

Answers to Problem Set 3

Group name: Ferienspass

Sebastian Kühnl: 5642348
Alexander Dück (as: reebyte): 5504077
Patrick Blank (as: paddyblank): 6729110
Christian Wierschem: 6729288

1 Question 1

Newton Optimization Algorithm

```
1 function [root,sol]=Newton_opt(fun,x_start,eps,delta,  
    maxiter)  
2 i=0;  
3 x=zeros(25,1);  
4 x(1)=x_start;  
5 %to get in while  
6 absdiff=1;  
7 check=0;  
8 %check input parameters  
9 if (eps<=0) || (delta<=0) || (maxiter<=0)  
10     disp('invalid input parameters')  
11     return;  
12 end  
13 %function at x_start already in optimum?  
14 syms a  
15 f=fun(a);  
16 grad=eval(subs(diff(f,a,1),a,x(1)));  
17 if (grad==0)  
18     root=x(1);  
19     return;  
20 end  
21 while (i<maxiter) && (absdiff>check)  
22     i=i+1;  
23     grad=eval(subs(diff(f,a,1),a,x(i)));  
24     he=eval(subs(diff(f,a,2),a,x(i)));  
25     x(i+1)=x(i)-he*grad;  
26     absdiff=abs(x(i)-x(i+1));  
27     check=eps*(1+abs(x(i+1)));  
28 end  
29 crit=delta*(1+abs(fun(x(i+1))));  
30 val=abs(eval(subs(diff(f,a,1),a,x(i+1))));  
31 if (i<maxiter) && (val<=crit)
```

```

32     disp('success');
33     sol=true;
34     root=x(i+1);
35 else
36     disp('failure');
37     sol=false;
38     root=-Inf;
39 end
40 disp(['number iterations: ',int2str(i)])
41 end

```

The functions:

```

1 function y=f1(x)
2 y=2.*x.^3-x.^2-3.*x+2;
3 end

```

```

1 function y=f2(x)
2 y=-x.*exp(-x);
3 end

```

2 Question 2

2.1 Subquestion 1

From the slides, we get the following conditions for the variables on the balanced growth path

$$s_k f(k^*, h^*) - (\delta_k + g + n + ng) k^* = 0 \quad (1)$$

$$s_h f(k^*, h^*) - (\delta_h + g + n + ng) h^* = 0. \quad (2)$$

solving for k^* and h^* , respectively, gives

$$k^* = \frac{s_k f(k^*, h^*)}{(\delta_k + g + n + ng)} \quad (3)$$

$$h^* = \frac{s_h f(k^*, h^*)}{(\delta_h + g + n + ng)}. \quad (4)$$

However, since $f(k^*, h^*)$ depends on the variables that we try to solve for, this is not the final form yet.

Since we are assuming $F(K_t, H_t, A_t L_t) = K_t^\alpha H_t^\beta (A_t L_t)^{1-\alpha-\beta}$, we can further solve the expression

$$f(k^*, h^*) = \frac{F(K_t, H_t, A_t L_t)}{A_t L_t} = \frac{K_t^\alpha H_t^\beta (A_t L_t)^{1-\alpha-\beta}}{A_t L_t} = \left(\frac{K_t}{A_t L_t} \right)^\alpha \left(\frac{H_t}{A_t L_t} \right)^\beta. \quad (5)$$

By definition of k^* and h^* , this is gives

$$f(k^*, h^*) = k_t^\alpha h_t^\beta. \quad (6)$$

Now, inserting this into 3 and 4

$$k^* = \frac{s_k k_t^\alpha h_t^\beta}{(\delta_k + g + n + ng)} \Leftrightarrow k^* = \left(\frac{s_k h_t^\beta}{(\delta_k + g + n + ng)} \right)^{\frac{1}{1-\alpha}} \quad (7)$$

$$h^* = \frac{s_h k_t^\alpha h_t^\beta}{(\delta_h + g + n + ng)} \Leftrightarrow h^* = \left(\frac{s_h k_t^\alpha}{(\delta_h + g + n + ng)} \right)^{\frac{1}{1-\beta}}. \quad (8)$$

It becomes evident, that k^* and h^* are a function of each other. However, we can simply substitute and then solve for the expression depending on parameters only

$$k^* = \left(\frac{s_k \left(\left(\frac{s_h k_t^\alpha}{(\delta_h + g + n + ng)} \right)^{\frac{1}{1-\beta}} \right)^\beta}{(\delta_k + g + n + ng)} \right)^{\frac{1}{1-\alpha}} \quad (9)$$

$$h^* = \left(\frac{s_h \left(\left(\frac{s_k h_t^\beta}{(\delta_k + g + n + ng)} \right)^{\frac{1}{1-\alpha}} \right)^\alpha}{(\delta_h + g + n + ng)} \right)^{\frac{1}{1-\beta}}. \quad (10)$$

which can now be solved for the respective variable. For k^* :

$$\begin{aligned} k^* &= \left(\frac{s_k (s_h k_t^\alpha)^{\frac{\beta}{1-\beta}}}{(\delta_h + g + n + ng)^{\frac{\beta}{1-\beta}} (\delta_k + g + n + ng)} \right)^{\frac{1}{1-\alpha}} \\ &= \left(\frac{s_k s_h^{\frac{\beta}{1-\beta}}}{(\delta_h + g + n + ng)^{\frac{\beta}{1-\beta}} (\delta_k + g + n + ng)} \right)^{\frac{1}{1-\alpha}} k_t^{\frac{\alpha\beta}{1-\beta-\alpha+\alpha\beta}} \\ &= \left(\frac{s_k s_h^{\frac{\beta}{1-\beta}}}{(\delta_h + g + n + ng)^{\frac{\beta}{1-\beta}} (\delta_k + g + n + ng)} \right)^{\frac{1-\beta-\alpha+\alpha\beta}{(1-\alpha)^2+\alpha\beta-\beta}} \end{aligned}$$

For h^* :

$$\begin{aligned}
h^* &= \left(\frac{s_h \left(s_k h_t^\beta \right)^{\frac{\alpha}{1-\alpha}}}{(\delta_k + g + n + ng)^{\frac{\alpha}{1-\alpha}} (\delta_h + g + n + ng)} \right)^{\frac{1}{1-\beta}} \\
&= \left(\frac{s_h s_k^{\frac{\alpha}{1-\alpha}}}{(\delta_k + g + n + ng)^{\frac{\alpha}{1-\alpha}} (\delta_h + g + n + ng)} \right)^{\frac{1}{1-\beta}} h_t^{\frac{\alpha\beta}{1-\beta-\alpha+\alpha\beta}} \\
&= \left(\frac{s_h s_k^{\frac{\alpha}{1-\alpha}}}{(\delta_k + g + n + ng)^{\frac{\alpha}{1-\alpha}} (\delta_h + g + n + ng)} \right)^{\frac{1-\beta-\alpha+\alpha\beta}{(1-\beta)^2+\alpha\beta-\alpha}}.
\end{aligned}$$

Thus, the solution vector becomes

$$\begin{bmatrix} k^* \\ h^* \end{bmatrix}' = \begin{bmatrix} \left(\frac{s_k s_h^{\frac{\beta}{1-\beta}}}{(\delta_h + g + n + ng)^{\frac{\beta}{1-\beta}} (\delta_k + g + n + ng)} \right)^{\frac{1-\beta-\alpha+\alpha\beta}{(1-\alpha)^2+\alpha\beta-\beta}} \\ \left(\frac{s_h s_k^{\frac{\alpha}{1-\alpha}}}{(\delta_k + g + n + ng)^{\frac{\alpha}{1-\alpha}} (\delta_h + g + n + ng)} \right)^{\frac{1-\beta-\alpha+\alpha\beta}{(1-\beta)^2+\alpha\beta-\alpha}} \end{bmatrix}'$$

2.6

All questions that lie between the first and this one are excluded from this document since they only involved programming exercises.

The path to be interpreted:

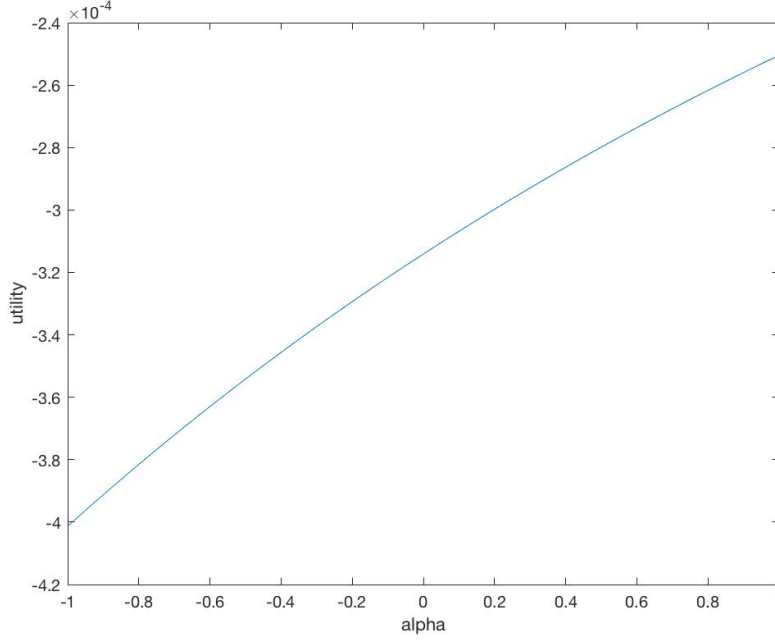


Figure 1: Paths of k^* and h^*

Both variables converge to the solution at the same time/speed. Optimal allocation human capital exceeds the optimal level of capital in all iterations.

A possible interpretation: That the balanced growth path level of human capital is greater than the balanced growth path level of physical capital could have two (simultaneous) reasons:

1. Human capital has a greater marginal product compared to physical capital
2. Human capital has a lower depreciation rate than physical capital

The reasoning behind this is that if an additional unit of human capital yields more than an additional unit of physical capital, and lasts longer than it as well, and investment in human capital is preferred over an investment in physical capital. This goes on this way until a point is reached, where an investment in human capital has the same marginal-product-to-depreciation-rate as physical capital. The investment in the two goods will increase such that their levels always give the same marginal-product-to-depreciation-rate (which implies a constant ratio of human to physical capital (which can be seen in the graph as well)) until the entire income is spent. At this point, the balanced growth path level is reached.

3 Question 3

3.1

To begin with, recall two properties of the logistic function characterising y_n :

$$y(x) := \Lambda(x) = \frac{1}{1 + \exp(-x)} = \frac{\exp(x)}{1 + \exp(x)} \quad (*)$$

Applying the quotient rule, yields the partial derivative w.r.t. x

$$\begin{aligned} \lambda(x) &= \frac{\exp(x)}{(1 + \exp(x))^2} \\ &= \frac{\exp(x)}{1 + \exp(x)} \frac{1}{1 + \exp(x)} \\ &= \Lambda(x)\Lambda(-x) && \text{(using (*))} \\ &= \Lambda(x)(1 - \Lambda(x)) && \text{(by symmetry around zero)} \\ &= y(x)(1 - y(x)) && (**) \end{aligned}$$

The log-likelihood function is given by

$$\mathbb{E}(\beta) = - \sum_{n=1}^N t_n \log(y_n) + (1 - t_n) \log(1 - y_n)$$

Accordingly, the first order condition w.r.t. coefficient β_i is given by

$$\begin{aligned} \frac{\partial \mathbb{E}(\beta)}{\partial \beta_i} &= - \sum_{n=1}^N \frac{(t_n - \Lambda(X'_n \beta)) \lambda(X'_n \beta) X_{ni}}{\Lambda(X'_n \beta) (1 - \Lambda(X'_n \beta))} && \text{(for } i = 1, \dots, K) \\ &= - \sum_{n=1}^N (t_n - \Lambda(X'_n \beta)) X_{ni} && \text{(using (**))} \\ &= \sum_{n=1}^N (y_n - t_n) X_{ni} && \text{(by definition of } y_n) \\ &= 0 && \text{(for } i = 1, \dots, K) \end{aligned}$$

Putting these K likelihood equations together yields the result

$$\begin{aligned} \nabla \mathbb{E}(\beta) &= \sum_{n=1}^N (y_n - t_n) X'_n \\ &= (y - t)' X \end{aligned}$$

In order to characterise the Hessian, take derivative w.r.t. β_j

$$\begin{aligned}\frac{\partial \mathbb{E}(\beta)}{\partial \beta_i \partial \beta_j} &= \sum_{n=1}^N \lambda(X'_n \beta) X_{ni} X_{nj} && (\text{for any } i, j = 1, \dots, K) \\ &= \sum_{n=1}^N \Lambda(X'_n \beta) (1 - \Lambda(X'_n \beta)) X_{ni} X_{nj} \\ &= \sum_{n=1}^N y_n (1 - y_n) X_{ni} X_{nj} && (\text{by definition of } y_n)\end{aligned}$$

which represents the (i, j) th element of the Hessian. In summary, we have

$$\begin{aligned}H &= \sum_{n=1}^N y_n (1 - y_n) X_n X'_n \\ &= X' R X\end{aligned}$$

3.2

It is well known that the log-likelihood function is globally concave in β because any contribution n is globally concave in the parameters and the sum of concave functions is again concave. Multiplying by (-1) yields a globally convex function in β . Moreover, any real number is feasible for β_i and the set of real numbers is convex. To sum up, a globally convex function is minimised over a convex set, i.e., we face a convex minimisation problem. If a local minimum exists, then it is also a global minimum point. It furthermore implies that the likelihood equations uniquely define β (except for a very special case).

3.3

3.4

Masterfile for the following subquestions:

```
1 %% Problem set 3; problem 3: logit estimation
2 %% 3.3
3 % Load data set
4 filename = 'MRW92QJE-data.xlsx';
5 sheet = 1;
6 %[num, txt, raw] = xlsread(filename, sheet);
7 OECDdummy = xlsread(filename, sheet, 'E2:E122');
8 gdp85pad = xlsread(filename, sheet, 'G2:G122');
9 school = xlsread(filename, sheet, 'K2:K122');
10 X = [OECDdummy, gdp85pad, school];
11 nrow0 = size(X,1);
12 % Delete rows with NaN-values
13 X(any(isnan(X), 2), :) = [];
14 nrow1 = size(X,1);
```

```

15 diffrow = nrow0 - nrow1;
16 fprintf( 'N.b.: %g observations deleted due to missing
    values! \n', diffrow);
17
18 %% 3.4
19 % Dependent variable
20 t = X(:,1);
21 % Independent variables
22 const = ones(nrow1,1);
23 lgdp85pad = log(X(:,2));
24 lschool = log(X(:,3)/100);
25 X = [const, lgdp85pad, lschool];
26
27 %% 3.5
28 beta0 = zeros(3,1);
29 maxiter = 200;
30 tol = 0.00001;
31 % (a) Newton
32 [beta, iter, t] = mynewton(t, X, beta0, maxiter, tol);
33 beta
34 iter
35 t
36
37 % (b) Broyden's method
38 [beta, iter, t] = broyden(t, X, beta0, maxiter, tol);
39 beta
40 iter
41 t
42
43 % (c) Fixed point iteration
44 [beta, iter, t] = fixedpoint(t, X, beta0, maxiter, tol
    );
45 beta
46 iter
47 t
48
49 % (d) fsolve
50
51 %% 3.6 see 3.5 (b)
52
53 %% 3.7 Standardization
54 M = max(X);
55 m = min(X);
56 maxmin = M - m;
57
58 for j = 1:nrow1

```



```

59     % Standardize lgdp85adult
60     X(j,2) = ( X(j,2) - m(2) )/maxmin(2);
61     % Standardize lschool
62     X(j,3) = ( X(j,3) - m(3) )/maxmin(3);
63 end
64
65 beta0 = [-10; 10; 1];
66 [beta, iter, t] = broyden(t, X, beta0, maxiter, tol);
67 beta
68 iter
69 t
70
71 %% 3.8
72 % (a) Newton's method
73 [beta, iter, t] = broyden(t, X, beta0, maxiter, tol);
74 beta
75 iter
76 t
77
78 %% 3.9 tic-toc is already implemented above

```

3.5

1. Newton's method converges to $\beta = (-19.9159 \quad 2.7321 \quad 2.0589)'$ after 9 iterations and 0.0142 seconds.
2. Broyden's method does not converge.
3. For any dampening factor $\lambda > 0$, the fixed point iteration did not converge.

Matlab Code

Newton's Algorithm:

```

1 function [beta, iter, t] = mynewton( t, X, beta0,
    maxiter, tol )
2 % Newton's method: algorithm using exact Hessian
3 % function [beta, iter] = newton(t, X, beta0, maxiter,
    tol)
4
5 %% (0) Initialise
6 tic
7 iter = 0;
8 beta = beta0;
9 exit_flag = 0;
10
11 %% (1) Compute gradient and Hessian, then iterate

```

```

12 for j = 1:maxiter
13     iter = j;
14     y = sigmoid(X*beta);
15     grad = X'*(y - t);
16     h1 = y.*(1-y);
17     R = diag(h1);
18     Hess = X'*R*X;
19     iHess = inv(Hess);
20     p = - iHess*grad;
21     % For clarity, display iteration steps:
22     fprintf( 'Step made in iteration %g: \n', iter )
23     p'
24     beta_new = beta + p;
25     %% (2) Check stopping criterion
26     check = ( norm( p ) <= tol*( 1+norm(beta_new) ) &&
                norm(grad) <= tol );
27     if check == 1
28         exit_flag = 1; break;
29     else
30         beta = beta_new;
31     end
32 end
33 %% (3) Report success or failure
34 if exit_flag == 0
35     fprintf( 'Algorithm has not converged after %g
                iterations\n That is bad luck, my friend!\n',
                maxiter)
36 end
37 toc
38 t = toc;

```

Broyden's Algorithm:

```

1 function [beta, iter, t] = broyden(t, X, beta0,
    maxiter, tol)
2 % Broyden's algorithm: Quasi-Newton method using
    approximated Hessian
3 % function [beta, iter] = broyden(t, X, beta0, maxiter
    , tol)
4
5 %% (0) Initialise
6 tic
7 iter = 0;
8 beta = beta0;
9 % Evaluate likelihood equations for first guess
10 y = sigmoid(X*beta);

```

```

11 grad = X'*(y - t);
12 % Trivial initial guess for Hessian: I(K)
13 ncol = size(X,2);
14 Hess = eye(ncol);
15 Hess = inv(Hess); % Calculate inverse for first
    iteration
16
17 exit_flag = 0;
18 %% (1) Compute next iterate
19 for j = 1:maxiter
20     iter = j;
21     p = - Hess*grad;
22     beta_new = beta + p;
23
24     %% (2) Update approximate Hessian
25     dbeta = beta_new - beta;
26     y = sigmoid(X*beta_new);
27     grad_new = X'*(y - t);
28     dgrad = grad_new - grad;
29     a = dgrad - Hess*dbeta;
30     b = dbeta'*dbeta;
31     c = a*dbeta';
32     Hess = Hess + c/b;
33     condition = cond(Hess);
34     fprintf( 'The condition number of the Jacobian in
        iteration %g \n is %.4f \n', iter, condition)
35     %% (3) Check stopping criterion
36     check = ( norm( p ) <= tol*( 1+norm(beta_new) ) &&
        norm(grad_new) <= tol );
37     beta = beta_new;
38     if check == 1
39         exit_flag = 1; break;
40     else
41         grad = grad_new;
42     end
43 end
44 %% (4) Report success or failure
45 if exit_flag == 0
46     fprintf( 'Algorithm has not converged after %g
        iterations\n That is bad luck, my friend! \n',
        maxiter)
47 end
48 toc
49 t = toc;

```

Fixed Point Algorithm:

```

1 function [beta, iter, t] = fixedpoint( t, X, beta0,
    maxiter, tol)
2 % Fixed point iteration <=>: gradient method/ steepest
    descent
3 % lambda is not obtained via linesearch but fixed at
    the beginning
4 % function [beta, iter] = fixedpoint( t, X, beta0,
    maxiter, tol, lambda0)
5
6 %% (0) Initialise
7 tic
8 iter = 0;
9 beta = beta0;
10 exit_flag = 0;
11 lambda = input( 'Please insert a dampening factor
    lambda: ' );
12
13 %% (1) Compute gradient and Hessian, then iterate
14 for j = 1:maxiter
15     iter = j;
16     y = sigmoid(X*beta);
17     grad = X'*(y - t);
18     p = - lambda*grad;
19     % For clarity, display iteration steps:
20     fprintf( 'Step made in iteration %g: \n', iter )
21     p'
22     beta_new = beta + p;
23
24     %% (2) Check stopping criterion
25     check = ( norm( p ) <= tol*( 1+norm(beta_new) ) &&
        norm(grad) <= tol );
26     if check == 1
27         exit_flag = 1; break;
28     else
29         beta = beta_new;
30     end
31 end
32 %% (3) Report success or failure
33 if exit_flag == 0
34     fprintf( 'Algorithm has not converged after %g
        iterations\n That is bad luck, my friend!\n',
        maxiter)
35 end
36 toc
37 t = toc;

```

Sigmoid:

```
1 function y = sigmoid( z )  
2 y = 1 ./ ( 1 + exp(-z) );
```

3.6

The function `broyden.m` used in the previous question returns the condition number of the Jacobian at every iteration. The condition number is consistently extremely large. Large values indicate that the problem is ill-conditioned. This is why the iteration did not converge.

3.7

Standardisation significantly reduces the condition of the Jacobian. Unfortunately, the condition is still far from being optimal and the Broyden's method does not converge again.

3.8

Unfortunately, the Newton's method did not converge again.

3.9

Newton's method is the most time-consuming algorithm since it requires to evaluate and invert the Hessian at every iteration. Unfortunately, other algorithms often failed to converge. Theoretically, Broyden's method is faster because only one inversion is necessary. Afterwards, it is possible to approximate the Hessian using information already available. However, Broyden's method is more imprecise and, therefore, requires more iterations for convergence than Quasi-Newton methods that directly update the inverse of the Hessian such as BFGS and DGP.

4 Question 4

4.1 Analytical Solution

From the Euler equation and the budget constraint we get that

$$c_t = (\beta(1+r))^{-\frac{1}{\theta}} c_{t+1} \quad (11)$$

$$\sum c_t \left(\frac{1}{1+r} \right)^t = \sum w_t \left(\frac{1}{1+r} \right)^t + a_0 \quad (12)$$

Since we can express all c_t as a function of c_0 , we can write

$$c_0 \sum \left(\beta^{-\frac{1}{\theta}} (1+r)^{\frac{\theta-1}{\theta}} \right)^t = \sum w_t \left(\frac{1}{1+r} \right)^t + a_0 \quad (13)$$

which allows to compute all c_t after solving for c_0

$$c_0 = \frac{\sum w_t \left(\frac{1}{1+r} \right)^t + a_0}{\sum \left(\beta^{-\frac{1}{\theta}} (1+r)^{\frac{\theta-1}{\theta}} \right)^t} \quad (14)$$

$$c_{t+1} = c_t (\beta(1+r))^{\frac{1}{\theta}}. \quad (15)$$

$$(16)$$

This can now be implemented into an algorithm by using a for-loop.

4.2 Algorithm

```

1 %PS3 Problem 4
2 %No T is given, so declare it at the beginning of the
   program, and for any
3 %theta
4 clear;
5 close all;
6 clc;
7 T=20;
8 theta=1;
9 %variable initialization (fixed)
10 time=linspace(0,T,T+1); %just for plot
11 w=zeros(T+1,1);
12 w(1)=10;
13 beta=0.99;
14 r=0.05;
15 %these variables will be determined
16 a=zeros(T+1,1);
17 c=zeros(T+1,1);
18 %computation (could also be written as a function)
19 num=zeros(T+1,1);
20 denom=zeros(T+1,1);
21 if theta==1
22     factor=beta*(1+r);
23 else
24     factor=(1+r)^((1-theta)/theta)*beta^(1/theta);
25 end
26 for i=0:T
27     num(i+1)=w(i+1)*(1+r)^(-i);
28     denom(i+1)=(factor)^i;
29 end
30 c(1)=sum(num)/sum(denom);
31 for i=1:T
32     a(i+1)=a(i)*(1+r)+w(i)-c(i);

```

```

33     c(i+1)=c(i)*(beta*(1+r))^(1/theta);
34 end
35 figure
36 plot(time,c,time,w,time,a);
37 legend('C','W','A')
38 acheck=a(end);
39 ccheck=c(end);
40 if (round(ccheck*1000)/1000==round(acheck*(1+r)*1000)
    /1000)
41     display('True --> a_{T+1}= 0')
42 elseif (round(ccheck*1000)/1000>round(acheck*(1+r)
    *1000)/1000)
43     display('a_{T+1}<0')
44     disc=a(end);
45     subst=zeros(T+1,1);
46     for i=0:T
47         subst(i+1)=(1+r)^i*((beta*(1+r))^(i/theta))^T-i;
48     end
49     d=sum(subst);
50     c(1)=c(1)-d;
51     for i=1:T
52         c(i+1)=c(i)*(beta*(1+r))^(i/theta);
53         a(i+1)=a(i)*(1+r)+w(i)-c(i);
54     end
55 else
56     display('error, a_{T+1}>0')
57 end

```

4.3 What to do if $\theta = 1$

in the case that $\theta = 1$, we get

$$\frac{c_t^{1-\theta}-1}{1-\theta} = \frac{1-1}{0} = \frac{0}{0}.$$

In this case, an application of L'Hôpital's rule has to be applied

$$\frac{f(x)}{g(x)} = \frac{\left(\frac{\partial f(x)}{\partial x}\right)}{\left(\frac{\partial g(x)}{\partial x}\right)}.$$

We now have to use

$$f(\theta) = c_t^{1-\theta} - 1 \tag{17}$$

$$g(\theta) = 1 - \theta \tag{18}$$

giving

$$\frac{\partial f(\theta)}{\partial \theta} = (-1) \ln(c_t) \exp((1 - \theta) \ln(c_t)) \quad (19)$$

$$\frac{\partial g(\theta)}{\partial \theta} = -1 \quad (20)$$

$$\frac{\left(\frac{\partial f(x)}{\partial x}\right)}{\left(\frac{\partial g(x)}{\partial x}\right)} = \ln(c_t) \exp((1 - \theta) \ln(c_t)) \quad (21)$$

in the limit

$$\lim_{\theta \rightarrow 1} \frac{\left(\frac{\partial f(x)}{\partial x}\right)}{\left(\frac{\partial g(x)}{\partial x}\right)} = \ln(c_t) \quad (22)$$

which can then be used in the maximization problem to obtain a new Euler equation.

5 Question 5

1. First of all, let us short notation in terms of $r_L := r_{low}$ and $r_H := r_{high}$. In order to check concavity of the objective function, consider its SOC:

$$-\gamma p(w_0(1+r^f+\alpha(r_L-r^f)))^{-(1+\gamma)}(r_L-r^f)^2 - \gamma(1-p)(w_0(1+r^f+\alpha(r_H-r^f)))^{-(1+\gamma)}(r_H-r^f)^2$$

Obviously its sign is ambiguous. Therefore, even though the (two) inequality constraints are (weakly) concave and no equality constraints arise, the first "convexity-condition" is not satisfied generally. That is, KTK-conditions are just necessary (as usual) but not sufficient for an optimum.

2. a)

FOC:

$$p(w_0(1+r^f+\alpha(r_L-r^f)))^{-\gamma}(r_L-r^f) + (1-p)(w_0(1+r^f+\alpha(r_H-r^f)))^{-\gamma}(r_H-r^f) = 0 \quad (1)$$

Assume the opposite, i.e., that the optimal portfolio share α^* (the value of α for which the equation above holds) depends on initial wealth w_0 . Denote this maximizer contingent on w_0 by $\alpha(w_0)$. As the optimal portfolio share responds on w_0 , the first order derivative of the LHS of (1) (setting $\alpha = \alpha(w_0)$) with respect to wealth should be equal zero. Namely, the total differentiation of the FOC w.r.t. α and w_0 :

$$\begin{aligned} \frac{dE u(w_1)}{d\alpha dw_0} = & -\gamma w_0^{-(1+\gamma)} p(1+r^f+\alpha(r_L-r^f))^{-\gamma}(r_L-r^f) + w_0^{-\gamma} p \alpha'(w_0)(r_L-r^f)^2 \\ & - \gamma w_0^{-(1+\gamma)} (1-p)(1+r^f+\alpha(r_H-r^f))^{-\gamma}(r_H-r^f) + w_0^{-\gamma} p \alpha'(w_0)(r_H-r^f)^2 = 0. \end{aligned}$$

Clearly, this equality can be rearranged as follows:

$$\begin{aligned} & w_0^{-\gamma} p \alpha'(w_0) (r_L - r^f)^2 + w_0^{-\gamma} p \alpha'(w_0) (r_H - r^f)^2 \\ &= \frac{1}{\gamma} w_0^{-(1+\gamma)} p (1 + r^f + \alpha(r_L - r^f))^{-\gamma} (r_L - r^f) + \frac{1}{\gamma} w_0^{-(1+\gamma)} (1-p) (1 + r^f + \alpha(r_H - r^f))^{-\gamma} (r_H - r^f) \end{aligned}$$

and now, the RHS of this equality corresponds to the FOC multiplied by γw_0^{-1} which has to be still equal zero. Thus, dividing the equality by the LHS (except $\alpha'(w_0)$) yields

$$\alpha'(w_0) = \frac{0}{w_0^{-\gamma} p (r_L - r^f)^2 + w_0^{-\gamma} p (r_H - r^f)^2} = 0$$

which contradicts that the optimal portfolio share depends on w_0 .

Alternatively, notice that the initial objective (which we aim to maximize) can be divided by w_0^ϕ and thus, wealth cancels out. Clearly, the optimal portfolio shares for the old resp. the new problem, will coincide.

b) The associated FOC becomes:

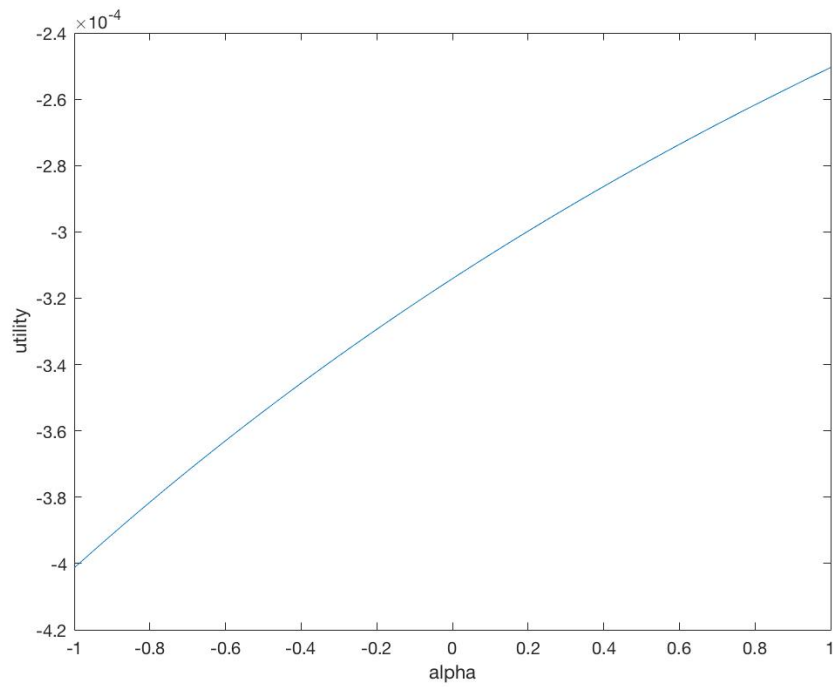
$$p(1 + r^f + \alpha(r_L - r^f))^{\phi-1} (r_L - r^f) + (1-p)(1 + r^f + \alpha(r_H - r^f))^{\phi-1} (r_H - r^f) = 0 \quad (2)$$

using $(r_L - r^f) = -(r_H - r^f) = -0.1$ and rearranging we obtain

$$p^{\frac{1}{\phi-1}} (1 + r^f + \alpha(-0.1)) = (1-p)^{\frac{1}{\phi-1}} (1 + r^f + \alpha 0.1) \quad (3)$$

$$\Leftrightarrow \alpha = 10(1 + r^f) \frac{p^{\frac{1}{\phi-1}} - (1-p)^{\frac{1}{\phi-1}}}{p^{\frac{1}{\phi-1}} + (1-p)^{\frac{1}{\phi-1}}} = 2.73308... \quad (4)$$

evaluated at $p = 0.1$, $r^f = 0.02$ and $\phi = -3$.



3. a) Intuitively , $\alpha \geq 0$ prevents the case where the household supplies the portfolio and $\alpha \leq 1$ rules out that the household borrows money in order to invest more in the portfolio.

b)

```
1 clear;
2 close all;
3
4 % Variables
5
6 phi=-3;
7 prob_low=0.1;
8 prob_high=1-prob_low;
9 alpha_upper=1;
10 alpha_lower=0;
11 w_init=10;
12 r_f=0.02;
13 r_high=0.12;
14 r_low=-0.08;
15
16 % Apply fminbnd
17 fun= @(alpha) (-utility(alpha,phi,prob_low,prob_high,
18     w_init,r_f,r_high,r_low));
19 alpha_fminbnd = fminbnd(fun,alpha_lower,alpha_upper);
20
21 % Apply fmincon
22 A=[];
23 b=[];
24 Aeq=[];
25 beq=[];
26 lb=alpha_lower;
27 ub=alpha_upper;
28 alpha_fmincon = fmincon(fun,0,A,b,Aeq,beq,lb,ub);
29
30 % Plot Utility and Marginal Utility
31 alpha_v=nan(100,1);
32 a=nan(100,1);
33 a_marg=nan(100,1);
34
35 for i=1:100
36     a(i,1)=utility(i/100,phi,prob_low,prob_high,w_init,
37         r_f,r_high,r_low);
38     a_marg(i,1)=marginal_utility(i/100,phi,prob_low,
39         prob_high,w_init,r_f,r_high,r_low);
40     alpha_v(i,1)=i/100;
41 end
42 figure
```

```

41 subplot(2,1,1); plot(alpha_v,a);
42 hline=refline((a(100,1)-a(1,1)),a(1,1));
43 hline.Color = 'r';
44 xlabel('alpha');
45 ylabel('utility');
46 legend('utility','Location','northwest');
47 subplot(2,1,2); plot(alpha_v,a_marg);
48 hline=refline((a_marg(100,1)-a_marg(1,1)),a_marg(1,1))
49 ;
49 hline.Color = 'r';
50 xlabel('alpha');
51 ylabel('marginal utility');
52 legend('marginal utility','Location','northeast');
53
54 disp(['fminbnd solution at alpha=', num2str(
    alpha_fminbnd), '. fmincon solution at alpha=',
    num2str(alpha_fmincon)]);
55
56 % Define functions for utility and marginal utility
57 function util=utility(alpha,phi,prob_low,prob_high,
    w_init,r_f,r_high,r_low)
58 r=prob_high*r_high+prob_low*r_low;
59 util=(1/phi)*(w_init*(1+r_f+alpha*(r-r_f)))^(phi);
60 end
61
62 function mutil=marginal_utility(alpha,phi,prob_low,
    prob_high,w_init,r_f,r_high,r_low)
63 r=prob_high*r_high+prob_low*r_low;
64 mutil=(w_init*(r-r_f))/((w_init*(1+r_f+alpha*(r-r_f)))
    ^(1-phi));
65 end

```

The fminbnd solution is at $\alpha=0.99993$. The first fmincon solution, starting from the upper boundary is at $\alpha=0.99828$. The second fmincon solution, starting from the lower boundary is at $\alpha=0.99767$.

The two methods can yield different results because

1. The way the constraints are defined differ. In fmincon, they can hold with equality, in fminbnd they have to hold with strict inequality.
2. The way the algorithm searches for a minimum. fmincon starts at some point x_0 and moves along the x -axis while evaluating the function. fminbnd searches for a local minimum on the entire interval.

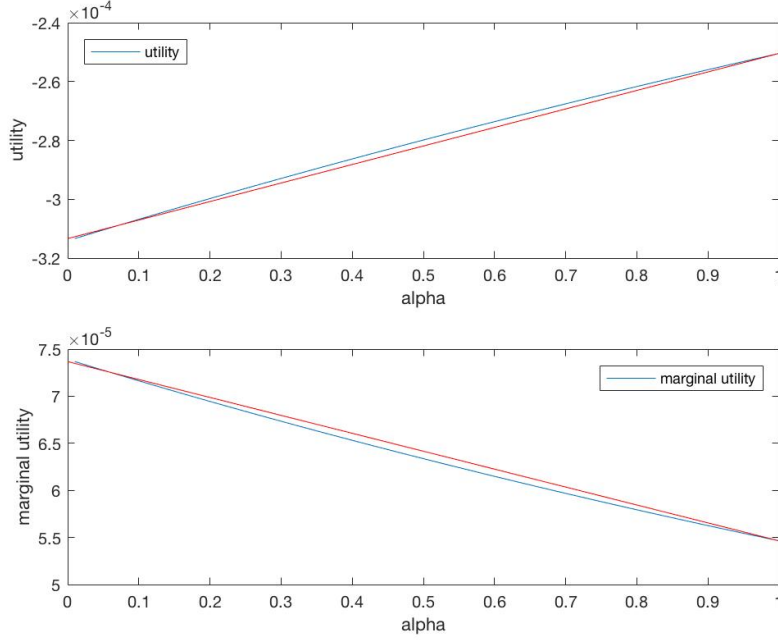


Figure 2: Illustration of the utility and marginal utility, depending on α . (The red line is a reference line with linear slope)

Comment: After running the algorithm for the first two times, the solutions produced by `fmincon` were $= 1$. The difference between the results from the two algorithms was then easily explainable by the difference in the interval constraints mentioned above ($lb \leq \alpha \leq ub$ (`fmincon`) vs $lb < \alpha < ub$ (`fminbnd`)).

c)

Part two of this question resulted in $\alpha = 2.73308$. However, the solution in part three was bounded on the upside at $\bar{\alpha} = 1$. The solution, derived numerically, resulted in α close to one (and one the first tries with `fmincon` resulted in $\alpha = 1$, exactly the boundary). If one runs the maximization without the upper constrain, the argmax increases with the starting point (for `fmincon` (that the solution does not converge to infinity can be explained as the differences in function value lie below machine precision which results in a maximum being reported)). This implies that the household would like to borrow at the risk free rate (negative α^f) to invest more in stocks to increase his expected portfolio return.

Since $\phi = 1 - \gamma$ and $\phi = -3$, it can be implied that $\gamma = 4$. Thus, the agent is risk averse. However, one would expect a risk averse agent not to borrow to invest into the risky asset.

Looking at the utility function with the parameters specified in the question, $u(w_1) = \frac{\left(\frac{1}{w_1^3}\right)}{-3}$, it becomes obvious that with increasing wealth, the utility moves closer to zero from below as wealth increases. Since the probability to receive a high interest rate is nine times as large as the probability to receive a low interest rate, the expected return from the risky asset is much larger than the return of the safe asset.

Since the maximization problem can be rewritten as

$$\max_{\alpha} \underbrace{\frac{1}{\phi}}_{<0} \left(\underbrace{w_0 (1 + r^f + \alpha (E[r] - r^f))}_{>0} \right) \underbrace{\phi}_{<0} \quad (23)$$

which explains that α exceeds 1 since there is no more uncertainty in the model.