.

By varying the initial guesses, we can see fminsearch performs from very poorly to relatively reasonable but still far from satisfactory, whereas fminunc functions in a consistently stable manner.

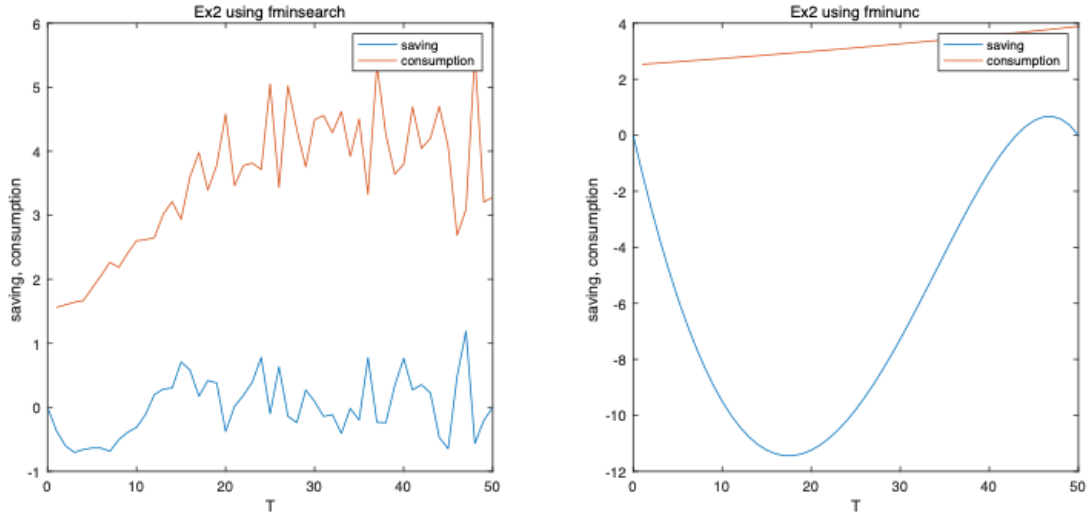

Figure 1. Initial Guess: saving = 0 * wage)

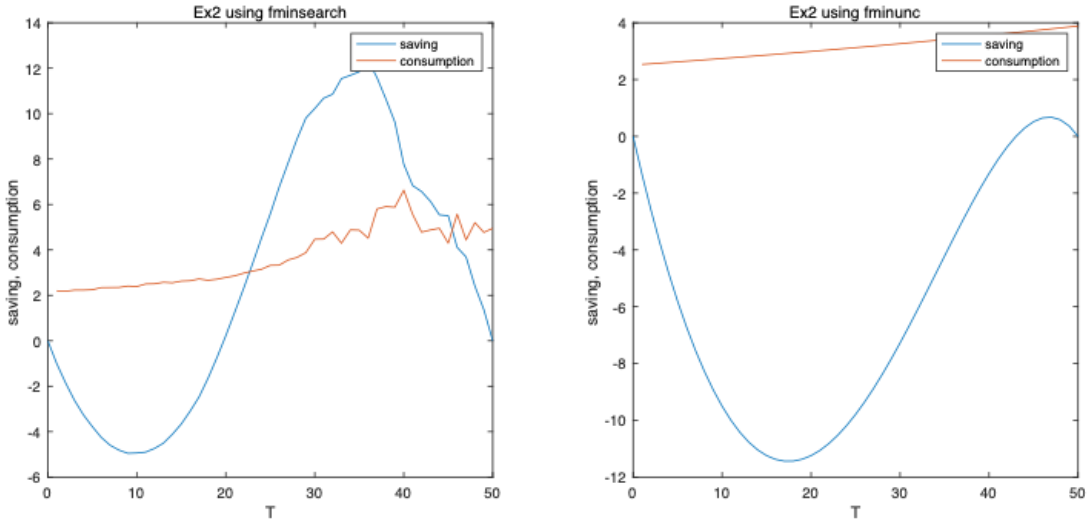

Figure 2. Initial Guess: saving = 0.5 * wage)

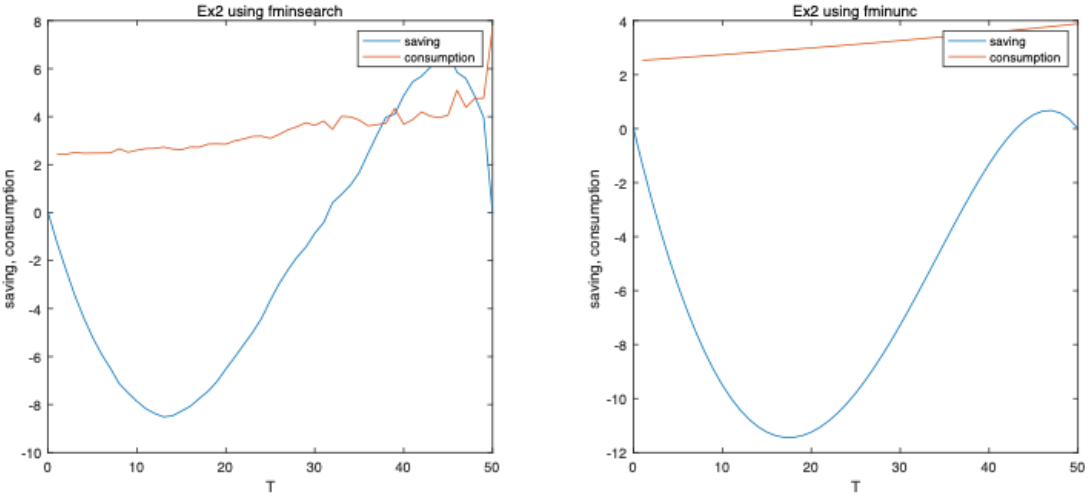

Figure 3. Initial Guess: saving = -0.5 * wage)

We can see the result comes out most preposterously with 0-vector as the initial guess, and more and more similar to the one from fminunc with better and better initial guesses, although still far from satisfactory.

Question 3

We transform the Principal Agent problem into an unconstrained problem, where any violation of the individual rationality and incentive compatibility constraint is penalized by a penalty term equal to the penalty factor P times the squared size of the violation. Computationally, we formulate the problem as a minimization of the unconstrained penalty function. We feed this penalty function into Matlab's fminunc optimization routine to obtain the following results. Note that as we increase P the optimal wage schedule increases and the less we violate the constraints.

P	(w_1, w_2)	IC	IR
10	$(0.4605, 0.7889)$	0.0233	0.1094
10^2	$(0.5355, 1.1110)$	0.0063	0.0199
10^3	$(0.5452, 1.2142)$	$8.36(-4)$	0.0024
10^4	$(0.5462, 1.2289)$	$8.7(-5)$	$2.48(-4)$
10^5	$(0.5463, 1.2304)$	$8.73(-6)$	$2.5(-5)$

For the Adverse Selection problem, we coded our own algorithm using Newton's method with line search. Similar to the Principal Agent problem, we formulate an unconstrained penalty function that incorporates all IR, IC and break-even constraints. The step size is determined by the gradient and the Hessian evaluated at the current guess. We factor the step size by a scalar λ which we obtain by minimizing the objective $f(x+\lambda s)$ using Matlab's fminbnd. The interval we choose for λ is $[-1, 1]$. We obtain the gradient and the Hessian using numerical derivatives. We obtain the following results:

Table 1. Adverse Selection: Newton's Method with Line Search

π^L	θ^H	(y_1^H, y_1^L)	(y_2^H, y_2^L)	IC	Π
0.8	0.1	(0.80085,0.79649)	(0.80247,0.79013)	-3.3168e-15	-6.1627e-09
0.79	0.1	(0.76345,0.89525)	(0.76343,0.89534)	-6.0043e-07	-2.5144e-09
0.7	0.1	(0.7325,0.65618))	(0.72801,0.66529)	-4.4932e-07	7.5604e-11
0.6	0.1	(0.56133,0.7151)	(0.56117,0.71601)	-0.00010835	-3.5453e-09
0.5	0.1	(0.63928,0.4248)	(0.52263,0.52263)	-0.0014091	1.1211e-06
0.8	0.75	(0.80052,0.79792)	(0.8005,0.79799)	-1.9922e-14	-5.5381e-09
0.79	0.75	(0.7978,0.7962)	(0.79582,0.80415)	-4.002e-05	5.6981e-08
0.7	0.75	(0.77507,0.77475)	(0.77507,0.77475)	4.2162e-09	2.0994e-07
0.6	0.75	(0.7501,0.74971)	(0.7501,0.74971)	-6.4867e-09	7.0786e-07
0.5	0.75	(0.69126,0.31015)	(0.58876,0.58645)	-0.085613	-0.11682

Question 4

We solve the 3x3 Normal Form game for its Nash equilibria by setting up the Lyapunov value function along with penalty terms such that we require any mixed strategy to satisfy $\sigma^j = (\sigma_1^j, \sigma_2^j, \sigma_3^j) = 1$ and $\sigma_i^j \leq 0$. We try different initial values to find a multiplicity of equilbria, one of them being for example:

$\sigma^1 = (0, 0.25, 0.75)$ and $\sigma^2 = (0.5524, 0.3105, 0.1371)$

which is one element from the set of solutions of the form:

$\sigma^1 = (0, 0.25, 0.75)$ and $\sigma^2 = (\lambda\frac{2}{3} + (1-\lambda)\frac{7}{13}, \lambda\frac{1}{3} + (1-\lambda)\frac{4}{13}, (1-\lambda)\frac{2}{3})$ for $\lambda = [0, 1]$

Similarly, we solve the 4x4 Normal Form game:

$\sigma^1 = (0, 0.25, 0.75, 0)$ and $\sigma^2 = (0, 0, 0.9091, 0.0909)$

$\sigma^1 = (0, 0, 0.5, 0.5)$ and $\sigma^2 = (0, 0.2536, 0.7464, 0)$

with the second EQ being one element from the set of solutions of the form:

$\sigma^1 = (0, 0, 0.5, 0.5)$ and $\sigma^2 = (0, \lambda, 1-\lambda, 0)$ for $\lambda = [0.25, 1]$

Question 5

Reproduce Table 5.2

We use the Fixed-Point Iteration method for the following system of nonlinear equations, generating the sequence:

$$y_{k+1} = \omega(\Pi_1^Y(e^{y_k}, e^{z_k}) + y_k) + (1-\omega)y_k$$

$$z_{k+1} = \omega(\Pi_2^Z(e^{y_k}, e^{z_k}) + z_k) + (1-\omega)z_k$$

The initial guess (y_0, z_0) used for this method, is (-2,-2) as in the book. Generating this sequence further to produce more than 100 iterates, say 1000, it converges to the true equilibrium given in the book:

$$(y^*, z^*) = (-.137, -.576)$$

After computing the sequence for the first 100 iterates, we display the errors (y_0, z_0)-(y^*, z^*) in the following tables, where in the first one, we set $\omega = 1$, and in the second and third we use extrapolation to try to improve performance by trying different values of ω:

Table 2. Fixed-Point Iteration

ω=1	
Iterate	*Error*
20	(-0.74567, -0.47235)
40	(-0.30854, -0.13596)
60	(-0.13485, -0.024466)
80	(-0.062043, 0.0088345)
100	(-0.030133, 0.015448)

Table 3. Fixed-Point Iteration

ω=.5	
Iterate	*Error*
20	(-1.1866, -0.83951)
40	(-0.7532, -0.47918)
60	(-0.48339, -0.2659)
80	(-0.31452, -0.14105)
100	(-0.20736, -0.068586)

Table 4. Fixed-Point Iteration

ω=9.9	
Iterate	*Error*
5	(-0.12055, 0.0021256)
10	(-0.015565, 0.020403)
15	(-0.0037149, 0.0070944)
20	(-0.0013391, 0.0021808)
25	(-0.00071414, 0.00069405)

As can be observed from the above tables, setting $\omega = 1$ yields errors virtually identical to those in Table 5.3 of the book. When varying the values of ω, it can easily be observed that as omega decreases more and more from 1, the rate of convergence becomes slower and slower. It can also be observed that setting a large $\omega > 1$, such as 9.9 in Table 4, significantly accelerates the rate of convergence, this particular choice yields errors quite close to those in Table 5.2 in the book.

Question 6

Reproduce Table 5.3

In this question we continue our analysis of the system of nonlinear equations presented in the previous question (with ω=1). This time, we will compute the errors of the iterates from two different methods: Newton's Method and Broyden's Method. In both methods we again use the initial guess (y, z)=(-2,-2)

For Newton's Method we first compute the Jacobian at the initial guess and solve the following equation:

$$A_k * s^k = -f(x^k)$$

and then we update the guess with $x^{k+1} = x^k + s^k$ and repeat the process.

Again, Newton's Method generates a sequence that converges to the true equilibrium, this time much more rapidly than via Fixed-Point Iteration, as can be seen in Table 5, where the errors quickly converge to zero in fewer than 10 iterations. (Note that for y the equilibrium converges to -.1375 which seems to have been rounded in the book to -.137, hence why the errors for y do not go exactly to zero in both methods)

For Broyden's Method we similarly generate a sequence, this time updating the Jacobian as well via the rule:

$$A_{k+1} = A_k + \frac{(y_k - A_k s^k)(s^k)^T}{(s^k)^T s^k}$$

where $y_k = f(x^{k+1}) - f(x^k)$.

As can be observed in Table 5 below, Broyden's Method is slower in converging to the solution when compared to Newton's Method, with Broyden needing more iterations for the errors to converge to zero. However, Broyden's Method is still much faster than Fixed-Point Iteration. It is important to note that this was a small system of nonlinear equations, for large systems Broyden's Method may outperform Newton's Method in terms of speed.

Table 5. Errors of Newton and Broyden Methods

Iterate	*Newton*	*Broyden*
0	(-1.863, -1.424)	(-1.863, -1.424)
1	(0.54499, 0.28236)	(0.54499, 0.28236)
2	(-0.058537, 0.0089488)	(0.013679, 0.0064621)
3	(-0.00038048, 0.00089451)	(-0.0023365, 0.00049096)
4	(-0.00046516, 8.229e-05)	(-2.2934e-05, -0.0001634)
5	(-0.00046514, 8.1556e-05)	(-0.00049585, 9.8275e-05)
6	(-0.00046514, 8.1555e-05)	(-0.0004652, 8.158e-05)
7	(-0.00046514, 8.1555e-05)	(-0.00046515, 8.1556e-05)
10	(-0.00046514, 8.1555e-05)	(-0.00046514, 8.1555e-05)