Theory and applications of spline functions for a servo-mechanism

Citation for published version (APA):

Document status and date:
Published: 01/01/1985

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
- Theory and applications of spline functions for a servo-mechanism.

Hugues de ROSEMONT: Université de Technologie de Compiègne.

February-July 1985

N. WPB-0200
## Summary

- 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
I   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 compiles 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 hardware.
IV) Theorie of spline functions

a) Mathematical theory


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 $1, 2, \ldots m-1$ are continuous.

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

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

with $q_m \in \mathfrak{w}_m$ and $\mathfrak{w}_m$ denotes the class of polynomials of degree $m$ or less.

proof

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

$$q_m(n) - q_{m-1}(n) = C_j(n-n_j)^m \quad (1)$$

thus

$$q_m(n) = q_{m0}(n) + (q_m(n) - q_{m0}(n)) + (q_m(n) - q_{m1}(n)) + (q_m(n) - q_{m2}(n)) + \ldots + (q_m(n) - q_{m1-1}(n))$$

$$q_m(n) = q_{m0}(n) + \sum_{j=1}^{n} C_j(n-n_j)^m$$

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

for $n < n_1$: $S(n) = q_{m0}(n)$ therefore $q_m(n) = q_{m0}(n)$

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

$$q_m^{(m)}(n) - q_m^{(m)}(n) = C_j x^m$$

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

The natural spline interpolation
A spline function of odd degree $2k-1$ with knots $n_1, n_2, \ldots, n_n$ is called a natural spline function if the two polynomials by which it is represented in the two intervals $(-\infty, n_1)$ and $(n_n, +\infty)$ are of degree $k-1$, or less.

Let $(n_i, j_i) \in 1 \ldots n$ 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:

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

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 $n_1, n_2, \ldots, n_n$ as knots.

So a smoothest spline function can be written like:

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

for $n < n_1$, $S(n)$ is automatically a function of degree $k-1$.

For $n > n_n$, $S(n) = \sum_{j=1}^{n} C_j (n-n_j)^{2k-1}$ reduces to a polynomial of degree $k-1$ if and only if the coefficient of every power of $n$ higher than $k-1$ vanishes.

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

Theorem:

Let $(n_i, J_i)$, $i = 1, \ldots, n$ 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 $n_i$ such that:

$$S(n_i) = J_i \quad i = 1, 2, \ldots, n$$
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 \( 2k-1 \) \((k\in\mathbb{N})\). 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:

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

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 \( S^*(n_{i+1}) \).

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

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

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

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:

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

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

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:

\[
(9) \frac{S(n_{i+1})-S(n_i)}{n_{i+1} - n_i} + \frac{1}{6} (n_{i+1}-n_i)[2S^*(n_i)+S^*(n_{i-1})]
\]

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

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

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

From this equation (7) we can pull out the coefficients of the spline function. If \( S_{i+1}(n) = C_0 + C_1 n + C_2 n^2 + C_3 n^3 \) 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 functions. 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:

\[
\frac{S(n_3) - S(n_2)}{n_3 - n_2} - \frac{n_3 - n_2}{6} (2 \times S''(n_2) + S''(n_3))
\]

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

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

Let call \( \alpha = \frac{6}{(\Delta n)^2} [(S(n_3) - S(n_2)) - (S(n_2) - S(n_1))] \)

\( \beta = \frac{6}{(\Delta n)^2} [(S(n_4) - S(n_3)) - (S(n_3) - S(n_2))] \)

\( \delta = \frac{6}{\Delta n} \left[ \frac{S(n_2) - S(n_1)}{\Delta n} - S'(n_1) \right] \)

\( \delta = \frac{6}{\Delta n} \left[ S'(n_4) - \frac{S(n_4) - S(n_3)}{\Delta n} \right] \)

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:

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

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):

\[
\begin{align*}
C_0 &= S(n_1) \\
C_1 &= \frac{[S(n_2) - S(n_1)]}{\Delta n} - \frac{\Delta n}{6} \left[ 2S'(n_1) + S'(n_2) \right]
\end{align*}
\]
\[ c_2 = \frac{S^*(n_1)}{2} \]

\[ c_3 = \frac{1}{6x\Delta n} (S^*(n_2) - S^*(n_1)) \]
V 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 expected with the aid of a simulating program, 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 spaces
. of 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 ≥ 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 \( (n_i, n_{i+1}, ..., n_{i+n-1}) \). 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.

![Diagram of welding path and sensor](image)

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:

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 $\varepsilon(n)$ ($\varepsilon=\text{error}$) 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 data: $R$(radius), $\Omega M$ (angular speed), $N$ (number of knots on which the interpolation must be done).

- SP.C.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:
- SPC.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 SPC.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)

\[ SPWF \]

\[ SPDF1 \]

\[ SPCF \]

\[ SPSF \]

\[ SPCFB \]

---

**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 R1 and R2 is directly proportional to the ratio R2/R1 if the boundary conditions and the angle between each knot are the same. (See SPCF)

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)

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 servo-mechanism. The spline interpolator is only one part of the servo mechanism 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 users'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.
EXIINT : transforms 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.

Initialise the 4 first values

Initialise the boundary conditions

Calculations of the 4 coefficients

Calculations of the set points. Write them to the screen.

Next knot

Continue?

Y

Exit

N
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 m.s. 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

Output port.

running time
```

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.PLM</th>
<th>SP96.PLM</th>
<th>CALM96.PLM</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.PLM) 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.PLM, SPM96.PLM, CAL96.PLM and CALM96.PLM can be seen in the appendix 0.

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 pyramidal 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 de 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 apprécié 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 points values are calculated.

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

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

10 WRITE (1, 12)
12 FORMAT (1, 12)
   N, 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
14 WRITE (1, 14)
16 FORMAT ('ITYPER, VALL, ITYPER, VALR')
READ (1, *, ERR=14) ITYPEL, VALL, ITYPER, VALR
C
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))
C
20 WRITE (1, 22)
22 FORMAT ('IMIN, IMAX')
   IMIN, IMAX
READ (1, *, ERR=20) IMIN, IMAX
   IF (IMAX.EQ.0) GOTO 10
CALL BIAVL1 (C, X, Y, N, XREF, YREF, DYREF, IMAX)
WRITE (1, 24) (XREF(I), YREF(I), DYREF(I), I=IMIN, IMAX)
24 FORMAT (3E15.6)
   DO 30
30 WRITE (1, 31)
   IMIN, IMAX
31 FORMAT (3E15.6)
   END
C

This subroutine calculates the N second derivative values of the spline function at each knot.
SUBROUTINE B1AC (COEFF, XNOD, YNOD, NNOO, WORK, *
*    TYPEL, VALL, TYPER, VALR, CODE)

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

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

C For the appropriate equations refer to ...
C
C In this implementation, all equations are multiplied by 6 (six).
C Moreover:
C 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)
C 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.
C
FAIL=3
IF (NNOD.LT.3) GOTO 90
C
BEGIN CONDITION

C Equations for interval 1 (X(1) ... X(2)):
  dY1 = (Yr - Y1) / H - H/6 * (2 * C1 + Cr)
  dYr = (Yr - Y1) / H + H/6 * (C1 + 2 * Cr)

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

C TYPEL = 1 : first derivative prescribed
WORK(1)=2.*H
COEFF(1)=YH-6.*VALL
WORK1=H
IBACK=1
GOTO 20

C 10 IF (TYPEL.NE.2) GOTO 90
C
C TYPEL = 2 : second derivative prescribed
Use only the equation for dYr in interval 1 and substitute
VALL for C1
C
WORK(1)=1.
COEFF(1)=VALL
WORK1=0.
IBACK=2

C 20 N1=NNOO-1
C
Equations for interval i (X(i) ... X(i+1)), i = 2 ... N-1:

dYl = (Yr - Yl) / H - H/6 * (2 * Cl + Cr)
dYr = (Yr - Yl) / H + H/6 * (Cl + 2 * Cr)

Implement equation i with: dYr(i-1) - dYl(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

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

TYPER = 2: second derivative prescribed

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

BACKSUBSTITUTION

50 COEFF(NNOD)=COEFF(NNOD)/WORK(NNOD)
1=NNOD
60 1=1-1
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 II, J2
C SP.FTN:

REAL DX, S, CL, CR, YL, YR, F, A0, 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=COEFF(11)
CR=COEFF(12)
YL=YNOD(11)
YR=YNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=YR-YL-F*(2.*CL+CR)
A0=YL
S=(X-XNOD(11))/DX
Y=((A3*S+A2)*S+A1)*S+A0

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, A0, 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
C Interval is 11 - 12. Value may be at either boundary
C

DX=ZNOD(12)-ZNOD(11)
F=DX**2/6.
CL=COEFF(11)
CR=COEFF(12)
YL=ZNOD(11)
YR=ZNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=YR-YL-#*(2.*CL+CR)
A0=YL
S=(X-ZNOD(11))/DX
DY=((3.*A3*S+2.*A2)*S+A1)/DX
Y=((A3*S+A2)*S+A1)*S+A0
90 RETURN
END

C
C This subroutine calculates the ordinate of a set point,
C and the first and second derivative of the spline function.
C
SUBROUTINE B1AV2 (COEFF, ZNOD, YNOD, NNOD, X, Y, DY, DDY)
INTEGER NNOD
REAL COEFF(NNOD), ZNOD(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.ZNOD(1)) GOTO 90
Y=YNOD(NNOD)
IF (X.GT.ZNOD(NNOD)) GOTO 90
C
C Linear search is done here
C
I2=1
10 I1=I2
I2=I1+1
IF (ZNOD(I2).LT.X) GOTO 10
C
C Interval is 11 - 12. Value may be at either boundary
C
DX=ZNOD(12)-ZNOD(11)
F=DX**2/6.
CL=COEFF(11)
CR=COEFF(12)
YL=ZNOD(11)
YR=ZNOD(12)
A3=F*(CR-CL)
A2=3.*F*CL
A1=YR-YL-#*(2.*CL+CR)
A0=YL
S=(X-ZNOD(11))/DX
DDY=2.*((3.*A3*S+A2)/DX)**2
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 II, I2, IL
REAL DX, S, CO, C1, YO, Y1, F, A0, A1, A2, A3

IL = 1

Before first XNOD

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

Linear search to find appropriate interval

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

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

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

Beyond last XNOD

80 YL(II) = YNOD(NNOD)
   IL = IL + 1
   IF (IL. LE. NL) GOTO 80
This subroutine calculates the set point values between C 0 and I MAX 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 IL, J, IL
REAL DX, S, CO, C1, Y0, Y1, F, A0, A1, A2, A3

IL = 1

Before first XNOD

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

Linear search to find appropriate interval

20 J = 1
30 IL = I
   J2 = I
   IF (I2.GT. NNOD) GOTO 80
   IF (XNOD(I2).LT. XL(IL)) GOTO 30
   IF (XNOD(I2).LT. XL(IL)) GOTO 90
   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)
Y0 = YNOD(I1)
Y1 = YNOD(I2)
A3 = F*(C1 - CO)
A2 = 3.*F*CO
A1 = Y1 - Y0 - F*(2.*CO + C1)
A0 = Y0
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)
C SP.FTN:

IL=IL+1
IF (IL.LE.NL) GOTO 80
C All done
C
90 RETURN
END

This subroutine calculates the set point ordonates between 0 and IMAX, the first and second derivatives of the spline function.

SUBROUTINE BI AVL2 (COEFF, XNOD, YNOD, NNOD, XL, YL, DYL, DDYL, NL)
INTEGER NNOD(NL), NL
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), XL(NL), YL(NL), DYL(NL),
     DDYL(NL)

INTEGER I1, I2, IL
REAL DX, S, CO, C1, YO, Y1, F, A0, A1, A2, A3
C
IL=1
C Before first XNUD
C
10 IF (XL(IL).GE.XNOD(1)) GOTO 20
   DYL(IL)=0.
   DYL(IL)=0.
   YL(IL)=YNOD(1)
   IL=IL+1
   IF (IL.GT.NL) GOTO 90
   GOTO 10
C
C Linear search to find appropriate interval
C
20 I2=1
30 I1=I2
   I2=I1+1
   IF (I2.GT.NNOD) GOTO 80
   IF (XNUD(I2).LT.XL(IL)) GOTO 30
C
C Interval is I1 - I2. Value may be at either boundary
C
DX=XNUD(I2)-XNOD(I1)
F=DX*2/S.
CO=COEFF(I1)
C1=COEFF(I2)
YO=YNOD(1)
Y1=YNOD(I2)
A3=F*(C1-CO)
A2=3.*F*CO
A1=Y1-YO-*(2.*CO+C1)
A0=YO
40 S=-(XL(IL)-XNOD(I1))/DX
   DYL(IL)=2.*((3.*A3*S+A2)/DX**2
   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(1).LE.XNOD(12)) GOTO 40
GOTO 30

C Beyond last XNOD
C
80 DDYL(1L)=0.
   DYL(1L)=0.
   YL(1L)=YNUD(NND(1))
   IL=IL+1
   IF (IL.LE.NL) GOTO 80

C All done
C
90 RETURN
END
This program calculates a spline interpolated curve with the knots placed on a 1:4 circle. It restitutes the maximal error between two knots. The interpolation is done on the curve y=f(x) and z=g(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(k)(i), y(k)(i), z(k)(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 derivative values for the spline functions y=(function of time) and z=(function of time) for each knot
- dyref, dzref=first derivative value
- xn(i), yn(i), zn(i)=coordinates of the set point where the maximal error occurs
- dr(max) maximum error between two knots
- val y, val z=left boundary conditions
- val y, val z=right boundary conditions
- T: angular speed
- li=nrc1 before arriving on the circle
- R: radius
- T: angle which separates each knot
- pas: given xref(in time)
- the: boundaries conditions are the first derivative's. The angular speed is constant.

REAL *x (un, l, d), r, teta, tetao, anc, t, val y, val z,
*val y, val z, dyref, dzref, ld1, pas

REAL *x (x(50), y(50), z(50), xB(10), yB(10), zB(10), DR3(400),
*xref(400), yref(400), zref(400), cy(50), cz(50), dyref(400),
*dzref(400), DR(400), DR2(400), DR3(400), zM(50), yM(50), xM(50)

INTEGER 1, k1, k2, k, itype1, itype, n, i1, j, imax, i2

10 WR11: (j, j')
12 FORMAT ('n, un, ld, r, teta, pas')
read (1, *: expr=10) n, om, ld, r, teta, pas
1f (1 > teta, g1, 0, 3141592654) goto 10
1f = teta /un
ld1 = r* x1
1f (1, g1, l1) goto 10
1f (1, r, 0) goto 13
1t3 = teta /a1
goto 14
13 tetao = (teta1 - a1) / r

Calculation of the coordinates of the knots for the cases Teta < (p)/2LP or
\[ PP \) and Teta > (p)/2}.
Calculation of the different boundary conditions for the cases TETA<(PI/2) and TETA>(PI/2).

Calculation of the coefficients and the set points for N knots.

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

REAL *8 H, OLH, F, WORK1, COEFF1, YH
INTEGER I, NJ, IBACK, FAIL

C For the appropriate equations refer to ...
C In this implementation, all equations are multiplied by 6 (six).
C Moreover:
C 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)
C 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.
C
FAIL = 3
IF (NNNUUD, LT, 3) GOTO 90

C BEGIN CONDITION
C Equations for interval 1 (X(1) ... X(2)):
C dY1 = (Y1 - Y1) / H - H/6 * (2 * C1 + Cr)
C dY1r = (Y1r - Y1) / H + H/6 * (C1 + 2 * Cr)
C
FAIL = 2
IF (XNNUUD(2) - XNOD(1)) > 0.0 GOTO 70
YH = 6.0 * (YNUD(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 derivative values for the spline functions y = (function of time) and z = (function of time) for each knot
DYREF, DZREF = first derivative value
XM(i), YH(i), ZM(i) = coordinates of the set point where the maximal error occurs
DR = error between the circle and the interpolated curve
VALV, VALZ = right boundary conditions
VALV, 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 derivative's. The angular speed is constant.

Block Data

Common / 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(100), YARG(500), GEG(1)
INTEGER NP, IRES
LOGICAL UK

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

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

N=GE+1(1)
DM=GE+1(2)
LD=GE+1(3)
R=GE+1(4)
TETA=GE+1(5)
PAS=GE+1(6)
I=1:ETA=UN
TT=1:570796327/OM
IF (LD,NE.0) GOTO 13
ETA0=ETA
GOTO 14
13 ETA0=(K*ETA-LD)/R
14 Z(1)=0
Y(1)=0
X(1)=0
1=2
15 ANG=ETA0*1.1:ETA*(I-2)
IF (ANG,GT.1.570796327) GOTO 16
X(I)=(I-1)*T
Y(I)=R*(X(I))COS(ANG)
Z(I)=LD+R*(X(I))SIN(ANG)
I=I+1
GOTO 15
16 K1=1
K2=1+6
DO 18 K=K1,K2
X(K)=(K-1)*T
Y(K)=R*(1:ETA0-0.570796327+ETA*(K-2))
18 Z(K)=(K+L)
ITYPEL=-1
ITYPER=-1
I=0
KL=0
19 ANG=ETA0*1.1:ETA*(I+(N-2))
IF (ANG,GT.1.570796327) GOTO 20
VALYR=R*ON*SIN(ANG)
VALZR=R*ON*DCOS(ANG)
GOTO 22
20 VALYR=R*UN
VALZR=0
22 IF (I,NE.0) GOTO 24
VALYL=0
VALYL=R*UN
GOTO 25
24 VALYL=DYLF
VALYL=DZLF
25 DO 26 K=1,N
11-1*K
X(K)=X(11)
Y(K)=Y(11)
26 Z(K)=Z(11)
CALL BJAC (CY, XB, YB, N, DYPF, ITYPEL, VALYL, ITYPER, VALYR)
CALL BJAC (CZ, XB, ZB, N, DZPF, ITYPEL, VALZL, ITYPER, VALZR)
J=0
30 J=J+1
XREF(J)=MA(J+T*1)
IF (XREF(J),LE.XB(2)) GOTO 30
1MAX=J-1
CALL B1AVL.1 (CY, XB, YB, N, XREF, YREF, DYPF, IMAX, DYLF)
CALL B1AVL.1 (CZ, XB, ZB, N, XREF, ZREF, DZPF, IMAX, DZLF)
IF (XREF(IMAX),GT.TT) GOTO 40
K=0
12=1+1
32 IF (K,GE.1MAX) GOTO 34
K=K+1
KL=KL+1
DR(KL)=R-D*DSQRT((R-YREF(K))**2+(ZREF(K)-LD)**2)
XREF1(KL)=XREF(K)
GOTO 32
34 I=I+1
GOTO 19
40 DK=. TRUE.
NP=KL
DO 50 I=1,NP
YARG(I)=XREF1(I)*OM-TETA+TETA0
YARG(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(NNOD), YNOD(NNOD), NNOD, WORK(NNOD), VALL, VALR
REAL *8 H, OLDH, 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 built ("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.3) GOTO 90

BEGIN CONDITION
Equations for interval 1 (X(1) ... X(2)):

\[ \frac{dY_1}{dx} = \frac{Y_r - Y_1}{H} - H/6 \times (2 \times C1 + C2) \]
\[ dY_r = \frac{Y_r - Y_1}{H} + H/6 \times (C1 + 2 \times C2) \]

FAIL=2
IF (XNOD(2).EQ.XNOD(1)) 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 (TYPE=1 .NE. 2) GOTO 90

C TYPE = 2 : second derivative prescribed
C Use only the equation for dYr in interval 1 and substitute
C VAL. for Cl
C
WORK(1)=1.
COEFF(1)=VAL.
WORK1=0.
IBACK=2

N1=NNOD-1

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

\[ dY1 = \frac{(Yr - Y1)}{H - H/6 \times (2 + C1 + C2)} \]
\[ dYr = \frac{(Yr - Y1)}{H + H/6 \times (C1 + 2 + C2)} \]

Implement equation i with: dYr(i-1) - dY1(i) = 0

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

END CONDITION

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

C TYPE = 1 : first derivative prescribed
C
F=H/WORK(NNOD-1)
WORK(NNOD)=2.*H - F*H
COEFF(NNOD)=6.*VALR-COEFF1 - F*COEFF(NNOD-1)
GOTO 50

40 IF (TYPE .NE. 2) GOTO 90

C TYPE = 2 : second derivative prescribed
C
WORK(NNOD)=1.
COEFF(NNOD)=VALR

BACKSUBSTITUTION
C
50 COEFF(NNOD)=COEFF(NNOD)/WORK(NNOD)
1=NNOD
60 I=1-1
SPCF.IN:

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

90 COEFF:=FAIL
RETURN
END

SUBROUTINE H1AVL1 (COEFF, XNOD, YNOD, NNOD, XL, YL, DYL, NL, DLF)
INTEGER NNOD, NL
REAL COEFF(NNOD), XNOD(NNOD), YNOD(NNOD), XL(NL), YL(NL), DYL(NL)

INTEGER I1, I2, IL
REAL DX, S, C1, Y0, Y1, F, SF, A0, A1, A2, A3, DLF

IL=1
C Before first XNOD

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

C Linear search to find appropriate interval

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

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

DX=XNOD(2)-XNOD(1)
1=DX**2/SF.
C0=COEFF(1)
C1=C0*F*(2)
Y0=YNOD(1)
Y1=YNOD(2)
A3=F*(C1-C0)
A2=3.*F*C0
A1=Y1-Y0-1.*(2.*C0+C1)
A0=Y0
SF=1
DLF=((3.*A3*SF+2.*A2)*SF+A1)/DX
40 (I=(XL(IL)-XNOD(I1))/DX
DYL(IL)=((3.*A3*SF+2.*A2)*SF+A1)/DX
YL(IL)=((A3*SF+A2)*SF+A1)*SF+A0
IL=IL+1
IF (IL.GT.NL) GOTO 90
IF (XI(IL).LE.XNOD(I2)) GOTO 40
GOTO 30

C Beyond last XNOD
C      $PCF.F1N$

C
80  BYL(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 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 B1AC and B1AVL1 are the same as in SPC.FTN. The variables are:

- N: number of knots.
- DM: 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).

```
BLOCK DATA
COMMON /FNAMES/ N1(20), N2(20)
DATA N1 / 'SPCF', 'I', '18*' /,
   N2 / 'SPCF', 'H', '18*' /
END
SUBROUTINE FUNCT (TARG, YARG, NP, OK, GEG, IRES)
REAL TARG(500), YARG(500), GEG(1)
INTEGER NP, IRES
LD(1) CAL OK

REAL * R, DM, LD, R, TETA, TETA0, ANQ, T, VALVR, VALZR,
   VALYL, VALZL, DVLF, DZLF, PAS

REAL * B 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, ITYPER, N, I1, J, IMAX, I2, NP, KL

N= GEG(1)
DM= GEG(2)
LD= GEG(3)
R= GEG(4)
TETA= GEG(5)
PAS= GEG(6)
T=TETA/DM
TT= 1. 570796327/DM
IF (I.D. NE. 0) GOTO 13
TETA0=TETA
GOTO 14
13 TETA0=(R*TETA-LD)/R
14 Z(I)=0
   Y(I)=0
   X(I)=0
   I=2
15 ANQ=TETA0+TETA*(I-2)
   IF (ANG. GT. 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 = I
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
KZ = 0
19 ANG = TETAO + TETA * (I + (N-2))
IF (ANG, GT, 1.570796327) GOTO 20
VALYR = R * (OM**2) * DCOS(ANG)
VALZR = - R * (OM**2) * DSIN(ANG)
GOTO 22
20 VALYR = 0
VALZR = 0
22 IF (I, NE, 0) GOTO 24
VALYL = 0
VALZL = R * OM
GOTO 25
24 VALYL = DYL F
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, ITYPE, VALYR)
CALL BIAC (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 - 1
CALL BIAVL(1) (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYL F)
CALL BIAVL(1) (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DZLF)
IF (XREF(IMAX), GT, TT) GOTO 40
K = 0
I2 = 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)
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) * OM - TETA + TETAO
YARG(I) = DR(I)
50 CONTINUE
RETURN
END
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 DIAC 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).

! COMMON /FNames/ N1(20), N2(20)

DATA N1 / 'SPDF', 'I', '18*', '/,
* N2 / 'SPDF', 'H', '18*', '/
END
SUBROUTINE FUNCT (TARG,YARG,NP,OK,GEG, IRES)
REAL TARG(500), YARG(500), GEG(1)
INTEGER NP, IRES
LOGICAL OK
REAL OM, LD, R, TETA, TETAO, ANG, T, VALYR, VALZR,
* VALYL, VALZL, DYLDF, DZLDF, PAS, ZREF1
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), XREF1(500), TT
INTEGER I, K1, K2, K, ITYPEL, ITYPER, N, I1, J, IMAX, I2, NP, KL, N1

N=GEQ(1)
OM=GEQ(2)
LD=GEQ(3)
R=GEQ(4)
TETA=GEQ(5)
PAS=GEQ(6)
T=TETA/OM
N1=N-1
TT=1.570796327/OM+9*T

IF (LD .NE. 0) GOTO 10
TETAO=TETA
GOTO 13
10 TETAO=(R*TETA-LD)/R
13 DO 14 I=1, N1
   X(I)=(I-1)*T
   Z(I)=R*OM*X(I)
   14 Y(I)=0
   1=4
15 ANG=TETAO+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=1
   K2=I+10
   DO 18 K=K1, K2
      X(K)=(K-1)*T
      Y(K)=R*(TETAO-0.570796327+TETA*(K-4))
   18 Z(K)=R+Z(3)+LD
      ITYPEL=1
      ITYPER=1
      I=0
      KL=0
      ZREF1=Z(3)+LD
19 ANG=TETAO+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
   VALZL=R*OM
   GOTO 25
24 VALYL=DYLF
   VALZL=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 BIAVL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYL)
   CALL BIAVL1 (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)
33 DR(KL)=YREF(K)
   GOTO 35
34 DR(KL)=ZREF1+R-ZREF(K)
35 XREF1(KL)=XREF(K)
   GOTO 32
36  I=I+1
    GOTO 19
40  OK=. TRUE.
    NP=KL
    DO 50 J=1,NP
        TARG(I)=XREF1(I)
        YARG(I)=DR(I)
    50 CONTINUE
    RETURN
END
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.
- VMAX: maximum angular speed to reach.
- AMAX: angular acceleration
- R: radius of the circle
- T: time between each knot
- PAS: time between each set points

The subroutines BIA and BIAVL1 are the same as in SPC.FTN. We display to the screen the curve YARG=f(TARG).

**BLOCK DATA**

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

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

**SUBROUTINE FUNCT (TARG, YARG, NP, QK, QEG, IRES)**

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

INTEGER N, I, IA, ITYPEP, ITYPEL, K1, K2, K, KL, I1, I2,
* J, IMAX, NP, KA

REAL VMAX, AMAX, R, T, PAS

REAL*8 T1, T2, T3, TETA1, TETA2,
* VALYR, VAL2R, VALYL, VALZL, DYL, DZL

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=CEQ(1)
VMAX=CEQ(2)
AMAX=CEQ(3)
R=CEQ(4)
T=CEQ(5)
PAS=CEQ(6)
T1=VMAX/AMAX
T2=(1.570796327-AMAX*(T1**2))/VMAX
T3=T1+T1
TETA1=AMAX*(T1**2)/2
TETA2=TETA1+(1.570796327-AMAX*(T1**2))
IF (T1.GT.0) GOTO 08
WRITE (1,06)
06 FORMAT ('T1 OR TI . LE. T')
08 X(1)=0
Y(1)=0
Z(1)=0
I=1
IA=1
10 X(I)=T1*X(I-1)
    IF (X(I).GE.T1) GOTO 12
    ANG(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
    GOTO 10

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

14 X(I)=T3*X(I-1)
    IF (X(I).GE.T3) GOTO 16
    ANG(IA)=TETA2+(VMAX-AMAX*(X(I)-T2)/2)*(X(I)-T2)
    Y(I)=R-R*DCOS(ANG(IA))
    Z(I)=R*DSIN(ANG(IA))
    I=I+1
    IA=IA+1
    GOTO 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 ITYPEL=1
    ITYPER=1
    I=0
    KA=N
    IA=N-1
    GOTO 42

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

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

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

23 VALYR=0
    VALZR=0

24 IF (I.NE.0) GOTO 26
    VALYL=0
    VALZL=0
    GOTO 27

26 VALYL=DYLF
    VALZL=DZLF

27 DO 28 K=1,N
    I=I+1

C
XB(K)=X(I1)
YB(K)=Y(I1)
28  ZB(K)=Z(I1)
    CALL B1AC (CY, XB, YB, N, DYREF, ITYPEL, VALYL, ITYPE, VALYR)
    CALL B1AC (CZ, XB, ZB, N, DZREF, 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 B1AVL1 (CY, XB, YB, N, XREF, YREF, DYNREF, IMAX, DYLF)
    CALL B1AVL1 (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DZLF)
    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)
    XREF1(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*'/

END

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

REAL B OM, LD, R, TETA, TETA0, ANG, T, VALYR, VALZR,
* VALYL, VALZL, DYLFL, DZLFL, PAS

REAL X(50), Y(50), Z(50), XB(10), YB(10), ZB(10),
*XREF(400), YREF(400), ZREF(400), CY(50), CZ(50), DZREF(400),
*DZ2REF(400), DR(500), XREF(500), TT

INTEGER I, K1, K2, K, ITYPEL, ITYPEI, N, I1, J, IMAX, I2, NP, KL
C SAME PROGRAM AS SPCF.FTN BUT THERE IS A KNOT

19 ANG=TE1AO+TETA*(I+(N-2))
   IF (ANG.GE.1.570796327) GOTO 20
   VALY=R*OM*DSIN(ANG)
   VALZR=R*OM*DCOS(ANG)
   GOTO 22
20 VALYR=R*OM
   VALZR=0
22 IF (I.NE.O) GOTO 24
   VALYL=0
   VALYL-R*OM
   GOTO 25
24 VALYL=DIYLF
   VALYL=DIZLF
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, ITYPEL, VALYR)
   CALL BJAC (CZ, XB, ZB, N, DZREF, ITYPEL, VALZL, ITYPEL, 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 BIAVL1 (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DIYLF)
   CALL BJAVL1 (CZ, XB, ZB, N, XREF, ZREF, DZREF, IMAX, DIZLF)
   IF (XREF(IMAX).GT.TT) GOTO 40
   K=0
32 K=K+1
34 I=I+1
30 CONTINUE
40 OK=.TRUE.
40 NP=KL
   DO 50 I=1,NP
   TARG(I)=XREF(I)*OM-TETA+TETAO
   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:
- \( V_{MAX} \): maximum angular speed to reach
- \( A_{MAX} \): angular acceleration
- \( R \): radius of the circle
- \( T \): time between two knots
- \( \text{PAS} \): angle between two set points

We display on the screen the curve \( ERA_1 = f(ANG_1) \).

Here is the code:

```fortran
BLOCK DATA
COMMON \( \text{FNAMES} \) / N1(20), N2(20)/

DATA N1 / 'ILF.', 'I ', '18*/'/,
* N2 / 'ILF.', 'H ', '18*/'/
END

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

REAL*8 MAX, AMAX, R, T, PAS, T1, T2, T3, TI, TETA1, TETA2

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

MAX=GEQ(1)
AMAX=GEQ(2)
R=GEQ(3)
T=GEQ(4)
PAS=GEQ(5)
T1=MAX/AMAX
T1=(1.570796327-AMAX*(T1**2))/MAX
T2=T1+1
T3=2*T1+1
TETA1=AMAX*(T1**2)/2
TETA2=TETA1+(1.570796327-AMAX*(T1**2))
IF (TI.GT.0) GOTO 0B
WRITE (1,06) 06 FORMAT('TI . LT. 0')
0B I=1
10 TC(I)=(I-1)*T
IF (TC(I).G.T1) GOTO 12
ANG(I)=AMAX*(TC(I)**2)/2
X(I)=R-R*DOS(ANG(I))
Y(I)=R*DSIN(ANG(I))
I=I+1
GOTO 10
12 TC(I)=(I-1)*T
IF (TC(I).G.T2) GOTO 14
ANG(I)=TETA1+(TC(I)-T1)*MAX
```
X(I)=R-R*DCOS(ANG(I))
Y(I)=R*DSIN(ANG(I))
I=I+1
GOTO 12
14 TC(I)=(I-1)*T  
   IF (TC(I).GT.T3) GOTO 16  
   ANG(I)=TETA2+(VMAX-AMAX*(TC(I)-T2)/2)*(TC(I)-T2)  
   X(I)=R-R*DCOS(ANG(I))  
   Y(I)=R*DSIN(ANG(I))  
   I=I+1  
   GOTO 14
16 N=I-1
   L=1
   K=1
   I=2
17 J=1
18 TARG(J)=ANG(I-1)+(J-1)*PAS  
   XREF(J)=R-R*DCOS(TARG(J))  
   IF (XREF(J).GT.X(I)) GOTO 20  
   J=J+1  
   GOTO 18
20 IMAX=J-1
   DO 22 K1=1, IMAX  
   YREF(K1)=Y(I-1)+(Y(I)-Y(I-1))/(X(I)-X(I-1))  
   XREF(K1)=XREF(K1)-X(I-1))  
   YREF(K1)=YREF(K1)  
   ERA(K1)=R-DSQR((R-XREF(K1))**2+YREF(K1)**2)  
   ERA1(K)=ERA(K1)  
   ANG1(K)=TARG(K1)
22 K=K+1
   I=I+1
   IF (I.LE.N) GOTO 17
   OK=.TRUE.
   NP=K-1  
   RETURN
END
This program interpolates a waving curve \( A\sin(X(I)) \)
by spline functions.
The boundary conditions are given by the first derivatives.
The subroutines B1AC and B1AVL1 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 \( ERA1=f(XREF1) \).

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

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

SUBROUTINE FUNCT (XREF1, ERA1, NP, OK, QEG)
REAL XREF1(500), ERA1(500), QEG(1)
INTEGER NP
LOGICAL OK
REAL OM, T, PAS, A, VALYR, VALYL, DYL

REAL*8 X(100), Y(100), XB(10), YB(10), CY(10), XREF(100),
* YREF(100), ERA(100), DYLE(100)

N=QEG(1)
OM=QEG(2)
T=QEG(3)
PAS=QEG(4)
A=QEG(5)
K=1
K2=1
ITYPE1=1
ITYPE2=1
DO 10 J=1, 100
X(I)=OM*T*(I-1)
10    Y(I)=A*cos(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=DYL
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, DYLE, ITYPE1, VALYL, ITYPE2, VALYR)
    L=0
```
18 \( L=L+1 \)
\[ XREF(L) = PAS*(L-1) + T*OM*(I-1) \]
IF \((XREF(L) \leq XB(2))\) QOTO 18
\( \text{IMAX} = L-1 \)
CALL BIAVI (CY, XB, YB, N, XREF, YREF, DYREF, IMAX, DYLF)
DO 20 \( K1=1, IMAX \)
\[ \text{ERA}(K1) = A*DCOS(XREF(K1)) - YREF(K1) \]
\[ \text{ERA1}(K2) = \text{ERA}(K1) \]
\[ XREF1(K2) = XREF(K1) \]
20 \( K2=K2+1 \)
IF \((I \lt N)\) QOTO 11
\( \text{NP} = K2-1 \)
\( \text{OK} = \text{TRUE.} \)
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
OM(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 1.5707963275.235987756 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)
```

1.5707963275.235987756 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
<table>
<thead>
<tr>
<th>T(s)</th>
<th>Error-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>3.00</td>
</tr>
<tr>
<td>0.2</td>
<td>2.50</td>
</tr>
<tr>
<td>0.3</td>
<td>2.00</td>
</tr>
</tbody>
</table>
$SAVE NOLIST

DG$ATTACH :PROCEDURE (PATH$P, EXCEP$P) WORD EXTERNAL;
    DECLARE PATH$P POINTER;
    DECLARE EXCEP$P POINTER;
    END;

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

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

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

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

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

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

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

$RESTORE
MM2 : DO;

/* DECLARATION OF THE RUN-TIME PROCEDURES;

DQ$ATTACH: CREATES A CONNECTION TO AN EXISTING FILE.
   'CI': CONSOLE INPUT (EXISTING FILE),
   CI: NAME OF THE CONNECTION.
DQ$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(‘FD: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 BYTES TRANSFERNRED.
   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;
MM1 :DO;

/* DECLARATION OF THE RUN-TIME PROCEDURES (SEE: UDI.EXT): */
DQ$CREATE: CREATES A CONNECTION TO AN EXISTING FILE.
' :CO: ': CONSOLE OUTPUT (EXISTING FILE).
CO: NAME OF THE CONNECTION.
DQ$OPEN: PREPARES THE CONNECTION FOR WRITE COMMANDS.
2 MEANS WE HAVE A WRITE ACCESS ONLY.
0 SIGNIFIES THAT NO BUFFERING SHOULD OCCUR.
DQ$WRITE: TRANSFERS DATA FROM MAIN MEMORY TO A FILE.
1 IS THE NUMBER OF BYTE TO BE WRITTEN.
DQ$CLOSE: WAITS FOR COMPLETION OF I/O OPERATIONS
TAKING PLACE ON THE FILE.
DQ$DETACH: BREAKS THE CONNECTION ESTABLISHED
BY DQ$CREATE. */

$INCLUDE(:F1:UDI.EXT)

OUTCAR :PROCEDURE (J) PUBLIC;

/* J: ASCII CHARACTER TO DISPLAY TO THE SCREEN.
STATUS: WORD IN WHICH THE OPERATING SYSTEM
    RETURNS AN EXCEPTION CODE.
CO: SEE ABOVE. */

DECLARE J BYTE;
DECLARE CO WORD;
DECLARE STATUS WORD;

CO=DQ$CREATE('4',':CO:',@STATUS);
CALL DQ$OPEN(CO,2,0,@STATUS);
CALL DQ$WRITE(CO,J,1,@STATUS);
CALL DQ$CLOSE(CO,@STATUS);
CALL DQ$DETACH(CO,@STATUS);
END OUTCAR;
END MM1;
GETINT : PROCEDURE GETS AN ASCII STRING FROM THE KEYBOARD, CALCULATES ITS INTEGER VALUE, AND PUTS THE CURSOR TO THE NEXT LINE. */

MM4 : DO;

/* DECLARATION OF THE EXTERNAL PROCEDURES: */

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 '0DH';
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 = '*'
    END;
  END;
END;
ELSE DO;
  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;
/* 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 SPF.FTN. THE PROGRAM IS WRITTEN FOR AN 8086 APLICATION. */

SPM : DO;


DECLARE N LITERALLY '4EH';
DECLARE X LITERALLY '58H';
DECLARE Y LITERALLY '59H';
DECLARE CR LITERALLY '0DH';
DECLARE LF LITERALLY '0AH';
DECLARE VALR REAL INITIAL (0.);
DECLARE CONTI BYTE INITIAL (59H);
DECLARE COOYI (4) INTEGER;
DECLARE COOY (4) REAL;
DECLARE (A0,A1,A2,A3) REAL;
DECLARE (I1,I,J) BYTE;

declare all external procedures; DQEXIT: PROCEDURE (EXCEP) EXTERNAL;
DECLARE EXCEP WORD;
END;

GETINT: PROCEDURE INTEGER EXTERNAL;
END;

EXIINT: 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 CALCULATION 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 PY INTEGER;

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

ASK : PROCEDURE BYTE;
/* TXT CONTAINS THE ASCII VALUES OF THE LETTERS 'CONTINUE'.
STRING: STRING OF ASCII CHARACTERS PICKED FROM THE KEYBOARD.
I: 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 OUTCAR(TXT(J));
END;
CALL INTCAR(@STRING);
I=0;
J=STRING(0);
DO WHILE J<>Y AND J<>N ;
   J=STRING(I);
   I=I+1;
END;
CALL OUTCAR(J);
CALL OUTCAR(CR);
CALL OUTCAR(LF);
RETURN J;
END ASK;

CALCC: PROCEDURE (PY$PTR, VI$PTR, VI$PTR, VI$PTR,
        CO$PTR, CI$PTR, C2$PTR, C3$PTR);
/* WE CAN RECOGNISE HERE THE SAME CALCULATION METHOD
AS IN THE FORTRAN PROGRAM SP_FTIL. WORK IS CALCULATED
WITH THE NUMERICAL VALUE: X(I)-X(I-1)=100. */

DECLARE PY$PTR POINTER;
DECLARE (VI$PTR, VI$PTR, CO$PTR, CI$PTR, C2$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 WORK (3) REAL DATA (200., 350., 371., 92857);
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);
END;
COEFF(3)=6.*VITL-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$PTR POINTER;
DECLARE JI INTEGER;
DECLARE (PY BASED PY$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 JI=0 TO 100 BY 51;
  YLI=FIX((C3*FLOAT(JI)/100.+C2)*FLOAT(JI)/100. 
           +C1)*FLOAT(JI)/100.+C0);
  CALL EXITINT@YLI);
END;
CALL OUTCAR(LF);
VITL=((3.*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 INITREALMATMUNIT;

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

DO II=0 TO 3;
    CALL POINT(@CODYI(I1));
    CODY(I1)=FLOAT (CODYI(I1));
END;
CALL OUTCAR(LF);
VALR=(CODY(3)-CODY(2))/100.; /* LINEAR INTERPOLATION */
DO WHILE CONTI=1;
    CALL CALCC(@CODY,YVAL,VALR,A0,A1,A2,A3);
    CALL SPLI(@CODY,A0,A1,A2,A3,YVALL);
    DO II=0 TO 2;
        CODY(II)=CODY(II+1);
    END;
    CALL POINT(@CODYI(3));
    CALL OUTCAR(LF);
    CODY(3)=FLOAT(CODYI(3));
    VALR=(CODY(3)-CODY(2))/100.;
    CONTI=ASK;
END;
CALL DOEXIT(0);
END SP1;
/*SP: 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 DESCRIBED IN THE PARAGRAPH
 'APPLICATION OF CALCULATIONS FOR FOUR KNOTS'. IT IS WRITTEN FOR
 AN 8086 APPLICATION. */

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.
 CODYI: INTEGER VALUE OF THE KNOT ORDONATE.
 CODY: REAL VALUE OF THE KNOT ORDONATE. */

DECLARE (II, I) BYTE;
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 BYTE INITIAL (59H);
DECLARE CODYI (4) INTEGER;
DECLARE CODY (4) REAL;

/*DECLARATION OF ALL THE EXTERNAL PROCEDURES:
 DHXEXIT: FINISHES A PROGRAM.
 GETINT: GETS AN ASCII STRING FROM THE KEY-BRoard
 AND TRANSFORMS IT TO AN INTEGER VALUE.
 EXINT: 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. */

DIXELT:PROCEDURE (EXCEPT) EXTERNAL;
DECLARE EXCEPT 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 ORDONATE OF THE KNOTS.
 ASK: ASKS IF WE WANT TO GET OUT OF THE PROGRAM.
 SPLI: CALCULATES THE COEFFICIENTS OF THE SPLINE FUNCTION
 BETWEEN THE TWO FIRST KNOTS; CALCULATES THE VALUES
 OF THE SET POINTS ALL THE SAME, AND RESTITUTES THEM.
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,54H,49H,4EH,55H,45H,3AH);
DECLARE STRING (6) BYTE;
DO J=0 TO 9;
   CALL OUTCAR(TXT(J));
END;
CALL INTCAR(@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 (3) REAL;
DECLARE YLI INTEGER;
DECLARE DIF (3) REAL;
DIF ARE (3) REAL = REAL;
DECLARE J1 INTEGER;
DECLARE I INTEGER;
DECLARE 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);
CA = DIF(0) * 0.01 - V11;
S2(1) = -0.0053333 * BE + 0.018666 * AL - 0.933333 * GA;
S2(0) = 3. * CA - 0.5 * S2(1);
C(0) = PY(0);
C(2) = 0.5 * S2(0);
C(1) = DIF(0) - 16.6666666 * (2. * S2(0) + S2(1));
C(3) = 0.001666666 * (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;

Y11 = (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.
VALL: left boundary condition.
VALR: right boundary condition.
CONTi: test value to know if the program must continue.
COOYI: integer value of the knot ordinate.
COOY: real value of the knot ordinate. */

CALL INIT$REAL$MATH$UNIT;

DO II = 0 TO 3;
   CALL POINT@COOYI(II));
   COOY(II) = FLOAT(COOYI(II));
END;
VALR = (COOY(3) - COOY(2)) / 100;
DO WHILE CONTi = Y;
   CALL SPLI@COOY + (VALL + VALR);
   DO II = 0 TO 2;
      COOY(II) = COOY(II + 1);
   END;
   CALL POINT@COOYI(3));
   COOY(3) = FLOAT(COOYI(3));
   VALR = (COOY(3) - COOY(2)) / 100;
   CONTi = ASK;
END;
CALL DOEXIT(0);
END SP;
### Results of sp.ftn

<table>
<thead>
<tr>
<th>X(I), Y(I), I=1...N</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>100 900</td>
</tr>
<tr>
<td>200 20</td>
</tr>
<tr>
<td>300 500</td>
</tr>
</tbody>
</table>

**ITYPEL, VALL, ITYPER, VALR**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 4.8</td>
</tr>
</tbody>
</table>

**C =**

```
  0.51440E 00  -0.48880E 00  0.37280E 00  -0.18640E 00
```

**IMIN, IMAX**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X(I), Y(I), I=1...N</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>100 900</td>
</tr>
<tr>
<td>200 20</td>
</tr>
<tr>
<td>300 500</td>
</tr>
<tr>
<td>400 -200</td>
</tr>
</tbody>
</table>

**ITYPEL, VALL, ITYPER, VALR**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1.2799 1 -7</td>
</tr>
</tbody>
</table>

**C =**

```
  -0.50784E 00  0.41088E 00  -0.31968E 00  0.15984E 00
```

**IMIN, IMAX**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X(I), Y(I), I=1...N</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>100 900</td>
</tr>
<tr>
<td>200 20</td>
</tr>
<tr>
<td>300 500</td>
</tr>
<tr>
<td>400 -200</td>
</tr>
</tbody>
</table>

**ITYPEL, VALL, ITYPER, VALR**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1.2799 1 -7</td>
</tr>
</tbody>
</table>

**C =**

```
  0.100000E 03  0.900000E 03  0.127990E 01  0.105000E 03  0.900243E 03  -0.114444E 01  0.110000E 03  0.888938E 03  -0.333911E 01  0.115000E 03  0.867235E 03  -0.530409E 01  0.120000E 03  0.836280E 03  -0.703940E 01  0.125000E 03  0.797223E 03  -0.854503E 01  0.130000E 03  0.751213E 03  -0.982098E 01  0.135000E 03  0.699351E 03  -0.108673E 01  0.140000E 03  0.642923E 03  -0.116838E 01  0.145000E 03  0.582941E 03  -0.122708E 01  0.150000E 03  0.520598E 03  -0.126280E 02  0.155000E 03  0.457044E 03  -0.127555E 02
```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.160000E+03</td>
<td>0.393426E+03</td>
<td>-0.126534E+02</td>
</tr>
<tr>
<td>0.165000E+03</td>
<td>0.330892E+03</td>
<td>-0.123216E+02</td>
</tr>
<tr>
<td>0.170000E+03</td>
<td>0.270592E+03</td>
<td>-0.117601E+02</td>
</tr>
<tr>
<td>0.175000E+03</td>
<td>0.213674E+03</td>
<td>-0.109690E+02</td>
</tr>
<tr>
<td>0.180000E+03</td>
<td>0.161285E+03</td>
<td>-0.094814E+01</td>
</tr>
<tr>
<td>0.185000E+03</td>
<td>0.114575E+03</td>
<td>-0.086976E+01</td>
</tr>
<tr>
<td>0.190000E+03</td>
<td>0.746919E+02</td>
<td>-0.072174E+01</td>
</tr>
<tr>
<td>0.195000E+03</td>
<td>0.427838E+02</td>
<td>-0.055075E+01</td>
</tr>
<tr>
<td>0.200000E+03</td>
<td>0.199993E+02</td>
<td>-0.035679E+01</td>
</tr>
</tbody>
</table>
Results of SPM.PLH

<table>
<thead>
<tr>
<th>POINT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
</tbody>
</table>

Appendix N

CONTINUITY

<table>
<thead>
<tr>
<th>POINT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
<tr>
<td>+0.0000</td>
<td></td>
</tr>
</tbody>
</table>

CONTINUE...
/* SPM96: 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 SPM.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
  PRECEEDING 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 I BYTE;
DECLARE COEFF0 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=NOT PORT;
  COEFF0=0.06*(PY(1)-PY(0));
  COEFF(0)=COEFF0-6.*VITL;
  DO 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;
  COEFF3=6.*VITR-COEFF0-0.269230*C0Eфф3(2);
  COEFF3=C0Eфф3/173.077;
  I=3;
  DO WHILE I>=1;
    COEFF(I-1)=COEFF(I-1)-100.*COEFF(I)/WORK(I-1);
  I=I-1;
  END;
  C3=0.00166666666*(C0Eфф3(I)-C0Eфф3(0));
  C2=0.5*C0Eфф3(0);
  C1=(PY(1)-PY(0))*0.01-16.6666666666*2.*C0Eфф3(0)+C0Eфф3(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.PLX. */

// SP96:00:

// THE AT ATTRIBUTE ALLOWS US TO SAVE MEMORIES BY STORING VALUES IN PLACES NO MORE USED.

THE DIFFERENT VARIABLES ARE:

PY: ORDI NATE 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 INITIALIZATION;
DISABLE;
DO WHILE I=1:
    PORT=NOT 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)=-0.00533333*BE+0.01666666*AL-0.9333333*GA;
    S2(0)=1.5*GA-0.25*S2(1);
    C0=FIX(PY(0)*100,);
    C1=FIX(DIF(0)-16.6666666*(4*S2(0)+S2(1)));
    C2=FIX(S2(0));
    C3=FIX(0.0016666666*(S2(1)-S2(0)-S2(0)));
END;
END SP96;

/* TOTAL RAM SPACE MEMORY NEEDED: 38 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 */
C7L9E : THIS PROGRAM CALCULATES THE SET POINT VALUES OF A
SPLINE FUNCTION, DEFINED BY ITS COEFFICIENTS CALCULATED
IN SPK06.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 1DO:

*/ PORT: BYTE SENSES TO THE OUTPUT PORT.
 YLI: VALUE OF THE SET POINT.
 C0...C3: COEFFICIENTS OF THE SPLINE FUNCTION.
 VITL: LEFT BOUNDARY CONDITION.
 PX: ABSOLUTE 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 INITSREAL#MATH#UNIT;
PX=0.;
DO WHILE I=1;
   PORT=NOT PORT;
   PX=PX+5.;
   YLI=(((C3*PX)+C2)*PX+C1)*PX+C0;
   VITL=(((C3*PX)+C3)*PX+C2)*PX+C1;
END;
END CAL96;

*/ RAM SPACE MEMORY NEEDED: 2B 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.PLM, BETWEEN THE TWO FIRST KNOTS. THIS PROGRAM CAN
BE RUN ON A B076 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.
YL1: VALUE OF THE SET POINT.
C0...C3: COEFFICIENTS OF THE SPLINE FUNCTION.
V1L1: LEFT BOUNDARY CONDITION.
PX: ABSYSCA OF THE SPLINE FUNCTION TO CALCULATE
THE SET POINT VALUES. */
DECLARE PORT BYTE AT (000FH);
DECLARE YL1 LONGINT;
DECLARE (C0,C1,C2,C3) LONGINT DATA (50,-48,37,-18);
DECLARE V1L1 LONGINT;
DECLARE PX INTEGER;

DO WHILE PX=PX;
    PORT=NOT PORT;
    PX=5*PX;
    YL1=(((C3*PX)+C2)*PX+C1)*PX+C0)/100;
    V1L1=(((C3+C3+C3)*PX+C2+C2)*PX+C1)/100;
END;
END CAL96;

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