inputweblogo.tex

Dipl.-Ing. D. Krause

December 8, 2008

Contents

1 Overview

1.1 Purpose

The nsplines package contains *.m files and a *.sci file for natural spline interpolations using GNU Octave or SciLab.

1.2 License conditions

The software is licensed to you under the terms of a BSD-style license:

- Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
  - Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
  - Redistributions in binary form must reproduce the above opyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
  - Neither the name of the Dirk Krause nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
- This software is provided by the copyright holders and contributors "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed.
  In no event shall the copyright owner or contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

2 Installation

2.1 Installation for Octave

2.1.1 System installation (for all users)

Run GNU Octave, on the prompt enter

```
1 DEFAULT_LOADPATH
```

The programs response contains a list of directories GNU Octave inspects when looking for a function file. Directories are separated by a colon. Two trailing slashes at the end of a directory name indicate that the directory is inspected recursively.
From the list choose a directory showing the following features:

- two trailing slashes at the directory name,
- no GNU Octave version number in the name,
- no API version number in the name and
- no hardware architecture description in the name.

If "/usr/share/octave/site/m" is in the list I suggest to use this directory.
In the choosen directory, create a "nsplines" subdirectory and copy the m/*.m files into the new "nsplines" subdirectory.

2.1.2 Installation for one user

In the first step create a user-specific directory for Octave files.
For the example user "J. James" having login name "jjames" and home directory "/home-/jjames", create "/home/jjames/octave".
Create a file ".octaverc" in your home directory (if it does not yet exist) and add a line

```
1 LOADPATH='/home/jjames/octave//::';
```

The two trailing slashes at the end of the directory tell GNU Octave to inspect the directory recursively. The two colons are a placeholder for the default load path.
In /home/jjames/octave create a subdirectory "nsplines", copy the m/*.m files into the new subdirectory.

```
1 mkdir /home/jjames/octave/nsplines
2 cp m/*.m /home/jjames/octave/nsplines
```

2.2 Installation for SciLab

2.2.1 System installation (for all users)

In the SciLab directory (i.e. c:\programme\scilab-4.0 – your directory name may differ depending on language and OS version) change into the "macros" subdirectory and create a new directory "nsplines" there. Copy sci*.sci into the new "nsplines" directory.
As root or Administrator run SciLab and enter the command

```
genlib('nsplines', 'SCI/macros/nsplines');
```

Optionally insert

```
load('SCI/macros/nsplines/lib');
```

into c:\programme\scilab-4.0\scilab.star so users have the functions available without having to load the file explicitly.

2.2.2 Installation for one user

Copy sci*.sci into da directory you have write permissions to. The complete resulting file names (i.e. "c:\jjames\octave\...\ns_create.sci" must not contain spaces.
Optionally create a file ".scilab" either in your home directory or in the directory you use for calcuations and add a line

```
exec('c:\jjames\octave\...\ns_create.sci');
exec('c:\jjames\octave\...\ns_polynom.sci');
exec('c:\jjames\octave\...\ns_value.sci');
exec('c:\jjames\octave\...\ns_gpfile.sci');
exec('c:\jjames\octave\...\nssp_create.sci');
exec('c:\jjames\octave\...\nssp_polynom.sci');
exec('c:\jjames\octave\...\nssp_value.sci');
exec('c:\jjames\octave\...\nssp_gpfile.sci');
exec('c:\jjames\octave\...\ns_nssp.sci');
exec('c:\jjames\octave\...\ns_psfile.sci');
exec('c:\jjames\octave\...\nssp_psfile.sci');
```

to have the functions available immediately after starting SciLab.

3 Theory

3.1 Natural Splines

Natural splines are a well-known mechanism to interpolate a curve which is given as a set of points. The curve is constructed piece by piece using cubic polynomials.
The polynomial

$$f_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

describes the curve between the points i and $i + 1$, this means $x_i \leq x \leq x_{i+1}$.
Betweeen n points there are $n - 1$ intervals, described by $n - 1$ polynomials. Each polynomial uses four polynomial coefficients, in summary we have $4(n - 1)$ polynomial coefficients, so we need $4(n - 1)$ equations.
The equations are as follows:

(a) $n - 1$ equations $(1 \leq i \leq n - 1)$
for the specified value at the "left" interval border:

$$d_i = y_i$$

(b) $n - 1$ equations $(1 \leq i \leq n - 1)$
for the specified value at the "right" interval border:

$$(x_{i+1} - x_i)^3 a_i + (x_{i+1} - x_i)^2 b_i + (x_{i+1} - x_i)c_i + d_i = y_{i+1}$$

(c) $n - 2$ equations $(2 \leq i \leq n - 1)$
from the following condition: In all inner points the first derivatives of the left and right polynomial must match in the point.

$$3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1})b_{i-1} + c_{i-1} = c_i$$
$$3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1})b_{i-1} + c_{i-1} - c_i = 0$$

(d) $n - 2$ equations $(2 \leq i \leq n - 1)$
from the following condition: In all inner points the second derivatives of the left and right polynomial must match in the point.

$$6(x_i - x_{i-1})a_{i-1} + 2b_{i-1} = 2b_i$$
$$6(x_i - x_{i-1})a_{i-1} + 2b_{i-1} - 2b_i = 0$$

(e) 2 equations
from the following condition: In outer points the second derivative of the polynomial is 0.

$$\begin{aligned} 2b_1 &= 0 \\ 6a_{n-1}(x_n - x_{n+1}) + 2b_{n-1} &= 0 \end{aligned}$$

3.2 Modified Natural Splines

If the first derivative of the curve in some or all points is given we have to replace some of the equations above by equations expressing the derivative:

- Inner points
  The equations from (c) and (d)

$$\begin{aligned} 3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1})b_{i-1} + c_{i-1} - c_i &= 0 \\ 6(x_i - x_{i-1})a_{i-1} + 2b_{i-1} - 2b_i &= 0 \end{aligned}$$

  are replaced by

$$\begin{aligned} 3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1})b_{i-1} + c_{i-1} &= y_i' \\ c_i &= y_i' \end{aligned}$$

- Outer points
  For the first (last) point replace the first (second) equation from

$$\begin{aligned} 2b_1 &= 0 \\ 6a_{n-1}(x_n - x_{n+1}) + 2b_{n-1} &= 0 \end{aligned}$$

  by the first (second) equation from

$$\begin{aligned} c_1 &= y_1' \\ 3(x_n - x_{n-1})^2 a_{n-1} + 2(x_n - x_{n-1})b_{n-1} + c_{n-1} &= y_n' \end{aligned}$$

So we have again $4(n-1)$ equations to find $4(n-1)$ polynomial coefficients.

3.3 NSSP

For some problems the use of simple polynomials

$$f_i(x) = p_i x^3 + q_i x^2 + r_i x + s_i$$

eases up the solution.

$$\begin{aligned} f_i(x) &= a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \\ &= a_i x^3 + (b_i - 3a_i x_i)x^2 + (c_i - 2b_i x_i + 3a_i x_i^2)x + d_i - c_i x_i + b_i x_i^2 - a_i x_i^3 \end{aligned}$$

results in:

$$\begin{aligned} p_i &= a_i \\ q_i &= b_i - 3a_i x_i \\ r_i &= c_i - 2b_i x_i + 3a_i x_i^2 \\ s_i &= d_i - c_i x_i + b_i x_i^2 - a_i x_i^3 \end{aligned}$$

NSSP is an acronym for natural splines, simple polynomials.

4 Usage

The package provides the following functions:

- M = ns_create(mw)
  The function argument *mw* contains the measurement values, one point per line. x-values are in the first column, y-values in the second column. If the first derivative is specified for some points, the matrix has four columns. The third column contains an indicator whether or not the derivative is specified for a point: If the value is larger than 0 the fourth column contains the derivative value. If the value is 0 or negative the fourth column is ignored.

  The function returns a matrix containing a line for each curve interval. Each line consists of x_{start}, x_{end}, a, b, c and d.

- y = ns_value(M, x)
  Argument $\mathcal{M}$ is the matrix returned by *ns_ create()*. The function calculates the interpolation value for a specified x.

- ns_gpfile(mw, M, name)
  The function writes a rudimentary GnuPlot file which can be used to show the point set and the interpolation curve. The *name* argument specifies the output file name.

- ns_psfile(mw, M, name)
  The function writes PostScript code to calculate the function to the file. This PostScript code is intended for use with the "pst-plot" package for LaTeX.

- nssp_. . . ()
  The functions are equivalent to the *ns_ . . . ()* functions but use simple polynomials. *Note:* Only use the *ns_ . . . ()* functions for matrices created by *ns_ create()*. Only use *nssp_ . . . ()* functions for matrices created by *nssp_ create()*.

5 Examples

5.1 BH-curve of a permanent magnet

For a permanent magnetic material we have the B-H-curve as a set of points. (see table 1).

Table 1: B-H-curve

H/Acm^{-1}	B/T
-525	0
-470	0,50
-420	0,72
-330	0,94
-220	1,12
-120	1,22
0	1,30

We are searching for B if $H = -262\,\mathrm{Acm}^{-1}$ is given. The file ns0001.m[1]

```
# Messwerte / measurement results
#          H          B
mw = [   -525.0    0.0
         -470.0    0.5
         -420.0    0.72
         -330.0    0.94
         -220.0    1.12
         -120.0    1.22
         0.0       1.3  ];

global M;
M = ns_create(mw);

Hvalue = -262.0;
Bvalue = ns_value(M, Hvalue);
printf('B = %g\n', Bvalue);
```

is used by the command

```
octave -q ns0001.m
```

and creates the output

[1]The SciLab file ns0001.sce is shown in appendix A.1.1 on page 23

```
B = 1.06071
```

So we have $B = 1,06\,\mathrm{T}$.

To print the interpolation curve we create a file ns0002.m[2]:

```
# Messwerte / measurement results
#          H          B
mw = [   -525.0    0.0
         -470.0    0.5
         -420.0    0.72
         -330.0    0.94
         -220.0    1.12
         -120.0    1.22
          0.0      1.3  ];

M = ns_create(mw);
ns_gpfile(mw, M, "ns0002.gp", "native");
```

We run

```
octave -q ns0002.m
```

and create ns0002.gp. At the start of file we insert to instructions to specify output file type and name:

```
set terminal mp c latex psnfss
set output "ns0002.mp"
```

Further modifications are possible to set labels...
We run

```
gnuplot ns0002.gp
mpost -tex=latex ns0002
mv ns0002.0 ns0002.mps
```

to create a *.mps file we can use in a LaTeX source (see figure 1 on the next page).

[2]Appendix A.1.2 on page 24 shows the SciLab file ns0002.sce

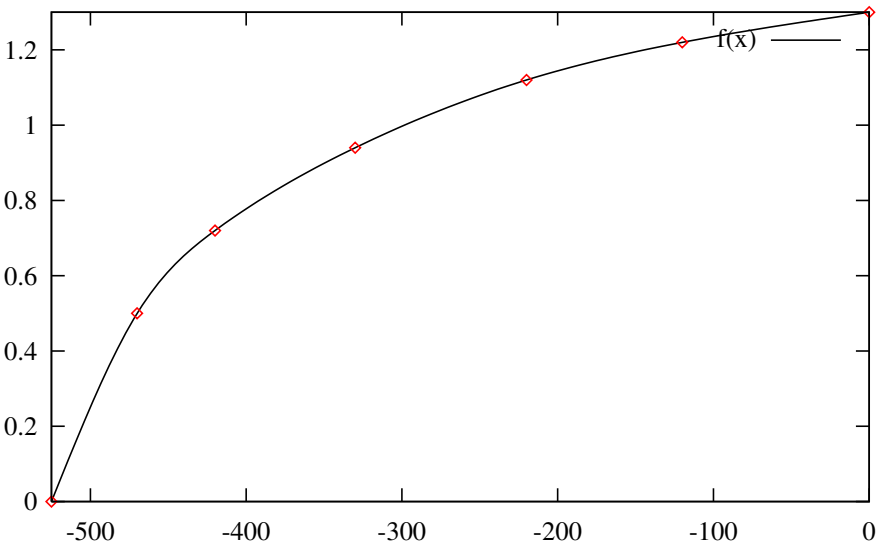

Figure 1: Interpolation curve

Now we want to know which H is needed for $B_x = 0,8\,\mathrm{T}$. We do a natural spline interpolation first and use the interpolation matrix $\mathcal{M}$ to define a function

$$f(H) = B(H) - B_x$$

The zero of this function is our H_x. We use GNU Octave to solve a system of nonlinear equations[3].

```
# Messwerte / measurement results
#         H         B
mw = [   -525.0    0.0
         -470.0    0.5
         -420.0    0.72
         -330.0    0.94
         -220.0    1.12
         -120.0    1.22
         0.0       1.3  ];

global M;
M = ns_create(mw);

global wishValue;
wishValue = 0.8;

function y = f(x)
  global M; global wishValue;
  y = ns_value(M, x) - wishValue;
endfunction

printf('For H = -262A/cm we have B = %gT\n', ns_value(M, -262.0));

[x, info] = fsolve("f", [-330.0]);
if (info == 1)
  printf('Successfully finished iteration.\n');
  printf('For B = 0.8T we need H = %g\n', x);
else
  perror("fsolve", info);
endif
ns_gpfile(mw, M, "ns0007.gp");
```

The output

```
Success.
H = -391.629
```

shows $H_x = -391,6\,\mathrm{Acm}^{-1}$.

[3]Our system is small – only one equation.

5.2 Example curve if first derivative is given

A fictionary curve y depending on x is specified in table 2. Furthermore we know that

$$0 \leq y \leq 1$$

Table 2: Points

x	y
1	0
2	0
3	0
4	0,5
5	1
6	1
7	1

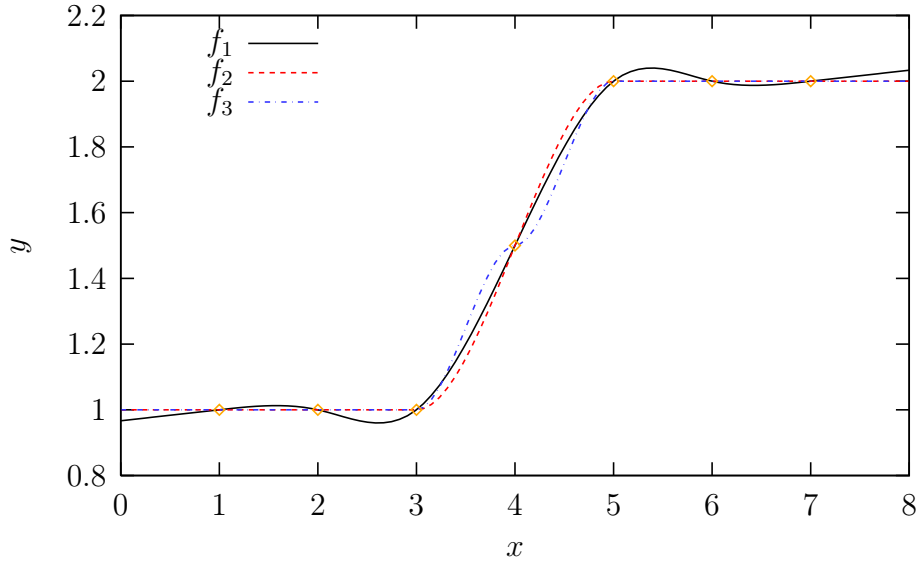

Figure 2: Different curves for the same set of points

```
mw = [
            1           1
            2           1
            3           1
            4         1.5
            5           2
            6           2
            7           2
];
[mwr, mwc] = size(mw);
M = ns_create(mw);
ns_gpfile(mw, M, "ns0004.gp", "native");
```

Specifying points without the derivative the natural spline interpolation results in curve f_1 (see figure 2). There are oscillations, the curve is not correctly bound to the specified y-range.

To avoid oscillations we specify the derivatives for the points $x = 3$ and $x = 5$. In the matrix lines for the points we enter a positive value in column 3 and 0 (the derivative) in column 4.

```
mw = [
        1       1       0       0
        2       1       0       0
        3       1       1       0
        4       1.5     0       0
        5       2       1       0
        6       2       0       0
        7       2       0       0
];
```

The resulting curve f_2 does not show oscillations.
We can apply further modifications to the curve by specifying the derivative in other points too.

```
mw = [
        1       1       0       0
        2       1       0       0
        3       1       1       0
        4       1.5     1       0
        5       2       1       0
        6       2       0       0
        7       2       0       0
];
```

If – for example – we specify a derivative 0 for $x = 4$ this results in f_5.
To compare the functions all the three curves are plotted in one diagram.

5.3 Optimized permanent magnet

For the magnetic material (see section 5.1 on page 10) we search for the optimum bias point (the curve point where $|B \cdot H|$ reaches the maximum).
Two methods are known:

- A graphical method (first approximation).
  We draw a rectangle, edge widths are B_r and H_c. The optimum bias point is where the diagonal line meets the $B = f(H)$ curve.
  The description formula looks like:

$$B = f(H) = H \cdot \frac{B_r}{H_c}$$

  So we have to find the zero of

$$g(H) = f(H) - H \cdot \frac{B_r}{H_c}$$

- A numeric method.

$$\begin{aligned} BH &= f_i(H) \cdot H \\ &= (p_i H^3 + q_i H^2 + r_i H + s_i) \cdot H \\ &= p_i H^4 + q_i H^3 + r_i H^2 + s_i H \end{aligned}$$

  The derivative of the product is

$$\frac{\mathrm{d}}{\mathrm{d}H}(BH) = 4p_i H^3 + 3q_i H^2 + 2r_i H + s_i$$

  To find the maximum of the product we have to search for a zero of:

$$h_i(H) = 4p_i H^3 + 3q_i H^2 + 2r_i H + s_i$$

  This function can be shown as a natural spline too, we can obtain the interpolation matrix by first creating a copy of the interpolation matrix for $B = f(H)$ and doing multiplications (all column 3 values are increased by factor 4, all column 4 values are increased by factor 3 and all column 5 values are increased by factor 2).

We calculate natural splines with simple polynomials here, so we have to use the *nssp_...()* functions.

The input file ns0010.m[4] applies both methods:

```
mw = [   -525.0   0.0
         -470.0   0.5
         -420.0   0.72
         -330.0   0.94
         -220.0   1.12
         -120.0   1.22
         0.0      1.3  ];

global M1;
M1 = nssp_create(mw);

global M2;
M2 = M1;

[r,c] = size(M2);

for i=1:r
  M2(i,3) = 4.0*M2(i,3);
  M2(i,4) = 3.0*M2(i,4);
  M2(i,5) = 2.0*M2(i,5);
endfor

function back = g(H)
  global M1;
  back = nssp_value(M1, H) - 1.3 * H / (-525.0);
endfunction

function back = h(H)
  global M2;
  back = nssp_value(M2, H);
endfunction

[H, info] = fsolve("g", [-525.0/2.0]);
if (info == 1)
  printf('Grapical approximation solved successfully.\n');
  printf('H    = %g\n', H);
  printf('B    = %g\n', nssp_value(M1, H));
  printf('|BH| = %g\n', abs(H*nssp_value(M1,H)));
else
  perror("fsolve", info);
endif
```

[4] See appendix A.1.3 on page 25 for SciLab file ns0007.sce

```
42
43 [H, info] = fsolve("h", [-525.0/2.0]);
44 if (info == 1)
45   printf('Optimizatzion solved successfully.\n');
46   printf('H    = %g\n', H);
47   printf('B    = %g\n', nssp_value(M1, H));
48   printf('|BH| = %g\n', abs(H*nssp_value(M1,H)));
49 else
50   perror("fsolve", info);
51 endif
```

The GNU Octave output

```
1 Grapical approximation solved successfully.
2 H    = -356.844
3 B    = 0.883614
4 |BH| = 315.313
5 Optimizatzion solved successfully.
6 H    = -368.153
7 B    = 0.857926
8 |BH| = 315.848
```

shows that there is only a small difference between the results of the two methods.

6 Internals

6.1 Polynomial creation

As already noted we have to solve a system of $4(n-1)$ equations to find $4(n-1)$ polynomial coefficients. The equations from (a) result directly in values d_i.
The remaining system contains $3(n-1)$ equations for $3(n-1)$ polynomial coefficients. We leave it up to GNU Octave to solve the system but we have to prepare a coefficients matrix $\mathcal{M}$ and a result vector $\overrightarrow{y}$ to describe the system of equations.
The variables vector $\overrightarrow{x}$ contains a_1, b_1, c_1, $a_2 \ldots c_{n-1}$ in one column. The position of the a_i in the vector is given by $3(i-1)+1$, the position of b_i is $3(i-1)+2$ and the position of c_i is $3(i-1)+2$.
In the coefficients matrix and the result vector we enter the equations as follows:[5]

(b) $n-1$ equations for the right interval borders. For equation number i, $1 \leq i \leq (n-1)$ the line number (in the coefficients matrix and the result vector) is $z = i$.

$$(x_{i+1} - x_i)^3 a_i + (x_{i+1} - x_i)^2 b_i + (x_{i+1} - x_i) c_i + d_i = y_{i+1}$$

is transformed into

$$(x_{i+1} - x_i)^3 a_i + (x_{i+1} - x_i)^2 b_i + (x_{i+1} - x_i) c_i + y_i = y_{i+1}$$
$$(x_{i+1} - x_i)^3 a_i + (x_{i+1} - x_i)^2 b_i + (x_{i+1} - x_i) c_i + = y_{i+1} - y_i$$

(c) $n-2$ equations for the inner points. The first derivatives of the left and right polynomial must match in the point. For equation number i, $2 \leq i \leq (n-1)$ the line number is $z = n + i - 2$.

$$3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1}) b_{i-1} + c_{i-1} - c_i = 0$$

(d) $n-2$ equations for the inner points. The second derivatives of the left and right polynomial must match in the point. For equation number i, $2 \leq i \leq (n-1)$ the line number is $z = 2n + i - 4$.

$$6(x_i - x_{i-1}) a_{i-1} + 2b_{i-1} - 2b_i = 0$$

(e) 2 equations for points 1 and n. The second derivative of the polynomial is 0 in the point. The line numbers are $z = 3n - 4$ and $z = 3n - 3$.

$$2b_1 = 0$$
$$6a_{n-1}(x_n - x_{n+1}) + 2b_{n-1} = 0$$

[5] The enumerations starts at (b) for correlation to section 3.1 on page 6

If the derivative is specified we have to apply the following changes to our system of equations:

- For the first point ($i = 1$), we replace equation number $z = 3n - 4$ from (e) by:

$$c_1 = y'_1$$

- For the last point ($i = n$), we replace equation number $z = 3n - 3$ from (e) by:

$$3(x_n - x_{n-1})a_{n-1}^2 + 2(x_n - x_{n-1})b_{n-1} + c_{n-1} = y'_n$$

- For inner points ($1 < i < n - 1$), we replace the equations $z = n + i - 2$ from (c) and $z = 2n + i - 4$ from (d) by

$$\begin{aligned} 3(x_i - x_{i-1})^2 a_{i-1} + 2(x_i - x_{i-1})b_{i-1} + c_{i-1} &= y'_i \\ c_i &= y'_i \end{aligned}$$

After solving the system of equations we read the a_i, b_i and c_i from the variables vector and transfer them to the interpolation matrix, the d_i are inserted into the interpolation matrix too.
The interpolation matrix looks like

$$\begin{pmatrix} x_1 & x_2 & a_1 & b_1 & c_1 & d_1 \\ x_2 & x_3 & a_2 & b_2 & c_2 & d_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n-1} & x_n & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \end{pmatrix}$$

6.2 Interpolation value calculation

To calculate interpolation value we first check whether the x is in the x-range specified in the point set.
Outside the x-range the curve is continued as a linear function.
For valid x values we first determine the interval, retrieve the polynomial coefficients from the interpolation matrix and calculate the polynomial result.

A SciLab files

A.1 BH-curve of a permanent magnet

A.1.1 Calculation

```
// Messwerte / measurement results
//        H         B
mw = [   -525.0   0.0
         -470.0   0.5
         -420.0   0.72
         -330.0   0.94
         -220.0   1.12
         -120.0   1.22
         0.0      1.3  ];

global M;
M = ns_create(mw);

Hvalue = -262.0;
Bvalue = ns_value(M, Hvalue);
printf('B = %g\n', Bvalue);
```

A.1.2 GnuPlot file creation

```
mw = [   -525.0   0.0
         -470.0   0.5
         -420.0   0.72
         -330.0   0.94
         -220.0   1.12
         -120.0   1.22
         0.0      1.3  ];

M = ns_create(mw);
ns_gpfile(mw, M, "ns0002.gp");
```

A.1.3 Calculate H for a given B

```
mw = [   -525.0   0.0
         -470.0   0.5
         -420.0   0.72
         -330.0   0.94
         -220.0   1.12
         -120.0   1.22
         0.0      1.3  ];

global M;
M = ns_create(mw);

global wishValue;
wishValue = 0.8;

function y = f(x)
  global M; global wishValue;
  y = ns_value(M, x) - wishValue;
endfunction

printf('For H = -262A/cm we have B = %gT\n', ns_value(M, -262.0));

[x, v, info]=fsolve([-330.0],f);
if (info == 1)
  printf('For B = 0.8T we need H = %g\n', x);
else
  printf('Failed to solve problem!\n');
end

ns_gpfile(mw, M, "ns0007.gp");
```

A.2 Example curve is first derivative is given

```
mw = [
            1           1
            2           1
            3           1
            4           1.5
            5           2
            6           2
            7           2
];
[mwr, mwc] = size(mw);
M = ns_create(mw);
ns_gpfile(mw, M, "ns0004.gp");
```

A.3 Optimized permanent magnet

```
mw = [   -525.0   0.0
         -470.0   0.5
         -420.0   0.72
         -330.0   0.94
         -220.0   1.12
         -120.0   1.22
         0.0      1.3  ];

global M1;
M1 = nssp_create(mw);

global M2;
M2 = M1;
[r,c] = size(M2);
for i=1:r
  M2(i,3) = 4.0*M2(i,3);
  M2(i,4) = 3.0*M2(i,4);
  M2(i,5) = 2.0*M2(i,5);
end

function back = g(H)
  global M1;
  back = nssp_value(M1, H) - 1.3 * H / (-525.0);
endfunction

function back = h(H)
  global M2;
  back = nssp_value(M2, H);
endfunction

[H, v, info] = fsolve([-525.0/2.0], g);
printf('Grapical approximation solved successfully.\n');
if (info == 1)
  printf('H    = %g\n', H);
  printf('B    = %g\n', nssp_value(M1, H));
  printf('|BH| = %g\n', abs(H*nssp_value(M1,H)));
else
  printf('Error: Failed to solve problem!\n');
end

[H, v, info] = fsolve([-525.0/2.0], h);
if (info == 1)
```

```
43   printf('Optimizatzion solved successfully.\n');
44   printf('H    = %g\n', H);
45   printf('B    = %g\n', nssp_value(M1, H));
46   printf('|BH| = %g\n', abs(H*nssp_value(M1,H)));
47 else
48   printf('Error: Failed to solve problem!\n');
49 end
```