8. NUMERICAL APPROXIMATION OF FUNCTIONS

Objectives of the paper:

- Familiarizing with some of the numerical approximation methods based on the use of polynomials,
- Fixing knowledge regarding the solving of the problem of numerical approximation of functions by different methods, using the Matlab programming environment, by studying some examples and solving some problems.

It is recommended to go through Annex M8 before studying paragraphs 8.1 and 8.2.

8.1. ELEMENTS REGARDING THE NUMERICAL APPROXIMATION OF FUNCTIONS IN MATLAB

Numerical approximation of functions

Let `f` be a function specified by a set of *n* points (x_i, $y_i=f(x_i)$), i=1,2,...,n.

a. Linear interpolation. Hermite interpolation. Spline interpolation

For the numerical approximation of the function *f* of a real variable through ***linear interpolation***, ***Hermite interpolation*** or ***spline interpolation***, Matlab provides the user with the function *interp1* with the syntax:

```
vy=interp1(x,y,vx,'method')
```

where:

- *x* is the vector of the points $\{x_i\}$;
- *y* is the vector of the points $\{y_i\}$;
- *vx* is the vector of the points where the approximation of the function *f* is desired;
- *vy* is the vector obtained by approximating the function *f* in the points *vx*;
- `method` represents a string of characters that specifies the desired interpolation method:
  - *linear* – for linear interpolation (is the implicit method);
  - *cubic* or *pchip* – for piecewise cubic Hermite interpolation polynomial;
  - *spline* – for cubic spline interpolation.

Spline interpolation can also be realized with the Matlab function `spline`. The syntax of the *spline* function is:

```
vy=spline(x,y,vx)
```

in which the parameters *x,y,vx* and *vy* have the same meanings as for the *interp1* function.

Comment: Actually, the *spline* method of the *interp1* function uses without user involvement the Matlab *spline* function to perform the interpolation.

b. Least squares approximation method

The approximation of the function f with the least squares method is done in Matlab through performing the following two steps:

1. Determining the interpolation polynomial using the Matlab *polyfit* function, which has the syntax:

```
P=polyfit(x,y,m)
```

where:

- *x* is the vector of the points $\{x_i\}$;
- *y* is the vector of the points $\{y_i\}$;
- *m* is the degree of the approximation polynomial, $0 \le m \le n-1$;
- *P* is the vector of the interpolation polynomial coefficients, the coefficients being in decreasing order of the powers of the undetermined of the polynomial.

2. Calculation of the approximation polynomial values at the desired points, using the Matlab *polyval* function, which has the syntax:

```
vy=polyval(P,vx)
```

where:

- *vx* is the vector of points where approximation of the function *f* is desired;
- *P* is the vector of the interpolation polynomial coefficients obtained in the first step;
- *vy* is the vector obtained by approximating the function *f* in the *vx* points with the polynomial *P* values.

8.2. Examples

Example 8.1: Let *f* be a real function of a real variable, specified in the table of values:

Table 8.1. *Tableof values for the example 8.1.*

x_i	0	1.2	1.6	2	2.7	3
$y_i=f(x_i)$	-2.5	0	2	1.7	-4	1

Determine the approximation values of the function*f* for the points 0.7, 1.3, 1.7, 2.5, 2.9, using:

a) linear interpolation method;

b) piecewise cubic Hermite interpolation polynomial method.

Solution: a) The following Matlab program sequence is executed:

```
% vector of the points xi
x=[0 1.2 1.6 2 2.7 3];
% vector for the values yi of the function for the points xi
y=[-2.5 0 2 1.7 -4 1];
% vector of the points where the approximation is performed
vx=[0.7 1.3 1.7 2.5 2.9];
% approximation through linear interpolation
disp('The values of f obtained through linear interpolation:')
vy=interp1(x,y,vx,'linear')
```

After executing the sequence above the following results are obtained:

```
 The values of f obtained through linear interpolation:
 vy=    -1.0417    0.5000    1.9250   -2.3714   -0.6667
```

b) The following Matlab program sequence is executed, which differs from that from the point a) only by the specified interpolation method:

```
% vector of the points xi
x=[0 1.2 1.6 2 2.7 3];
% vector for the values yi of the function for the points xi
y=[-2.5 0 2 1.7 -4 1];
% vector of the points where the approximation is performed
vx=[0.7 1.3 1.7 2.5 2.9];
% approximation through cubic Hermite interpolation
disp(['The values of f obtained through cubic '...
    'Hermite interpolation: '])
vy=interp1(x,y,vx,'cubic')
```

obtaining the values:

```
 The values of f obtained through cubic Hermite interpolation:
 vy=    -1.4777    0.4901    1.9771   -2.9221   -1.3678
```

Example 8.2: Represent graphically in the same graphical window the linear interpolation and cubic spline functions, for the function *f* defined by the following table of values:

Table 8.2. *Table of values for the example 8.2.*

x_i	-1.5	0	1	3
$y_i=f(x_i)$	7.8	5	6.3	6.8

as well as the points in the table.

Solution: The representation interval is given by the lowest value x_i and the highest value x_i: [-1.5,3]. The following Matlab program sequence is executed:

```
% vector of the points xi
x=[-1.5 0 1 3];
% vector for the values yi of the function for the points xi
y=[7.8 5 6.3 6.8];
% vector of the approximation interval points
vx=-1.5:0.1:3;
% approximation through linear interpolation
vy_linear=interp1(x,y,vx,'linear');
```

```
% graphic representations
plot(x,y,'ro',vx,vy_linear,'g',vx,vy_spline,'b--')
legend('points','linear','spline')
```

The resulting graphs are shown in the figure 8.1.:

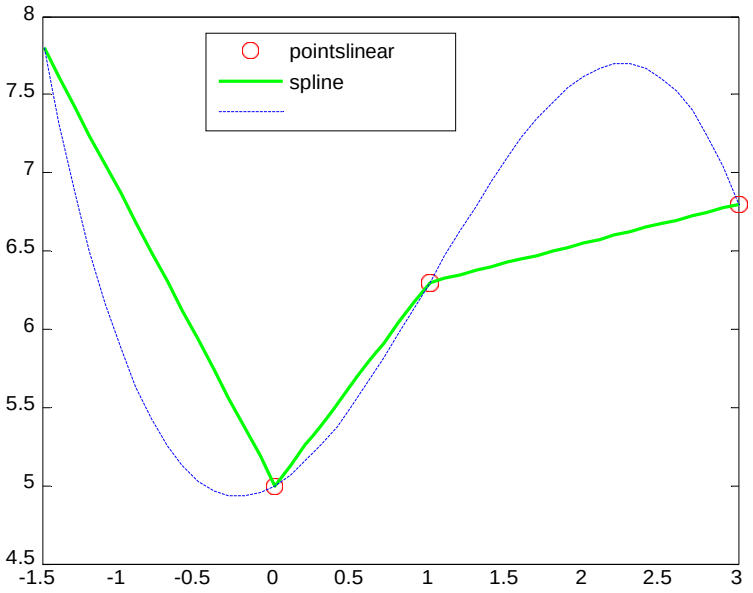

Fig.8.1. *The graphs of the approximation functions from example 8.2.*

Example 8.3: Having the function *f* given by the table of values:

Table 8.3. *Table of values for the example 8.3.*

x_i	-8	-6	-4	-2	0	2	4
$y_i=f(x_i)$	30	10	9	6	5	4	4

a) Approximate, in the sense of the least squares, the function *f* in the points -7, -4.2, -0.75, 1, 2.15, 3, using parabolic regression, cubic regression and 6th order polynomial regression.

b) Represent graphically in the same graphical window the approximation polynomials of point a) and the points in the table.

Solution: a) To approximate the function given by the least squares method (polynomial regression), the two steps specified in the paragraph 8.1 are followed. The following Matlab program sequence is executed:

```
% Step 1: determination of approximation polynomials
% vector of the points xi
x=-8:2:4;
% vector for the values yi of the function for the points xi
y=[30 10 9 6 5 4 4];
% vector of the coefficients of the interpolation polynomials
P2=polyfit(x,y,2);    % parabolic regression
P3=polyfit(x,y,3);    % cubic regression
P6=polyfit(x,y,6);    % 6th order polynomial regression
% Step 2: approximation
% vector of the points where the approximation is performed
vx=[-7 -4.2 -0.75 1 2.15 3];
```

```
% approximation through parabolic regression
disp('cmmp: parabolic regression:')
vy2=polyval(P2,vx)
% approximation through cubic regression
disp('cmmp: cubic regression:')
vy3=polyval(P3,vx)
% approximation through 6th order polynomial regression
disp('cmmp: 6th order polynomial regression:')
vy6=polyval(P6,vx)
```

obtaining the following results:

```
 cmmp: parabolic
 regression: vy2 =
    20.8929    9.9529    3.1473    2.5119    3.1266    4.1071
 cmmp: cubic
 regression: vy3 =
    20.0595    7.1222    4.9833    5.6786    5.6114    4.9405
 cmmp: 6th order polynomial
 regression: vy6 =
    13.6680    9.2095    5.0584    4.8867    3.7901    2.6289
```

b) Also for the graphical representation of approximation functions, it is necessary to go through the two stages specified in the paragraph 8.1. However, the first step is identical to the first step of point a). Below is the instruction sequence corresponding to the second step and the graphs obtained are shown in the figure 8.2.:

```
% Step 2: graphs for the approximation polynomials
% graphic representation interval
vxg=min(x):(max(x)-min(x))/100:max(x);
% approximation through parabolic regression
vy2g=polyval(P2,vxg);
% approximation through cubic regression
vy3g=polyval(P3,vxg);
% approximation through 6th order polynomial regression
vy6g=polyval(P6,vxg);
% graphs
% table points
plot(x,y,'bd')
hold on
% approximation polynomials
plot(vxg,vy2g,'r',vxg,vy3g,'--g',vxg,vy6g,'k:')
axis([min(x)-1 max(x)+1 ...
    min([y vy2g vy3g vy6g])-5 max([y vy2g vy3g vy6g])+5])
legend('points',' parabolic regression ', ...
    'cubic regression ','6th order regression')
hold off
```

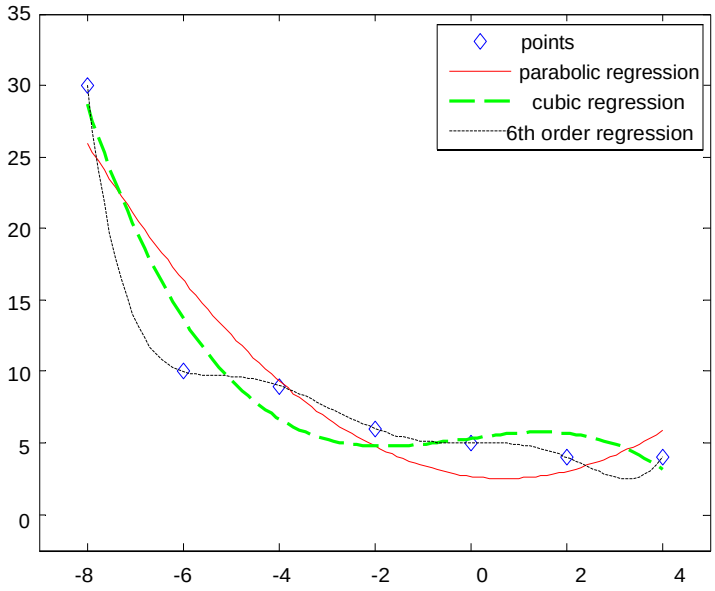

Fig.8.2. *The graphs of the approximation functions from example 8.3.*

8.3. Problems to solve

P8.1. Write a Matlab function *langrange* which performs the Lagrange polynomial interpolation. The arguments of the Matlab function will be: the vector of the points $\{x_i\}$ and the vector of the points $\{y_i\}$ of a function f known through points and the vector vx of the points in which approximation is desired. The Matlab function will return the vector vy of the values obtained by approximating the function f with the Lagrange interpolation polynomial in the points vx.

P8.2. The following table, which represents the values of the speed of a moving vehicle read at various times, is considered:

time [sec]	0	1	2	3	4	5	6
speed [m·sec^{-1}]	15	30	75	60	60	40	55

Estimate the speed values at the following times t_1=0.5 sec, t_2=3.2 sec and t_3=5.7 sec, using:

a) linear interpolation;
b) Lagrange polynomial interpolation (use the function from P8.1.);
c) piecewise cubic Hermite interpolation polynomial;
d) spline cubic interpolation;
e) parabolic regression;
f) approximation with the least squares5th order polynomial method.

P8.3. In the hypotheses of the problem P8.2., graphically represent in the same graphical window the points in the table and the approximation functions corresponding to the points a)-f).

P8.4. Write a Matlab function which receives as arguments the vector of the points $\{x_i\}$, and the vector of the points $\{y_i\}$ of a function *f* known through points and the vector *vx* of the points in which approximation is desired. The Matlab function will return the vectors obtained through linear interpolation and through cubic regression of the function *f* in the points *vx*.

P8.5. Write a Matlab function which receives as arguments the vector of the points $\{x_i\}$, and the vector of the points $\{y_i\}$ of a function *f* known through points. The Matlab function graphically represents in the same graphical window, the coordinate points (x_i, y_i) and the corresponding approximation functions obtained through spline cubic interpolation and piecewise cubic Hermite interpolation polynomial.

ANNEX M8. ELEMENTS REGARDING THE NUMERICAL APPROXIMATION OF FUNCTIONS

M8.1. Numerical approximation of functions

a. The function approximation problem

Having the function $f: I \subseteq \boldsymbol{R} \to \boldsymbol{R}$ and $[a,b]$ a subinterval of the definition domain I. There is the question of determining another function $g: I \to \boldsymbol{R}$, of relatively simple expression, which best approximates the function f on the interval $[a,b]$, namely $g(x) \approx f(x), \forall x \in [a,b]$.

The problem of approximating a function arises in the following two situations:

a) the expression of the function f is known, but sufficiently complicated, so its use in calculations is inconvenient or leads to large calculation errors;
b) the expression of the function f is not known, the function being specified only by a set of n points $\{(x_i, y_i)\}, y_i = f(x_i), i = \overline{1,n}$ with $x_1 = a$, $x_n = b$, $x_i \in (a,b)$, $i = \overline{2,n-1}$.

The most common situation is the second, in which case, the values of the function f are usually given in tabular form:

x_i	$x_1=a$	x_2	...	$x_n=b$
$y_i= f(x_i)$	y_1	y_2	...	y_n

In most applications, the values $\{x_i\}$ are equidistant with the discretization step

$h = \dfrac{b-a}{n}$, namely $x_{i+1} = x_i + h, \quad i = \overline{1,n-1}$.

In many practical situations, it is not necessary to determine the expression of the approximation function g, but only the approximation values $g(x)$ for any x in the interval $[a,b]$.

In case b), if for the approximation function g the condition $g(x_i) = y_i$, $i = \overline{1,n}$, is imposed, the approximation problem is also referred to as an ***interpolation*** problem. If the problem extends beyond the interval $[a,b]$, i.e. it is desired to approximate the function f in a point $x \in I-[a,b]$, then the term ***extrapolation*** is used.

b. Several methods for numerical approximation of functions

In continuation the function f is assumed to be is specified by a set of n points, $\{(x_i, y_i)\}, y_i = f(x_i), i = \overline{1,n}$, with $x_i = a$ and $x_n = b$.

Linear interpolation

It is required to determine the approximation function g to be affine on each subinterval $[x_i, x_{i+1}]$, $i = \overline{1,n-1}$, and to pass through the given points, therefore to verify the conditions:

$g(x_i) = y_i$, $i = \overline{1,n}$.

The function *g* with the required properties exists and is unique. It is given by expression:

$$g(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \cdot x + \frac{x_{i+1} y_i - x_i y_{i+1}}{x_{i+1} - x_i}, \; x \in [x_i, x_{i+1}], i = \overline{1, n-1}$$

Lagrange polynomial interpolation

It is required to determine the polynomial P_{n-1} (function *g* is, thus, a polynomial function, $g = P_{n-1}$), of order $P_{n-1} = n-1$, of the form:

$$P_{n-1}(x) = a_1 + a_2 x + a_3 x^2 + \ldots + a_n x^{n-1}, \forall x \in [a,b], \; a_i \in \mathbf{R} \; i = \overline{1,n}$$

passing through the given points, therefore to verify the conditions:

$P_{n-1}(x_i) = y_i$, $i = \overline{1,n}$.

The following system of linear equations is thus obtained, with the unknowns $a_1, a_2, \ldots, a_n$:

$$\begin{cases} a_1 + a_2 x_1 + a_3 x_1^2 + \ldots + a_n x_1^{n-1} = y_1 \\ a_1 + a_2 x_2 + a_3 x_2^2 + \ldots + a_n x_2^{n-1} = y_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_1 + a_2 x_n + a_3 x_n^2 + \ldots + a_n x_n^{n-1} = y_n \end{cases}$$

Because the system determinant is a Vandermonde determinant and the points x_i, $i = \overline{1,n}$, are distinct two by two, the system has a unique solution. Therefore, *there is a unique n-1 degree polynomial passing through the given points*. This polynomial is called ***the interpolation polynomial of the points*** (x_i, y_i), $i = \overline{1,n}$.

There are several ways of determining the interpolation polynomial, which in fact lead to equivalent expressions: Newton type, Gauss type, Stirling type, Bessel type, Lagrange type, and so on.From here on only the last one is referred to.

The Lagrange interpolation polynomial is determined in the following way: choose the approximation polynomial of the form:

$$L_{n-1}(X) = \sum_{k=1}^{n} y_k \, p_k(X),$$

where the p_k polynomials are of *n*-1 degree and have the property:

$$p_k(x_i) = \begin{cases} 1, & i = k \\ 0, & i \neq k \end{cases}.$$

The expression of these polynomials can easily be deduced:

$$p_k(x) \prod_{\substack{i=1 \\ i \neq k}}^{n} \frac{X - x_i}{x_k - x_i}$$

finally obtaining the expression of the Lagrange interpolation polynomial:

$$L_{n-1}(X) = \sum_{k=1}^{n} \left(y_k \prod_{\substack{i=1 \\ i \neq k}}^{n} \frac{X - x_i}{x_k - x_i} \right)$$

Hermite polynomial interpolation

It is assumed that for the function *f* the derivatives values up to a certain order for certain points $\{x_i\}$ are known:

$$f^{(r)}(x_i) = y_i^r, \quad r = \overline{0, r_i}, \; i = \overline{1, n}$$

It is required to determine the minimum degree polynomial *P* that fulfills the following conditions:

$$p^{(r)}(x_i) = y_i^r, \quad r = \overline{0, r_i}, \quad i = \overline{1, n}$$

Such a polynomial exists, is unique and has the degree $m = n + \sum_{i=1}^{n} r_i$. It's called ***the Hermite interpolation polynomial***.

Approximation using spline functions

It is required to approximate the function *f* with a polynomial spline function *g* with the degree $m << n$, such that:

$$g(x_i) = y_i, i = \overline{1, n}$$

A polynomial spline function of m degree is a $C_{[a,b]}^{m-1}$ class function, whose restrictions g_i on each subinterval $[x_{i-1}, x_i]$ are polynomials with a $m << n$ degree:

$$g_i(x) = P_m^i(x), \quad \forall x \in [x_{i-1}, x_i], \; i = \overline{2, n}, \; degree\, P_m^i = m.$$

If $m=3$ the function *g* is called ***a cubic spline function***.

Approximation using the least squares method

It is requested to determine the polynomial P_m, $degree\ P_m = m$, $m < n$, with the form:

$$P_m(x) = a_1 + a_2 x + a_3 x^2 + \ldots + a_{m+1} x^m, \forall x \in [a, b],$$

which approximates the function *f* so as to minimize the sum of the squares of the differences between the approximate and the exact values for the points $\{x_i\}$.

The problem is an optimization problem:

$$\hat{P}_m = \{P_m \mid \min_{a_1 \ldots a_{m+1}} \sum_{i=1}^{n} [P_m(x_i) - y_i]^2\}.$$

The resulting calculation method is called the least squares method ***(LSM)***. This method is used when either the pairs $(x_i, y_i = f(x_i)), i = \overline{1, n}$, are not exactly known or *n* is very large.

The approximation of the function *f* through a polynomial with the above presented form through the LSM method is called generally ***polynomial regression*** and particularly **linear regression** if $m=1$, ***parabolic regression*** if $m=2$, respectively ***cubic regression*** if $m=3$.

Approximation through the LSM method can also be applied to other approximation functions *g*, different from the polynomial ones.