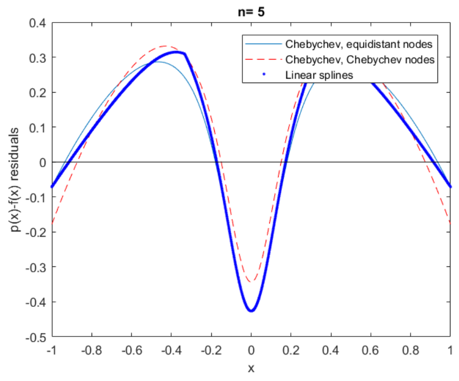
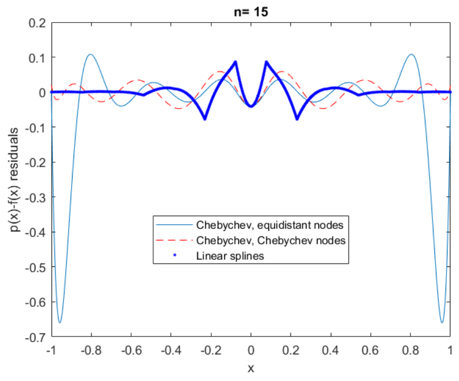
Answers to Problem Set 4
Group name: Ferienspass

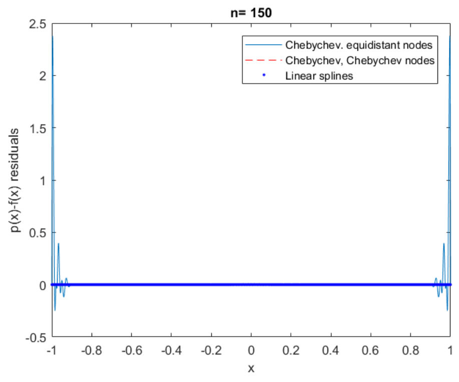
Sebastian Kühnl: 5642348
Alexander Dück (as: reebyte): 5504077
Patrick Blank (as: paddyblank): 6729110
Christian Wierschem: 6729288

1 Question 1

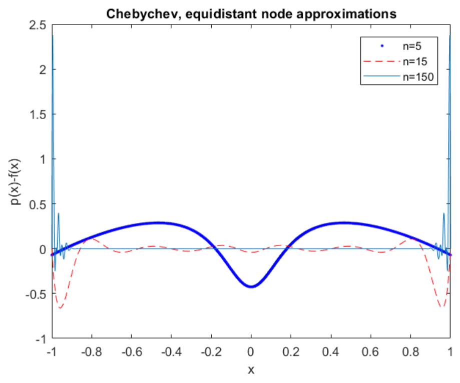
Chebyshev approximation using equidistant nodes and Chebyshev nodes. However, there is a difference between lecture slides 7 and the notes from the lecture, as you will see in the code provided below. Plots are ordered in chronological order! (For comparison, the residuals to actual function are plotted.)

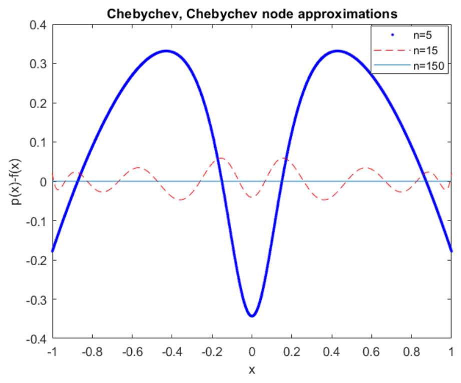


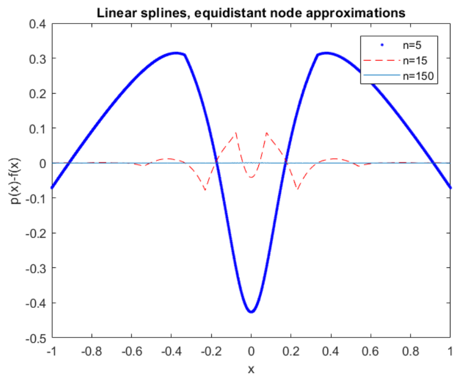




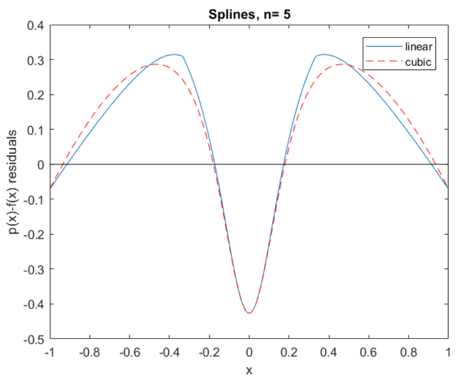
For n=5, equidistant nodes and Chebyshev nodes as well as linear splines are very similar. For n=15, linear splines become more edgy. It performs well at the edges and average else. Both Chebyshev approximations are very similar in [-0.75;0.75], while equidistant nodes fall off at the edges (as expected). 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.

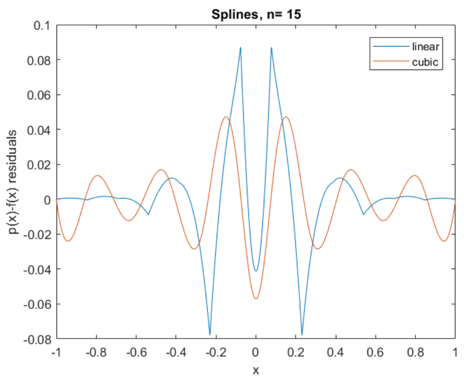


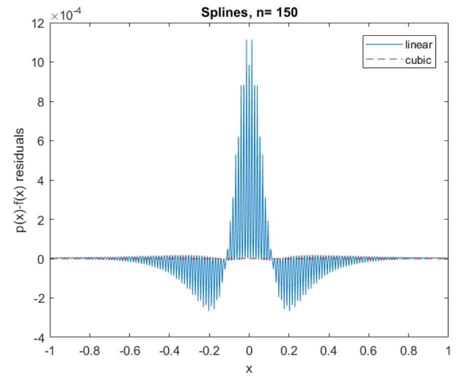




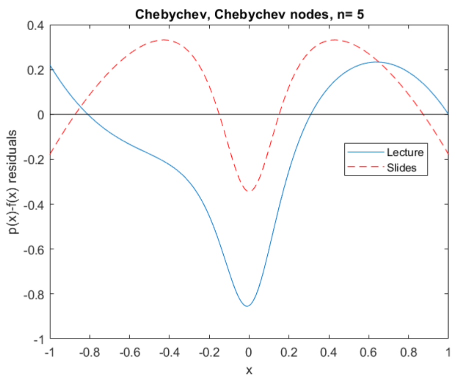
In this figure again the effect of equidistant nodes when using Chebyshev can be seen as large fluctuations at the corner. Besides this, as the number of nodes increases, the approximation gets closer to the real function.

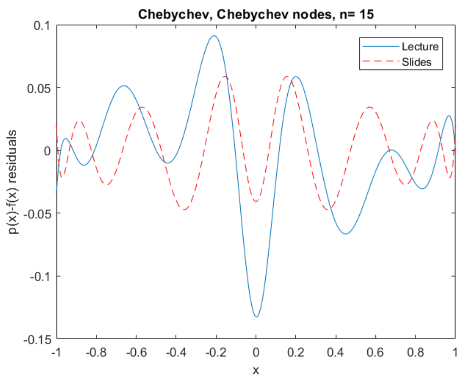


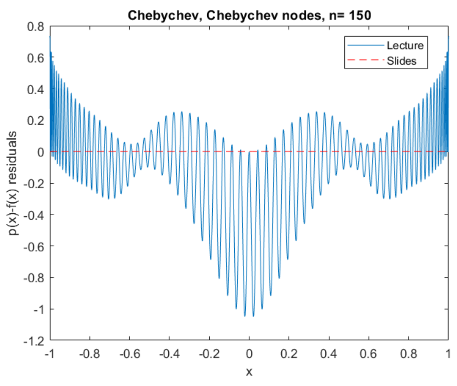




Linear and cubic splines are very similar when n=5. For n=15 one can observe that cubic splines are smoother than linear splines and perform better in the center (around 0), whereas linear splines perform better at the corner (it becomes smooth and then becomes nearly a straight line). For n=150, at 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

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

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

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

Moreover,

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

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.

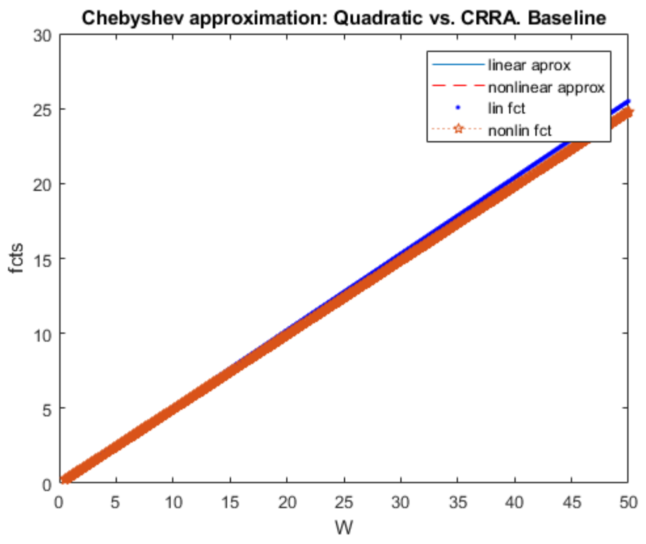
CRRA utility In case of CRRA utility the three derivatives are given by

$$u'(C_t) = C_t^{-\gamma} \qquad \text{for any} \gamma \neq 1 \qquad \text{(Marginal utility)}$$
$$u''(C_t) = -\gamma C_t^{-(\gamma+1)} < 0 \qquad \text{(Risk aversion)}$$
$$u'''(C_t) = \gamma(1+\gamma) C_t^{-(\gamma+2)} > 0 \qquad \text{(Prudence)}$$

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:

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



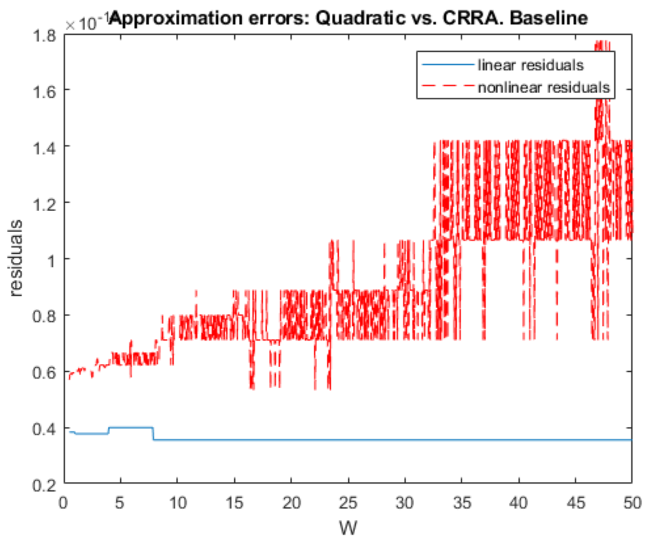


Table 1: Maximum percentage errors of deviation

	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

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

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

3.2 Analytical Solution of α_i

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

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

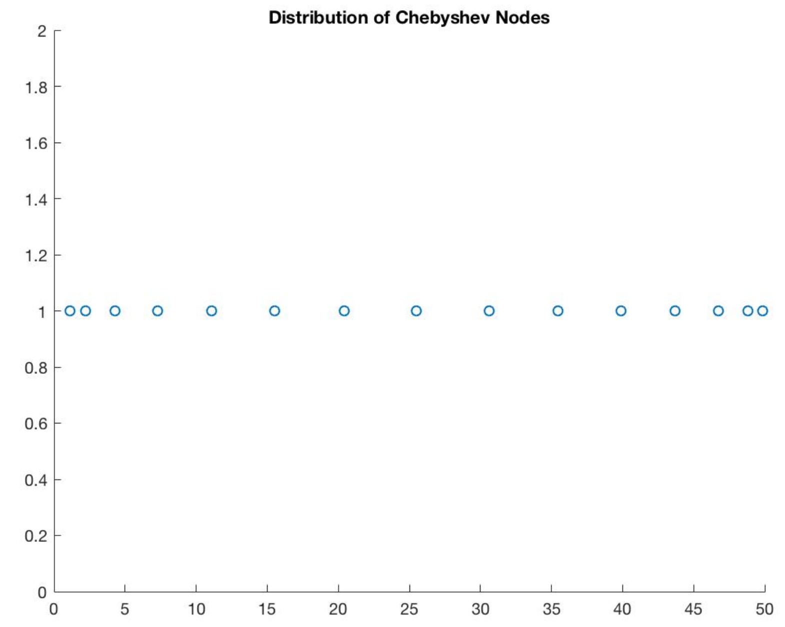
which can now be rearranged

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

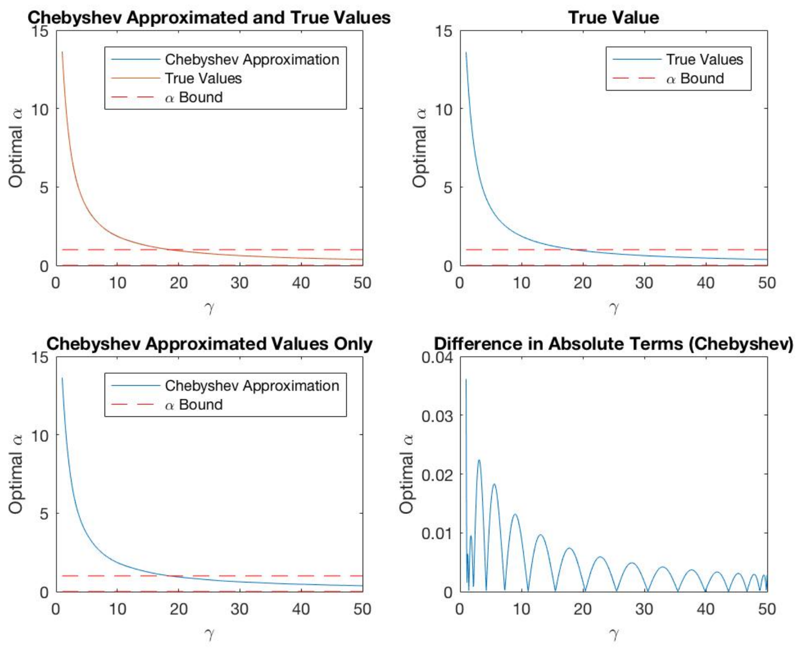
This equation has then been approximated using Chebyshev and Spline interpolation.

3.2.1 Chebyshev

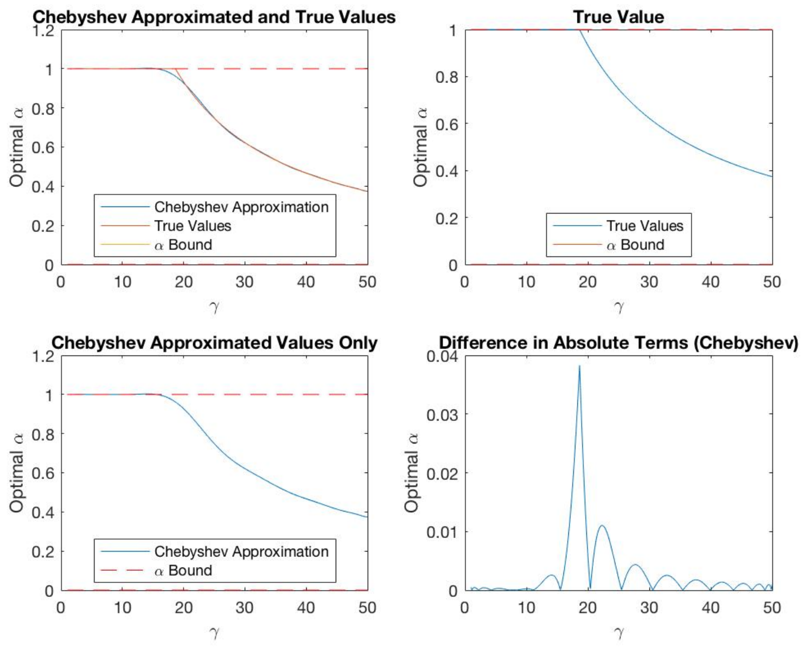
To approximate the function to the highest accuracy possible, Chebyshev nodes had to be created.



Then, the actual approximation could be performed. First, the unconstrained α has been approximated.



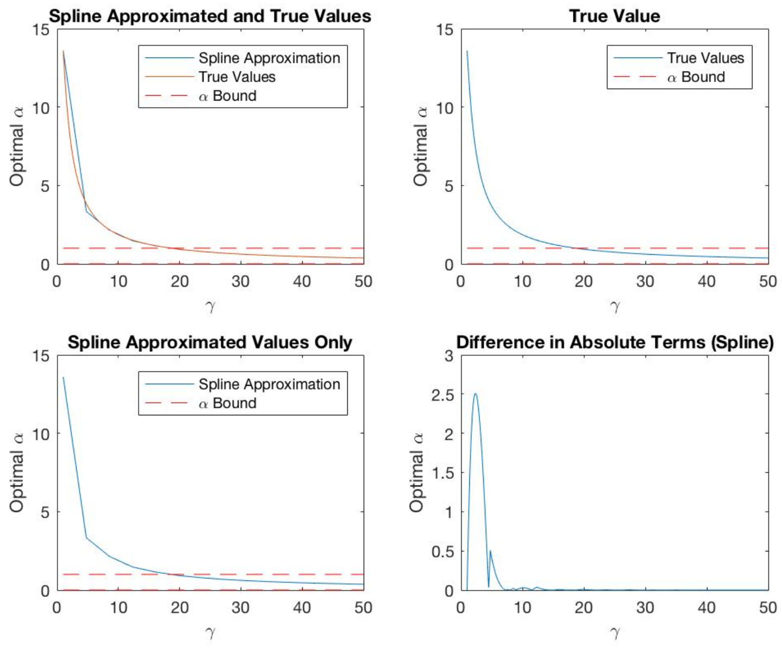
For the constrained $\alpha \in [0,1]$, we get:



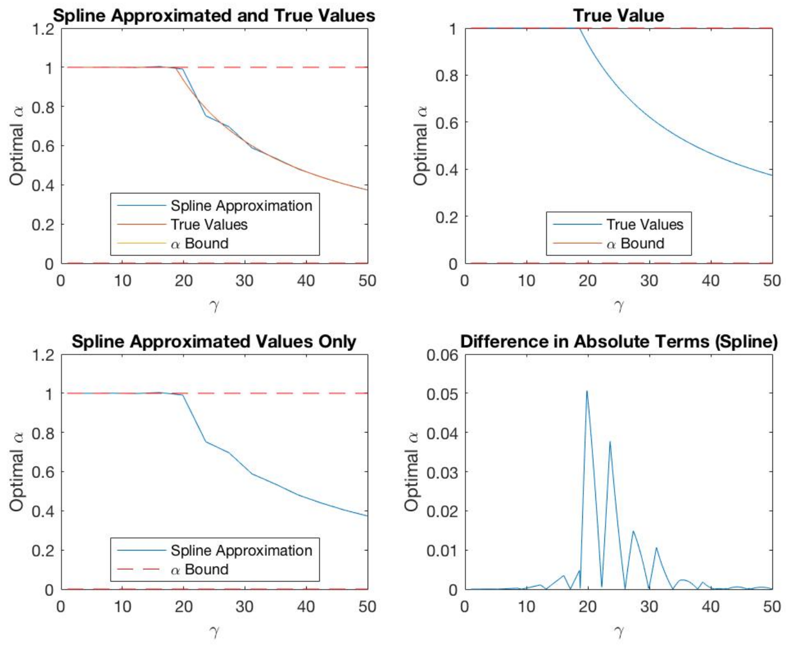
One can clearly see the spike in inaccuracy around the kink in the true function.

3.2.2 Spline Interpolation

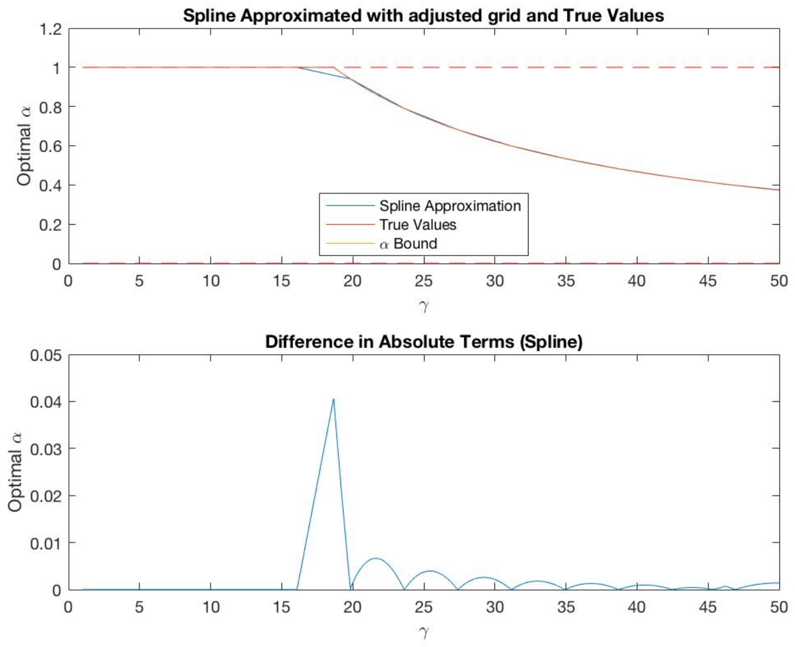
The Spline Interpolation has been performed by simply using the code provided as solution for the first exercise of this Problem set. Again, the interpolation is first performed on the unconstrained α and on the constrained one thereafter.



As before for the Chebyshev approximation, the approximation with linear splines for the constrained α differs:



Again, the error increases around the point of the kink. The quality of this approximation technique could be increased by adaptive grid methods, adding nodes where the difference between the approximation is the largest (i.e. right at the point of the kink). This has been tried out but did not fully work out as planned. However, an improvement can be seen from using an adjusted grid:



3.3 Analytical Solution of γ_i

The formula stated on the exercise sheet displays that the first derivate 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 (using $p := p_l$ and therefore $1 - p = p_h$):

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

With our specification, this gives:

$$\gamma_i^* = \frac{\ln(0.9) - \ln(0.1)}{\ln(1.02) - \ln(0.96)} \approx 36.2431$$

Appendices

A Code to Question 1

Chebyshev Approximation:

```
function [yequi,ychebsli,ycheblec]=cheb(fct,x,m,xmin,
   xmax)
% In Miranda-Fackler, in fundefn, n is the degree of
   approximation, which
% is the number of nodes (m) -1. However, there is a
   problem with 2 nodes,
% so this is also set to 2 and is kept in mind.
c=max(m-1,2);

%define function space with fundefn
fspace=fundefn('cheb',c,xmin,xmax);
distance=(xmax-xmin)/(m-1);
nodesequi=zeros(m,1);
ynodesequi=zeros(m,1);
nodeschebslides=zeros(m,1);
ynodeschebsli=zeros(m,1);
ynodescheblec=zeros(m,1);
nodescheblecture=zeros(m,1);
%create nodes
%also, calculate function values at x
for j=1:m
  nodesequi(j)=xmin+(j-1)*distance;             %
     equidistant nodes
  ynodesequi(j)=fct(nodesequi(j));              %
     function values
  nodeschebslides(j)=-cos((2*j-1)*pi/(2*m));     %
     Chebyshev nodes according to slide set 7
  nodescheblecture(j)=-cos((2*j-1)*pi/(m));     %
     Chebyshev nodes according to lecture notes
  ynodeschebsli(j)=fct(nodeschebslides(j));
  ynodescheblec(j)=fct(nodescheblecture(j));
end

%calculate the matrix of basis functions
Bequi=funbas(fspace,nodesequi);  %equidistant
Bchebsli=funbas(fspace,nodeschebslides);  %Chebyshev
Bcheblec=funbas(fspace,nodescheblecture);  %Chebyshev


```

```
%get polynomial coefficients
cequi=Bequi\ynodesequi;  %equidistant
cchebsli=Bchebsli\ynodeschebsli;  %chebychev
ccheblec=Bcheblec\ynodescheblec;

%approximate the function
yequi=funeval(cequi,fspace,x);
ychebsli=funeval(cchebsli,fspace,x);
ycheblec=funeval(ccheblec,fspace,x);

end
```

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

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

  fspacespllin=fundefn('spli',m-1,xmin,xmax,1);  %
      linear splines
  fspacesplcub=fundefn('spli',m-1,xmin,xmax,3);  %
      cubic splines
  distance=(xmax-xmin)/(m-1);
  nodesspl=zeros(m,1);
  ynodes=zeros(m,1);
  %nodes
  for i=1:m
    nodesspl(i)=xmin+(i-1)*distance;   %eqidistant
        nodes
    ynodes(i)=fct(nodesspl(i));        %fct values at
        nodes
  end

  %calculate the matrix of basis functions
  Bspllin=funbas(fspacespllin,nodesspl);
  Bsplcub=funbas(fspacesplcub,nodesspl);

  %get polynomial coefficients
  cspllin=Bspllin\ynodes;
  csplcub=Bsplcub\ynodes;

  %approximate the function
  yspllin=funeval(cspllin,fspacespllin,x);
  ysplcub=funeval(csplcub,fspacesplcub,x);

```

```
end
```

Function to be approximated:

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

Main code:

```
%PS4P1
clear;
close all;
clc;

%Chebychev

%variable declaration
n1=5;  %number of nodes
n2=15;
n3=150;
%f(x) is simplef.m
f=@simplef;
xmin=-1;
xmax=1;
b=linspace(xmin,xmax,2000); %x-space
b=b';

[yapequi,yapchebsli,yapcheblec]=cheb(f,b,n1,xmin,xmax)
    ;
[yapequi2,yapchebsli2,yapcheblec2]=cheb(f,b,n2,xmin,
    xmax);
[yapequi3,yapchebsli3,yapcheblec3]=cheb(f,b,n3,xmin,
    xmax);


%SPLINES equidistant nodes
[yapspllin,yapsplcub]=spl(f,b,n1,xmin,xmax);
[yapspllin2,yapsplcub2]=spl(f,b,n2,xmin,xmax);
[yapspllin3,yapsplcub3]=spl(f,b,n3,xmin,xmax);

%actual function
yact=simplef(b);


%plots compare with same n
figure
```

```
plot(b,yapequi-yact,b,yapchebsli-yact,'--r',b,
    yapspllin-yact,'.b')
line([-1, 1],[0, 0],'color','black')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('n= 5')
legend('Chebychev, equidistant nodes','Chebychev,
    Chebychev nodes','Linear splines')

figure
plot(b,yapequi2-yact,b,yapchebsli2-yact,'--r',b,
    yapspllin2-yact,'.b')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('n= 15')
legend('Chebychev, equidistant nodes','Chebychev,
    Chebychev nodes','Linear splines')

figure
plot(b,yapequi3-yact,b,yapchebsli3-yact,'--r',b,
    yapspllin3-yact,'.b')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('n= 150')
legend('Chebychev. equidistant nodes','Chebychev,
    Chebychev nodes','Linear splines')


%plots comparison same node method (no cheb lecture
    and no cubic splines)

figure
plot(b,yapequi-yact,'.b',b,yapequi2-yact,'--r',b,
    yapequi3-yact)
xlabel('x')
ylabel('p(x)-f(x)')
title('Chebychev, equidistant node approximations')
legend('n=5','n=15','n=150')

figure
plot(b,yapchebsli-yact,'.b',b,yapchebsli2-yact,'--r',b
    ,yapchebsli3-yact)
xlabel('x')
ylabel('p(x)-f(x)')
title('Chebychev, Chebychev node approximations')
legend('n=5','n=15','n=150')
```

```

figure
plot(b,yapspllin-yact,'.b',b,yapspllin2-yact,'--r',b,
    yapspllin3-yact)
xlabel('x')
ylabel('p(x)-f(x)')
title('Linear splines, equidistant node approximations
    ')
legend('n=5','n=15','n=150')


%compare linear splines and cubic splines

%plots compare with same n

figure
plot(b,yapspllin-yact,b,yapsplcub-yact,'--r')
line([-1, 1],[0, 0],'color','black')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Splines, n= 5')
legend('linear','cubic')

figure
plot(b,yapspllin2-yact,b,yapsplcub2-yact)
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Splines, n= 15')
legend('linear','cubic')

figure
plot(b,yapspllin3-yact,b,yapsplcub3-yact,'--r')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Splines, n= 150')
legend('linear','cubic')

%compare slides and lecture

%plots compare with same n
figure
plot(b,yapcheblec-yact,b,yapchebsli-yact,'--r')
line([-1, 1],[0, 0],'color','black')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Chebychev, Chebychev nodes, n= 5')
```

```
legend('Lecture','Slides')

figure
plot(b,yapcheblec2-yact,b,yapchebsli2-yact,'--r')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Chebychev, Chebychev nodes, n= 15')
legend('Lecture','Slides')

figure
plot(b,yapcheblec3-yact,b,yapchebsli3-yact,'--r')
xlabel('x')
ylabel('p(x)-f(x) residuals')
title('Chebychev, Chebychev nodes, n= 150')
legend('Lecture','Slides')
```

B Code to Question 2

```
%% Problem set 4, exercise 2
close all;
clear;
% Set parameters
rmin = -0.08;
rmax = 0.12;
p = 0.5;
% CRRA
gamma = 2;
% Grid
Wmin = .5;
Wmax = 50;
% Set number of nodes & order of polynomial
m = 15;
n = 1;

prob = [p 1-p]';
R = [1+rmin 1+rmax]';

%% Quadratic utility
% Define linear optimal consumption
linMU = @(W) .5*(prob'*R).*W;

%% CRRA utility
% Define nonlinear optimal consumption s.t. it
   constitutes a root-finding
```

```
% problem; implicitly defined by Euler equation.
nonlinMU = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
    gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
    .^-(1./gamma) - C0;

% Plot implicit function C0 of W
fimplicit(nonlinMU, [Wmin Wmax 0 30])

%% Interpolation of quadratic utility using Chebyshev
x=linspace(Wmin,Wmax,1000);
[ylin, ftilde1, yhat1] = chebyshev_approx(linMU, Wmin,
     Wmax, m, n, 'explicit', x');

%% Interpolation of CRRA utility using Chebyshev
[ynonlin, ftilde2, yhat2] = chebyshev_approx(nonlinMU,
     Wmin, Wmax, m, n, 'implicit', x');
ynonlin=ynonlin';  % it gives 1X1000 matrix instead of
     1000X1 (?!)

%% Plot residuals
figure(1)
plot(x',ftilde1-ylin,x',ftilde2-ynonlin,'--r')
xlabel('W')
ylabel('residuals')
legend('linear residuals','nonlinear residuals')
title('Approximation errors: Quadratic vs. CRRA.
    Baseline')
% Accuracy
acclin = max(abs(ftilde1-ylin));
fprintf( 'Approximation error*e+13 for quadratic
    utility: %.4f \n', acclin*10^13)
accnonlin = max(abs(ftilde2-ynonlin));
fprintf( 'Approximation error*e+13 for CRRA utility:
    %.4f \n', accnonlin*10^13)
% Maximum percentage deviation
maxdev = max(abs(ynonlin - ylin)./ylin);
fprintf(  'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)

figure(2)
plot(x',ftilde1,x',ftilde2,'--r',x',ylin,'.b',x',
    ynonlin,':p')
xlabel('W')
ylabel('fcts')
legend('linear aprox','nonlinear approx','lin fct','
    nonlin fct')
```

```
title('Chebyshev approximation: Quadratic vs. CRRA. 
    Baseline')

% What happens if setting is changed?
%% (i) Increase in gamma
gamma = 4;
nonlinMUgg = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
    gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
    .^-(1./gamma) - C0;
[ynonlingg, ftilde2gg, yhat2gg] = chebyshev_approx(
    nonlinMUgg, Wmin, Wmax, m, n, 'implicit', x');
ynonlingg=ynonlingg';

% Accuracy
accnonlin = max(abs(ftilde2gg-ynonlingg));
fprintf( '(i) Increase in gamma. For example, set
    gamma = %.2f \n', gamma)
fprintf( 'Approximation error*e+13 for CRRA utility:
    %.4f \n', accnonlin*10^13)
% Maximum percentage deviation
maxdev = max(abs(ynonlingg - ylin)./ylin);
fprintf(  'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)

%% (ii) Decrease in p
gamma = 2;
p = 0.2;
prob = [p 1-p]';
nonlinMUpp = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
    gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
    .^-(1./gamma) - C0;
[ynonlinpp, ftilde2pp, yhat2pp] = chebyshev_approx(
    nonlinMUpp, Wmin, Wmax, m, n, 'implicit', x');
ynonlinpp=ynonlinpp';

% Accuracy
accnonlin = max(abs(ftilde2pp-ynonlinpp));
fprintf( '(ii) Decrease in p. For example, set p = %.2
    f \n', p)
fprintf( 'Approximation error*e+13 for CRRA utility:
    %.4f \n', accnonlin*10^13)
% Maximum percentage deviation
maxdev = max(abs(ynonlinpp - ylin)./ylin);
fprintf(  'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)

```

```

%% (iii) Increase spread
p = 0.5;
prob = [p 1-p]';
inc = 0.2;
rmin = rmin - inc;
rmax = rmax + inc;
R = [1+rmin 1+rmax]';
nonlinMUsp = @(W, C0) ( prob(1).*( ( W.*R(1) - C0 ).^-
    gamma) + prob(2).*( ( W.*R(2) - C0 ).^-gamma) )
    .^-(1./gamma) - C0;
[ynonlinsp, ftilde2sp, yhat2sp] = chebyshev_approx(
    nonlinMUsp, Wmin, Wmax, m, n, 'implicit', x');
ynonlinsp=ynonlinsp';

% Accuracy
accnonlin = max(abs(ftilde2sp-ynonlinsp));
fprintf( '(iii) Change spread by +/- inc. For example,
     spread increase = %.2f \n', 2*inc)
fprintf( 'Maximum absolute error*e+13 for CRRA utility
    : %.4f \n', accnonlin*10^13)
% Maximum percentage deviation
maxdev = max(abs(ynonlinsp - ylin)./ylin);
fprintf(  'The maximum percentage deviation is %.2f
    percent \n', maxdev*100)

function [yact, yapp, yhat] = chebyshev_approx( fun, a
    , b, m, n, funtype, x)
% [yact, yapp, yhat] = chebyshev_approx( fun, a, b, m,
     n, funtype, x)
% USAGE: Chebychev interpolation
% INPUT:
%     fun   := function handle, e.g., @exp(-x)
%   [a, b]  := domain on which fun is interpolated
%       m   := nb. of nodes, j = 1,...,m
%       n   := degree of chebyshev polynomial; n.b.: n
    < m
% funtype   := 'explicit' or 'implicit' function
% OUTPUT:
%    coeff  := Chebyshev coefficients alpha_i, i =
    0,...,n
%     xhat  := Chebyshev nodes
%     yhat  := Function values at Chebyshev nodes

%% (0) Initialisation
if n > m
```

```
    error( 'Error. It must hold that n < m.' )
end

%% (1) Compute row vector of m Chebyshev nodes in
    [-1,1]
row = 1:m;
tmp = ( 2*row - 1 )*pi;
zhat = - cos( tmp / (2*m) );

%% (2) Rescale Chebyshev nodes to [a,b]
xhat = a + .5*( b - a )*( zhat + 1 );

%% (3) Evaluate function at Chebyshev nodes
if strcmp(funtype, 'implicit') % implicit optimal
    consumption function
    % Plug in xhat for W
    % Calculate actual values of y for x instead of
        nodes only
    tmp2 = length(xhat);
    tmp3 = length(x);
    yhat = ones(1,tmp2);
    yact = ones(1,tmp3);
    for i = 1:tmp3
       Wnew = x(i);
       myfunnew= @(C0) fun(Wnew,C0);
       x0new=0;
       yact(i)=fzero( myfunnew,x0new);
    end
    for j = 1:tmp2
        W = xhat(j);
        myfun = @(C0) fun(W,C0);
        x0 = 0;
        yhat(j) = fzero( myfun,x0 ); % Rootfinder
            evaluates C0(W)
    end
else % exlicit optimal consumption function
    yhat = feval( fun, xhat );
    yact = feval( fun, x ); %same here
end

%% (4) Polynomial coeffs are solution to linear
    equation Tx*coeff = yhat
% Construct interpolation matrix Tx of size m*(n+1)
Tx = ones(m, n+1); % Returns a vector of ones only if
    n = 0
if n >= 1
```

```
    Tx(:,2) = xhat';
end
% Recursively define rest of matrix Tx
if n >= 2
    for j = 3:(n+1)
        Tx(:,j) = 2*xhat*Tx(:,j-1) - Tx(:,j-2);
    end
end
% Then, polynomial coefficients are given by
coeff = Tx\yhat';

%% Evaluate approximation yapp for larger x
tmp4 = length(x);
Txnew = ones(tmp4, n+1); % Returns a vector of ones
    only if n = 0
if n >= 1
    Txnew(:,2) = x';
end
% Recursively define rest of matrix Tx
if n >= 2
    for j = 3:(n+1)
        Txnew(:,j) = 2*x*Txnew(:,j-1) - Txnew(:,j-2);
    end
end
yapp = Txnew*coeff;

end
```

C Code to Question 3

Main code:

```
%PS4P3
clear;
close all;
clc;

%variable declaration
n=15;  %number of nodes;
f=@simplefQ4P3;
f_const=@simplefQ4P3const;
%Set up Gamma space
xmin=1;
xmax=50;
b=linspace(xmin,xmax,1000);
```

```
b=b';

for i=1:2
    if i==1

%Unconstrained alpha
approximated_alpha=chebi(f,b,n,xmin,xmax);
real_alpha=simplefQ4P3(b);
difference= abs(approximated_alpha - real_alpha);
alphaspline=spl(f,b,n,xmin,xmax);
difference_spline= abs(alphaspline - real_alpha);
    else
%Constrained alpha
approximated_alpha=chebi(f_const,b,n,xmin,xmax);
real_alpha=simplefQ4P3const(b);
difference= abs(approximated_alpha - real_alpha);
alphaspline=spl(f_const,b,n,xmin,xmax);
difference_spline= abs(alphaspline - real_alpha);
    end
figure
subplot(2,2,1)
plot(b,approximated_alpha,b,real_alpha)
title("Chebyshev Approximated and True Values")
line([min(b),max(b)],[0,0],'Color','red','LineStyle','
    --')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','
    --')
legend("Chebyshev Approximation","True Values","\alpha
     Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,2)
plot(b,real_alpha)
title("True Value")
line([min(b),max(b)],[0,0],'Color','red','LineStyle','
    --')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','
    --')
legend("True Values","\alpha Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,3)
plot(b,approximated_alpha)
line([min(b),max(b)],[0,0],'Color','red','LineStyle','
--')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','--')
title("Chebyshev Approximated Values Only")
legend("Chebyshev Approximation","\alpha Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,4)
plot(b,difference)
title("Difference in Absolute Terms (Chebyshev)")
xlabel("\gamma")
ylabel("Optimal \alpha")

figure
subplot(2,2,1)
plot(b,alphaspline,b,real_alpha)
title("Spline Approximated and True Values")
line([min(b),max(b)],[0,0],'Color','red','LineStyle','--')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','--')
legend("Spline Approximation","True Values","\alpha Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,2)
plot(b,real_alpha)
title("True Value")
line([min(b),max(b)],[0,0],'Color','red','LineStyle','--')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','--')
legend("True Values","\alpha Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,3)
plot(b,alphaspline)
line([min(b),max(b)],[0,0],'Color','red','LineStyle','--')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','--')
title("Spline Approximated Values Only")
```

```
legend("Spline Approximation","\alpha Bound",'location
    ','best')
xlabel("\gamma")
ylabel("Optimal \alpha")

subplot(2,2,4)
plot(b,difference_spline)
title("Difference in Absolute Terms (Spline)")
xlabel("\gamma")
ylabel("Optimal \alpha")

end

% Adjust grid for Spline Interploation:
[V, I]=max(difference_spline);
alphaspline_adapt=spladapt(f_const,b,n,xmin,xmax,I,b);
difference_spline_adapt= abs(alphaspline_adapt -
    real_alpha);

figure
subplot(2,1,1)
plot(b,alphaspline_adapt,b,real_alpha)
title("Spline Approximated with adjusted grid and True
     Values")
line([min(b),max(b)],[0,0],'Color','red','LineStyle','
    --')
line([min(b),max(b)],[1,1],'Color','red','LineStyle','
    --')
legend("Spline Approximation","True Values","\alpha
    Bound",'location','best')
xlabel("\gamma")
ylabel("Optimal \alpha")
subplot(2,1,2)
plot(b,difference_spline_adapt)
title("Difference in Absolute Terms (Spline)")
xlabel("\gamma")
ylabel("Optimal \alpha")

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

  fspacespllin=fundefn('spli',m-1,xmin,xmax,1);  %
      linear splines
  fspacesplcub=fundefn('spli',m-1,xmin,xmax,3);  % cubic splines
  %distance=(xmax-xmin)/(m-1);
  nodesspl=zeros(m,1);
  ynodes=zeros(m,1);
  %nodes
  distance_nodes=round((xmin-b(I(1,1),1))/(xmax-xmin)*(m-1),0);
  distance=(xmin-b(I(1,1),1))/distance_nodes;

  for i=1:m
    nodesspl(i)=xmin+(i-1)*distance;   %eqidistant nodes
    ynodes(i)=fct(nodesspl(i));        %fct values at nodes
  end

  %calculate the matrix of basis functions
  Bspllin=funbas(fspacespllin,nodesspl);
  Bsplcub=funbas(fspacesplcub,nodesspl);

  %get polynomial coefficients
  cspllin=Bspllin\ynodes;
  csplcub=Bsplcub\ynodes;

  %approximate the function
  yspllin=funeval(cspllin,fspacespllin,x);
  ysplcub=funeval(csplcub,fspacesplcub,x);

end
```

The function to be approximated:

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

With constraint:

```
function a=simplefQ4P3const(x)
 b=(9.^(1./x).*1.02-1.02)./(0.06+0.06.*9.^(1./x));
 a=NaN(length(b));
 for j=1:length(b)
 if b(j)>1
     a(j)=1;
 else
     a(j)=b(j);
 end
end
```

Chebyshev:

```
function [y]=chebi(fct,x,m,xmin,xmax)

%Chebyshev interpolation m=n
%define function space using fundefn

fspace=fundefn('cheb',m,-1,1);

anodes=NaN(m,1);
nodes=NaN(m,1);

%create nodes and evaluate fct values at nodes
for j=1:m
  nodes(j,1)=-cos((2*j-1)*pi/(2*m)); %Chebyshev nodes
  nodes(j,1)=(nodes(j,1)+1)*(xmax-xmin)/2+xmin; %
      Rescale nodes
  anodes(j,1)=fct(nodes(j,1)); %Evaluate function at
      nodes
end

figure
scatter(nodes,ones(m,1))
title("Distribution of Chebyshev Nodes")

%calculate  matrix of basis functions
B=funbas(fspace,nodes);

%solve for polynomial coefficients
c=B\anodes;

%approximate function a
y=funeval(c,fspace,x);

end
```

Spline:

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

```

```
  fspacesplin=fundefn('spli',m-1,xmin,xmax,1);  %
     linear splines
  fspacesplcub=fundefn('spli',m-1,xmin,xmax,3);  %
     cubic splines
  distance=(xmax-xmin)/(m-1);
  nodesspl=zeros(m,1);
  ynodes=zeros(m,1);
  %nodes
  for i=1:m
    nodesspl(i)=xmin+(i-1)*distance;   %eqidistant
       nodes
    ynodes(i)=fct(nodesspl(i));        %fct values at
       nodes
  end

  %calculate the matrix of basis functions
  Bspllin=funbas(fspacespllin,nodesspl);
  Bsplcub=funbas(fspacesplcub,nodesspl);

  %get polynomial coefficients
  cspllin=Bspllin\ynodes;
  csplcub=Bsplcub\ynodes;

  %approximate the function
  yspllin=funeval(cspllin,fspacespllin,x);
  ysplcub=funeval(csplcub,fspacesplcub,x);

end
```