- Theory and applications of spline functions for a servo-mechanism.

Hugues de ROSEMON: Université de Technologie de Compiègne.

February-July 1985

N. WPB-0200
Thanks

Presentation of the work environment
- the Eindhoven University
- the education at the university
- the mechanical engineering department

Presentation of the work
- the FAIR project
- the particular work itself

Theory of spline functions
- mathematical theory
- calculations with third degree cubic spline functions
- application of the calculations for four knots

Simulations of the interpolation on the PRIME
- how to use interpolation
- the basic flowchart of the fortran program
- the different programs
- results of the different programs
- conclusion

Development of the programs on the INTELLEC development system
- introduction to the work
- presentation of the development system
- development on the 8086 version
- measurements of the running times on the 8086 version
- conclusion

General conclusion

Appendix
- summary in French
- documentation
Thanks

At the end of this training period, I want to express my greatest thanks to Mr. Heuvelman, my coach, who by his advice, his competence and his kindness has directed me and my investigation. Thanks too APA which provided me an application of my subject and specially to Jan and Rolland who helped me during the simulation step on the INTELLEC system. The first part of this work would have been impossible without the participation of Mr. Banens who introduced me to his fortran program and to the PRIME facilities. There are many people, especially Frits, who I will not forget because they made my stay at the university very pleasant. I also want to thank Lia who accepted to type this report.

Once back to France, one part of my heart will stay Dutch. Not only because of the arts, the landscapes with water and windmills, the people, the flowers or the beers, but because Dutch people know how to combine all of this to make their land a land of warm hospitality.
II  Presentation of the work environment

a)  The Eindhoven University

The Eindhoven university of technology has been founded in 1956. It offers nine courses of study in which students can qualify as graduate engineers specialising in the following subjects:

- Technology in its social application
- Industrial engineering and management science
- Mathematics
- Computing science
- Technical physics
- Mechanical engineering
- Electrical engineering
- Chemical engineering
- Architecture, structural engineering and urban planning

Since the Eindhoven university of technology opened in 1957 more than 6600 students have graduated from it. The degree of doctor in the Technical sciences can be obtained by students submitting a doctorate thesis on research they have carried out.

b)  The education at the university

A full university course in Netherlands used to take at least 4 years. But generally university studies can be divided in two phases: The first phase has a duration of four years and comprises two examinations: the first or preliminary examination at the end of the first year and the final examination at the end of the fourth year. Students are allowed 2 extra years to complete the first phase. The first examination has to be passed at the end of the second year at the latest.

In the second phase will be introduced in 1986/87, covers three types of training:
- further professional training as physist, pharmacist with a maximum of two years.
- professional training of teachers for secondary schools with a maximum of 1 year
- training for research and technological design, with a maximum of 1 a 2 years.

c)  The mechanical engineering department

Design and production are the two main groups into which the highly varied tasks of mechanical engineers are divided. The nature of the tasks carried out by mechanical engineers varies from scientific research and development to industrial organisation.

Apart from their theoretical knowledge mechanical engineers must possess specific practical skills. To this end, the curriculum includes among other
things, participation in the work done by the department in its four divisions:
  . fundamentals of mechanical engineering
  . product design and development
  . design for industrial processing
  . production engineering and automation
  In this department there are about two hundred employees (teachers, technical personnel) and seven hundred students. I have worked during five months among them.
III Presentation of the work

a) The FAIR project (Flexible Automation and Industrial Robots)

The research project FAIR is financed and directed by the Dutch government. The aim is to develop, for most applications of flexible automation, an industrial robot system. The mechanical and electrical engineering departments of the university as well as several companies, are involved in this project. In a very general view the university applies its theoretical experience and the companies their practical experience. Both are linked by a contract. For organisational reasons the project is subdivided into five small parts:
1. the general aspects of automation
2. the handling of parts
3. kinematics and dynamics of mechanical structures
4. the drive systems, the control systems and the applications of the systems
5. the arc-welding and the sensory systems.
My project has been done in the project group 4.

b) The practical work itself

I worked with the firm APA (Advanced Product Automation) which is developing a new welding-robot, in view of flexible automation, with the help of the university. So my practical work took place at the university but was directly connected to an industrial project. The developed welding-robot has several motors to move a torch from one point to another. Each motor has its own feedback control loop which, to reach a desired accuracy, needs a pilot value every 5 m.s. Knowing the position value in the space of the next point, the job of the main computer is to calculate the motion of each motors. These calculations are very complicated and require a long time to be done. Consequently, this computer is too busy to send information every 5 m.s to each motor.

We resolved the problem by using slave computers. They receive from the main computer a piece of information (for example every 100 m.s) and calculate by an interpolating method, the intermediate points every 5m.s. If we know two points the interpolation is linear and if more than three points are known we can use a polynomial method. Each motor is piloted by a slave computer and possesses its own interpolator.
Because the interpolation must be accurate we must find a good method and so the first part of the project will be a simulation of different kinds of interpolations. This first part is purely theoretical. The linear functions and polynomial functions are well known and, consequently, only spline functions are described.
The second part of this project will concern the real time simulation, the running time measurements and the implementation with hard-ware.
IV) **Theorie of spline functions**

a) **Mathematical theory**

The theory comes from the books: "Data fitting by spline functions" T.N.E. Greville, MRC Technical report No. 893

**Properties**

A spline function $s(n)$ of degree $m$ with knots $n_1, n_2 \ldots n_n$ is defined having the following properties:

(a). in each interval $(n_i, n_{i+1})$ for $i = 1, \ldots n$ the spline function is given by some polynomial of degree $m$ or less.

(b). $S(n)$ and each derivatives of order 1, 2, .. $m-1$ are continuous.

$s(n) \in S_m (n_1, n_2, \ldots n_n)$ has a unique representation of the form

$$s(n) = \varphi_m(n) + \sum_{j=1}^{n} C_j (n - n_j)^m$$

with $\varphi_m \in \Pi_m$ and $\Pi_m$ denotes the class of polynomials of degree $m$ or less.

**proof**

For $j = 0, 1, \ldots n$ let $\varphi_{mj}(n)$ be the polynomial that gives the value of $S(n)$ in the interval $(n_j, n_{j+1})$. It follows of conditions (a) and (b) that $\varphi_{mj}(n)-\varphi_{mj-1}(n)$ is a polynomial of degree $m$ having $m$ fold-zero at $n = n_j$ that is: $\varphi_{mj}(n)-\varphi_{mj-1}(n) = C_j(n-n_j)^m$ (1)

thus $\varphi_m(n) = \varphi_{m0}(n) + (\varphi_{m1}(n)-\varphi_{m0}(n))+(\varphi_{m2}(n)-\varphi_{m1}(n))+(\varphi_{m3}(n)-\varphi_{m2}(n)) + \ldots + (\varphi_{ml}(n)-\varphi_{ml-1}(n))$

$$\varphi_m(n) = \varphi_{m0}(n) + \sum_{j=1}^{l} C_j(n-n_j)^m$$

We must show that this representation is unique or in other words, that for a given $S(n)$ the polynomial $\varphi_m(n)$ and the $C_j$ are uniquely determined.

for $n < n_1: S(n) = \varphi_{m0}(n)$ therefore $\varphi_m(n) = \varphi_{m0}(n)$

Further $m$ fold differentiation of (1) gives:

$$\varphi_{mj}(n) - \varphi_{mj-1}(n) = C_j n^m$$

taking $n = n_j$, this may be written as $C_j = \frac{1}{m!} (s^{(m)}(n_{j+0}) - s^{(m)}(n_{j-0}))$

The natural spline interpolation
A spline function of odd degree $2k-1$ with knots $n_1', n_2'... n_m'$ is called a natural spline function if the two polynomials by which it is represented in the two intervals $(-\infty, n_1')$ and $(n_m', +\infty)$ are of degree $k-1$, or less. Let $(n_i', j_i)$ $i=1...n$ be given data points, with the abscissas $n_i$ in strictly increasing sequence and containing a finite interval $(a, b)$. A function $f(n)$ of class $C^k$ fits the $n$ data points with the conditions:

$$f(n_i) = J_i \quad i = 1...n$$

this is the smoothest function (smoothest being interpreted to mean that the integral $\int_a^b (f_k(n))^2 \, dn$ shall be made as small as possible)

- For $k>n$: the problem doesn't have a unique solution, as there is an infinity of polynomials of degree $k-1$ satisfying (2) for all of which $J = 0$.
- For $k = n$: unique solution given by the Lagrange interpolation polynomial
- For $k < n$: there is a unique function $f(n)$ for which the minimum is attained. This function turns out to be a spline function of degree $2k-1$ having the abscissas $n_1', n_2'... n_n'$ as knots.

So a smoothest spline function can be written like:

$$S(n) = \phi_{k-1}(n) + \sum_{j=1}^{n} C_j (n-n_j)^{2k-1}$$

for $n<n_1$, $S(n)$ is automatically a function of degree $k-1$ for $n>n_n$: $S(n) = \phi_{k-1}(n) + \sum_{j=1}^{n} C_j (n-n_j)^{2k-1}$ reduces to a polynomial of degree $k-1$ if and only if the coefficient of every power of $n$ higher than $k-1$ vanishes. So for $r = 0, 1..., k-1$ the coefficients of $n^{2k-1-r}$ is $(-1)^r (2k-1) \frac{n}{r} C_j n_j^r$ and we must have $\sum_{j=1}^{n} C_j n_j^r = 0$ (4)

Thus an expression of the form (3) is a natural spline function of degree $2k-1$ with the knots $n_1', n_2'... n_n'$ if and only if conditions (4) are satisfied.

Theorem:

let $(n_i', j_i)$, $i=1...n$ be given data points where the $n_i'$ form a strictly increasing sequence and let $k$ be a positive integer not exceeding $n$. There is a unique natural spline function $s(n)$ of degree $2k-1$ with the knots $n_1'$ such that:

$$s(n_i') = J_i \quad i = 1, 2,... n$$
b) **Calculation with third degree spline function**

If the coordinates of \( n \) data points are given, the smoothest interpolating function is a natural spline function \( S(n) \) of degree \( 2k-1 \) (\( k/n \)). The parameters can be obtained by solving the system of equations consisting of condition (4) and (5) with \( S(n_i) \) given by (3). Third degree spline functions (\( k = 2, 2k-1 = 3 \)) are the most useful and interesting one.

A third degree natural spline function \( S(n) \) is given by a three degree polynomial in each interval \( (n_i, n_{i+1}) \) \( i = 1 \ldots n-1 \) and by a linear function in each interval \( (n_i, n_{i+1}) \) and vanishes in \((-\infty, n_1) \) and \((n_n, +\infty) \). In a general way:

\[
S^*(n) = S^*(n_i) + \frac{n-n_i}{n_{i+1}-n_i} [S^*(n_{i+1}) - S^*(n_i)]
\]  
(6)

with \( n_i \leq n \leq n_{i+1} \) \( i = 1, 2, \ldots n-1 \)

\( S(n) \) is easily calculated if we know \( S(n_i), S(n_{i+1}), S^*(n_i) \) and \( S^*(n_{i+1}) \).

\[
S(n) = S(n_i) + \frac{n-n_i}{n_{i+1}-n_i} [S(n_{i+1}) - S(n_i)] - \frac{1}{6} (n-n_i)(n_{i+1}-n)[S^*(n) + S^*(n) + S^*(n_{i+1})]
\]  
(7)

Differentiation of (7) with respect to \( n \) gives:

\[
(8) \quad S'(n) = \frac{S(n_{i+1}) - S(n_i)}{n_{i+1} - n_i} + \frac{1}{6} (2n-n_i-n_{i+1})[S^*(n_i) + S^*(n) + S^*(n_{i+1})] - \frac{1}{6} (n-n_i)(n_{i+1}-n)[\frac{S^*(n_{i+1}) - S^*(n_i)}{n_{i+1} - n_i}]
\]

Now, we call \( S_i(n) \) the spline function between the knots \([n_{i-1}, n_i]\) and \( S_{i+1}(n) \) the spline function between \([n_i, n_{i+1}]\). From (8) we deduce that:

\[
(8.1): \quad S'_i(n_i) = \frac{S(n_i) - S(n_{i-1})}{n_i - n_{i-1}} + \frac{1}{6} (n_i-n_{i-1})[2S^*(n_i) + S^*(n_{i-1})]
\]

\[
(8.2): \quad S'_{i+1}(n_i) = \frac{S(n_{i+1}) - S(n_i)}{n_{i+1} - n_i} - \frac{1}{6} (n_{i+1} - n_i)[S^*(n_{i+1}) + 2S^*(n_i)]
\]

To satisfy the condition (b) we must have the equality of (8.1) and (8.2) for each knot \( n_i \) \( i = 2, 3, \ldots n-1 \). So, from (8.1) and (8.2) we can write:

\[
(9) \quad \frac{S(n_i) - S(n_{i-1})}{n_i - n_{i-1}} + \frac{1}{6} (n_i-n_{i-1}) [2S^*(n_i) + S^*(n_{i-1})] = \frac{S(n_{i+1}) - S(n_i)}{n_{i+1} - n_i} - \frac{1}{6} (n_{i+1} - n_i)[S^*(n_{i+1}) + 2S^*(n_i)]
\]
For the knots $n_1$ and $n_n$, we can write the boundary conditions:

\begin{align}
(10) \quad S'(n_1) &= \frac{S(n_2) - S(n_1)}{n_2 - n_1} - \frac{1}{6} (n_2 - n_1) [2S''(n_1) + S''(n_2)] \\
(11) \quad S'(n_n) &= \frac{S(n_n) - S(n_{n-1})}{n_n - n_{n-1}} + \frac{1}{6} [n_n - n_{n-1}] [2S''(n_n) + S''(n_{n-1})]
\end{align}

Generally $S'(n_1)$ and $S'(n_n)$ are given. We will call them boundary conditions.

The system of equations (9), (10), (11) is a system of $n$ equations with $n$ unknowns $(S''(n_1); S''(n_2) \ldots S''(n_n))$. After solution of this system, we can replace the calculated values of $S''(n_i)$ in the equation (7) rewritten:

\begin{align}
(7) \quad S_{i+1}(n) &= \frac{n_{i+1} - n}{n_{i+1} - n_i} [S(n_i) - \frac{1}{6} S''(n_i) [(n_{i+1} - n_i)^2 - (n_{i+1} - n_i)^2]] \\
&\quad + \frac{n - n_i}{n_{i+1} - n_i} [S(n_{i+1}) - \frac{1}{6} S''(n_{i+1}) [(n_{i+1} - n_i)^2 - (n_i - n)^2]]
\end{align}

From this equation (7) we can pull out the coefficients of the spline function. If $S_{i+1}(n) = C_0 + C_1n + C_2n^2 + C_3n^3$ with $n_i \leq n \leq n_{i+1}$, then $C_0$, $C_1$ and $C_3$ are easily calculated.

The reasoning to calculate the coefficients has been made for cubic natural spline functions, but it will be exactly the same if using cubic spline function. It only means, in this last case, that either first or second derivative boundary condition can be chosen and the case is simpler.

c) Application of the calculations for four knots

We suppose that $[(n_1, S(n_1)), (n_2, S(n_2)), (n_3, S(n_3)), (n_4, S(n_4))]$ is a set of knots and the boundary condition $S'(n_1)$ and $S'(n_4)$ are known.

The boundary conditions given by (10) and (11) are:

\begin{align}
S'(n_1) &= \frac{S(n_2) - S(n_1)}{n_2 - n_1} - \frac{1}{6} (n_2 - n_1) (2S''(n_1) + S''(n_2)) \\
S'(n_4) &= \frac{S(n_4) - S(n_3)}{n_4 - n_3} + \frac{1}{6} (n_4 - n_3)(2S''(n_4) + S''(n_3))
\end{align}

The continuity conditions given by (9) are:

\begin{align}
\frac{S(n_3) - S(n_2)}{n_3 - n_2} - \frac{n_3 - n_2}{6} (2 \times S''(n_2) + S''(n_3)) = \frac{S(n_2) - S(n_1)}{n_2 - n_1} + \frac{n_2 - n_1}{6} (2 \times S''(n_2) + S''(n_1))
\end{align}
In our case \( n_i \) is the time and \( n_i - n_{i-1} = \Delta n = \text{cste.} \) With this remark, the general system to resolve becomes:

\[
\begin{aligned}
S^*(n_3) + S^*(n_1) + 4 S^*(n_2) &= \frac{6}{(\Delta n)^2} [(S(n_3) - S(n_2)) - (S(n_2) - S(n_1))] \\
S^*(n_4) + S^*(n_2) + 4 S^*(n_3) &= \frac{6}{(\Delta n)^2} [(S(n_4) - S(n_3)) - (S(n_3) - S(n_2))] \\
2S^*(n_4) + S^*(n_3) &= \frac{6}{\Delta n} [S'(n_4) - \frac{S(n_4) - S(n_2)}{\Delta n}] \\
2S^*(n_1) + S^*(n_2) &= \frac{6}{\Delta n} \left[ \frac{S(n_2) - S(n_1)}{\Delta n} - S'(n_1) \right]
\end{aligned}
\]

Let call

\[
\begin{aligned}
\alpha &= \frac{6}{(\Delta n)^2} [(S(n_3) - S(n_2)) - (S(n_2) - S(n_1))] \\
\beta &= \frac{6}{(\Delta n)^2} [(S(n_4) - S(n_3)) - (S(n_3) - S(n_2))] \\
\delta &= \frac{6}{\Delta n} \left[ \frac{S(n_2) - S(n_1)}{\Delta n} - S'(n_1) \right] \\
\xi &= \frac{6}{\Delta n} \left[ S'(n_4) - \frac{S(n_4) - S(n_3)}{\Delta n} \right]
\end{aligned}
\]

Always in our case, we calculate \( S'(n_4) \) as \( S'(n_4) = \frac{S(n_4) - S(n_3)}{\Delta n} \) and \( \delta = 0 \) in any case.

The results of this system are:

\[
\begin{aligned}
S^*(n_1) &= \frac{52}{45} \delta - \frac{2}{45} \beta - \frac{7}{45} \alpha \\
S^*(n_2) &= -\frac{4}{45} \beta + \frac{14}{45} \alpha - \frac{7}{45} \delta \\
S^*(n_3) &= -\frac{4}{45} \alpha + \frac{2}{45} \delta + \frac{14}{45} \beta \\
S^*(n_4) &= -\frac{7}{45} \beta + \frac{2}{45} \alpha - \frac{1}{45} \delta
\end{aligned}
\]

If the spline function between the two first knots \((n_1, n_2)\) is written:

\[C_0 + C_1 n + C_2 n^2 + C_3 n^3 = S_1(n)\]

we obtain the coefficients values from (7):

\[
\begin{aligned}
C_0 &= S(n_1) \\
C_1 &= \frac{[S(n_2) - S(n_1)]}{\Delta n} - \frac{\Delta n}{6} [2S^*(n_1) + S^*(n_2)]
\end{aligned}
\]
\[ c_2 = \frac{S^*(n_1)}{2} \]
\[ c_3 = \frac{1}{6x \Delta n} (S^*(n_2) - S^*(n_1)) \]
Simulation of interpolation on the PRIME

a) How to use the interpolations

In research work, simulation is of great importance because it gives an idea of reality simply by using a computer. In our case, we expect to obtain an idea of the accuracy of the interpolation method. We attempted to calculate and visualize the error between a real curve and the interpolated curve. We wanted especially to try spline interpolation and linear interpolation on circles and straight lines.

In a welding application we can not enter all the knot values in the memories of a computer because:

- it would take too much memory space
- variations in the parts to be welded together.

To get around this disadvantage we can imagine that a sensitiv sensor follows the welding path just in front of the torch and detects a new knot every 100ms. The interpolation can be done on n knots \((n \geq 3)\) stored in only n memory places.

In reality, we calculate the interpolating function between the two first knots \(n_i\) and \(n_{i+1}\) with the knowledge of n knots \((n_i', n_{i+1}', \ldots n_{i+n-1}')\). At the moment when the \(n_{i+n}\) knot is detected, we can calculate the next interpolating function between the knots \(n_{i+1}\) and \(n_{i+2}\). This is a simulation of what will happen in reality. The number of knots \(n\) must be determined in order to have the best accuracy possible. Of course, when an interpolating function is calculated, we are able to give the set values every 5 m.s.

b) The basic flowchart of the general fortran program

The program calculates cubic spline functions in the more general way. The boundary conditions can be given either by the first derivative, or the second derivative, or both and calculations can be done on n knots. The set values can be given as desired. This basic program was already written and I have just modified it in different ways to find the expected results.
The calculations are carried out as explained in the paragraph "Calculations with third degree spline functions". To resolve the general system of \( n \) unknowns, \( n \) equations, the program uses a gaussian elimination method.

The computer used was a high computer PRIME and the different programs have been written in Fortran. We can say that we have two kinds of programs:
- The first calculates the errors and displays them on the screen
- the second calculates the errors and draws the curve \((a, \varepsilon)\) on the screen. \((\varepsilon=\text{error}, \ a=\text{angle or time})\)

The general flowchart is:

![Flowchart](image)

From one program to another what changes is the "calculation of the coordinates of the \( n \) knots". They are not done in the same fashion if we are interpolating a circle or a straight line. The other parts are common to all the programs. To insure the continuity of the first derivative, the boundary condition on the first knot is picked from the preceeding calculations.
c) **The different programs**

The basic program SP.FTN displays the values of $S(n)$, $S'(n)$, and $\varepsilon(n)$ ($\varepsilon$=error) to the screen. Each set of knots is given by hand and results are presented as figures. (See appendix A)

In reality, knots are separated by an equal time but the distance between two knots can vary a lot because of the welding speed. To take into account this phenomena one program generating knots with a range of speed has been written. The other programs generate knots with a constant speed.

The trials are done on circles and straight lines, the error is in the later programs, displayed as a curve $\varepsilon=f(n)$. The calculations of the knots is automatic, we just give the datas: $R$(radius), $\Omega M$ (angular speed), $N$ (number of knots on which the interpolation must be done).

- **SPC.FTN** generates knots with a constant speed on a circle, calculates the spline functions and displays to the screen the maximum error between two knots and the position (coordinates) of this error. (See appendix B)

The different graphic spline programs are:

- **SPCF.FTN** uses a graphic function. The knots are generated on 1/4 circle with a constant speed. The boundaries conditions are given by the first derivatives. (See appendix C)

- **SPCF2.FTN** and **SPCF12.FTN** are the same as **SPCF.FTN** but, within the first case both of the boundary conditions of order 2 and, in the second case, a start boundary condition of order 1 and the arrival of order 2. (See appendix D)

- **SPDF1.FTN** calculates spline functions on a straight line followed by 1/4 circle with knots generated with a constant speed. The boundary conditions are given by the first derivatives. (See appendix E)

- **SPSF.FTN** draws 1/4 circle with knots generated with a range of speed. The boundary conditions are given by the first derivatives. (See appendix F)

- **SPCFB.FTN** gives the error on 1/4 circle with the first and last knot spaced out a $\pi/2$ angle. (See appendix G)

We have seen in the paragraph "Theory of spline" that the spline functions give the best polynomial curve (smoothest) passing through knots. Consequently it is not necessary to try the well known and classical polynomial functions. Only linear functions have been compared to spline functions:

- **ILF.FTN** is a program interpolating 1/4 circle with linear functions. Knots are generated with a range of speed. (See appendix H)
The last program will be a program giving the accuracy of a waving movement interpolated by spline functions: SPW.FTN. The waving movement is the movement of the torch following the welding path. (See appendix I)

d) Results of the different programs

The parameters, for the graphic spline programs, are given in appendix J. The results are displayed in appendix K. First the theory of natural spline functions is confirmed. The spline functions calculated with first derivative boundary conditions give more accurate results than spline functions calculated with second derivatives or
mixed first and second derivative boundary conditions. (See SPCF2, SPCF, SPCF12)
The error between the spline interpolated curve and two circles of different radius $R_1$ and $R_2$ is directly proportional to the ratio $R_2/R_1$ if the boundary conditions and the angle between each knot are the same. (See SPCF)

![Diagram](image)

On a circle interpolated by spline functions, the maximum errors are situated at the entrance and the exit of the set of knots, and the errors are very small between these two sides. (See SPCF and SPSF)
Calculations of spline functions with 5 knots don't give better results than calculations done with 4 knots. On the other hand, calculations with 4 knots are much better than calculations with 3 knots. Consequently, the best number of knots to calculate the spline functions is 4. (See SPDF1)
After interpolating a circle, we don't arrive tangentially to the straight line. In this case, the error is the biggest at first and decreases after that. (See SPDF1.COMO)

![Diagram](image)

Because spline functions are polynomial functions, a straight line, with good boundary conditions, is interpolated as a straight line.
Spline functions give results about ten times better than linear functions when the chosen path is a circle. (compare SPSF and ILF)

e) Conclusion

After this simulation step, we have a general idea of how the spline functions work. We have seen that the accuracy of this kind of interpolation is good enough for our applications. The trials have been done essentially on circles because one of the applications of this welding robot is to weld cylinders together.
VI Development of the programs on the INTELLEC development system

a) Introduction to the work

In the first part of the practical work we studied the spline functions but we must keep in mind that the program will be implemented in a real servomechanism. The spline interpolator is only one part of the servomechanism and some other tasks must be done during the 5 m.s. (calculation of the position...). Consequently, the running time of the program is an important parameter we must know.

To obtain this running time, we must write first a spline program with a real time configuration which can display the results to verify if they are good (comparison with the general fortran program results). After that we will implement the program in a hardware support to measure the running time itself.

There are two different parts in the spline program. The calculation of the coefficients $C_0$, $C_1$, $C_2$ and $C_3$ every 100 ms and the calculation of the set values every 5 ms. The calculation of the coefficients must be, obviously, done during the 5 ms interval.

b) Presentation of the development system

The INTELLEC serie III micro-computer development system is more than a keyboard, a video display and disk drives: it is a real tool for designing microcomputer software for the IAPX86,88 processor. We are able to write programs, debug programs, link them, locate them and run them on the board. We can connect, in addition to the INTELLEC serie III, an emulator for running in a hardware environment programs written in the 9096 version. The board allows only software entrances or exits via the keyboard and the screen. This is a tool to develop software programs.

The emulation is the controlled execution of the prototype software in an artificial hardware environment. It has the ability to externally control program execution while operating in the user's prototype.

In our case, the final and definitive environment of the program will be the 9096 processor. The serie III development system has been designed to run, on its board, programs written for the 8086 applications. It means that to display results on the screen, the program must be first developed with an 8086 version. Then to measure its running time, it will be written in the 8096 version and loaded in the emulator.

c) Development on the 8086 version

This system offers the possibility to write source programs in a high level language (PLM86) using the facilities of a run-time support. This run-time support is a kind of library with special functions:
DQ$CREATE : creates an input device as console input (:CI:)
DQ$DETACH : deletes an input device
DQ$EXIT : finishes a program
DQ$OPEN : opens a file
DQ$CLOSE : closes a file
DQ$READ : reads an ASCII character into a file
DQ$WRITE : writes an ASCII character into a file

These functions allow us to enter data via the keyboard and to display results to the screen.

Before writing the real program it is necessary to think about the intermediary subprograms. (See appendix L)

INTCAR : gets an ASCII sting of characters from the keyboard
GETINT : gets an transforms an ASCII string into an integer value.
EXIST : transformes an INTEGER value to an ASCII string and writes it on the screen
OUTCAR : writes an ASCII string from memories to the screen.

All the programs are written in PLM because:
- this language has a bloc structure and control construction that aid structured programming.
- this is a high level language with all the advantages (no need to be concerned by the details of the target processor, use of data types and structure).
- it has the facilities for such data structures as structured array and pointer-based dynamic variables.
- PLM programs are portable across different INTEL's processor.

We can write the spline program in two different ways. The first idea is to take the general fortran program and to rewrite it in the same way for our application (See SPM.PLM in appendix M), the second one is to write something completely different on the basis of what is said in "application of the calculations for four knots". (See SP.PLM in appendix M) The two flowcharts are the same, only the programming method changes). The results of the fortran spline program SP.FTN and the SPM.PLM program are displayed in the appendix N.
All the calculations are made with real values, but we display on the screen some integers values which are easier to do.

Flowchart:

1. Initialise the 4 first values
2. Initialise the boundary conditions
3. Calculations of the 4 coefficients
4. Calculations of the set points. Write them to the screen.
5. Next knot
6. Continue?
   - Yes: Go to step 4
   - No: Exit
d) Measurements of the running time on the 8096 version

As explained before during the 5 m.s interval a lot of calculations must be done: calculations of the next coefficients $C_0$, $C_1$, $C_2$, $C_3$, calculation of the position and calculation of the next set point.

So the running time of a program takes a lot of importance and we want to verify that all the calculations can be done during these 5 ms. To measure the running time we use an emulator. The hardware part of the emulator is composed of two timers, one PWM output, a 10 bits A/D converter, a high speed I/O unit, 5 I/O ports.

The idea is to trigger an output port each time the program is finished and to run it indefinitely in a loop. In this way, the time between each change of state of the output port is the running time of the program. The output of the part is visualized on the screen of an oscilloscope.

The single flowchart is:

```
  Program to test

Trigger the output port
```

We want to measure the different running times of calculation (coefficients and set points). As explained in the § "Development of the 8086 version", the programs can be written in two different versions. The results are:

<table>
<thead>
<tr>
<th>SPM96.PL</th>
<th>SPM6.PL</th>
<th>CALM96.PL</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.2 m.s</td>
<td>6.8 m.s</td>
<td>3.4 m.s</td>
</tr>
</tbody>
</table>
In this first trial, all the values are declared as real values. It means that all the calculations (x, /, +, -) are executed in a special "real unit" of the processor and take a lot of time. We can reduce this time only by using "long integer" values (32 bits) but we must take care of the overflow and of the accuracy. We can consider, here, that the calculation time of the coefficient (SP96.PLМ) is satisfying but we have to reduce the time of calculation of the set points.

After rewriting the setpoint calculation program with long integer values, we find a new time of:

\[ \text{CAL96} : 0.48 \text{ m.s} \]

The different programs SP96.PLМ, SPM96.PLМ, CAL96.PLМ and CALM96.PLМ can be seen in the appendix O.

e) Conclusion

The other programs, which must be included in the servo mechanism, have not yet been written. Consequently we do not know if the different running times are short enough to make all the calculations during the required time. Some documentation does not appear in this report because it is very big and not necessary for understanding the work. Nevertheless, for more information you can consult the INTELL's books: "PL/M-86 user's guide", "PL/M-96 user's guide", "Run time support manual for IAPX 86,88 applications", "ISIS-II user's guide", "IAPX 86,88 family utilities user's guide", "MCS-96 utilities user's guide", "Microcontroller handbook".
VII General conclusion

This practical work gave me a general view of how to develop a project. The different steps, with their significantes, are:
- the theoretical approach helps to understand all the details of the study and allows to have performance results.
- the general simulation shows which parameters are important, how the theory is working and if the choice of method is accurate.
- the simulation on a micro-computer is important to see if the particular application gives some coherent results compared with the general theory.
- the trials in a hardware environment give a real idea of what happens in real time. This is the final step where we decide if the method is acceptable.

This pyramidial scheme (from the more general to the particular shape) finds some applications in other research projects. I discovered the important role of the simulation.

The spline function program, as part of the whole servo-mechanism, has been developed in a certain context. The team responsible for the servo design followed my project and gave me the necessary indications. I learned how to include a particular work in a general design with all the constraints that apply.

The large simulation part gave a chance to become familiar with the computers and with some programming problems especially real-time problems. The hardware simulation taught me a lot about micro-processors.
Résumé en français:

Le projet s'est déroulé dans les locaux de l'université de technologie de Eindhoven et a été effectué au bénéfice de la société APA. Celle-ci travaille actuellement à la conception d'un robot de soudure devant prendre place dans des ateliers flexibles.

Afin d'avoir un asservissement cohérent, chaque moteur du robot doit recevoir une valeur pilote toute les 5 m.s. Le microprocesseur central, occupé à calculer les points de passage obligé pour chaque moteur, ne peut accomplir cette tâche: il peut seulement fournir une valeur toute les 100 m.s. à chaque moteur.

L'idée consiste alors à munir chaque moteur d'un microprocesseur esclave recevant une valeur toute les 100 m.s. et restituant, entre autre, la valeur des points intermédiaires toutes les 5 m.s.

Mon sujet était de réaliser une étude sur les fonctions d'interpolation appelé 'spline' et d'en appliquer la théorie à la conception d'un programme temps réel. Celui-ci devait restituer des valeurs pilotes toutes les 5 m.s à partir des points connus toutes les 100 m.s.

Les différentes étapes ont été:
- l'étude théorique des fonctions spline.
- la simulation d'interpolations par fonctions spline sur un ordinateur de grande puissance.
- l'application du programme à notre cas particulier et sa simulation sur un système de développement INTELLEC.
- l'écriture et la mise au point des programmes temps réel.

Les résultats obtenus permettent d'apprécier la précision et la rapidité d'une telle interpolation.

Ce projet s'est révélé particulièrement enrichissant. Il m'a permis de comprendre les processus temps réel mis en jeu dans une commande de robot et d'acquérir une expérience générale en informatique. J'ai également appris de pouvoir participer à un travail d'équipe.
This program is the basic spline program. The programmer gives the following data: N, X(I), Y(I), ITYPEL, ITYPER, VALL, VALR, IMIN, IMAX.

The variables are:

- **N**: number of knots.
- **X(I), Y(I)**: coordinates of the knots.
- **XREF(I), YREF(I)**: coordinates of the set points.
- **ITYPER, ITYPEL**: left or right boundary condition of order one (1) or two (2).
- **VALL, VALR**: left or right boundary condition.
- **DYREF**: first derivative value of the spline function on each set point.
- **C**: second derivative values of the spline function on each knot.
- **XMIN, XMAX**: interval in which the set point values are calculated.

The boundary conditions can be given either by the first derivative either by the second.

```fortran
REAL*8 X(400), Y(400), C(400), XREF(400), YREF(400), DYREF(400), TVALL, VALR
INTEGER I, N, ITYPEL, ITYPER, CODE, IMIN, IMAX

DO 2 I=1,400
  XREF(I)=0.5*I
20 WRITE (1,12)
12 FORMAT ('I, X(I), Y(I), I=1,N')
  READ (1,*,ERR=10) N, (X(I), Y(I), I=1,N)
  IF (N.EQ.0) GOTO 90
40 WRITE (1,14)
14 FORMAT ('ITYPEL, VALL, ITYPER, VALR')
  READ (1,*,ERR=14) ITYPEL, VALL, ITYPER, VALR
  CALL BIAC (C,X,Y,N,DYREF, ITYPEL, VALL, ITYPER, VALR, CODE)
  WRITE (1,18) (C(I), I=1,N)
18 FORMAT ('C = '/(5E15.5))

20 WRITE (1,22)
22 FORMAT ('IMIN, IMAX')
  READ (1,*,ERR=20) IMIN, IMAX
  IF (IMAX.EQ.0) GOTO 10
  CALL BIVALI (C, X, Y, N, XREF, YREF, DYREF, IMAX)
  WRITE (1,24) (XREF(I), YREF(I), DYREF(I), I=IMIN, IMAX)
24 FORMAT (3E15.6)
60 CONTINUE
```

This subroutine calculates the N second derivative values of the spline function at each knot.
SUBROUTINE BIAC (COEFF, XNOD, YNOD, NNOD, WORK,
* TYPEL, VALL, TYPER, VALR, CODE)

INTEGER NNOD, TYPEL, TYPER, CODE
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), WORK(NNOD), VALL, VALR

REAL H, OLDH, F, WORK1, COEFF1, YH
INTEGER I, N1, IBACK, FAIL

For the appropriate equations refer to ...

In this implementation, all equations are multiplied by 6 (six).
Moreover:

1. The equations are build ("assembled") for the intervals 1, 2 etc.
   So they are made in two rounds (normally two intervals are
   involved in the formulation of each equation)

2. Solution of the tridiagonal set of equations is done by
   Gaussian elimination (without pivoting). The elimination
   is done immediately, backwards-substitution ends the
   solution-process.

FAIL=3
IF (NNOD.LT.3) GOTO 90

BEGIN CONDITION

Equations for interval 1 (X(1) ... X(2)):
\[ \text{dY1} = (Y_r - Y_1) / H - H/6 \times (2 \times C1 + C_r) \]
\[ \text{dYr} = (Y_r - Y_1) / H + H/6 \times (C1 + 2 \times C_r) \]

FAIL=2
H=XNOD(2)-XNOD(1)
IF (H.LE.0.) GOTO 90
YH=6.*(YNOD(2)-YNOD(1))/H
COEFF1=YH

FAIL=1
IF (TYPEL.NE.1) GOTO 10

TYPEL = 1 : first derivative prescribed

WORK(1)=2.*H
COEFF(1)=YH-6.*VALL
WORK1=H
IBACK=1
GOTO 20

10 IF (TYPEL.NE.2) GOTO 90

TYPEL = 2 : second derivative prescribed

Use only the equation for dYr in interval 1 and substitute
VALL for C1

WORK(1)=1.
COEFF(1)=VALL
WORK1=0.
IBACK=2

20 N1=NNOD-1
SP. FTN:

Equations for interval i (X(i) ... X(i+1)), i = 2 ... N-1:

\[ dy_1 = \frac{(Y_r - Y_l)}{H - H/6} * (2 * C1 + C_r) \]
\[ dy_r = \frac{(Y_r - Y_l)}{H + H/6} * (C_1 + 2 * C_r) \]

Implement equation i with: \( dy_r(i-1) - dy_1(i) = 0 \)

FAIL = 2
DO 30 I = 2, N1
OLDH = H
H = XNOD(I+1) - XNOD(I)
IF (H .LE. 0.) GOTO 90
F = OLDH/WORK(I-1)
WORK(I) = 2. * (OLDH + H) - F * WORK1
WORK1 = H
YH = 6. * (YNOD(I+1) - YNOD(I))/H
COEFF(I) = YH - COEFF1 - F * COEFF(I-1)
COEFF1 = YH
30 CONTINUE

FAIL = 1
IF (TYPER .NE. 1) GOTO 40

C TYPER = 1 : first derivative prescribed

F = H / WORK(NNOD-1)
WORK(NNOD) = 2. * H - F * H
COEFF(NNOD) = 6. * VALR - COEFF1 - F * COEFF(NNOD-1)
GOTO 50

40 IF (TYPER .NE. 2) GOTO 90

C TYPER = 2 : second derivative prescribed

WORK(NNOD) = 1.
COEFF(NNOD) = VALR

C BACKSUBSTITUTION

50 COEFF(NNOD) = COEFF(NNOD) / WORK(NNOD)
1 = NNOD
60 I = I - 1
COEFF(I) = (COEFF(I) - (XNOD(I+1) - XNOD(I)) * COEFF(I+1))/WORK(I)
IF (I .GT. 1BACK) GOTO 60
FAIL = 0
90 CODE = FAIL RETURN
END

This subroutine calculates the ordinate of a set point.

SUBROUTINE B1AVO (COEFF, XNOD, YNOD, NNOD, X, Y)
INTEGER NNOD
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), X, Y
INTEGER 11, I2
REAL DX, S, CL, CR, YL, YR, F, AO, A1, A2, A3

C Find interval to be used

Y=YNOD(1)
IF (X.LT.XNOD(1)) GOTO 90
Y=YNOD(NNOD)
IF (X.GT.XNOD(NNOD)) GOTO 90

C Linear search is done here

12=1
10 11=12
12=11+1
IF (XNOD(12).LT.X) GOTO 10

C Interval is 11 - 12. Value may be at either boundary

DX=XNOD(12)-XNOD(11)
F=DX**2/6.
CL=CUEFF(11)
CR=CUEFF(12)
YL=YNOD(11)
YR=YNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=YR-YL-F*(2.*CL+CR)
AO=YL
S=(X-XNOD(11))/DX
Y=((A3*S+A2)*S+A1)*S+AO

90 RETURN
END

This subroutine calculates the ordinate of a set point and the first derivative value of the spline function.

SUBROUTINE U1AV1 (COEFF, XNOD, YNOD, NNOD, X, Y, DY)
INTEGER NNOD
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), X, Y, DY

INTEGER 11, 12
REAL DX, S, CL, CR, YL, YR, F, AO, A1, A2, A3

C Find interval to be used

DY=0.
Y=YNOD(1)
IF (X.LT.XNOD(1)) GOTO 90
Y=YNOD(NNOD)
IF (X.GT.XNOD(NNOD)) GOTO 90

C Linear search is done here

12=1
10 11=12
12=11+1
IF (XNOD(12).LT.X) GOTO 10
C Interval is 11 - 12. Value may be at either boundary
C
DX=XNOD(12)-XNOD(11)
F=DX**2/6.
CL=COEFF(11)
CR=COEFF(12)
YL=YNOD(11)
YR=YNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=3.*F*CL+CR
A0=YL
S=(X-XNOD(11))/DX
DY=((3.*A3*S+2.*A2)*S+A1)/DX
Y=((A3*S+A2)*S+A1)*S+A0
90 RETURN
END

C This subroutine calculates the ordinate of a set point,
C and the first and second derivatives of the spline function.
C
SUBROUTINE B1AV2 (COEFF, XNOD, YNOD, NNOD, X, Y, DY, DDY)
INTEGER NNOD
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), X, Y, DY, DDY
C
INTEGER 11,12
REAL DX, S, CL, CR, YL, YR, F, A0, A1, A2, A3
C
C Find interval to be used
C
DY=0.
DDY=0.
Y=YNOD(1)
IF (X.LT.XNOD(1)) GOTO 90
Y=YNOD(NNOD)
IF (X.GT.XNOD(NNOD)) GOTO 90
C
C Linear search is done here
C
I2=1
10 I1=I2
I2=I1+1
IF (XNOD(I2).LT.X) GOTO 10
C
C Interval is 11 - 12. Value may be at either boundary
C
DX=XNOD(12)-XNOD(I1)
F=DX**2/6.
CL=COEFF(11)
CR=COEFF(12)
YL=YNOD(11)
YR=YNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=3.*F*CL+CR
A0=YL
S=(X-XNOD(11))/DX
DDY=2.*((3.*A3*S+A2)/DX**2

Appendix A
This subroutine calculates the ordinates of the set point values between 0 and IMAX.

SUBROUTINE H1AVLO (COEFF, XNOD, YNOD, NNOD, XL, YL, NL)
INTEGER NNOD, NL
REAL COEFF (NNOD), XNOD (NNOD), YNOD (NNOD), XL (NL), YL (NL)

INTEGER I1, I2, IL
REAL DX, S, CO, C1, YO, Y1, F, A0, A1, A2, A3

IL = 1

_BEFORE FIRST XNOD

10 IF (XL (IL) .GE. XNOD (1)) GOTO 20
   YL (IL) = YNOD (1)
   IL = IL + 1
1F (IL .GT. NL) GOTO 90
   GOTO 10

_LINEAR SEARCH TO FIND APPROPRIATE INTERVAL

20 I2 = 1
30 I1 = I2 + 1
   I2 = I1 + 1
   IF (I2 .GT. NNOD) GOTO 80
   IF (XNOD (I2) .LT. XL (IL)) GOTO 30

_INTERVAL IS I1 - I2. VALUE MAY BE AT EITHER BOUNDARY

DX = XNOD (I2) - XNOD (I1)
F = DX**2/6.
CO = COEFF (I1)
C1 = COEFF (I2)
YO = YNOD (I1)
Y1 = YNOD (I2)
A3 = F*(C1-C0)
A2 = 3.*F*C0
A1 = Y1 - YO + (2.*CO+C1)
A0 = YO
40 S = (XL (IL) - XNOD (I1)) / DX
   YL (IL) = ((A3*S+A2)*S+A1)*S+A0
   IL = IL + 1
1F (IL .GT. NL) GOTO 90
   IF (XL (IL) .LE. XNOD (I2)) GOTO 40
   GOTO 30

_Beyond last XNOD

80 YL (IL) = YNOD (NNOD)
   IL = IL + 1
   IF (IL .LE. NL) GOTO 80
This subroutine calculates the set point values between
0 and IMAX and the first derivative value of the spline function.

SUBROUTINE BIAVL1 (COEFF, XNOD, YNOD, NNOD, XL, YL, DYL, NL)
INTEGER NNOD, NL
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), XL(NL), YL(NL), DYL(NL)

INTEGER I1, I2, IL
REAL DX, S, CO, C1, YO, Y1, F, A0, A1, A2, A3

IL=1

Before first XNOD

10 IF (XL(IL) .GE. XNOD(I)) GOTO 20
   DYL(IL)=0.
   YL(IL)=YNOD(I)
   IL=IL+1
   IF (IL .GT. NL) GOTO 90
   GOTO 10

Linear search to find appropriate interval

20 I2=1
30 I1=I?
   I2=I1+1
   IF (I2 .GT. NNOD) GOTO 80
   IF (XNOD(I2) .LT. XL(IL)) GOTO 30

Interval is I1 - I2. Value may be at either boundary

DX=XNOD(I2)-XNOD(I1)
   F=DX**2/6.
   CO=COEFF(I1)
   C1=COEFF(I2)
   YO=YNOD(I1)
   Y1=YNOD(I2)
   A3=F*(-C1-CO)
   A2=3.*F*CO
   A1=Y1-YO-F*(*2.*CO+C1)
   A0=YO
40 S=(XL(IL)-XNOD(I1))/DX
   DYL(IL)=((3.*A3*S+2.*A2)*S+A1)/DX
   YL(IL)=((A3*S+A2)*S+A1)*S+A0
   IL=IL+1
   IF (IL .GT. NL) GOTO 90
   IF (XL(IL) .LE. XNOD(I2)) GOTO 40
   GOTO 30

Beyond last XNOD

80 DYL(IL)=0.
   YL(IL)=YNOD(NNOD)
This subroutine calculates the set point ordonates between 0 and IMAX, the first and second derivatives of the spline function.

SUBROUTINE B1AVL2 (COEFF, XNOD, YNOD, NNOD, XL, YL, DYL, DDYL, NL)
INTEGER NNOD, NL
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), XL(NL), YL(NL), DYL(NL),
      DDYL(NL)

INTEGER 11, 12, IL
REAL DX, S, CO, C1, YO, Y1, F, AO, A1, A2, A3

IL = 1

Before first XNOD

10 IF (XL(IL) .GE. XNOD(1)) GOTO 20
   DYL(IL) = 0.
   DYL(IL+1) = 0.
   YL(IL) = YNOD(1)
   IL = IL + 1
   IF (IL .LE. NNOD) GOTO 90
   GOTO 10

Linear search to find appropriate interval

20 12 = 1
30 11 = 12
   12 = 11 + 1
   IF (12 .GE. NNOD) GOTO 80
   IF (XNOD(12) .LT. XL(IL)) GOTO 30

Interval is 11 - 12. Value may be at either boundary

DX = XNOD(12) - XNOD(11)
F = DX**2 / 6.
CO = COEFF(11)
C1 = COEFF(12)
YO = YNOD(11)
Y1 = YNOD(12)
A3 = F * (C1 - CO)
A2 = 3. * F * CO
A1 = Y1 - YO + * (2. * CO + C1)
AO = YO
40 S = (XL(IL) - XNOD(11)) / DX
   DYL(IL) = 2. * (3. * A3 * S + A2) / DX**2
   DYL(IL+1) = (S * A3 * S + 2. * A2 * S + A1) / DX
   YL(IL) = (A3 * S + A2) * S + A1) * S + AO
   IL = IL + 1
   IF (IL .LE. NL) GOTO 90
IF (XL(IL).LE.XNOD(I2)) GOTO 40
GOTO 30
C
C Beyond last XNOD
C
80 DDYL(IL)=0.
   DYIL(IL)=0.
   YL(IL)=YNOD(NNOD)
   IL=IL+1
   IF (IL.LE.NL) GOTO 80
C
C All done
C
90 RETURN
END
This program calculates a spline interpolated curve with the knots placed on a circle. It restitutes the maximal error between two knots. The interpolation is done on the curve $y=f(x)$ and $z=f(x)$ where $x$ is the time.

The parameters are:
- N: number of knots
- $x(i), y(i)$: coordinates of the knots in space
- $z(i)$: coordinate in time of the knots
- $x_{ref}(i), y_{ref}(i), z_{ref}(i)$: coordinates of the calculated set points.
- $y_{ref}(i), z_{ref}(i)$: coordinates of the calculated set points.

Functions $y = f(x)$ and $z = f(x)$ where $x$ is the time.

Where the maximal error occurs:
- DR: maximum error between two knots
- VALY, VALZ: left boundary conditions
- VALY, VALZ: right boundary conditions
- UN: angular speed
- LR: length of the circle before arriving on the circle
- K: angular speed
- PAS: given $x_{ref}(in time)$
- TH: boundaries conditions are the first derivatives. The angular speed is constant.

REAL *R (UN, LD, R, TETA, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETA0, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
*VALY, VALZ, TETAO, TETAO, ANC, T, VALY, VALZ,
14  
Z(1)=0
Y(1)=0
X(1)=0
I=2

15  
ANG=TETA+TETA*(J-2)
IF (ANG.GT.1.570796327) GOTO 16
X(I)= (1-1)*I
Y(I)= R-R*DCOS(ANG)
Y(I)= L+R*DSIN(ANG)
I=I+1
GOTO 15

16  
K1=1
K2=I+6
DU J= K1, K2
X(K)= (K-1)*I
Y(K)= R*EL0.570796327+TETA*(K-2))

18  
Z(K)=K*6I
C Calculation of the different boundary conditions for
C the cases TETA<(PI/2) and TETA>(PI/2).
C
19  
ANG=TETA+(1+(N-2))
IF (ANG.GT.1.570796327) GOTO 20
VALYK=R*UM*DSIN(ANG)
VALZR=R*UM*DCOS(ANG)
GOTO 21

20  
VALYK=R*UM
VALZR=0

22  
IF (I.NE.0) GOTO 24
VALYL=0
VALYL=R*UM
GOTO 25

24  
VALYL=ZYL

26  
Z(K+2)=Z(11)
CALL B1AC (CY, XB, YB, N, DYREF, ITYPEF, VALYL, ITYPE, VALYR)

C Calculation of the coefficients and the set points for N knots.
C
25  
DU J=1, N
XI=I+K
XB(K)=X(I)
YI=I+K

28  
ZB(K)=Z(I)
CALL B1AC (CY, XB, YB, N, XREF, ITYPEF, VALYL, ITYPE, VALYR)

J=0

30  
J=J+1
XREF(J)=PAS*J+T*1
IF (XREF(J).LE.XB(2)) GOTO 30

CALL B1AVJ (CY, XB, YB, N, XREF, YREF, DYREF, ITYPEF, VALYL, ITYPE, VALYR)
CALL B1AVJ (CY, XB, YB, N, XREF, ZREF, DZREF, ITYPEF, VALYL, ITYPE, VALYR)

1F (YREF.1.IMAX).GT.R) GOTO 10
K=0
I=1+I

C Find the maximal error between two knots.
This subroutine calculates the second derivative values of the spline function at each knot.

SUBROUTINE H1AC (COEFF, XNOD, YNOD, NNOD, WORK, *
    * TYPEL, VALL, TYPEP, VALR)
    INTEGER NNOD, TYPEL, TYPEP, CODE
    REAL*8 COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), WORK(NNOD), VALL, VALR

    REAL*8 H, OL, DH, F, WORK1, COEFF1, YH
    INTEGER I, NI, IBACK, FAIL

    For the appropriate equations refer to ...

In this implementation, all equations are multiplied by 6 (six).

Moreover:
1. The equations are build ("assembled") for the intervals 1, 2 etc.
   So they are made in two rounds (normally two intervals are involved in the formulation of each equation)
2. Solution of the tridiagonal set of equations is done by Gaussian elimination (without pivoting). The elimination is done immediately, backwards-substitution ends the solution-process.

FAIL = 3
IF (NNOD.LT.3) GOTO 90

C BEGIN CONDITION
C Equations for interval 1 (X(1) ... X(2)):
  dY1 = (Y1 - Y2) / H - H/6 * (2 * C1 + Cr)
  dY2 = (Y2 - Y1) / H + H/6 * (C1 + 2 * Cr)

FAIL = 2
IF (NNOD.LT.2) GOTO 90
YH = 6. * (YNOD(2) - YNOD(1)) / H
COEFF1 = YH
SPCF.FTN:

THIS PROGRAM IS THE SAME AS SPC.FTN BUT USES A GRAPHIC
FUNCTION. IT CALCULATES FIRST THE POSITION OF THE
KNOTS, THEN THE ERRORS AND RESTITUTES THE CURVE ERROR=f(ANGLE)
WHERE X IS THE TIME.
THE INTERPOLATION IS DONE ON THE CURVES Y=f(X) AND Z=f(X).
THE PARAMETERS ARE:

N=NUMBER OF KNOTS
X(I),Y(I)=COORDINATES OF THE KNOTS IN SPACE
Z(I)=COORDINATE IN TIME OF THE KNOTS
XB(I),YB(I),ZB(I)=COORDINATES OF THE KNOTS ON WHICH
THE SPLINE FUNCTIONS ARE CALCULATED.
XREF(I),YREF(I),ZREF(I)=COORDINATES OF THE CALCULATED
SET POINTS.
CY(I),CZ(I)=SECOND DERIVATIV VALUES FOR THE SPLINE
FUNCTIONS Y=(function of time ) AND
Z=(function of time) FOR EACH KNOT
DYREF,DXREF=FIRST DERIVATIV VALUE
XM(I),YM(I),ZM(I):COORDINATES OF THE SET POINT
WHERE THE MAXIMAL ERROR OCCURS
DR=ERROR BETWEEN THE CIRCLE AND THE INTERPOLATED CURVE
VALYR,VALZR=RIGHT BOUNDARY CONDITIONS
VALYL,VALZL=LEFT BOUNDARY CONDITIONS
OM=ANGULAR SPEED
LD=LENGTH BEFORE ARRIVING ON THE CIRCLE
R=RADIUS
TETA=ANGLE WHICH SEPARATES EACH KNOT
PAS=TIME BETWEEN EACH SET POINT
THE BOUNDARIES CONDITIONS ARE THE FIRST
DERIVATIVS. THE ANGULAR SPEED IS CONSTANT.

COMMON /F-NAMES/ N1(20),N2(20)

DATA N1 / 'SPCF', 'I', '18*' /,
* N2 / 'SPCF', 'H', '18*' /
END
SUBROUTINE FUNCT (TARG, YARG, NP, OK, GEG, IRES)
REAL TARG(600), YARG(500), GEG(1)
INTEGER NP, IRES
LOGICAL UK

REAL X(30), Y(30), Z(30), XB(10), YB(10), ZB(10),
*XREF(400), YREF(400), ZREF(400), CY(50), CZ(50), DXREF(400),
*ZDXREF(400), DR(500), XREF(500), TT

INTEGER I, K1, K2, K, ITYPEL, ITYPEH, N, I1, J, IMAX, I2, NP, KL

N=GEG(1)
NP=GEG(2)
LD=GEG(3)
R=GEG(4)
TETA=GEG(5)
PAS=GEG(6)
I=1
TT=1./570796327/2
1F (LD. ME. 0) GOTO 13
TETA=TETA
GOTO 14
13 TETA=(K*TETA-LD)/R
14 Z(1)=0
Y(1)=0
X(1)=0
1=2
15 ANG=11: TAO*11: TETA*(1-2)
1F (ANG. GT. 1.570796327) GOTO 16
X(1)=(1-1)*T
Y(1)=R-K*X)*COS(ANG)
Z(1)=LD+(KIDIN(ANG)
1=1+1
GOTO 15
16 K1=1
K2=1+6
DO 18 K=K1, K2
X(K)=(K-1)*T
Y(K)=R*(11: TAO-0.570796327+TETA*(K-2))
18 Z(K)=(K+L)D
ITYPEL=1
ITYPER=1
1=0
KL=0
19 ANG=11: TETA*11: TETA*(1+(N-2))
1F (ANG. GT. 1.570796327) GOTO 20
VALYR=K*OM*OSIN(ANG)
VALZR=K*OM*DOS(ANG)
GOTO 22
20 VALYR=K*UN
VALZR=0
22 1F (1. NE. 0) GOTO 24
VALYL=0
VALYL=K*UN
GOTO 25
24 VALYL=DYLF
VALYL=IDLF
25 DO 26 K=1, N
11=1+K
XH(K)=X(11)
YH(K)=Y(11)
26 ZH(K)=Z(11)
CALL BIAC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPER, VALYR)
CALL BIAC (CZ, XB, ZB, N, DZREF, ITYPEL, VALZL, ITYPER, VALZR)
J=0
30 J=J+1
XREF(J)=PA(J)+T*J
1F (XREF(J), LE. XB(2)) GOTO 30
1MAX=J-1
CALL BIAVL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYLFI
CALL BIAVL1 (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DZLF)
1F (XREF(IMAX). GT. TT) GOTO 40
K=0
1Z=1+1
32 1F (K. GE. IMAX) GOTO 34
K=K+1
KL=KL+1
**Appendix C**

```fortran
DR(KL)=R-DSQRT((R-YREF(K))**2+(ZREF(K)-LD)**2)
XREF1(KL)=XREF(K)
GOTO 32
34 I=I+1
GOTO 19
40 CK=.TRUE.
NP=KL
DO 50 I=1, NP
YARC(I)=XREF1(I)*OM-TETA+TETA0
YARC(I)=DR(I)
50 CONTINUE
RETURN
END,

SUBROUTINE BIAC (COEFF, XNOD, YNOD, NNOD, WORK,
* TYPEL, VALL, TYPER, VALR)
* INTEGER NNOD, TYPEL, TYPER, CODE
REAL*8 COEFF, XNOD, YNOD (NNOD), WORK (NNOD), VALL, VALR

REAL*8 H, OLDF, F, WORK1, COEFF1, YH
INTEGER 1, NI, IBACK, FAIL

For the appropriate equations refer to ...

In this implementation, all equations are multiplied by 6 (six).
Moreover:
1. The equations are build ("assembled") for the intervals 1, 2 etc.
   So they are made in two rounds (normally two intervals are
   involved in the formulation of each equation)
2. Solution of the tridiagonal set of equations is done by
   Gaussian elimination (without pivoting). The elimination
   is done immediately, backwards-substitution ends the
   solution-process.

FAIL=1
IF (NNOD. LT. 1) GOTO 90

BEGIN CONDITION
Equations for interval 1 (X(1) ... X(2)):
   dY1 = (Yr - Yl) / H - H/6 * (2 * C1 + C2)
   dYr = (Yr - Yl) / H + H/6 * (C1 + 2 * C2)

FAIL=2
IF (XNOD(2)-XNOD(1))
   IF (H, LE, 0.) GOTO 90
   YH=6. * (YNOD(2)-YNOD(1))/H
   COEFF1=YH

FAIL=1
IF (TYPEL. NE. 1) GOTO 10

TYPEL = 1 : first derivative prescribed
   WORK(1)=2. *H
   COEFF(1)=YH-6. *VALL
```


WORK1=H
1BACK=1
GOTO 20

10 IF (TYPE1 .NE. 2) GOTO 90

C TYPE1 = 2 : second derivative prescribed

C Use only the equation for dYr in interval 1 and substitute
C VAL for Cl

WORK(1)=1.
COEFF(1)=VAL.
WORK1=0.
1BACK=2

20 N1=NNUD-1

C Equations for interval i (X(i) ... X(i+1)), i = 2 ... N-1:

C dY1 = (Yr - Y1) / H - H/6 * (2*C1 + Cr)
C dYr = (Yr - Y1) / H + H/6 * (Cl + 2*Cr)

C Implement equation i with: dYr(i-1) - dYl(i) = 0

I-FAIL=2
DO 30 I=2, N1
OLDH=H
H=XNOD(I+1)-XNOD(I)
IF (H .LE. 0.) GOTO 90
F=OLDH/WORK(I-1)
WORK(I)=2.*(OLDH+H) - F*WORK1
WORK1=H
YH=6.*((YNOD(I+1)-YNOD(I))/H)
COEFF(I)=YH-COEFF1 - F*COEFF(I-1)
COEFF(I)=YH
30 CONTINUE

C END CONDITION

FAIL=1
IF (TYPE1 .NE. 1) GOTO 40

C TYPE1 = 1 : first derivative prescribed

F=H/WORK(NNUD-1)
WORK(NNUD)=2.*H - F*H
COEFF(NNUD)=6.*VALR-COEFF1 - F*COEFF(NNOD-1)
GOTO 50

40 IF (TYPE1 .NE. 2) GOTO 90

C TYPE1 = 2 : second derivative prescribed

WORK(NNOD)=1.
COEFF(NNOD)=VALR

C BACKSUBSTITUTION

50 COEFF(NNOD)=COEFF(NNOD)/WORK(NNOD)
1=NNOD
60 I=1-1
C SPC., F-IN:

COEFF(I) = (COEFF(I) - (XNOD(I+1) - XNOD(I)) * COEFF(I+1)) / WORK(I)
IF (I .GT. BACK) GOTO 60
FAIL = 0

90 COEFF = FAIL
RETURN
END

C
C
C
SUBROUTINE BIAVL1 (COEFF, XNOD, YNOD, NNOD, XL, YL, DYL, NL, DLF)
INTEGER NNOD, NL
REAL*8 COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), XL(NL), YL(NL), DYL(NL)

INTEGER I1, I2, IL
REAL*8 DX, S, C0, C1, Y0, Y1, SF, AO, A1, A2, A3, DLF

1L = 1
C Before First XNOD
C
10 IF (XL(IL).GE. XNOD(I1)) GOTO 20
DYL(IL) = 0.
YL(IL) = YNOD(I1)
IL = IL + 1
IF (I1 .GT. NNOD) GOTO 90
GOTO 10

C Linear search to find appropriate interval
C
20 12 = 1
30 11 = 1?
12 = 11 + 1
IF (I2. GT. NNOD) GOTO 80
IF (XNOD(I2).LT. XL(IL)) GOTO 50

C Interval is 11 - I2. Value may be at either boundary
C
DX = XNOD(2) - XNOD(1)
FX = DX*2/A.
C0 = COEFF(I1)
C1 = COEFF(I2)
Y0 = YNOD(I1)
Y1 = YNOD(I2)
A3 = F*CO
A2 = 3. *F*CO
A1 = Y1 - Y0 - (*2. *CO+C1)
AO = Y0
SF = 1
DLF = ((3. *A3*SF+2. *A2)*SF+AO)/DX
40 (= (XL(IL) - XNOD(I1))/DX
DYL(IL) = ((3. *A3*S+S+2. *A2)*S+AO)/DX
YL(IL) = ((A3*S+A2)*S+A1)*S+AO
IL = IL + 1
IF (I1 .GT. NL) GOTO 90
IF (XI(IL).LE. XNOD(I2)) GOTO 40
GOTO 30

C
C Beyond last XNOD
Appendix C

C SPCF.F7N:

C

80 BYL(1L)=0.
Y.L(1L)=YNQJ(NNOD)
1L=1L+1
IF (IL.LE.NL) GOTO 80

C All done
C

90 RETURN
END
SPCF12.FTN:

This program interpolates 1/4 circle by spline functions, with a first derivative boundary condition and a second one respectively at the entrance and the exit of the set of knots.

The speed is constant.

The subroutines BIAC and B1AVL1 are the same as in SPC.FTN.

The variables are:

- **N**: number of knots.
- **OM**: constant angular speed.
- **LD**: length of the straight line before entering the circle. It must be smaller than R*TETA.
- **R**: radius of the circle.
- **TETA**: angle between each knot.
- **PAS**: time between each set point.

We display to the screen the curve \( YARG=f(TARG) \).

---

```plaintext

BLOCK DATA
COMMON /FNAMES/ N1(20), N2(20)

DATA N1 / 'SPCF', 'I', '18*' /,
* N2 / 'SPCF', 'H', '18*' /

SUBROUTINE FUNCT (TARG, YARG, NP, OK, GEG, IRES)
REAL TARG(500), YARG(500), GEG(1)
INTEGER NP, IRES

LD=1CAL OK

REAL*8 OM, LD, R, TETA, TETAO, ANG, T, VALYR, VALZR,
*VALYL, VALZL, DYL, DZL, PAS

REAL*8 X(50), Y(50), Z(50), XB(10), YB(10), ZB(10),
*XREF(400), YREF(400), ZREF(400), CY(50), CZ(50), DYREF(400),
*DZREF(400), DR(500), XREF1(500), TT

INTEGER I, K1, K2, K, ITYPEL, ITYPE, N, I1, J, IMAX, I2, NP, KL

N=GEG(1)
OM=GEG(2)
LD=GEG(3)
R=GEG(4)
TETA=GEG(5)
PAS=GEG(6)
T=TETA/OM
TT=1.570796327/OM
IF (LD .NE. 0) GOTO 13
TETAO=TETA
GOTO 14

13 TETAO=(R*TETA-LD)/R
14 Z(I)=0
Y(I)=0
X(I)=0
I=2
15 ANG=TETAO+TETA*(I-2)
IF (ANG, GT, 1.570796327) GOTO 16
X(I)=(I-1)*T
Y(I)=R-R*DCOS(ANG)
```
C SPCF12.FIN:

Z(I)=LD+R*DSIN(ANG)
I=I+1
GOTO 15

16 K1=1
K2=I+6
DO 18 K=K1,K2
X(K)=(K-1)*T
Y(K)=R*(TETAO-0.570796327+TETA*(K-2))
18 Z(K)=R+LD
ITYPEL=1
ITYPER=2
I=0
KL=0

19 ANG=TETAO+TETA*(I+(N-2))
IF (ANG.GT.1.570796327) GOTO 20
VALYR=R*(DM**2)*DCOS(ANG)
VALZR=-R*(DM**2)*DSIN(ANG)
GOTO 22
20 VALYR=0
VALZR=0
22 IF (I.NE.0) GOTO 24
VALYL=0
VALZL=R*DM
GOTO 25
24 VALYL=DYLF
VALZL=DZLF
25 DO 26 K=1,N
II=I+K
XB(K)=X(I1)
YB(K)=Y(I1)
26 ZB(K)=Z(I1)
CALL BIAC (CY,XB,YB,N,DYREF,ITYPEL,VALYL,ITYPER,VALYR)
CALL BIAC (CZ,XB,ZB,N,DZREF,ITYPEL,VALZL,ITYPER,VALZR)
J=O
30 J=J+1
XREF(J)=PAS+J+T*I
IF (XREF(J).LE.XB(2)) GOTO 30
IMAX=J-1
CALL BIAVL (CY,XB,YB,N,XREF,YREF,DYREF,IMAX,DYLF)
CALL BIAVL (CZ,XB,ZB,N,XREF,ZREF,DZREF,IMAX,DZLF)
IF (XREF(IMAX).GT.TT) GOTO 40
K=O
I2=I+1
32 IF (K.GE.IMAX) GOTO 34
K=K+1
KL=KL+1
DR(KL)=R-DSORT((R-YREF(K))**2+(ZREF(K)-LD)**2)
XREF1(KL)=XREF(K)
GOTO 32
34 I=I+1
GOTO 19
40 OK=.TRUE.
NP=KL
DO 50 I=1,NP
TARG(I)=XREF1(I)*DM-TETA+TETAO
YARG(I)=DR(I)
50 CONTINUE
RETURN
END
SPDF1.FTN:

The program interpolates 1/4 circle by spline functions with 3 knots given on a straight line just before the circle. The boundary conditions are given by the first derivatives, and the speed is constant. The subroutines BIAC and BIAVL1 are the same as in SPC.FTN.

The variables are:
- N: number of knots.
- OM: constant angular speed.
- LD: length of the straight line before entering the circle. It must be smaller than R*TETA.
- R: radius of the circle.
- TETA: angle between each knot.
- PAS: time between each set point.

We display to the screen the curve YARG=f(TARG).

```

```

```
15 ANG=TETA0+TETA*(I-4)
   IF (ANG.GT.1.570796327) GOTO 16
   X(I)=(I-1)*T
   Y(I)=R-R*COS(ANG)
   Z(I)=LD+Z(3)+R*SIN(ANG)
   I=I+1
   GOTO 15
16 K1=I
   K2=I+10
   DO 18 K=K1,K2
   X(K)=(K-1)*T
   Y(K)=R*(TETA0-0.570796327+TETA*(K-4))
18 Z(K)=R+Z(3)+LD
   ITYPEL=1
   ITYPER=1
   I=0
   K1=0
   ZREF1=Z(3)+LD
19 ANG=TETA0+TETA*I
   IF (ANG.GT.1.570796327) GOTO 20
   VALYR=R*OM*SIN(ANG)
   VALZR=R*OM*COS(ANG)
   GOTO 22
20 VALYR=R*OM
   VALZR=0
22 IF (I.NE.0) GOTO 24
   VALYL=0
   VALYL=R*OM
   GOTO 25
24 VALYL=DYLF
   VALYL=DZLF
25 DO 26 K=1,N
   II=I+K
   XB(K)=X(II)
   YB(K)=Y(II)
26 ZB(K)=Z(II)
   CALL BIAC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPER, VALYR)
   CALL BIAC (CZ, XB, ZB, N, DZREF, ITYPEL, VALZL, ITYPER, VALZR)
   J=0
30 J=J+1
   XREF(J)=PAS*J+T*I
   IF (XREF(J).LE.XB(2)) GOTO 30
   IMAX=J-1
   CALL BIACL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYLF)
   CALL BIACL1 (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DZLF)
   IF (XREF(IMAX).GT.TT) GOTO 40
   K=0
32 IF (K.GE.IMAX) GOTO 36
   K=K+1
   KL=KL+1
   IF (ZREF(K).LE.ZREF1) GOTO 33
   IF (YREF(K).GT.R) GOTO 34
   DR(KL)=R-SQRT((R-YREF(K))**2+(ZREF(K)-ZREF1)**2)
   GOTO 35
33 DR(KL)=YREF(K)
   GOTO 35
34 DR(KL)=ZREF1+R-ZREF(K)
35 ZREF1(KL)=XREF(K)
   GOTO 32
36  I=I+1
    GOTO 19
40  OK=. TRUE.
    NP=KL
    DO 50  I=1, NP
    TARG(I)=XREF1(I)
    YARG(I)=DR(I)
50  CONTINUE
    RETURN
    END
SPSF.FTN:

This program interpolates 1/4 circle by spline functions, with first derivative boundary conditions and a range of speed.

The variables are:
- \( N \): number of knots on which the interpolation will be done.
- \( \text{VMAX} \): maximum angular speed to reach.
- \( \text{AMAX} \): angular acceleration
- \( R \): radius of the circle
- \( T \): time between each knot
- \( \text{PAS} \): time between each set points

The subroutines B1AC and BIAVL1 are the same as in SPC.FTN. We display to the screen the curve \( \text{YARG}=f(\text{TARG}) \).

BLOCK DATA
COMMON /FNAMES/ N1(20), N2(20)

DATA N1 / 'SPSF', 'I', '18*' /,
* N2 / 'SPSF', 'H', '18*' /
END

SUBROUTINE FUNCT (TARG, YARG, NP, QK, GEG, IRES)
REAL TARG(500), YARG(500), GEG(1)
INTEGER NP, IRES

INTEGER N, I, IA, ITYPEP, ITYPEL, K1, K2, K, KL, I1, I2,
* J, IMAX, NP, KA
REAL \( \text{VMAX}, \text{AMAX}, R, T, \text{PAS} \)

REAL*8 TI, T1, T2, T3, TETA1, TETA2,
*VALRY, VAL2R, VALYL, VALZL, DYLF, DZLF

REAL*8 X(100), Y(100), Z(100), ANG(100), XB(10), YB(10),
*ZB(10), CY(10), CZ(10), XREF(400), YREF(400), ZREF(400),
*DYREF(100), DIREF(100), DR(500), XREF1(500)

N=GEQ(1)
VMAX=GEQ(2)
AMAX=GEQ(3)
R=GEQ(4)
T=GEQ(5)

PAS=GEQ(6)

TI=VMAX/AMAX
T1=(1.570796327-AMAX*(T1**2))/VMAX
T2=TI+T1
T3=2*T1+T1
TETA1=AMAX*(T1**2)/2
TETA2=TETA1+(1.570796327-AMAX*(T1**2))

IF (TI.GT.0) GOTO 08
WRITE (1,06)
06 FORMAT ('TI OR TI .LE. T')
08 X(1)=0
Y(1)=0
Z(1)=0
I=2
IA=1
10 X(I)=(I-1)*T
    IF (X(I).GE.T1) GOTO 12
    ANQ(IA)=AMAX*(X(I)**2)/2
    Y(I)=R-R*DCOS(ANG(IA))
    Z(I)=R*DSIN(ANG(IA))
    I=I+1
    IA=IA+1
    QOTO 10

12 X(I)=(I-1)*T
    IF (X(I).GE.T2) GOTO 14
    ANQ(IA)=TETA1+(X(I)-T1)*VMAX
    Y(I)=R-R*DCOS(ANG(IA))
    Z(I)=R*DSIN(ANG(IA))
    IA=IA+1
    I=I+1
    QOTO 12

14 X(I)=(I-1)*T
    IF (X(I).GE.T3) GOTO 16
    ANQ(IA)=TETA2+(VMAX-AMAX*(X(I)-T2)/2)*(X(I)-T2)
    Y(I)=R-R*DCOS(ANG(IA))
    Z(I)=R*DSIN(ANG(IA))
    IA=IA+1
    I=I+1
    QOTO 14

16 K1=1
    K2=I+6
    DO 18 K=K1,K2
    X(K)=T3+(K-K1)*T
    Y(K)=R
    Z(K)=R
18  ITYPEP=1
    ITYPEL=1
    I=0
    KL=O
    KA=N
    IA=N-1
    QOTO 20

20 IF (KA.GE.K2) GOTO 42
    IF (X(KA).GE.T1) GOTO 21
    VALYR=R*DSIN(ANG(IA))*AMAX*X(KA)
    VALZR=R*DCOS(ANG(IA))*AMAX*X(KA)
    QOTO 24

21 IF (X(KA).GE.T2) GOTO 22
    VALYR=R*DSIN(ANG(IA))*VMAX
    VALZR=R*DCOS(ANG(IA))*VMAX
    QOTO 24

22 IF (X(KA).GE.T3) GOTO 23
    VALYR=R*DSIN(ANG(IA))*(VMAX-AMAX*(X(KA)-T2))
    VALZR=R*DCOS(ANG(IA))*(VMAX-AMAX*(X(KA)-T2))
    QOTO 24

23 VALYR=0
    VALZR=0
24 IF (I.NE.0) GOTO 26
    VALYL=0
    VALZL=0
    QOTO 27

26 VALYL=DYLFL
    VALZL=DZLFL
27 DO 28 K=1,N
    I1=I+K
$X_B(K)=X(I1)$
$Y_B(K)=Y(I1)$

28 $Z_B(K)=Z(I1)$
CALL B1AC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPE, VALYR)
CALL B1AC (CZ, XZ, ZB, N, DZF, ITYPEL, VALZL, ITYPE, VALZR)

J=0

32 J=J+1
IF (J.GT.400) QOTO 33
XREF(J)=PAS+J*T*I
IF (XREF(J).LE.XB(2)) QOTO 32

33 IMAX=J-1
CALL B1AL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYL)
CALL B1AL1 (CZ, XZ, ZB, N, XREF, ZREF, DZREF, IMAX, DZL)

K=0
I2=I+1

34 IF (K.GE.IMAX) QOTO 36
K=K+1
KL=KL+1
DR(KL)=R-DSQRT((R-YREF(K))**2+ZREF(K)**2)
XREF(KL)=XREF(K)
QOTO 34

36 I=I+1
KA=KA+1
IA=IA+1
QOTO 20

42 OK=. TRUE.
NP=KL
DO 50 I=1, NP
IF (XREF1(I).GT.T1) QOTO 44
TARG(I)=AMAX**(XREF1(I)**2)/2
QOTO 48

44 IF (XREF1(I).GT.T2) QOTO 46
TARG(I)=TETA1+(XREF1(I)-T1)**VMAX
QOTO 48

46 IF (XREF1(I).GE.T3) QOTO 47
TARG(I)=TETA2+**(VMAX-AMAX**(XREF1(I)-T2)**2*(XREF1(I)-T2)
QOTO 48

47 TARG(I)=1.570796327
48 YARG(I)=DR(I)

50 CONTINUE
RETURN
END
SAME PROGRAM AS SPCF.FTN BUT THERE IS A KNOT
AT THE ENTRANCE AND THE EXIT OF THE CIRCLE.

FIRST DERIVATIVES -LAST AND FIRST POINTS ON THE EXTREMA

BLOCK DATA
COMMON /FNAMES/ N1(20), N2(20)

DATA N1 / 'SPCF', 'I', '1B*', '/,
* N2 / 'SPCF', 'H', '1B*/

SUBROUTINE FUNCT (TARG, YARG, NP, OK, QEG, IRES)
REAL TARG(500), YARG(500), QEG(1)
INTEGER NP, IRES

REAL BM, LD, R, TETA, TETA0, ANG, T, VALYR, VALZR,
* VALYL, VALZL, DYLFR, DZLF, PAS

REAL X(50), Y(50), Z(50), XB(10), YB(10), ZB(10),
* XREF(400), YREF(400), ZREF(400), CY(50), CZ(50), DYREF(400),
* DZREF(400), DR(500), XREF(500), TT

INTEGER I, K1, K2, K, ITYPEL, ITYPER, N, I1, J, IMAX, I2, NP, KL

N=QEG(1)
DM=QEG(2)
LD=QEG(3)
R=QEG(4)
TETA=QEG(5)
PAS=QEG(6)
T=TETA/DM
TT=1.570796327/DM
IF (LD, NE. 0) GOTO 13
TETA0=TETA
GOTO 14
13 TETA0=(R*TETA-LD)/R
14 Z(1)=0
Y(1)=0
X(1)=0
I=2
15 ANG=TETA0+TETA*(I-2)
IF (ANG, GE. 1.570796327) GOTO 16
X(I)=(I-1)*T
Y(I)=R-R*DCOS(ANG)
Z(I)=LD+R*DSIN(ANG)
I=I+1
GOTO 15
16 K1=1
K2=I+6
DO 18 K=K1, K2
X(K)=(K-2)*T+(1.570796327-TETA0-(K1-3)*TETA)/DM
Y(K)=R+R*DCOS(ANG)
Z(K)=LD+R*DSIN(ANG)
18 Z(K)=LD+R
ITYPEL+1
ITYPEP=1
I=O
K1=0
19 ANG = TETA0 + TETA * (I + (N - 2))
   IF (ANG .GE. 1.570796327) GOTO 20
   VALYR = R*OM*DSIN(ANG)
   VALZR = R*OM*DCOS(ANG)
   GOTO 22
20 VALYR = R*OM
   VALZR = 0
22 IF (I.NE.0) GOTO 24
   VALY = 0
   VALZ = R*OM
   GOTO 25
24 VALY = DYLF
   VALZ = DZLF
25 DO 26 K=1, N
   XI = I + K
   XB(K) = X(I)
   YB(K) = Y(I)
26 ZB(K) = Z(I)
   CALL BIAC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPE, VALYR)
   CALL BJAC (CZ, XB, ZB, N, DZREF, ITYPEL, VALZL, ITYPE, VALZR)
   J = 0
30 J = J + 1
   XREF(J) = PAS*J + T*I
   IF (XREF(J) .LE. XB(2)) GOTO 30
   IMAX = J
   CALL BIAVL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYLF)
   CALL BIAVL1 (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DZLF)
   IF (XREF(IMAX) .GT. T) GOTO 40
   K = 0
   12 = I + 1
32 IF (K .GE. IMAX) GOTO 34
   K = K + 1
   KL = K + 1
   DR(KL) = R - DSORT((R - YREF(K))**2 + (ZREF(K) - LD)**2)
   XREF(KL) = XREF(K)
   GOTO 32
34 I = I + 1
   GOTO 19
40 OK = .TRUE.
   NP = KL
   DO 50 I = 1, NP
     TARG(I) = XREF(I)*OM - TETA + TETA0
     YARG(I) = DR(I)
50 CONTINUE
RETURN
END
This subroutine calculates the linear interpolation of a 1:4 circle with a range of speed.

The variables are:
- **VMAX**: maximum angular speed to reach
- **AMAX**: angular acceleration
- **R**: radius of the circle
- **T**: time between two knots
- **PAS**: angle between two set points

We display on the screen the curve ERA1=f(ANG1).

```fortran
SUBROUTINE FUNCT (ANG1, ERA1, NP, OK, GEG)
  REAL ANG1(500), ERA1(500), GEG(1)
  INTEGER NP
  LOGICAL OK

  REAL*8 X(100), Y(100), TARG(100),
  * XREF(100), YREF(100), ERA(100),
  * XREF1(500), YREF1(500)

  VMAX=GES(1)
  AMAX=GES(2)
  R=GES(3)
  T=GES(4)
  PAS=GES(5)

  TI=VMAX/AMAX
  T1=(1.570796327-AMAX*(TI**2))/VMAX
  T2=TI+11
  T3=2*TI+11
  TETA1=AMAX*(T1**2)/2
  TETA2=TETA1+(1.570796327-AMAX*(TI**2))
  IF (TI.GT.0) QOTO 08
  WRITE (1,06)
  06 FORMAT(‘TI , LT. 0’)
  08 I=1
  10 TC(I)=(I-1)*T
  IF (TC(I).GT.T1) QOTO 12
  ANG(I)=AMAX*(TC(I)**2)/2
  X(I)=R-R*DCOS(ANG(I))
  Y(I)=R*DSIN(ANG(I))
  I=I+1
  QOTO 10
  12 TC(I)=(I-1)*T
  IF (TC(I).GT.T2) QOTO 14
  ANG(I)=TETA1+(TC(I)-T1)*VMAX
```
C ILF.FTN:

\[ X(I) = R - R \times \text{DCOS}(\text{ANG}(I)) \]
\[ Y(I) = R \times \text{DSIN}(\text{ANG}(I)) \]
\[ I = I + 1 \]
\[ \text{GOTO 12} \]

14 \[ TC(I) = (I - 1) \times T \]
\[ \text{IF } (TC(I) \times GT. T3) \text{ GOTO 16} \]
\[ \text{ANG}(I) = T1A2 + (VMAX - AMAX \times (TC(I) - T2) / 2) \times (TC(I) - T2) \]
\[ X(I) = R - R \times \text{DCOS}(\text{ANG}(I)) \]
\[ Y(I) = R \times \text{DSIN}(\text{ANG}(I)) \]
\[ I = I + 1 \]
\[ \text{GOTO 14} \]

16 \[ N = I - 1 \]
\[ L = 1 \]
\[ K = 1 \]
\[ I = 2 \]

17 \[ J = 1 \]

18 \[ \text{TARG}(J) = \text{ANG}(I-1) + (J-1) \times \text{PAS} \]
\[ \text{XREF}(J) = R - R \times \text{DCOS}(\text{TARG}(J)) \]
\[ \text{IF } (\text{XREF}(J) \times GT. X(I)) \text{ GOTO 20} \]
\[ J = J + 1 \]
\[ \text{GOTO 18} \]

20 \[ \text{IMAX} = J - 1 \]
\[ \text{DO 22 K} = 1, \text{IMAX} \]
\[ \text{YREF}(K) = Y(I-1) + (Y(I) - Y(I-1)) / (X(I) - X(I-1)) \]
\[ = *(\text{XREF}(K) - X(I-1)) \]
\[ \text{XREF1}(K) = \text{XREF}(K) \]
\[ \text{YREF1}(K) = \text{YREF}(K) \]
\[ \text{ERA}(K) = R - \text{DSQRT}((R - \text{XREF}(K))^2 + \text{YREF}(K)^2) \]
\[ \text{ERA1}(K) = \text{ERA}(K) \]
\[ \text{ANG1}(K) = \text{TARG}(K) \]
\[ 22 \]
\[ K = K + 1 \]
\[ I = I + 1 \]
\[ \text{IF } (I \text{ LE. N}) \text{ GOTO 17} \]
\[ \text{OK} = \text{. TRUE.} \]
\[ \text{NP} = K - 1 \]
\[ \text{RETURN} \]
\[ \text{END} \]
This program interpolates a waving curve \( A \times \sin(X(I)) \) by spline functions. The boundary conditions are given by the first derivatives. The subroutines B1AC and B1AV1 are the same as in SPC.FTN. The parameters are:

- \( N \): number of knots
- \( \omega \): angular speed
- \( T \): time between two knots
- \( \text{PAS} \): angle between two set point
- \( A \): amplitude of the waving curve

We display on the screen the curve \( \text{ERA1}=f(X\text{REF1}) \).

```fortran
BLOCK DATA
COMMON /FNMES/ N1(20), N2(20)

DATA N1 / 'SPWF', 'I', '1B', '/.
DATA N2 / 'SPWF', 'H', '1B', '/
END

SUBROUTINE FUNCT (XREF1, ERA1, NP, OK, QEG)
REAL XREF1(S00), ERA1(S00), QEG(S)
INTEGER NP, OK
REAL*8 OM, T, PAS, A, VALYR, VALYL, DYLF
REAL*8 X(100), Y(100), XB(10), YB(10), CY(10), XREF(100),
9 YREF(100), ERA(100), DYREF(100)

N=QEG(1)
OM=QEG(2)
T=QEG(3)
PAS=QEG(4)
A=QEG(5)
K=1
K2=1
ITYPEL=1
ITYPER=1
DO 10 J=1, 100
X(I)=OM*T*(I-1)
10 Y(I)=A*DCOS(X(I))
I=0
11 I=I+1
VALYR=-A*OM*DSIN(X(I+N-1))
IF (I.NE.1) GOTO 12
VALYL=0
GOTO 14
12 VALYL=DYLF
14 DO 16 J=1, N
K=I+J-1
XB(J)=X(K)
16 YB(J)=Y(K)
CALL B1AC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPER, VALYR)
L=0
```
18 L=L+1
XREF(L)=PAS*(L-1)+T*OM*(I-1)
IF (XREF(L).LE.XB(2)) QOTO 18
IMAX=L-1
CALL B1AVL.1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYL)
DO 20 K1=1, IMAX
ERA(K1)=A*DCOS(XREF(K1))-YREF(K1)
ERA1(K2)=ERA(K1)
XREF1(K2)=XREF(K1)
20 K2=K2+1
IF (I.LT.10) QOTO 11
NP=K2-1
OK=.TRUE.
RETURN
END
These following files give you all the informations about the parameters used in the programs. If one of the parameters is changed when running a program, the new value appears on the column of the drawn. We can modify each parameter as we want.

First appears the names of all the variables, then the initial values of these variables (on the same line and separated by a blanc), then the names of the two axes of the drawn and, to finish, the initial values for the graphic function.

Content of the file SPCF.I:

N
DM(rad/s)
LD
R
TETA(rad)
PAS(s)

4 1. 0. 1. 0.1 0.01
ERROR-VALUE
ANGLE-VALUE:

0 0

1 0 0

Content of the file SPSF.I:

N
VMAX(rad/s)
AMAX(rad/s/s)
R
T(s)
PAS(s)

4 1.570796327 5.235987756 1 0.1 0.005
ERROR-VALUE
ANGLE-VALUE:

0 0

1 0 0

Content of the file ILF.I:

VMAX(rad/s)
AMAX(rad/s/s)
R
T(s)
PAS(rad)

1.570796327 5.235987756 1 0.1 0.005
ERROR-VALUE:
ANGLE-VALUE:
These following files give you all the informations

Content of the file SPWF.I

N
OM(rad/s)
T(s)
PAS(rad)
A
4 1.256 0.1 0.025 2.5
ERROR-VALUE
ANGLE(rad.)
0 0
1 0 0

Content of the file SPDF.I:

N
OM(rad/s)
I D
R
TETA(rad)
PAS(s)
4 1. 0. 1. 0.1 0.01
ERROR-VALUE
TIME-VALUE
0 0
1 0 0
DD$ATTACH: PROCEDURE (PATH$P, EXCEP$P) WORD EXTERNAL;
   DECLARE PATH$P POINTER;
   DECLARE EXCEP$P POINTER;
END;

DD$CREATE: PROCEDURE (PATH$P, EXCEP$P) WORD EXTERNAL;
   DECLARE PATH$P POINTER;
   DECLARE EXCEP$P POINTER;
END;

DD$DETACH: PROCEDURE (CONN, EXCEP$P) EXTERNAL;
   DECLARE CONN WORD;
   DECLARE EXCEP$P POINTER;
END;

DD$OPEN: PROCEDURE (CONN, ACCESS, NUM%BUF$, EXCEP$P) EXTERNAL;
   DECLARE CONN WORD;
   DECLARE (ACCESS, NUM%BUF$) BYTE;
   DECLARE EXCEP$P POINTER;
END;

DD$READ: PROCEDURE (CONN, BUF$,P, COUNT, EXCEP$P) WORD EXTERNAL;
   DECLARE CONN WORD;
   DECLARE (BUF$P, EXCEP$P) POINTER;
   DECLARE COUNT WORD;
END;

DD$WRITE: PROCEDURE (CONN, BUF$P, COUNT, EXCEP$P) EXTERNAL;
   DECLARE CONN WORD;
   DECLARE (BUF$P, EXCEP$P) POINTER;
   DECLARE COUNT WORD;
END;

DD$CLOSE: PROCEDURE (CONN, EXCEP$P) EXTERNAL;
   DECLARE CONN WORD;
   DECLARE EXCEP$P POINTER;
END;

DD$EXIT: PROCEDURE (EXCEP) EXTERNAL;
   DECLARE EXCEP WORD;
END;

$RESTORE
MM2 :DO:

/\ * DECLARATION OF THE RUN-TIME PROCEDURES:
  * DQ$ATTACH: CREATES A CONNECTION TO AN EXISTING FILE.
    ':
  :@C:" : CONSOLE INPUT (EXISTING FILE).
  CI: NAME OF THE CONNECTION.
  DQ$OPEN: OPEN A PREVIOUSLY ESTABLISHED CONNECTION.
    1 MEANS READ ACCESS ONLY.
    0 SIGNIFIES THAT NO BUFFERING SHOULD OCCUR.
  DQ$READ: FETCHES DATA FROM AN OPEN FILE.
    BUFFER$PTR POINTS TO THE AREA WHERE THE
    ASCII CHARACTERS WILL BE STORED.
    6 SPECIFIES THE DESIRED NUMBER OF BYTES TO
    BE READ.
  DQ$CLOSE: WAITS FOR COMPLETION OF I/O OPERATIONS
    TAKING PLACE ON THE FILE.
  DQ$DETACH: BREAKS THE CONNECTION ESTABLISHED BY
    DQ$ATTACH. */

#include("F1:UDI.EXT")

INTCAR :PROCEDURE (BUFFER$PTR) PUBLIC;

  /* CI: SEE ABOVE
   STATUS: WORD VALUE IN WHICH THE OPERATING SYSTEM
          RETURNS AN EXCEPTION CODE.
   ACTUAL: NUMBER OF BYTE TRANSFERRED.
   BUFFER: SEE ABOVE. */

DECLARE CI WORD;
DECLARE STATUS WORD;
DECLARE BUFFER$PTR POINTER;
DECLARE (BUFFER BASED BUFFER$PTR)(6) BYTE;
DECLARE ACTUAL WORD;

CI=DQ$ATTACH(4,':CI:'),@STATUS);
CALL DQ$OPEN(CI,1,0,@STATUS);
ACTUAL=DQ$READ(CI,BUFFER$PTR,6,@STATUS);
CALL DQ$CLOSE(CI,@STATUS);
CALL DQ$DETACH(CI,@STATUS);
END INTCAR;
END MM2;
DECLARE J BYTE;
DECLARE CO WORD;
DECLARE STATUS WORD;

CO=DO$CREATE(2, 'CO:', @STATUS);
CALL DO$OPEN(CO, 2, 0, @STATUS);
CALL DO$WRITE(CO, J, 1, @STATUS);
CALL DO$CLOSE(CO, @STATUS);
CALL DO$DETACH(CO, @STATUS);
END OUTCAR;
END MM1;
GETINT. THIS PROCEDURE GETS AN ASCII STRING FROM THE
KEYBOARD, CALCULATES ITS INTEGER VALUE,
AND PUT THE CURSOR TO THE NEXT LINE. */

MM4 : DO;

/*DECLARATION OF THE EXTERNAL PROCEDURES:
INTCAR: GETS AN ASCII STRING FROM THE KEYBOARD.
OUTCAR: DISPLAYS TO THE SCREEN AN ASCII VALUE. */

INTCAR :PROCEDURE (BUFFER$PTR) EXTERNAL;
DECLARE BUFFER$PTR POINTER;
END;

OUTCAR :PROCEDURE (J) EXTERNAL;
DECLARE J BYTE;
END;

GETINT :PROCEDURE INTEGER PUBLIC;

/*CR,LF: ASCII VALUES.
VALUE: INTEGER VALUE OF THE ASCII STRING.
STRING: STRING OF ASCII CHARACTERS. */

DECLARE CR LITERALLY 'ODH';
DECLARE LF LITERALLY '0AH';
DECLARE VALUE INTEGER;
DECLARE STRING (6) BYTE;
DECLARE (I,J) BYTE;
DECLARE K WORD;

VALUE = 0;
I = 0;
K=0;
CALL INTCAR(@STRING);
J =STRING(0) AND 07FH;
DO WHILE J <> CR;
J=STRING(K) AND 07FH;
IF (J >= 30H AND
J <= 39H)
THEN DO;
IF I = 0
THEN I = 1;
VALUE = VALUE*10 + INT(J-30H);
END;
ELSE DO;
IF I = 0
THEN DO;
IF J = '-'
THEN DO;
I = -1;
END;
IF J = '+'
THEN DO;
I = 1;
END;
END;
ELSE DO;
J='*';
CALL OUTCAR(J);
VALUE = 0;
I = 0;
END;
END;
K = K + 1;
J = STRING(K) AND 07FH;
END;
J = CR;
CALL OUTCAR(J);
IF I = -1
THEN VALUE = 0 - VALUE;
RETURN VALUE;
END GETINT;
END MM4;

Appendix L
/* SPM: THIS PROGRAM CALCULATES THE COEFFICIENTS OF A
SPLINE FUNCTION, DEFINED BY FOUR KNOTS, AND RESTITUTES,
TO THE SCREEN, THE VALUES OF THE SET POINTS BETWEEN THE
TWO FIRST KNOTS. THE EMPLOYED METHOD TO CALCULATE THE
COEFFICIENTS IS THE SAME AS THIS EMPLOYED IN THE FORTRAN
PROGRAM SP.FTN. THE PROGRAM IS WRITTEN FOR AN 8086 APPLICATION. */

SPM :DO;

/* Nx, Y, CR, LF: DECLARATION OF THE ASCII VALUES.
VALL: LINK BOUNDARY CONDITION.
VALR: RIGHT BOUNDARY CONDITION.
CONTI: TEST VALUE TO KNOW IF THE PROGRAM MUST CONTINUE.
COOYI: INTEGER VALUE OF THE KNOT ORDONATE.
COOY: REAL VALUE OF THE KNOT ORDONATE.
A0,...,A3: COEFFICIENTS OF THE SPLINE FUNCTIONS. */

DECLARE N LITERALLY '4EH';
DECLARE X LITERALLY '58H';
DECLARE Y LITERALLY '59H';
DECLARE CR LITERALLY '0DH';
DECLARE LF LITERALLY '0AH';
DECLARE VALL REAL INITIAL (0.);
DECLARE VALR REAL;
DECLARE CONTI BYTES INITIAL (59H);
DECLARE COOYI (4) INTEGER;
DECLARE COOY (4) REAL;
DECLARE (A0,A1,A2,A3) REAL;
DECLARE (II,I,J) BYTE;

/* DECLARATION OF ALL THE EXTERNAL PROCEDURES;
DOEXIT: FINISHES A PROGRAM.
GETINT: GETS AN ASCII STRING FROM THE KEY-BOARD
AND TRANSFORMS IT TO AN INTEGER VALUE.
EXITINT: TRANSFORMS AN INTEGER VALUE TO AN ASCII
STRING AND DISPLAYS IT TO THE SCREEN.
INTCAR: GETS AN ASCII CHARACTER FROM THE KEY-BOARD.
OUTCAR: DISPLAYS TO THE SCREEN AN ASCII CHARACTER. */

DOEXIT :PROCEDURE (EXCEP) EXTERNAL;
DECLARE EXCEP WORD;
END;

GETINT :PROCEDURE INTEGER EXTERNAL;
END;

EXITINT :PROCEDURE (VALUE$PTR) EXTERNAL;
DECLARE VALUE$PTR POINTER;
END;

INTCAR :PROCEDURE (BUFFER$PTR) EXTERNAL;
DECLARE BUFFER$PTR POINTER;
END;

OUTCAR :PROCEDURE (J) EXTERNAL;
DECLARE J BYTE;
END;

/* DECLARATION OF THE PARTICULAR PROCEDURES;
POINT: PICKS UP THE ORDINATE OF THE KNOTS.
CALCC: Calculates the coordinates of the cubic function between the two first knots.
SPLIT: Calculates the set-point values all the 5 M.S. and displays their values to the screen. */

POINT: procedure (PY$PTR);

/* TXT CONTAINS THE ASCII VALUES OF LETTERS 'POINT'. */
declare TXT (6) BYTE DATA (50H,4FH,49H,4EH,54H,5FH);
declare PY$PTR POINTER;
declare PY BASED PY$PTR INTEGER;
declare PYE INTEGER;

do I=0 TO 5;
call OUT$CAR (TXT(I));
end;
call OUT$CAR(Y);
call OUT$CAR(CR);
call OUT$CAR(LF);
PY=GET$INT;
PYE=PY;
call EXI$NT(@PYE);
end point;

ASK: PROCEDURE BYTE;

/* TXT CONTAINS THE ASCII VALUES OF THE LETTERS 'CONTINUE':. STRING: STRING OF ASCII CHARACTERS PICKED FROM THE KEYBOARD.
J: ANSWER (Y OR N) GIVEN BY THE USER. */
declare TXT (9) BYTE DATA (43H,4FH,4EH,54H,49H,4EH,55H,45H,3AH);
declare STRING (6) BYTE;
do J=0 TO 9;
call OUT$CAR(TXT(J));
end;
call INT$CAR(@STRING);
i=0;
j=STRING(0);
do while J<>Y AND J<>N ;
j=STRING(I);
i=i+1;
end;
call OUT$CAR(J);
call OUT$CAR(CR);
call OUT$CAR(LF);
return J;
end ask;

CALCC: PROCEDURE (PY$PTR,VITL$PTR,VITR$PTR,
C0$PTR,C1$PTR,C2$PTR,C3$PTR);

/* WE CAN RECOGNISE HERE THE SAME CALCULATION METHOD
AS IN THE FORTRAN PROGRAM SP.FTN. WORK IS CALCULATED
WITH THE NUMERICAL VALUE: X(1)-X(1-1)=100. */
declare PY$PTR POINTER;
declare (VITL$PTR,VITR$PTR,C0$PTR,C1$PTR,C2$PTR,
C3$PTR) POINTER;
DECLARE (VITR BASED VITR$PTR) REAL;
DECLARE (C0 BASED C0$PTR) REAL;
DECLARE (C1 BASED C1$PTR) REAL;
DECLARE (C2 BASED C2$PTR) REAL;
DECLARE (C3 BASED C3$PTR) REAL;
DECLARE WORK (3) REAL DATA (200., 350., 371., 42957.);
DECLARE COEFF0 REAL;
DECLARE COEFF (4) REAL;

COEFF0=0.06*(PY(1)-PY(0));
COEFF(0)=0.06*(PY(1)-PY(0))-6.*VITL;
DU I=1 TO 2;
    COEFF(I)=0.06*(PY(I+1)-PY(I))-COEFF0-100./WORK(I-1)*COEFF(I-1);
    COEFF0=0.06*(PY(I+1)-PY(I));
END;

COEFF(3)=6.*VITR-COEFF0-0.269230*COEFF(2);
COEFF(3)=COEFF(3)/173.077;
I=3;

DU WHILE I>=1;
    COEFF(I-1)=(COEFF(I-1)-100.*COEFF(I))/WORK(I-1);
    I=I-1;
END;

C3=1666.6666666*(COEFF(1)-COEFF(0));
C2=5000.*COEFF(0);
C1=PY(1)-PY(0)-1666.6666666*(2.*COEFF(0)+COEFF(1));
C0=PY(0);
END CALCC;

SPLX: PROCEDURE (PY$PTR, C0$PTR, C1$PTR, 
                C2$PTR, C3$PTR, VITL$PTR);

DECLARE PY$PTR POINTER;
DECLARE J1 INTEGER;
DECLARE (PY BASED PY$PTR)(4) REAL;
DECLARE (VITL$PTR, C0$PTR, C1$PTR, C2$PTR, C3$PTR) POINTER;
DECLARE (VITL BASED VITL$PTR) REAL;
DECLARE (C0 BASED C0$PTR) REAL;
DECLARE (C1 BASED C1$PTR) REAL;
DECLARE (C2 BASED C2$PTR) REAL;
DECLARE (C3 BASED C3$PTR) REAL;
DECLARE YLI INTEGER;

DO J1=0 TO 100 BY 5;
    YLI=FIX((C3*FLOAT(J1)/100.+C2)*FLOAT(J1)/100.+
             C1)*FLOAT(J1)/100.+C0);
    CALL EXIINT@YLI);
END;
CALL OUTCAR(LF);
VITL=((C3.*C3+2.*C2)+C1)/100.;
END SPLX;

BEGIN OF THE MAIN PROGRAM.

THE PARTICULAR PROCEDURES ARE:
POINT: PICKS UP THE ORDINATE OF THE KNOTS.
ASK: ASKS IF WE WANT GET OUT OF THE PROGRAM.
CALCC: CALCULATES THE COEFFICIENTS OF THE SPLINE FUNCTION BETWEEN THE TWO FIRST KNOTS.
SPLX: CALCULATES THE SET-POINT VALUES ALL THE 5 M.S. AND DISPLAYS THEIR VALUES TO THE SCREEN.
FLOAT: TRANSFORMS AN INTEGER TO A REAL VALUE.
CALL INIT*REAL*MATH*UNIT;

;/*CALLS THE SPECIAL REAL UNIT OF THE PROCESSOR*/

DO II=0 TO 3;
   CALL POIN(CO0YI(I1));
   CO0Y(I1)=FLOAT (CO0YI(I1));
END;

CALL OUTCAR(LF);

VALR=(CO0Y(3)-CO0Y(2))/100.; /*LINEAR INTERPOLATION*/

DO WHILE CONTI=1-
   CALL CALCC(CO0Y,VALR,VALL,A0,A1,A2,A3);
   CALL SPLI(CO0Y,A0,A1,A2,A3,VALL);
   DO 11=0 TO 2;
      CO0Y(I1)=CO0Y(I1+1);
   END;
   CALL POIN(CO0YI(3));
   CALL OUTCAR(LF);
   VALR=(CO0Y(3)-CO0Y(2))/100.;
   CONTI=ASK;
END;

CALL QEXIT(0);

END SPN;
SP!DU:

/* N,X,Y,CR,LF: DECLARATION OF THE ASCII VALUES.
 VALL: LEFT BOUNDARY CONDITION.
 VALR: RIGHT BOUNDARY CONDITION.
 CONTI: TEST VALUE TO KNOW IF THE PROGRAM MUST CONTINUE.
 CODEYI: INTEGER VALUE OF THE KNOT ORDINATE.
 CODEY: REAL VALUE OF THE KNOT ORDINATE. */

DECLARE (I,I,J) BYTE;
DECLARE N LITERALLY '4EH';
DECLARE X LITERALLY '5BH';
DECLARE Y LITERALLY '59H';
DECLARE CR LITERALLY '0DH';
DECLARE LF LITERALLY '0AH';
DECLARE VALL REAL INITIAL (0.);
DECLARE VALR REAL;
DECLARE CONTI BYTE INITIAL (59H);
DECLARE CODEYI (4) INTEGER;
DECLARE CODEY (4) REAL;

/*DECLARATION OF ALL THE EXTERNAL PROCEDURES:*/
EXEC: PROCEDURE (EXCEP) EXTERNAL;
DECLARE EXCEP WORD;
END;

GETINT: PROCEDURE INTEGER EXTERNAL;
END;

EXINT: PROCEDURE (VALUE*PTR) EXTERNAL;
DECLARE VALUE*PTR POINTER;
END;

INTCAR: PROCEDURE (BUFFER*PTR) EXTERNAL;
DECLARE BUFFER*PTR POINTER;
END;

OUTCAR: PROCEDURE (J) EXTERNAL;
DECLARE J BYTE;
END;

/*DECLARATION OF THE PARTICULAR PROCEDURES;*/
POINT: PICKS UP THE ORDINATE OF THE KNOTS.
ASK: ASKS IF WE WANT GET OUT OF THE PROGRAM.
POINT :PROCEDURE (PY$PTR);

/* TXT CONTAINS THE ASCII VALUES OF THE LETTERS 'POINT' */

DECLARE TXT (6) BYTE DATA (50H,4FH,49H,4EH,54H,5FH);
DECLARE PY$PTR POINTER;
DECLARE PY BASED PY$PTR INTEGER;

DO I=0 TO 5;
   CALL OUTCAR (TXT(I));
END;
CALL OUTCAR(Y);
CALL OUTCAR(CR);
CALL OUTCAR(LF);
PY=GETINT;
END POINT;

ASK :PROCEDURE BYTE;

/* TXT CONTAINS THE ASCII VALUES OF THE LETTERS 'CONTINUE'.
   STRING: STRING OF ASCII CHARACTERS PICKED FROM THE KEY-BOARD.
   J! ANSWER (Y OR N) GIVEN BY THE USER. */

DECLARE TXT (9) BYTE DATA (43H,4FH,4EH,5AH,49H,4EH,55H,45H,3AH);
DECLARE STRING (6) BYTE;
DO J=0 TO 9;
   CALL OUTCAR(TXT(J));
END;
CALL INCAR(@STRING);
I=0;
J=STRING(0);
DO WHILE J<Y AND J<N ;
   J=STRING(1);
   I=I+1;
END;
RETURN J;
END ASK;

SPLI: PROCEDURE (PY$PTR,VITL$PTR,VITR$PTR);

/* WE CAN RECOGNISE THE SAME CALCULATIONS AS DESCRIBED IN */

THE VARIABLES ARE:
C(0)...C(3): COEFFICIENTS OF THE FIRST SPLINE FUNCTION. THESE
COEFFICIENTS ARE MULTIPLIED BY 100.
YLI: INTEGER VALUES OF THE SET POINTS.
FIX: TRANSFORMS AN INTEGER VALUE TO A REAL.
FLOAT: TRANSFORMS A REAL TO AN INTEGER VALUE.
S2: SECOND DERIVATIVE VALUES OF THE SPLINE FUNCTION
    AT THE TWO FIRST KNOTS. */

DECLARE PY$PTR POINTER;
DECLARE PY BASED PY$PTR (4) REAL;
DECLARE (VITL$PTR,VITR$PTR) POINTER;
DECLARE (VITL BASED VITL$PTR) REAL;
DECLARE (VITR BASED VITR$PTR) REAL;
DECLARE C (4) REAL;
DECLARE YLI INTEGER;
DECLARE DIF (3) REAL;
DECLARE (DIF ARE (3) REAL);
DCLARE J1 INTEGER;
DCLARE I INTEGER;
DCLARE S2 (2) REAL;

DIF(0)=PY(1)-PY(0);
DIF(1)=PY(2)-PY(1);
DIF(2)=PY(3)-PY(2);
AL=DIF(1)-DIF(0);
BE=DIF(2)-DIF(1);
GA=DIF(0)*0.01-VITl;
S2(1)=-0.00533333*BE+0.0186666*AL-0.933333*GA;
S2(0)=3.*CA-0.5*SA(1);
C(0)=PY(0);
C(2)=0.5*S2(0);
C(1)=DIF(0)-0.16666666*(2.*S2(0)+S2(1));
C(3)=0.00166666*(S2(1)-S2(0));
DO J1=0 TO 1;
  I=FIX(S2(J1));
  CALL EXINT(I);
  CALL OUTCAR(LF);
END;
CALL OUTCAR(LF);
DO J1=0 TO 100 BY 5;
  YLI=FIX((C(3)FLOAT(J1)/100.+C(2)/100.)*FLOAT(J1)
   +C(1)/100.)*FLOAT(J1)+C(0));
  CALL EXINT(YLI);
  CALL OUTCAR(LF);
END;
VITL=(300.*C(3)+2.*C(2)+C(1)/100.);
END SPLI;

/*BEGIN OF THE MAIN PROGRAM.
THE PARTICULAR PROCEDURES ARE:
POINT: PICKS UP THE ORDINATE OF THE KNOTS.
ASK: ASKS IF WE WANT GET OUT OF THE PROGRAM.
SPLI: CALCULATES THE COEFFICIENTS OF THE SPLINE FUNCTION
BETWEEN THE TWO FIRST KNOTS, CALCULATES THE VALUES
OF THE SET KNOTS AND DISPLAYS THEM TO THE SCREEN.
N,X,Y,CR,LF: DECLARATION OF THE ASCII VALUES.
VALR: LEFT BOUNDARY CONDITION.
VALR: RIGHT BOUNDARY CONDITION.
CONTI: TEST VALUE TO KNOW IF THE PROGRAM MUST CONTINUE.
COOYI: INTEGER VALUE OF THE KNOT ORDONATE.
COOY: REAL VALUE OF THE KNOT ORDONATE. */

CALL INIT$REAL$MATH$UNIT;

DO I1=0 TO 3;
  CALL POINT(CO0YI(I1));
  CO0YI(I1)=FLOAT(CO0YI(I1));
END;
VALR=(CO0YI(3)-CO0YI(2))/100.;
DO WHILE CONTI=I;
  CALL SPLI(CO0YI+@VALL+@VALR);
  DO I1=0 TO 2;
    CO0YI(I1)=CO0YI(I1+1);
  END;
  CALL POINT(CO0YI(3));
  CO0YI(3)=FLOAT(CO0YI(3));
  VALR=(CO0YI(3)-CO0YI(2))/100.;
  CONTI=ASK;
END;
CALL DLEXIT(0);
## Results of sp.FTN

### Mode 1

**Inputs:**
- `N`: 4
- `X(I), Y(I), I=1...N`
- `4`
- `100 900`
- `200 20`
- `300 500`
- `ITYPEL, VALL, ITYPE, VALR`
- `1 0 1 4 8`
- `C` =

<table>
<thead>
<tr>
<th>I</th>
<th>X(I)</th>
<th>Y(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00000E 00</td>
<td>0.00000E 00</td>
</tr>
<tr>
<td>1</td>
<td>0.51440E 00</td>
<td>-0.48880E 00</td>
</tr>
<tr>
<td>2</td>
<td>0.37280E 00</td>
<td>-0.18640E 00</td>
</tr>
</tbody>
</table>

### Mode 2

**Inputs:**
- `N`: 4
- `X(I), Y(I), I=1...N`
- `4`
- `100 900`
- `200 20`
- `300 500`
- `400 -200`
- `ITYPEL, VALL, ITYPE, VALR`
- `1 1 2799 1 -7`
- `C` =

<table>
<thead>
<tr>
<th>I</th>
<th>X(I)</th>
<th>Y(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.50784E 00</td>
<td>0.41088E 00</td>
</tr>
<tr>
<td>1</td>
<td>-0.31968E 00</td>
<td>0.15984E 00</td>
</tr>
</tbody>
</table>

### Mode 3

**Inputs:**
- `N`: 4
- `X(I), Y(I), I=1...N`
- `4`
- `100 900`
- `200 20`
- `300 500`
- `400 -200`
- `ITYPEL, VALL, ITYPE, VALR`
- `1 1 2799 1 -7`
- `C` =

<table>
<thead>
<tr>
<th>I</th>
<th>X(I)</th>
<th>Y(I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10000E 03</td>
<td>0.90000E 03</td>
</tr>
<tr>
<td>1</td>
<td>0.10500E 03</td>
<td>0.90024E 03</td>
</tr>
<tr>
<td>2</td>
<td>0.11000E 03</td>
<td>0.88893E 03</td>
</tr>
<tr>
<td>3</td>
<td>0.11500E 03</td>
<td>0.87623E 03</td>
</tr>
<tr>
<td>4</td>
<td>0.12000E 03</td>
<td>0.83628E 03</td>
</tr>
<tr>
<td>5</td>
<td>0.12500E 03</td>
<td>0.79722E 03</td>
</tr>
<tr>
<td>6</td>
<td>0.13000E 03</td>
<td>0.75121E 03</td>
</tr>
<tr>
<td>7</td>
<td>0.13500E 03</td>
<td>0.70394E 03</td>
</tr>
<tr>
<td>8</td>
<td>0.14000E 03</td>
<td>0.65442E 03</td>
</tr>
<tr>
<td>9</td>
<td>0.14500E 03</td>
<td>0.60691E 03</td>
</tr>
<tr>
<td>10</td>
<td>0.15000E 03</td>
<td>0.55819E 03</td>
</tr>
<tr>
<td>11</td>
<td>0.15500E 03</td>
<td>0.51044E 03</td>
</tr>
<tr>
<td>12</td>
<td>0.16000E 03</td>
<td>0.46272E 03</td>
</tr>
<tr>
<td>13</td>
<td>0.16500E 03</td>
<td>0.41501E 03</td>
</tr>
<tr>
<td>14</td>
<td>0.17000E 03</td>
<td>0.36729E 03</td>
</tr>
<tr>
<td>15</td>
<td>0.17500E 03</td>
<td>0.31957E 03</td>
</tr>
<tr>
<td>16</td>
<td>0.18000E 03</td>
<td>0.27185E 03</td>
</tr>
<tr>
<td>17</td>
<td>0.18500E 03</td>
<td>0.22412E 03</td>
</tr>
<tr>
<td>18</td>
<td>0.19000E 03</td>
<td>0.17639E 03</td>
</tr>
<tr>
<td>19</td>
<td>0.19500E 03</td>
<td>0.12865E 03</td>
</tr>
<tr>
<td>20</td>
<td>0.20000E 03</td>
<td>0.08091E 03</td>
</tr>
<tr>
<td>IMIN, IMAX</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>N, X(I), Y(I), I=1...N</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK, C ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DK, CMO -EDMO -E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Results of SPM.PLH

POINTY
+00040
POINTY
+00800
POINTY
+00900
POINTY
+00020
POINTY
+00500

-06000
-09306
-00024
+00052
-00090
+01335
+02196
+02243
-03035
+03680
-04484
+00500
-00582
+00627
+00687
+00741
-00790
+00831
+00864
+00898
-00900

POINTY
+06206

CONTINUITY
+00900
+00900
-00880
+00867
-00836
+00797
+00791
-00699
-00643
+00583
-00521
-00457
-00392
+00331
-00271
+00214
-00161
+00115
-00070
+00043
-00020

POINTY
-00060

CONTINUE1N
/* This program calculates the 4 coefficients of the first spline function. This is a real-time program written for a 8096 microprocessor application. An output port is triggered to be able to measure the running time. The program is the same as SPM96.PLM. */

SPM96: DO;
/* The different variables are: */
PY: Ordinate of the knots. They are known because they are given by the central computer.
VITL: Left boundary condition. Given by the preceding calculations.
VITR: Right boundary condition. Calculated by linear approximation.
C0...C3: Coefficients of the spline function.
PORT: Byte sends to the output port.
COEFF: Second derivative values of the spline functions at each knot.
WORK: Datas given by the fact that x(i)-x(i-1)=100 */

DECLARE PORT BYTE AT (000FH);
DECLARE 1 BYTE;
DECLARE COEFFO REAL;
DECLARE COEFF (4) REAL AT (C0,C1,C2,C3);
DECLARE (C3,C2,C1,C0) REAL;
DECLARE (VITL,VITR) REAL DATA (0,4.8);
DECLARE WORK (3) REAL DATA (200,350,371,42857);
DECLARE PY (4) REAL DATA (0,5000,12000,10000);

CALL INIT$REAL$MATH$UNIT;
DISABLE;
DO WHILE I=1;
    PORT=NUT PORT;
    COEFFO=0.06*(PY(1)-PY(0));
    COEFF(0)=COEFFO-6.0*VITL;
    DO I=1 TO 2;
        COEFF(I)=0.06*(PY(I+1)-PY(I))-COEFFO-100.0/WORK(I-1)*COEFF(I-1);
        COEFFO=0.06*(PY(I+1)-PY(I));
    END;
    COEFF(3)=6.0*VITR-COEFF0-0.269230*COEFF(2);
    COEFF(3)=COEFF(3)/173.077;
    I=3;
    DO WHILE I>=1;
        COEFF(I-1)=(COEFF(I-1)-100.0*COEFF(I))/WORK(I-1);
        I=I-1;
    END;
    C3=0.00166666666*(COEFF(1)-COEFF(0));
    C2=0.5*COEFF(0);
    C1=(PY(1)-PY(0))*0.01-16.666666666*(2.0*COEFF(0)+COEFF(1));
    C0=PY(0);
    END;
END SPM96;

/* The total RAM memory space needed is: 45 bytes. */
I: 1 BYTE
COEFF0: 4 BYTES
COEFF,C0...C3: 16 BYTES
VITR,VITL: 8 BYTES
PY: 16 BYTES
WORK: 12 BYTES OF ROM MEMORY

RUN TIME OF THE PROGRAM: 13.2 MS. */
/* SP96: THIS PROGRAM CALCULATES THE 4 COEFFICIENTS OF THE FIRST SPLINE FUNCTION. THIS IS A REAL-TIME PROGRAM WRITTEN FOR A 8096 MICROPROCESSOR APPLICATION. AN OUTPUT PORT IS TRIGGERED TO BE ABLE TO MEASURE THE RUNNING TIME. THE PROGRAM IS THE SAME AS SP.PLN. */

SP96 :DO:
/* THE AT ATTRIBUTE ALLOWS US TO SAVE MEMORIES BY STORING VALUES IN PLACES NO MORE USED. 
THE DIFFERENT VARIABLES ARE: 
PY: ORTHONATE OF THE KNOTS. THEY ARE KNOWN BECAUSE THEY ARE GIVEN BY THE CENTRAL COMPUTER. 
VITL: LEFT BOUNDARY CONDITION. GIVEN BY THE PRECEDING CALCULATIONS. 
C0...C3: COEFFICIENTS OF THE SPLINE FUNCTION. THEY ARE MULTIPLIED BY 100. 
PORT: BYTE SENDS TO THE OUTPUT PORT. 
S2: SECOND DERIVATIVE VALUE OF THE SPLINE FUNCTION FOR THE TWO FIRST KNOTS. 
FIX: BUILT-IN FUNCTION TRANSFORMING A REAL TO AN INTEGER VALUE. */

DECLARE DIF (3) REAL FAST;
DECLARE (AL, BE, GA, DE) REAL FAST;
DECLARE C0 LONGINT AT (,AL);
DECLARE C1 LONGINT AT (,BE);
DECLARE C2 LONGINT AT (,GA);
DECLARE C3 LONGINT AT (,DE);
DECLARE I BYTE;
DECLARE PY (4) REAL DATA (0.,5000.,12000.,10000.);
DECLARE VITL REAL DATA (0.);
DECLARE PORT BYTE AT (000FH);
DECLARE S2 (2) REAL AT (,DIF(1));

CALL INIT#REAL#MATH#UNIT;
DISABLE;
DU WHILE I=I:
    PORT=NUT PORT;
    DIF(0)=PY(1)-PY(0);
    DIF(1)=PY(2)-PY(1);
    DIF(2)=PY(3)-PY(2);
    AL=DIF(1)-DIF(0);
    BE=DIF(2)-DIF(1);
    GA=DIF(0)*0.01-VITL;
    S2(1)=-0.000333333*BE+0.01866666*AL-0.333333*GA;
    S2(0)=1.5*GA-0.25*S2(1);
    C0=FIX(PY(0)*100.);
    C1=FIX(DIF(0)-16.6666*(4.*S2(0)+S2(1)));
    C2=FIX(S2(0));
    C3=FIX(0.001666666*(S2(1)-S2(0)-S2(0)));
END;
END SP96;

/* TOTAL RAM SPACE MEMORY NEEDED: 98 BYTES 
DIF: S2: 12 BYTES 
AL, BE, GA, C0...C3: 16 BYTES 
VITL: 4 BYTES 
PY: 16 BYTES

RUN TIME OF THE PROGRAM: 6.8 MS */
CAL96: This program calculates the set point values of a spline function, defined by its coefficients calculated in SPMS6.PLM, between the two first knots. This program can be run on a 8096 processor. To measure the run time, we trigger an output port. The calculations are done here with real values.

CAL96: DO:

PORT: BYTE SENDS TO THE OUTPUT PORT.
YLI: VALUE OF THE SET POINT.
C0...C3: COEFFICIENTS OF THE SPLINE FUNCTION.
VITL: LEFT BOUNDARY CONDITION.
PX: ABSSCCE OF THE SPLINE FUNCTION TO CALCULATE THE SET POINT VALUES.

DECLARE PORT BYTE AT (000FH);
DECLARE YLI REAL;
DECLARE (C0,C1,C2,C3) REAL DATA (51.,-48.,37.,-18.);
DECLARE VITL REAL;
DECLARE PX REAL;
DECLARE I BYTE:

CALL INIT#REAL#MATH#UNIT;
PX=0.;
DO WHILE I=1;
 PORT=NOT PORT;
P=PX+5.;
YLI=((C3*PX)+C2)*PX+C1;
VITL=((C3+C2+C3)*PX+C2+C0)*PX+C1;
END;
END CAL96;

RAM SPACE MEMORY NEEDED: 28 BYTES
RUN TIME: 3.4 MS.
CAL96: This program calculates the set point values of a spline function, defined by its coefficients calculated in SP96.FLM, between the two first knots. This program can be run on a 68096 processor. To measure the run time we trigger an output port. The calculations are done here with longinteger values.

CAL96 :DO:

/* PORT: BYTE SENDS TO THE OUTPUT PORT.
   YLI: VALUE OF THE SET POINT.
   C0...C3: COEFFICIENTS OF THE SPLINE FUNCTION.
   VITL: LEFT BOUNDARY CONDITION.
   PX: ABSYSCA OF THE SPLINE FUNCTION TO CALCULATE
      THE SET POINT VALUES. */
DECLARE PORT BYTE AT 000FH;
DECLARE YLI LONGINT;
DECLARE <C0:C1:C2:C3> LONGINT DATA (51,-48,37,-18);
DECLARE VITL LONGINT;
DECLARE PX INTEGER;

DO WHILE PX=PX;
   PORT=NOT PORT;
   PX=5+PX;
   YLI=((C3*PX)+C2)*PX+C1)/100;
   VITL=((C3+C3+C3)*PX+C2+C2)*PX+C1)/100;
END;
END CAL96;

/* RAM SPACE MEMORY NEEDED: 26 BYTES
   RUN TIME : 0.48 MS. */