Numerical Analysis
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The topics of the course are:

- Floating-point arithmetic: different sources of the computational error; absolute vs relative errors; the floating point representation of real numbers; the round-off unit; the machine epsilon; floating-point operations; over- and under-flow; numerical cancellation.

- Numerical approximation of nonlinear equations: the bisection and the Newton methods; the fixed-point iteration; convergence analysis (global and local results); order of convergence; stopping criteria and corresponding reliability; generalization to the system of nonlinear equations (hints).

- Numerical approximation of systems of linear equations: direct methods (Gaussian elimination method; LU and Cholesky factorizations; pivoting; sparse systems: Thomas algorithm for tridiagonal systems); iterative methods (the stationary and the dynamic Richardson scheme; Jacobi, Gauss-Seidel, gradient, conjugate gradient methods (hints); choice of the preconditioner; stopping criteria and corresponding reliability); accuracy and stability of the approximation; the condition number of a matrix; over- and under-determined systems: the singular value decomposition (hints).

- Numerical approximation of functions and data: Polynomial interpolation (Lagrange form); piecewise interpolation; cubic interpolating splines; least-squares approximation of clouds of data.

- Numerical approximation of derivatives: finite difference schemes of the first and second order; the undetermined coefficient method.

- Numerical approximation of definite integrals: simple and composite formulas; midpoint, trapezoidal, Cavalieri-Simpson quadrature rules; Gaussian formulas; degree of exactness and order of accuracy of a quadrature rule.

- Numerical approximation of ODEs: the Cauchy problem; one-step methods (forward and backward Euler and Crank-Nicolson schemes); consistency, stability, and convergence (hints).

Contents

Chapter 1

Introduction to MATLAB

1.1 Main MATLAB operators

Assignment operator:

```
% Print output
a = 1
% Does not print output
b = 2;
```

The active variables can be found in the workspace and the value can be checked on the command window with:

```
% Value of all variables
whos
% Value of a
whos a
```

If you want to save the file:

```
% Save the command history
diary file_name.txt
% Save the whole workspace
save file_name
% Save only the variable a
save file_name_only_a a
% Load only the variable a
load file_name_only_a
% Load the whole workspace
load file_name
```

It is possible to clear variables with the following commands:

```
% Clear only the variable a
clear a
% Clear the whole workspace
clear all
```

1.2 Vector and matrices

Most of the entities in MATLAB are matrices, even real numbers. The matrices can be defined in the following ways:

```
% Row vector definition
c = [1 2 3]
% Column vector definition
c = [1; 2; 3]
% Vectorn transposition
c = [1 2 3]'
% 2D matrix definition
D = [ 1 2 3;
       4 5 6;
       7 8 9 ]
```

It is also possible to define various types of matrices:

```
% Zeros vector/matrix
A = zeros(row_length,column_length)
% Ones vector/matrix
A = ones(row_length,column_length)
% Identity matrix
A = eye(row_length,column_length)
% Diagonal matrix
d = [1:4]
D = diag(d)
% Set a not principal diagonal
D = diag(d, diagonal_index)
% Select only upper o lower trinagular
Ml = tril(M)
Mu = triu(M)
% Access an element in vector
C(1)
% Access an element in matrix
```

```
C([2,3]);
% Access a part of the matrix
Q(rows,columns)
% Access the element in position (n,m)
Q(end, end)
% Dimension of a matrix
length(a);
numel(b);
size(a);
```

The operations on vectors are done in the following way:

```
% Given two row vectors a and b
% Vector sum
a + b
% Vector difference
a - b
% Scalar product
a * b'
dot(a,b)
% Tensor product
a' * b
% Elementwise product
a .* b
% Elementwise division
a ./ b
% Elementwise exponentiation
a .^ 2
```

The operations on matrices are done in the following way:

```
% Givcen two matrices A and B (both 3x2)
% Matrix sum
A + B
% Matrix difference
A - B
% Matrix product
K * L'
% Elementwise product
A .* B
% Elementwise division
A ./ B
% Elementwise exponentiation
A .^ 2
```

```
% Power matrix (useful only square)
A ^ 2
% Other useful values of the matrices
% Determinant
det(A)
% Trace
trace(A)
% Inverse of small matrix
inv(A)
% Given a column vector b ths olutio of Ax=b
A \ b
```

The function used to plot a graph are the following:

```
% To plot y=f(x) in [a,b]
x = a:step_length:b;
y = f(x);
figure
plot(x,y,color)
% To add y2=f2(x) in [c,d]
hold on
x2 = c:step_length:d;
y2 = f2(x);
plot(x2,y2,color)
% Show graph's grid
grid on
% Set the axis limit
axis([xmin xmax ymin ymax])
% Set the same scaling for both axis
axis equal
```

To handle functions the commands are:

```
% Define a function handle to g(x)
f = @g(x);
% Evaluation of f in a
f(a)
% Define an anonymous function
% It is useful to modify other functions
f = @(argument-list) expression
```

The operators that u logical values are:

```
% Smaller than
```

```
a < b
% Greater than
a > b
% Smaller or equal than
a <= b
% Equal to
a == b
% Different from
a ~= b
% And
(a < b) & (b > c)
% Or
(a < b) | (b > c)
```

The control-flow statement are:

```
% if-then-else statements
if (condition1)
    block1
elseif (condition2)
    block2
else
    block3
end
% for loops
for (index=start:step:end)
    instruction block
end
% while loops
while (condition)
    instruction block
end
```

There are two categories of m-files:

- Scripts: these files contain instructions that are executed in sequence in the command line if the script file is called. The variables are saved in the current workspace.
- Functions: they take some input arguments and return some outputs after a series of instructions are performed. The variables defined in the function are local to the scope of the function itself.

Chapter 2

Laboratory I

Exercise 1

Define the row vector:

$$\bar{v_k} = [1, 9, 25, \ldots, (2k+1)^2] \in \mathbb{R}$$

with $k = 8$ using the following strategies:

1. A for loop to define one by one each element of the vector.
2. The vector syntax to build it in just one shot.

Answer of exercise 1

```
k = 8;

% For loop strategy
vk = zeros(1, k+1);
for (ii = 0:k)
    vk(ii+1) = (2*ii + 1)^2;
end
vk
% Vector syntax
vk = [1:2:2*k+1].^2;
```

Exercise 2

Define a function which, for an input value k, returns the corresponding vector v_k as defined in the previous exercise.

Answer of exercise 2

```
function vk = ex_1_2(k)

vk = [1:2:2*k+1].^2;

end
```

Exercise 3

Using the function of the previous exercise write another function that returns, for a generic value k, the $2(k+1) \times 2(k+1)$ matrix.

$$m_k = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sqrt[2]{2} & 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \sqrt[3]{2} & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \sqrt[4]{2} & 0 & 0 & \cdots & 0 & 9 \\ 0 & 0 & 0 & 0 & \sqrt[5]{2} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt[6]{2} & \cdots & 0 & 25 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & \sqrt[(2k+1)]{2} & 0 \\ 1 & 1 & 9 & 9 & 25 & 25 & \cdots & (2k+1)^2 & (2k+1)^2 \end{bmatrix}$$

Answer of exercise 3

```
function Mk = ex_1_3(k)

% Create a diagonal matrix with the right elements and dimension
Mk = diag(2.^(1./[1:2*(k+1)]))
% The bottom right element will be overwritten

% Last column
Mk(2:2:end, end) = ex_1_2(k)

% Last line
Mk(end, 1:2:end) = ex_1_2(k)
Mk(end, 2:2:end) = ex_1_2(k)

end
```

Exercise 4

Compare the results of the following code segments:

```
% Code A
x = 0;
while (x ~= 1)
    x = x + 1/16
end

% Code B
x = 0;
while (x ~= 1)
    x = x + 0.1
end
```

Answer of exercise 4

```
% Code A
x = 0;
while (x ~= 1)
    x = x + 1/16
end

% Code B
x = 0;
k = 1;
format long
while (x ~= 1)
    k
    x = x + 0.1
    if k==10 || k==11
        x-1
    end
    k=k+1;
    if k==15
        break
    end
end
```

Exercise 5

Find the machine epsilon by implementing an ad hoc procedure. Comment and justify the obtained results.

Answer of exercise 5

```
% Pay attention, you cannot use "eps" since it is a built-in
    variable
k = 0;
EPS = 1/2;
while (1 + EPS) > 1
    EPS_old = EPS;  % keep track of the value
    EPS = EPS / 2;
    k = k + 1;
end
format long
EPS_old    % computed with the ad-hoc procedure
(1 + EPS_old) > 1
EPS
(1 + EPS) > 1
k       % Number of iterations, which is also the numer of digits
    in the mantissa, according to the standard
eps        % Octave/MATLAB built-in
```

Exercise 6

Given the following code:

```
realmax
a = 1.0e+308;
b = -a;
c = 1.1e+308;
(a + b) + c
(a + c) + b
```

Explain why the second result is Inf.

Answer of exercise 6

```
realmax
a = 1.0e+308;
b = 1.1e+308;
c = -a;
(a + b) + c
a + (b + c)
```

Exercise 7

Consider the following function:

$$f(x) = \frac{e^x - 1}{x}$$

1. Evaluate $f(x)$ for values of x around zero (try with $x_k = 10^{-k}$, $k \in [1, 20]$). What do you obtain? Explain the results.

2. Propose an approach for fixing the problem. (Hint: Use Taylor expansions to get an approximation of $f(x)$ around $x = 0$).

3. How many terms in the Taylor expansion are needed to get double precision accuracy (16 decimal digits) $\forall x \in \left[0, \frac{1}{2}\right]$?

Answer of exercise 7

```
%  Evaluate f(x) for values of x around zero (try with xk =
   10^{-k}, k in [1,20]). What do you obtain? Explain the results.


k = [1:20]';
x = 10.^(-k);
f = @(x) (exp(x) - 1) ./ x;
format long
[k x f(x)]
figure;
plot(x, f(x), '*')

%  The computed f(xk) have the expected behavior until k = 7, in
   agreement with
%  lim{x -> 0} f(x) = 1.
%
% k = 8 to k = 15: the sequence oscillates with increasing
   amplitude (bigger and bigger).
% k = 16: the result becomes 0! The cause of these effects is the
   loss of accuracy due to
% numerical cancellation.

% The numerical cancellation is the most serious consequence
   related to the
% floating point representation, that is the representation of the
   real
```

```
% numbers with a finite precision.
% It occurs when you subtract two numbers which are very close to
    each other.
% So you lose significant digits and consequently accuracy of the
    finite
% representation
% a = 1.11e-15, b = 1.12e-15, b-a=1.0e-17_ b-a very close to the
    machine
% epsilon

% For k = 8, ..., 15 e^{xk} and 1 are very close to each other,
% still being different in floating point arithmetic.
% We are subtracting them and the result suffers of a great
    relative error.
% Moreover, we are dividing by xk, that approaches 0, hence the
    error is magnified.

% Finally, for k = 16, e^{xk} and 1 are too close to have a
    different representation
% in the floating ploint arithmetic (eps ~ 10^{-16}), hence the
    difference is 0.
%
%  Write the Taylor expansion of e^x around x0 = 0, truncating it
    at the fifth order:
%
%  e^x = sum_{k=0}^{+infinity} {f^{(k)}(x0)}/{k!} (x-x0)^k =
%       = sum_{k=0}^{5}       {f^{(k)}(x0)}/{k!} (x-x0)^k + O(x^6) =
%      = 1 + x + {x^2}/{2} + {x^3}/{6} + {x^4}/{24} + {x^5}/{120} +
    O(x^6)
%
%  and substitute it in f(x).
%
%  f(x) = 1 + {x}/{2} + {x^2}/{6} + {x^3}/{24} + {x^4}/{120} +
    O(x^5)
%
%  With this formula, at least for x > 0, all the terms have the
    same sign, so no numerical cancellation occurs.


f_taylor_5 = @(x) 1 + 1/2*x + 1/6*x.^2 + 1/24*x.^3 + 1/120*x.^4;

[k x f(x) f_taylor_5(x)]
```

```
%%% How many terms in the Taylor expansion are needed to get
    double precision accuracy (16 decimal digits) for all x in [0,
    1/2]?
%
%  The error in the approximation of a function g(x) at a point x
    = a with its Taylor expansion up to order n is
%  E[g,n,a](x) = {g^{(n+1)}(xi)}/{(n+1)!} (x-a)^{n+1} for at least
    a xi
%  s.t. |xi -a| < |x -a|.
%
%  In this case
%
%  E[e^x,n+1,0](x) = {e^{xi}}/{(n+2)!} x^{n+2} for xi in [0,x]
%
%  and therefore, substituting in f(x)
%
%  E[f,n,0](x) = {E[e^x,n+1,0](x)}/{x} = {e^{xi}}{(n+2)!} x^{n+1}
    for xi in [0,x]
%
%  We want
%
%  E[f,n,0](x) < eps for all x in [0, 0.5]:
%
%  since both e^xi and x^{n+1} are increasing functions, it's
    enough to choose xi = x = 0.5 and seek the first n* such that
%
%  {e^{1/2}}/{(n*+2)!} (1/2)^{n*+1} < eps


format short e
n = [1:20]';
err = 1./factorial(n+2) .* (0.5).^(n+1).*exp(0.5);

[n err]

%   Therefore n* = 13$.
```

Exercise 8

The sequence:

$$1, \frac{1}{3}, \frac{1}{9}, \ldots, \frac{1}{3^n}, \ldots$$

can be generated with the following recursive relations:

$$\begin{cases} p_n = \dfrac{10}{3} p_{n-1} = p_{n-2} \\ p_1 = \dfrac{1}{3},\ p_0 = 1 \end{cases}$$

$$\begin{cases} q_n = \dfrac{1}{3} q_{n-1} \\ q_0 = 1 \end{cases}$$

1. Implement the two relations in order to generate the first 100 terms of the sequence.
2. Study the stability of the two algorithms and justify the obtained results.

Answer of exercise 8

```
           % Implement the two relations in order to generate the
              first 100 terms of the sequence.

% First recursive relation

p(1) = 1;
p(2) = 1/3;
for i = 2:100
    p(i+1) = 10/3*p(i) - p(i-1);
end

figure
subplot(2,1,1)
plot(0:100, p, 'LineWidth',3)
% gca return the current axes
% setting the fontsize on axes
set(gca,'FontSize',16)
xlabel('n','FontSize',16)
ylabel('p_n','FontSize',16)
% The sequence explodes!
```

```
% Second recursive relation

q(1) = 1;
for i=1:100
    q(i+1) = 1/3*q(i);
end

subplot(2,1,2)
plot(0:100, q, 'LineWidth',3)
set(gca,'FontSize',16)
xlabel('n','FontSize',16)
ylabel('q_n','FontSize',16)
% The sequence is ok.


%  Study the stability of the two algorithms and justify the
    obtained results.
%
%  Error analysis: since we use finite precision numbers, we know
    the initial
% data up to an error eps. Actually, if we start from 1, we know
    it exactly,
% but we may introduce anyway errors in the following steps.
%  Indeed, denoting by fl(pi) the floating point representative
    for pi:
%
%  fl(p0) = p0 + eps, fl(p1) = p1 + eps.
%
%  It follows
%
%  fl(p2) = {10}\{3} fl(p1) - fl(p0) = {10}\{3}(p1 + eps)- (p0 +
    eps) = p2 + {7}\{3} eps
%  fl(p3) = {10}\{3} fl(p2) - fl(p1) = {10}\{3}(p2 + {7}\{3}eps) -
    (p1 + eps) = p2 + {61}\{9} eps
%  fl(p4) = ...
%
%  The error is amplified!
%
%  Instead, with the second formula
%
%  fl(q0) = q0 + eps
%  fl(q1) = {1}\{3} fl(q0) = {1}\{3} q0 + {1}\{3} eps = q1 +
    {1}\{3} eps
```

```
%  fl(q2) = {1}\{3} fl(q1) = {1}\{3} q1 + {1}\{9} eps = q2 +
   {1}\{9} eps
%  fl(q3) = ...
%
%  The error is reduced!
```

Exercise 9

Find the minimum positive number representable in MATLAB/Octave by implementing an ad hoc procedure. Compare with *realmin*.

Answer of exercise 9

```
k = 0;
zero = 1/2;
while (0 + zero) > 0
  zero_old = zero;
  zero = zero / 2;
  k = k + 1;
end

% Remember: realmin is the minimum !normalized! positive floating
    point number
realmin
% The minimum positive floating point number is instead a
    !denormalized! number
zero_old % < realmin
k
```

Exercise 10

1. Use Taylor polynomial approximation to avoid the loss of significance errors in the following function when x approaches 0:

$$f(x) = \frac{1 - \cos(x)}{x^2}$$

2. Reformulate the following function $g(x)$ to avoid the loss of significance error in its evaluation for increasing values of x towards $+\infty$:

$$g(x) = x\left(\sqrt{x+1} - \sqrt{x}\right)$$

Answer of exercise 10

```
format long


%  Use Taylor polynomial approximation to avoid the loss of
   significance
%  errors in the following function when x approaches 0
%
%    f(x) = {1-cos(x)}/{x^2}
%
%  The limit of f(x) = {1-cos(x)}/{x^2} as x -> 0 is well known:
%
%  lim {x -> 0} {1-cos(x)}/{x^2} = {1}/{2}


clear all, close all, clc
k = [1:30]';
x = 2.^(-k);
f = @(x) (1 - cos(x))./(x.^2);
[k x f(x)]


%  If we try to evaluate f(x) directly with a sequence xk = 2^{-k}
   that approaches 0, we notice anomalous behaviour at k = 13, k =
   27, k = ...
%  The reason is numerical cancellation of the terms on the
   numerator. We can use Taylor expansion of cos(x) for x -> 0 to
   obtain a better approximation. Indeed
```

```
%
%  cos(x) = 1 - {1}/{2} x^2 + {1}/{24} x^4 - {1}/{720} x^6 + o(x^8)
%
%  so that
%
%  f(x) = {1}/{2} - {1}/{24} x^2 + {1}/{720} x^4 + o(x^6)


f_taylor_4 = @(x) 1/2 - x.^2/24 + x.^4/720;

[k x f(x) f_taylor_4(x)]

% The obtained formula is more stable that the direct evaluation
    of f(x), and the computed values approach {1}/{2}.


%  Reformulate the following function g(x) to avoid the loss of
%  significance error in its evaluation for increasing values of x
    towards +infinity.
%
%  g(x) = x (sqrt{x+1} - sqrt{x}).
%
%  It is well known that
%
%  lim{x -> +infinity} g(x) = +infinity


clear all, close all, clc
format short e

k = [1:20]';
x = 10.^(k);

g = @(x) x.*(sqrt(x+1) - sqrt(x));

[k x g(x)]


%  If we try to evaluate g(x) for a sequence xk = 10^{k} that
    approaches 0, we notice anomalous behaviour at k = 16, k = ...,
    for which the computed value is 0. This happens since, in
    floating point representation, x16 + 1 = x16!
%
```

```
% Rationalization of the expression in bracket solves this
  numerical cancellation: indeed, multiplying the numerator and
  the denominator by sqrt{x+1} + sqrt{x}
%
% g(x) = {x (sqrt{x+1} - sqrt{x})(sqrt{x+1} +
  sqrt{x})}/{(sqrt{x+1} + {x})} = {x}/{(sqrt{x+1} + sqrt{x})}.


g2 = @(x) x./(sqrt(x+1) + sqrt(x));

[k x g(x) g2(x)]

% This formula is more stable and does not suffer of numerical
  cancellation.
```

Exercise 11

We can compute e^{-x} around $x = 0$ using Taylor polynomials in two ways, either using:

$$e^{-x} \approx 1 - x - \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots$$

or using

$$e^{-x} = \frac{1}{e^x} \approx \frac{1}{1 - x - \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots}$$

Which approach is the most accurate?

Answer of exercise 11

```
clear all, close all, clc
format long

f_taylor_pos = @(x) 1 - x + 1/2*x.^2 - 1/6*x.^3 + 1/24*x.^4 -
    1/120*x.^5;
f_taylor_neg = @(x) 1./(1 + x + 1/2*x.^2 + 1/6*x.^3 + 1/24*x.^4 +
    1/120*x.^5);

k = [1:20]';

x_pos = 10.^(-k);
[x_pos f_taylor_pos(x_pos) f_taylor_neg(x_pos)]

x_neg = -10.^(-k);
[x_neg f_taylor_pos(x_neg) f_taylor_neg(x_neg)]


%  No bad behaviour is observed since the 1 in both formulas
    dominates the sum and numerical cancellation does not occur.
%   Therefore both expression can be accepted.
%
%  Instead, if the sum were not dominated by the term 1 (or if you
    didn't notice that),
%   the following combination could have been proposed
%
%  e^{-x} ~ {1}/{1 + x + {1}/{2}x^2 + {1}/{6}x^3 + ...}, x > 0;
%        ~ 1 - x + {1}/{2}x^2 - {1}/{6}x^3 +  ... , x <=0.
%
%  The choice of the previous expression for x > 0 should not
    display any numerical cancellation because all the terms in the
```

```
sum are positive. Similarly, the choice for x < 0 grants that
the terms -x^{2k+1} are positive, again avoiding any numerical
cancellation.
```

Exercise 12

Consider the following integral:

$$I_n(\alpha) = \int_0^1 \frac{x^n}{x+\alpha}\,dx \qquad \forall n \in \mathbb{N}, \alpha > 0$$

1. Give an upper bound for $I_n(\alpha)$, $\forall n \in \mathbb{N}, \alpha > 0$.
2. Prove the following recursive relation between $In(\alpha)$ and $I_{n-1}(\alpha)$:

$$\begin{cases} I_n(\alpha) = -\alpha I_{n-1}(\alpha) + \dfrac{1}{n} \\ I_o(\alpha) = \ln\left(\dfrac{\alpha+1}{\alpha}\right) \end{cases}$$

3. Employing the previous relation, compute $I_40(\alpha = 8)$ and comment the obtained results.
4. Write a numerically stable recursive relation for $I_40(\alpha = 8)$.

Answer of exercise 12

```
clear all;
format short e

n = 40;

% alpha = 1/8 not requested in the homework
for (alpha = [1/8, 8])
    I(1) = log((alpha+1)/alpha);
    for (k = 1:n)
        I(k+1) = -alpha*I(k) + 1/k;
    end
    recursion_integral = I(n+1);
    upper_bound = (1/alpha)*(1/(n+1));
    exact_integral = quadl(@(x) (x.^n)./(x+alpha), 0, 1, 1e-16); %
        Exact value (not requested)
    [recursion_integral, upper_bound, exact_integral]
end


%  The recursive approximation of I{40}(alpha = 8) is 1.6389 ...
   10^{18}: this result is for sure incorrect because it violates
   the upper bound for $I_{40}(\alpha = 8)$.
```

```
%
% This is due to the finite precision we use and the error
   propagation: the initial value I0(alpha) =
   ln({alpha+1}/{alpha}) is represented with a certain error eps;
   denoting by fl(y) the floating point representation of the
   number y, we have
%
% fl(I0(alpha)) = I0(alpha) + eps,
% fl(I1(alpha)) = -alpha fl(I0(alpha)) + 1 = -alpha (I0(alpha) +
   eps) + 1 = I1(alpha) - alpha eps,
% ...
% fl(Ik(alpha)) = In(alpha) + (-1)^k alpha^k eps,
%
% Therefore, at the final (n-th) step the error is multiplied by
   a factor alpha^n, which result in an amplification of the error
   if alpha > 1. Instead, for alpha < 1, the error is damped and
   the recursive relation is stable (e.g. see the previous test
   for alpha = 1/8).
%
% Now write a numerically stable recursive relation for
   I{40}(alpha = 8).
%
% The idea is to transform the factor alpha on the RHS of the
   recursive relation in a factor {1}/{alpha}; this can be
   achieved inverting the recursive relation:
%
%    I{k-1}(alpha) = - {1}/{alpha} I{k}(alpha) + {1}/{k alpha}
%    lim{k -> +infinity} I{k}(alpha) = 0
%
% The final value lim{k -> +infinity} I{k}(alpha) is equal to
   zero because 0 <= I{k}(alpha) <= {1}/{alpha} {1}/{k+1} -> 0, as
   k -> + infinity.


clear all;
n = 40;
big = 1000;

% alpha = 1/8 not requested in the homework
for (alpha = [1/8, 8])
    I(big + 1) = 0;
    for (k = big:-1:n+1)
        I(k) = -1/alpha*I(k+1) + 1/(k*alpha);
```

```
    end
    recursion_integral = I(n+1);
    upper_bound = (1/alpha)*(1/(n+1));
    exact_integral = quadl(@(x) (x.^n)./(x+alpha), 0, 1, 1e-16); %
        Exact value (not requested)
    [recursion_integral, upper_bound, exact_integral]
end

%  The recursion is now stable for alpha > 1 (and note that, now,
    an additional error is present, because the final condition can
    be set for an arbitrary large k = k* instead of k = +infinity).
```

Exercise 13

Given the following sequence:

$$\begin{cases} x_{n+1} = 2^{n+1}\left[\sqrt{1+\dfrac{x_n}{2^n}} - 1\right] \\ x_0 > -1 \end{cases}$$

for which $\lim_{n\to+\infty} x_n = \ln(1+x_0)$

1. Set $x_0 = 1$, compute $x_1, x_2, \ldots, x_{71}$ and explain the obtained results.
2. Transform the sequence in an equivalent one that converges to the theoretical limit.

Answer of exercise 13

```
% Set x0 = 1, compute x1, x2, ... , x{71} and explain the obtained
    results.

clear all, close all, clc
format long

n_max = 71;
x = zeros(n_max+1, 1);

x(1) = 1;        % x0 set to 1
for n = 0:n_max-1
  x(n+2) = 2^(n+1)*(sqrt(1 + x(n+1)/2^(n)) - 1); % Pay attention to
      the indexing!!!
end
x(end)
x_lim = log(1 + x(1))

figure
hold on, box on
plot([0:71], x, 'bx-','Linewidth',3)
plot([0:71], x_lim*ones(n_max+1,1)', 'r-','Linewidth',3)
axis([-1 72 -0.05 1.05])
set(gca,'LineWidth',2)
set(gca,'FontSize',16)

% When computing the sequence with Octave/MATLAB we find that
    x{71} differs a lot from ln(1+x0) = ln(2). The plots also shows
    that xn = 0, for all n >= n* = 52: this is due to numerical
```

```
    cancellation effects, because at n = n* it holds
    {x{n*}}/{2^{n*}} < eps, hence in finite precision arithmetic
    x{n*} = 0!

x(53)

%%

%  The value of n* can also be computed with the following
    argument: since (hopefully) numerical cancellation will occur
    for large n
%  x{n} ~ ln(1+x0)
%  therefore
%  {x{n}}/{2^{n}} < eps is approximately equivalent to
    {ln(1+x0)}/{2^{n}} < eps
%  which can be solved for n, finding
%  n > log2({ln(1+x0)}/{eps})


x_0 = 1;
n = log(log(1 + x_0)/eps)/log(2)


%  Transform the sequence in an equivalent one that converges to
    the theoretical limit.
%
%  After a razionalization, the recurrence can be written as
%  x{n+1} = 2^{n+1} [ sqrt{1+{xn}/{2^{n}}} - 1 ] = {2 xn}{ sqrt{1
    + {xn}/{2^{n}}} + 1}


clc, clear all

n_max = 71;
x = zeros(n_max+1, 1);

x(1) = 1;          % x0 set to 1
for n = 0:n_max-1
  x(n+2) = 2*x(n+1)/(sqrt(1 + x(n+1)/2^(n)) + 1);
end
x(end)

%   The error in this case is
```

```
x(end) - log(1 + x(1))

figure
hold on, box on
plot([0:n_max], x,'bx-','Linewidth',3)
plot([0:n_max], log(1+x(1))*ones(n_max+1), 'r-','Linewidth',3)
axis([-1 72 0.67 1.02])
set(gca,'LineWidth',2)
set(gca,'FontSize',14)

%   Using the previous calculations we know that (1 + xn/2^n)
    approximately 1 for n >= 52, so x(n+1) = 2*xn/(1 + 1) = xn
```

Chapter 3

Laboratory session II