

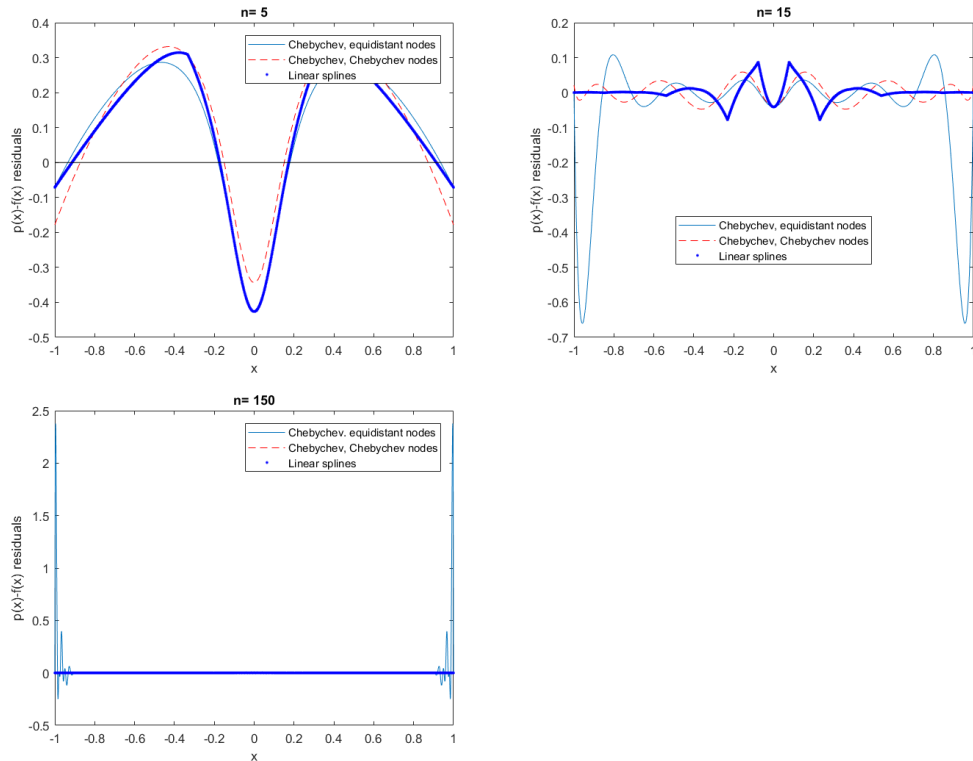
Answers to Problem Set 4

Group name: Ferienspass

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

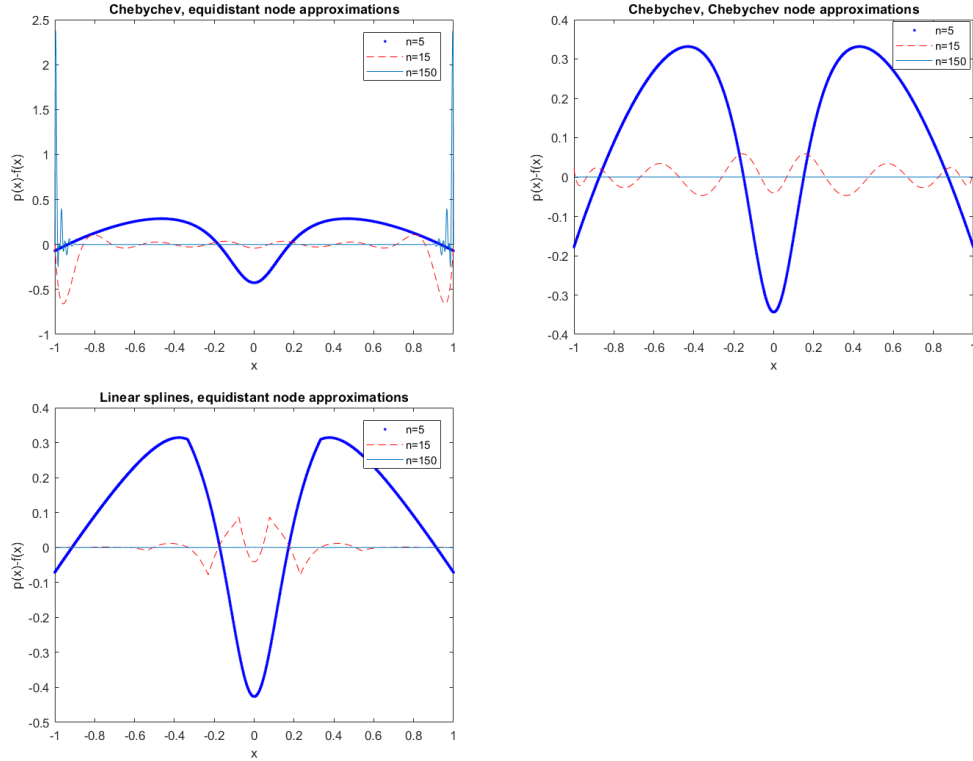
1 Question 1

Chebyshev approximation using equidistant nodes and Chebyshev nodes. However, there is a difference between lecture slides 7 and the notes from the lecture, as you will see in the code provided below. Plots are ordered in chronological order! (For comparison, the residuals to actual function are plotted.)

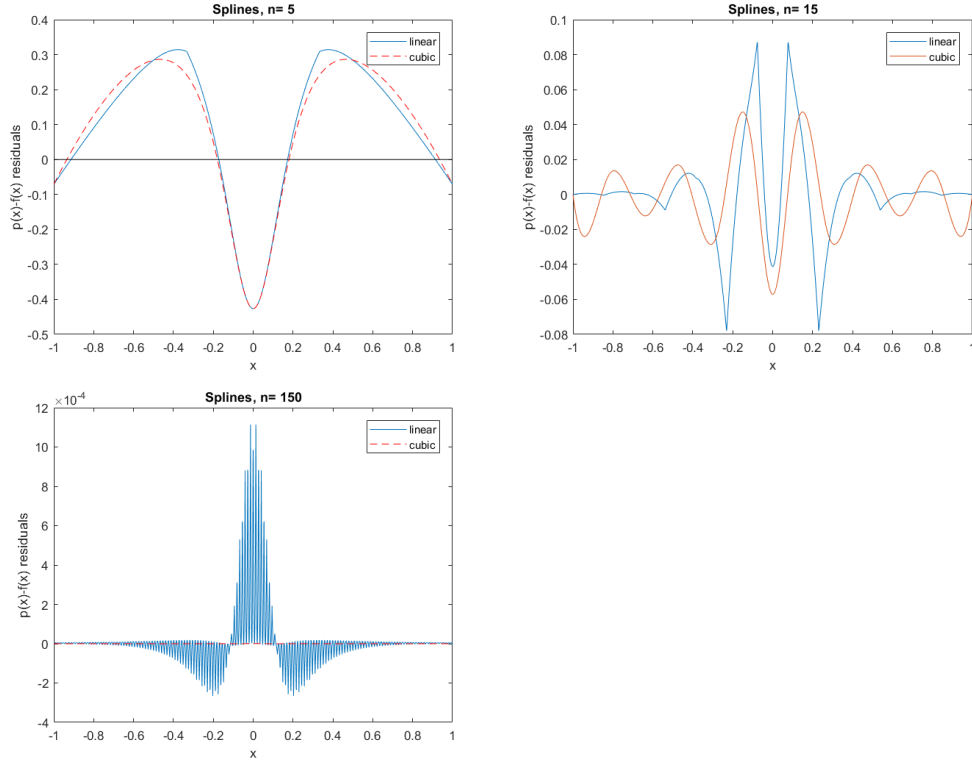


For $n=5$, equidistant nodes and Chebyshev nodes as well as linear splines are very similar. For $n=15$, linear splines become more edgy. It performs well at the edges and average else. Both Chebyshev approximations are very similar

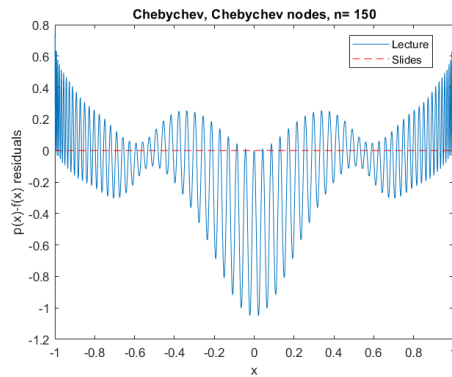
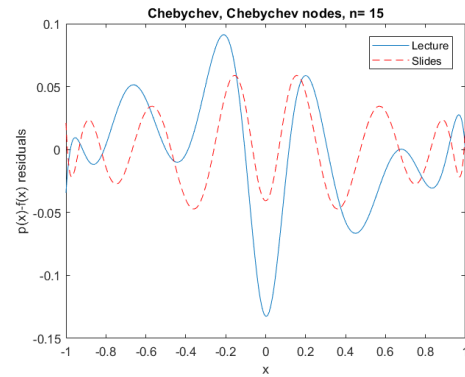
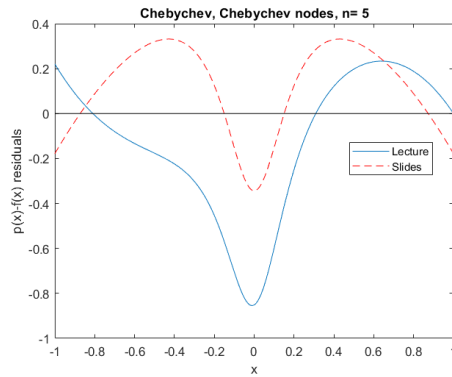
in $[-0.75; 0.75]$, while equidistant nodes fall off at the edges (as expected). For $n=150$ this effect is even stronger, but it moves closer to the corner. The others are not comparable due to the residual scale. The effect does not occur when using Chebyshev nodes because there are more nodes at the corner to prevent these large fluctuations.



In this figure again the effect of equidistant nodes when using Chebyshev can be seen as large fluctuations at the corner. Besides this, as the number of nodes increases, the approximation gets closer to the real function.



Linear and cubic splines are very similar when $n=5$. For $n=15$ one can observe that cubic splines are smoother than linear splines and perform better in the center (around 0), whereas linear splines perform better at the corner (it becomes smooth and then becomes nearly a straight line). For $n=150$, at first sight, linear splines perform badly, but it is only relative to cubic splines (look at the scale). As n increases, the approximation gets better when using splines.



The slides formula seems to be the right one. The function is symmetric and so is the approximation. The lecture formula leads to very odd (i.e. asymmetric) approximations.

In general, it seems to be very odd that the residuals at 0 are not 0, because there should be a node and thus the residual should be zero. Maybe it is due to the toolbox calculations. Other possibilities have been thought of and precluded.

2 Question 2

The first order condition of the unconstrained maximisation problem is given by

$$u'(C_0) - \mathbb{E}u'(W_0(1+r) - C_0) = 0$$

Accordingly, the optimal consumption plan obeys the Euler equation

$$u'(C_0) = \mathbb{E}u'(C_1) \quad (\text{Euler EQ})$$

Quadratic utility Let the utility function be quadratic. Then, marginal utility is given by

$$u'(C_t) = -(C_t - \bar{C}) = \bar{C} - C_t \quad (\text{Marginal utility})$$

Moreover,

$$u''(C_t) = -1 < 0 \quad (\text{Risk aversion})$$

$$u'''(C_t) = 0 \quad (\text{Prudence})$$

In order to obtain the optimal consumption, plug the marginal utility into the Euler equation

$$\begin{aligned} \bar{C} - C_0 &= \mathbb{E}(\bar{C} - C_1) \\ \bar{C} - C_0 &= \mathbb{E}(\bar{C} - (W_0(1+r) - C_0)) \\ 2C_0 &= W_0\mathbb{E}(1+r) \\ C_0 &= \frac{1}{2}W_0\mathbb{E}(1+r) \end{aligned}$$

Note that marginal utility is linear in C_t . Consequently, we could exploit linearity of the expectation operator which yields more generally

$$\mathbb{E}(u'(C_t)) = u'(\mathbb{E}(C_t))$$

This is why certainty equivalence holds, i.e., the intertemporal consumption decision remains unchanged when agents are exposed to more or even less uncertainty. Indeed, expected lifetime utility is reduced by income risks (concave utility). However, the comparative statics require to look at the third derivative which indicates that agents are not influenced by the degree of income uncertainty. In case of linear marginal utility agents are not prudent. It is quite hard to judge whether this result makes economic sense, i.e., such a function provides a meaningful utility representation. There is a lot of empirical work on the willingness to insure, it is true, but prudence is a different issue. If we believe in the precautionary savings motive (which makes intuitively sense), quadratic utility is inappropriate.

CRRA utility In case of CRRA utility the three derivatives are given by

$$u'(C_t) = C_t^{-\gamma} \quad \text{for any } \gamma \neq 1 \quad (\text{Marginal utility})$$

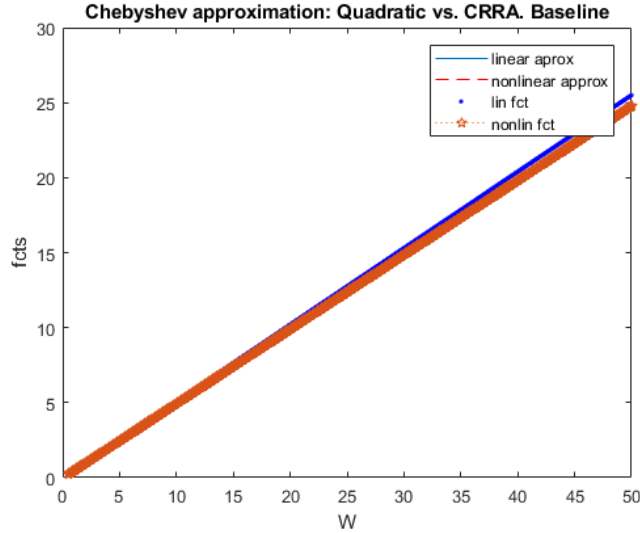
$$u''(C_t) = -\gamma C_t^{-(\gamma+1)} < 0 \quad (\text{Risk aversion})$$

$$u'''(C_t) = \gamma(1+\gamma)C_t^{-(\gamma+2)} > 0 \quad (\text{Prudence})$$

The last two derivatives tell us that marginal utility is strictly convex. Therefore, the agent is prudent. If agents are exposed to higher income uncertainty (i.e., higher variance in r) precautionary savings reduce present consumption. These savings allow them to prepare for the possibility of more severe income states.

Apparently, $\mathbb{E}(u'(C_t)) = u'(\mathbb{E}(C_t))$ will no longer hold. In order to derive the optimal consumption, plug the first derivative into the Euler equation:

$$\begin{aligned} C_0^{-\gamma} &= \mathbb{E}(C_1^{-\gamma}) \\ &= \mathbb{E}((W_0(1+r) - C_0)^{-\gamma}) \\ \Rightarrow C_0 &= \mathbb{E}[(W_0(1+r) - C_0)^{-\gamma}]^{-\frac{1}{\gamma}} \end{aligned}$$



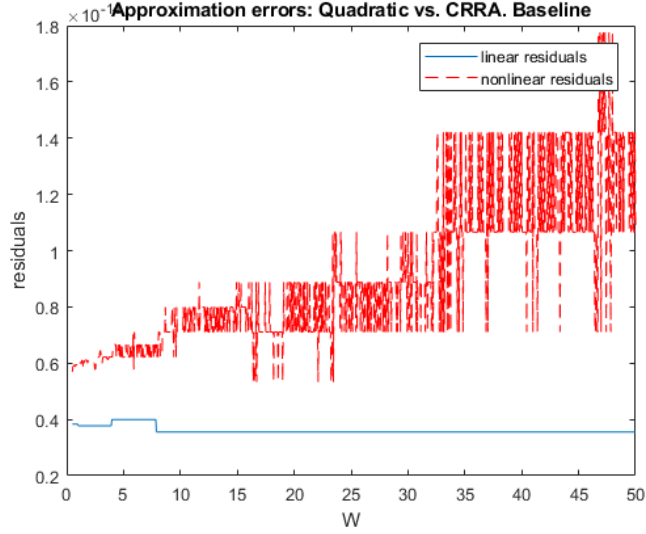


Table 1: Maximum percentage errors of deviation

	Setting	Example	MPE of deviation
	Baseline		2.75
(i)	Higher risk aversion γ	$\gamma = 4$	4.22
(ii)	Lower probability p	$p = 0.20$	3.55
(iii)	Higher mean-preserving interest rate spread	+0.40	19.17

The table shows that the maximum percentage error of deviation increases for all modifications (i)-(iii) compared to the baseline model. In particular, the error of deviation significantly increases if agents face higher spreads. However, it is naturally impossible to compare these change quantitatively since we plugged in some arbitrary numbers to mimic the new setting. Different values will produce different errors, but we can safely say that errors of deviation generally increase.

3 Question 3

3.1

Using $p := p_l$ and therefore $1 - p = p_h$ the first order condition of agent i becomes

$$\frac{1-\gamma_i}{1-\gamma_i} [p(1+r^f+\alpha(r_L-r^f))^{-\gamma_i}(r_L-r^f)+(1-p)(1+r^f+\alpha(r_H-r^f))^{-\gamma_i}(r_H-r^f)] = 0 \quad (1)$$

$$\Leftrightarrow E[(1+r^f+\alpha(r-r^f))^{-\gamma_i}(r-r^f)] = 0 \quad (2)$$

3.2 Analytical Solution of α_i

The equation has then been converted into the form $\alpha(\gamma_i)$, already using the presented calibration.

$$\begin{aligned} & E[(1+r^f+\alpha(r-r^f))^{-\gamma_i}(r-r^f)] = \\ & p_l [(1+r^f+\alpha(r-r^f))^{-\gamma_i}(r-r^f)] + p_h [(1+r^f+\alpha(r-r^f))^{-\gamma_i}(r-r^f)] = 0 \\ & 0.1(1.02+\alpha(-0.06))^{-\gamma_i}(-0.06) + 0.9(1.02+\alpha(0.06))^{-\gamma_i}(0.06) = 0 \end{aligned}$$

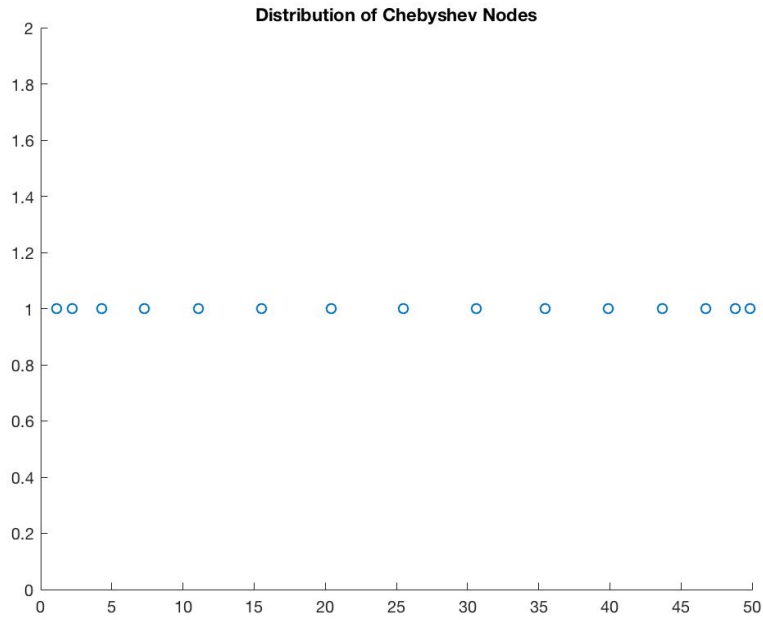
which can now be rearranged

$$\begin{aligned} & 0.1(1.02+\alpha(-0.06))^{-\gamma_i}0.06 = 0.9(1.02+\alpha(0.06))^{-\gamma_i}(0.06) \\ & 0.1(1.02+\alpha(-0.06))^{-\gamma_i} = 0.9(1.02+\alpha(0.06))^{-\gamma_i} \\ & \left(\frac{0.1}{0.9}\right)^{\frac{1}{\gamma_i}} (1.02+\alpha(-0.06)) = 1.02+\alpha(0.06) \\ & 9^{\frac{1}{\gamma_i}} (1.02+\alpha(-0.06)) = 1.02+\alpha(0.06) \\ & 9^{\frac{1}{\gamma_i}} 1.02 - 1.02 = \alpha(0.06) - \alpha(-0.06)9^{\frac{1}{\gamma_i}} \\ & \alpha = \frac{9^{\frac{1}{\gamma_i}} 1.02 - 1.02}{(0.06) + (0.06)9^{\frac{1}{\gamma_i}}} \end{aligned}$$

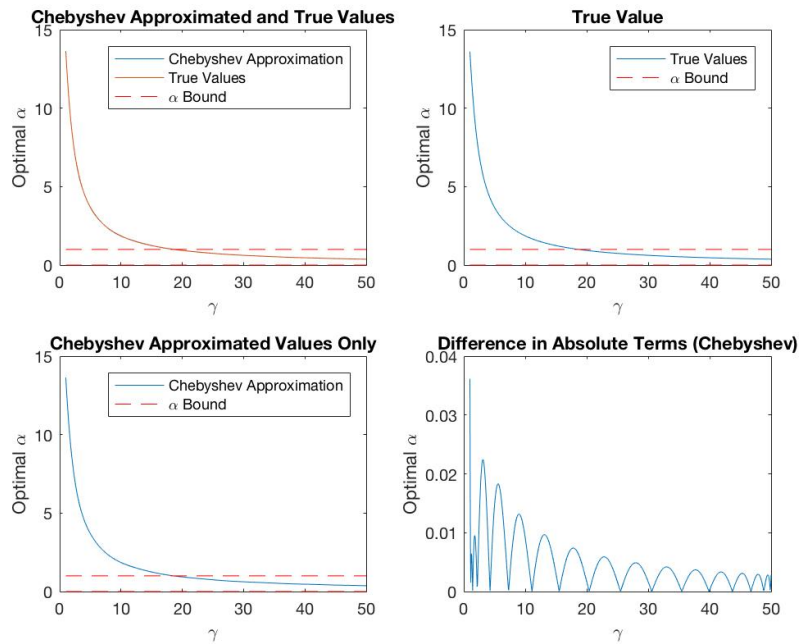
This equation has then been approximated using Chebyshev and Spline interpolation.

3.2.1 Chebyshev

To approximate the function to the highest accuracy possible, Chebyshev node had to be created.

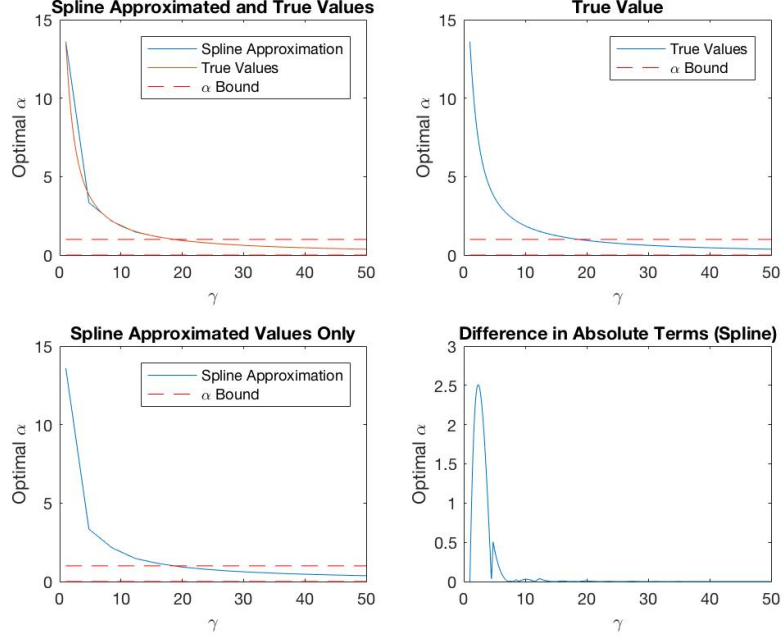


Then, the actual approximation could be performed.



3.2.2 Spline Interpolation

The Spline Interpolation has been performed by simply using the code provided as solution for the first exercise of this Problem set.



3.3 Analytical Solution of γ_i

The formula stated on the exercise sheet displays that the first derivative of the objective function with respect to α_i evaluated at $\alpha_i = 1$ has to be equal zero. That is, agent i 's optimal portfolio share is one or in other words, the constraint imposed just binds from above. Indeed, this expression can be rewritten in terms of its associated degree of risk aversion:

$$\gamma_i^* = \frac{\ln 1 - p - \ln p}{\ln 1 + r_H - \ln 1 + r_L}$$

Appendices

A Code to Question 1

Chebyshev Approximation:

```
1 function [yequi, ychebsli, ycheblec]=cheb(fct,x,m,xmin,
    xmax)
2 % In Miranda-Fackler, in fundefn, n is the degree of
    approximation, which
3 % is the number of nodes (m) -1. However, there is a
    problem with 2 nodes,
4 % so this is also set to 2 and is kept in mind.
5 c=max(m-1,2);
6
7 %define function space with fundefn
8 fspace=fundefn('cheb',c,xmin,xmax);
9 distance=(xmax-xmin)/(m-1);
10 nodesequi=zeros(m,1);
11 ynodesequi=zeros(m,1);
12 nodeschebslides=zeros(m,1);
13 ynodeschebsli=zeros(m,1);
14 ynodescheblec=zeros(m,1);
15 nodescheblecture=zeros(m,1);
16 %create nodes
17 %also, calculate function values at x
18 for j=1:m
19     nodesequi(j)=xmin+(j-1)*distance;           %
        equidistant nodes
20     ynodesequi(j)=fct(nodesequi(j));             %
        function values
21     nodeschebslides(j)=-cos((2*j-1)*pi/(2*m));   %
        Chebyshev nodes according to slide set 7
22     nodescheblecture(j)=-cos((2*j-1)*pi/(m));    %
        Chebyshev nodes according to lecture notes
23     ynodeschebsli(j)=fct(nodeschebslides(j));
24     ynodescheblec(j)=fct(nodescheblecture(j));
25 end
26
27 %calculate the matrix of basis functions
28 Bequi=funbas(fspace,nodesequi); %equidistant
29 Bchebsli=funbas(fspace,nodeschebslides); %Chebyshev
30 Bcheblec=funbas(fspace,nodescheblecture); %Chebyshev
31
32
```

```

33 %get polynomial coefficients
34 cequi=Bequi\ynodesequi; %equidistant
35 cchebsli=Bchebsli\ynodeschebsli; %chebychev
36 ccheblec=Bcheblec\ynodescheblec;
37
38 %approximate the function
39 yequi=funeval(cequi,fspace,x);
40 ychebsli=funeval(cchebsli,fspace,x);
41 ycheblec=funeval(ccheblec,fspace,x);
42
43 end

```

Linear and cubic splines, also using the Miranda-Fackler toolbox:

```

1 function [yspllin,ysplcub]=spl(fct,x,m,xmin,xmax)
2 % In Miranda-Fackler, in fundefn, n is the degree of
   approximation, which
3 % is the number of nodes (m) -1
4
5     fspacespllin=fundefn('spli',m-1,xmin,xmax,1); %
       linear splines
6     fspacesplcub=fundefn('spli',m-1,xmin,xmax,3); %
       cubic splines
7     distance=(xmax-xmin)/(m-1);
8     nodesspl=zeros(m,1);
9     ynodes=zeros(m,1);
10    %nodes
11    for i=1:m
12        nodesspl(i)=xmin+(i-1)*distance; %eqidistant
           nodes
13        ynodes(i)=fct(nodesspl(i)); %fct values at
           nodes
14    end
15
16    %calculate the matrix of basis functions
17    Bspllin=funbas(fspacespllin,nodesspl);
18    Bsplcub=funbas(fspacesplcub,nodesspl);
19
20    %get polynomial coefficients
21    cspllin=Bspllin\ynodes;
22    csplcub=Bsplcub\ynodes;
23
24    %approximate the function
25    yspllin=funeval(cspllin,fspacespllin,x);
26    ysplcub=funeval(csplcub,fspacesplcub,x);
27

```

28 `end`

Function to be approximated:

```
1 function y=simplef(x)  
2     y=1/(1+25.*x.^2);  
3 end
```

Main code:

```
1 %PS4P1  
2 clear;  
3 close all;  
4 clc;  
5  
6 %Chebychev  
7  
8 %variable declaration  
9 n1=5; %number of nodes  
10 n2=15;  
11 n3=150;  
12 %f(x) is simplef.m  
13 f=@simplef;  
14 xmin=-1;  
15 xmax=1;  
16 b=linspace(xmin,xmax,2000); %x-space  
17 b=b';  
18  
19 [yapequi,yapchebsli,yapcheblec]=cheb(f,b,n1,xmin,xmax)  
20 ;  
21 [yapequi2,yapchebsli2,yapcheblec2]=cheb(f,b,n2,xmin,  
22     xmax);  
23 [yapequi3,yapchebsli3,yapcheblec3]=cheb(f,b,n3,xmin,  
24     xmax);  
25  
26 %SPLINES equidistant nodes  
27 [yapspllin,yapsplcub]=spl(f,b,n1,xmin,xmax);  
28 [yapspllin2,yapsplcub2]=spl(f,b,n2,xmin,xmax);  
29 [yapspllin3,yapsplcub3]=spl(f,b,n3,xmin,xmax);  
30  
31 %actual function  
32 yact=simplef(b);  
33  
34 %plots compare with same n  
35 figure
```

```

35 plot(b,yapequi-yact,b,yapchebsli-yact,'--r',b,
      yapspllin-yact,'.b')
36 line([-1, 1],[0, 0],'color','black')
37 xlabel('x')
38 ylabel('p(x)-f(x) residuals')
39 title('n= 5')
40 legend('Chebychev, equidistant nodes','Chebychev,
      Chebychev nodes','Linear splines')
41
42 figure
43 plot(b,yapequi2-yact,b,yapchebsli2-yact,'--r',b,
      yapspllin2-yact,'.b')
44 xlabel('x')
45 ylabel('p(x)-f(x) residuals')
46 title('n= 15')
47 legend('Chebychev, equidistant nodes','Chebychev,
      Chebychev nodes','Linear splines')
48
49 figure
50 plot(b,yapequi3-yact,b,yapchebsli3-yact,'--r',b,
      yapspllin3-yact,'.b')
51 xlabel('x')
52 ylabel('p(x)-f(x) residuals')
53 title('n= 150')
54 legend('Chebychev, equidistant nodes','Chebychev,
      Chebychev nodes','Linear splines')
55
56
57 %plots comparison same node method (no cheb lecture
      and no cubic splines)
58
59 figure
60 plot(b,yapequi-yact,'.b',b,yapequi2-yact,'--r',b,
      yapequi3-yact)
61 xlabel('x')
62 ylabel('p(x)-f(x)')
63 title('Chebychev, equidistant node approximations')
64 legend('n=5','n=15','n=150')
65
66 figure
67 plot(b,yapchebsli-yact,'.b',b,yapchebsli2-yact,'--r',b,
      yapchebsli3-yact)
68 xlabel('x')
69 ylabel('p(x)-f(x)')
70 title('Chebychev, Chebychev node approximations')
71 legend('n=5','n=15','n=150')

```

```

72 |
73 | figure
74 | plot(b,yapspllin-yact, '.b',b,yapspllin2-yact, '--r',b,
    |     yapspllin3-yact)
75 | xlabel('x')
76 | ylabel('p(x)-f(x)')
77 | title('Linear splines, equidistant node approximations
    | ')
78 | legend('n=5', 'n=15', 'n=150')
79 |
80 |
81 | %compare linear splines and cubic splines
82 |
83 | %plots compare with same n
84 |
85 | figure
86 | plot(b,yapspllin-yact,b,yapsplcub-yact, '--r')
87 | line([-1, 1],[0, 0], 'color','black')
88 | xlabel('x')
89 | ylabel('p(x)-f(x) residuals')
90 | title('Splines, n= 5')
91 | legend('linear','cubic')
92 |
93 | figure
94 | plot(b,yapspllin2-yact,b,yapsplcub2-yact)
95 | xlabel('x')
96 | ylabel('p(x)-f(x) residuals')
97 | title('Splines, n= 15')
98 | legend('linear','cubic')
99 |
100 | figure
101 | plot(b,yapspllin3-yact,b,yapsplcub3-yact, '--r')
102 | xlabel('x')
103 | ylabel('p(x)-f(x) residuals')
104 | title('Splines, n= 150')
105 | legend('linear','cubic')
106 |
107 | %compare slides and lecture
108 |
109 | %plots compare with same n
110 | figure
111 | plot(b,yapcheblec-yact,b,yapchebsli-yact, '--r')
112 | line([-1, 1],[0, 0], 'color','black')
113 | xlabel('x')
114 | ylabel('p(x)-f(x) residuals')
115 | title('Chebychev, Chebychev nodes, n= 5')

```

```

116 legend('Lecture','Slides')
117
118 figure
119 plot(b,yapcheblec2-yact,b,yapchebsli2-yact,'--r')
120 xlabel('x')
121 ylabel('p(x)-f(x) residuals')
122 title('Chebychev, Chebychev nodes, n= 15')
123 legend('Lecture','Slides')
124
125 figure
126 plot(b,yapcheblec3-yact,b,yapchebsli3-yact,'--r')
127 xlabel('x')
128 ylabel('p(x)-f(x) residuals')
129 title('Chebychev, Chebychev nodes, n= 150')
130 legend('Lecture','Slides')

```

B Code to Question 2

```

1 %% Problem set 4, exercise 2
2 close all;
3 clear;
4 % Set parameters
5 rmin = -0.08;
6 rmax = 0.12;
7 p = 0.5;
8 % CRRA
9 gamma = 2;
10 % Grid
11 Wmin = .5;
12 Wmax = 50;
13 % Set number of nodes & order of polynomial
14 m = 15;
15 n = 1;
16
17 prob = [p 1-p]';
18 R = [1+rmin 1+rmax]';
19
20 %% Quadratic utility
21 % Define linear optimal consumption
22 linMU = @(W) .5*(prob'*R).*W;
23
24 %% CRRA utility
25 % Define nonlinear optimal consumption s.t. it
    constitutes a root-finding

```



```

26 % problem; implicitly defined by Euler equation.
27 nonlinMU = @(W, CO) ( prob(1).*( ( W.*R(1) - CO ).^-
    gamma) + prob(2).*( ( W.*R(2) - CO ).^-gamma) )
    .^-(1./gamma) - CO;
28
29 % Plot implicit function CO of W
30 fimplicit(nonlinMU, [Wmin Wmax 0 30])
31
32 %% Interpolation of quadratic utility using Chebyshev
33 x=linspace(Wmin,Wmax,1000);
34 [ylin, ftilde1, yhat1] = chebyshev_approx(linMU, Wmin,
    Wmax, m, n, 'explicit', x');
35
36 %% Interpolation of CRRA utility using Chebyshev
37 [ynonlin, ftilde2, yhat2] = chebyshev_approx(nonlinMU,
    Wmin, Wmax, m, n, 'implicit', x');
38 ynonlin=ynonlin'; % it gives 1X1000 matrix instead of
    1000X1 (?)
39
40 %% Plot residuals
41 figure(1)
42 plot(x',ftilde1-ylin,x',ftilde2-ynonlin,'--r')
43 xlabel('W')
44 ylabel('residuals')
45 legend('linear residuals','nonlinear residuals')
46 title('Approximation errors: Quadratic vs. CRRA.
    Baseline')
47 % Accuracy
48 acclin = max(abs(ftilde1-ylin));
49 fprintf( 'Approximation error*e+13 for quadratic
    utility: %.4f \n', acclin*10^13)
50 accnonlin = max(abs(ftilde2-ynonlin));
51 fprintf( 'Approximation error*e+13 for CRRA utility:
    %.4f \n', accnonlin*10^13)
52 % Maximum percentage deviation
53 maxdev = max(abs(ynonlin - ylin)./ylin);
54 fprintf( 'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)
55
56 figure(2)
57 plot(x',ftilde1,x',ftilde2,'--r',x',ylin,'.b',x',
    ynonlin,':p')
58 xlabel('W')
59 ylabel('fcts')
60 legend('linear aprox','nonlinear approx','lin fct','
    nonlin fct')

```

```

61 title('Chebyshev approximation: Quadratic vs. CRRA.
        Baseline')
62
63 % What happens if setting is changed?
64 %% (i) Increase in gamma
65 gamma = 4;
66 nonlinMUgg = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
        gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
        .^-(1./gamma) - C0;
67 [ynonlingg, ftilde2gg, yhat2gg] = chebyshev_approx(
        nonlinMUgg, Wmin, Wmax, m, n, 'implicit', x');
68 ynonlingg=ynonlingg';
69
70 % Accuracy
71 accnonlin = max(abs(ftilde2gg-ynonlingg));
72 fprintf( '(i) Increase in gamma. For example, set
        gamma = %.2f \n', gamma)
73 fprintf( 'Approximation error*e+13 for CRRA utility:
        %.4f \n', accnonlin*10^13)
74 % Maximum percentage deviation
75 maxdev = max(abs(ynonlingg - ylin)./ylin);
76 fprintf( 'The maximum percentage deviation is %.2f
        percent \n', maxdev*100)
77
78 %% (ii) Decrease in p
79 gamma = 2;
80 p = 0.2;
81 prob = [p 1-p]';
82 nonlinMUpp = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
        gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
        .^-(1./gamma) - C0;
83 [ynonlinpp, ftilde2pp, yhat2pp] = chebyshev_approx(
        nonlinMUpp, Wmin, Wmax, m, n, 'implicit', x');
84 ynonlinpp=ynonlinpp';
85
86 % Accuracy
87 accnonlin = max(abs(ftilde2pp-ynonlinpp));
88 fprintf( '(ii) Decrease in p. For example, set p = %.2
        f \n', p)
89 fprintf( 'Approximation error*e+13 for CRRA utility:
        %.4f \n', accnonlin*10^13)
90 % Maximum percentage deviation
91 maxdev = max(abs(ynonlinpp - ylin)./ylin);
92 fprintf( 'The maximum percentage deviation is %.2f
        percent \n', maxdev*100)
93

```

```

94
95 %% (iii) Increase spread
96 p = 0.5;
97 prob = [p 1-p]';
98 inc = 0.2;
99 rmin = rmin - inc;
100 rmax = rmax + inc;
101 R = [1+rmin 1+rmax]';
102 nonlinMUsp = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
    gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
    .^-(1./gamma) - C0;
103 [ynonlinsp, ftilde2sp, yhat2sp] = chebyshev_approx(
    nonlinMUsp, Wmin, Wmax, m, n, 'implicit', x');
104 ynonlinsp=ynonlinsp';
105
106 % Accuracy
107 accnonlin = max(abs(ftilde2sp-ynonlinsp));
108 fprintf( '(iii) Change spread by +/- inc. For example,
    spread increase = %.2f \n', 2*inc)
109 fprintf( 'Maximum absolute error*e+13 for CRRA utility
    : %.4f \n', accnonlin*10^13)
110 % Maximum percentage deviation
111 maxdev = max(abs(ynonlinsp - ylin)./ylin);
112 fprintf( 'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)
113
114 function [yact, yapp, yhat] = chebyshev_approx( fun, a
    , b, m, n, funtype, x)
115 % [yact, yapp, yhat] = chebyshev_approx( fun, a, b, m,
    n, funtype, x)
116 % USAGE: Chebychev interpolation
117 % INPUT:
118 %     fun      := function handle, e.g., @exp(-x)
119 %     [a, b]   := domain on which fun is interpolated
120 %     m        := nb. of nodes, j = 1,...,m
121 %     n        := degree of chebyshev polynomial; n.b.: n
    < m
122 % funtype     := 'explicit' or 'implicit' function
123 % OUTPUT:
124 %     coeff    := Chebyshev coefficients alpha_i, i =
    0,...,n
125 %     xhat     := Chebyshev nodes
126 %     yhat     := Function values at Chebyshev nodes
127
128 %% (0) Initialisation
129 if n > m

```

```

130         error( 'Error. It must hold that n < m.' )
131     end
132
133     %% (1) Compute row vector of m Chebyshev nodes in
134     % [-1,1]
135     row = 1:m;
136     tmp = ( 2*row - 1 )*pi;
137     zhat = - cos( tmp / (2*m) );
138
139     %% (2) Rescale Chebyshev nodes to [a,b]
140     xhat = a + .5*( b - a )*( zhat + 1 );
141
142     %% (3) Evaluate function at Chebyshev nodes
143     if strcmp(funtype, 'implicit') % implicit optimal
144         % consumption function
145         % Plug in xhat for W
146         % Calculate actual values of y for x instead of
147         % nodes only
148         tmp2 = length(xhat);
149         tmp3 = length(x);
150         yhat = ones(1,tmp2);
151         yact = ones(1,tmp3);
152         for i = 1:tmp3
153             Wnew = x(i);
154             myfunnew= @(C0) fun(Wnew,C0);
155             x0new=0;
156             yact(i)=fzero( myfunnew,x0new);
157         end
158         for j = 1:tmp2
159             W = xhat(j);
160             myfun = @(C0) fun(W,C0);
161             x0 = 0;
162             yhat(j) = fzero( myfun,x0 ); % Rootfinder
163             % evaluates C0(W)
164         end
165     else % explicit optimal consumption function
166         yhat = feval( fun, xhat );
167         yact = feval( fun, x ); %same here
168     end
169
170     %% (4) Polynomial coeffs are solution to linear
171     % equation Tx*coeff = yhat
172     % Construct interpolation matrix Tx of size m*(n+1)
173     Tx = ones(m, n+1); % Returns a vector of ones only if
174     % n = 0
175     if n >= 1

```

```

170     Tx(:,2) = xhat';
171 end
172 % Recursively define rest of matrix Tx
173 if n >= 2
174     for j = 3:(n+1)
175         Tx(:,j) = 2*xhat*Tx(:,j-1) - Tx(:,j-2);
176     end
177 end
178 % Then, polynomial coefficients are given by
179 coeff = Tx\yhat';
180
181 %% Evaluate approximation yapp for larger x
182 tmp4 = length(x);
183 Txnew = ones(tmp4, n+1); % Returns a vector of ones
    only if n = 0
184 if n >= 1
185     Txnew(:,2) = x';
186 end
187 % Recursively define rest of matrix Tx
188 if n >= 2
189     for j = 3:(n+1)
190         Txnew(:,j) = 2*x*Txnew(:,j-1) - Txnew(:,j-2);
191     end
192 end
193 yapp = Txnew*coeff;
194
195 end

```

C Code to Question 3

Main code:

```

1 %PS4P3
2 clear;
3 close all;
4 clc;
5
6 %variable declaration
7 n=15; %number of nodes;
8 f=@simplefQ4P3;
9
10 %Set up Gamma space
11 xmin=1;
12 xmax=50;
13 b=linspace(xmin,xmax,1000);

```

```

14 b=b';
15
16 approximated_alpha=chebi(f,b,n,xmin,xmax);
17 real_alpha=simplefQ4P3(b);
18 difference= abs(approximated_alpha - real_alpha);
19
20 figure
21 subplot(2,2,1)
22 plot(b,approximated_alpha,b,real_alpha)
23 title("Chebyshev Approximated and True Values")
24 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
25 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
26 legend("Chebyshev Approximation","True Values","\alpha
    Bound")
27 xlabel("\gamma")
28 ylabel("Optimal \alpha")
29
30 subplot(2,2,2)
31 plot(b,real_alpha)
32 title("True Value")
33 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
34 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
35 legend("True Values","\alpha Bound")
36 xlabel("\gamma")
37 ylabel("Optimal \alpha")
38
39 subplot(2,2,3)
40 plot(b,approximated_alpha)
41 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
42 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
43 title("Chebyshev Approximated Values Only")
44 legend("Chebyshev Approximation","\alpha Bound")
45 xlabel("\gamma")
46 ylabel("Optimal \alpha")
47
48 subplot(2,2,4)
49 plot(b,difference)
50 title("Difference in Absolute Terms (Chebyshev)")
51 xlabel("\gamma")
52 ylabel("Optimal \alpha")

```

```

53
54 %SPLINES equidistant nodes
55 alphaspline=spl(f,b,n,xmin,xmax);
56 difference_spline= abs(alphaspline - real_alpha);
57
58 figure
59 subplot(2,2,1)
60 plot(b,alphaspline,b,real_alpha)
61 title("Spline Approximated and True Values")
62 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
63 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
64 legend("Spline Approximation","True Values","\alpha
    Bound")
65 xlabel("\gamma")
66 ylabel("Optimal \alpha")
67
68 subplot(2,2,2)
69 plot(b,real_alpha)
70 title("True Value")
71 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
72 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
73 legend("True Values","\alpha Bound")
74 xlabel("\gamma")
75 ylabel("Optimal \alpha")
76
77 subplot(2,2,3)
78 plot(b,alphaspline)
79 line([min(b),max(b)], [0,0], 'Color','red','LineStyle','
    --')
80 line([min(b),max(b)], [1,1], 'Color','red','LineStyle','
    --')
81 title("Spline Approximated Values Only")
82 legend("Spline Approximation","\alpha Bound")
83 xlabel("\gamma")
84 ylabel("Optimal \alpha")
85
86 subplot(2,2,4)
87 plot(b,difference_spline)
88 title("Difference in Absolute Terms (Spline)")
89 xlabel("\gamma")
90 ylabel("Optimal \alpha")

```

The function to be approximated:

```
1 function a=simplefQ4P3(x)
2   a=(9.^(1./x).*1.02-1.02)./(0.06+0.06.*9.^(1./x));
3 end
```

Chebyshev:

```
1 function [y]=chebi(fct,x,m,xmin,xmax)
2
3 %Chebyshev interpolation m=n
4 %define function space using fundefn
5
6 fspace=fundefn('cheb',m,-1,1);
7
8 anodes=NaN(m,1);
9 nodes=NaN(m,1);
10
11 %create nodes and evaluate fct values at nodes
12 for j=1:m
13   nodes(j,1)=-cos((2*j-1)*pi/(2*m)); %Chebyshev nodes
14   nodes(j,1)=(nodes(j,1)+1)*(xmax-xmin)/2+xmin; %
      Rescale nodes
15   anodes(j,1)=fct(nodes(j,1)); %Evaluate function at
      nodes
16 end
17
18 figure
19 scatter(nodes,ones(m,1))
20 title("Distribution of Chebyshev Nodes")
21
22 %calculate matrix of basis functions
23 B=funbas(fspace,nodes);
24
25 %solve for polynomial coefficients
26 c=B\anodes;
27
28 %approximate function a
29 y=funeval(c,fspace,x);
30
31 end
```

Spline:

```
1 function [yspllin,ysplcub]=spl(fct,x,m,xmin,xmax)
2 % In Miranda-Fackler, in fundefn, n is the degree of
   approximation, which
3 % is the number of nodes (m) -1
```



```

4
5 fspacespllin=fundefn('spli',m-1,xmin,xmax,1); %
    linear splines
6 fspacesplcub=fundefn('spli',m-1,xmin,xmax,3); %
    cubic splines
7 distance=(xmax-xmin)/(m-1);
8 nodesspl=zeros(m,1);
9 ynodes=zeros(m,1);
10 %nodes
11 for i=1:m
12     nodesspl(i)=xmin+(i-1)*distance; %eqidistant
        nodes
13     ynodes(i)=fct(nodesspl(i)); %fct values at
        nodes
14 end
15
16 %calculate the matrix of basis functions
17 Bspllin=funbas(fspacespllin,nodesspl);
18 Bsplcub=funbas(fspacesplcub,nodesspl);
19
20 %get polynomial coefficients
21 cspllin=Bspllin\ynodes;
22 csplcub=Bsplcub\ynodes;
23
24 %approximate the function
25 yspllin=funeval(cspllin,fspacespllin,x);
26 ysplcub=funeval(csplcub,fspacesplcub,x);
27
28 end

```