TECHNICAL TRANSACTIONS 1/2019
CIVIL ENGINEERING
DOI: 10.4467/2353737XCT.19.008.10048
SUBMISSION OF THE FINAL VERSION: 28/11/2018

Agnieszka Makowska orcid.org/0000-0003-1067-7130
amakow@pk.edu.pl
Faculty of Civil Engineering, Cracow University of Technology

Verification of the spline method and its application to curvilinear objects

Weryfikacja metody splajn i jej zastosowanie do obiektów krzywoliniowych

Abstract

Two methods of interpolation are presented in this article: interpolation with the help of orthogonal polynomials and interpolation on a cubic spline path (with the help of glued-together functions). Two procedures have been written by the author: *WielOrt* and *Splajn*. A comparative analysis of these procedures was conducted by four verification methods that have been created by the author. The examples of verification were chosen so as to make it possible to compare the created by author interpolating function $f(x)$ and the known function $g(x)$. Graphics, numerical procedures and examples were prepared in the *Mathematica* program.
Keywords: interpolating, spline, orthogonal polynomial

Streszczenie

W artykule omówione zostały dwie metody interpolacji: interpolacja za pomocą wielomianów ortogonalnych oraz interpolacja za pomocą sześciennych funkcji sklejanych (splajnów). Napisane zostały przez autora dwie procedury *WielOrt* and *Splajn*. Przeprowadzono wnikliwą analizę porównawczą tych procedur, przykłady zostały dobrane tak, aby możliwe było porównanie utworzonej przez autora funkcji interpolacyjnej $f(x)$ ze znaną funkcją interpolowaną $g(x)$. Weryfikację przeprowadzono czterema opracowanymi przez autora metodami. Grafika, procedury numeryczne i przykłady zostały przygotowane w programie: *Mathematica*.
Słowa kluczowe: krzywoliniowość, interpolacja, splajn, wielomian ortogonalny

1. INTRODUCTION

The aim of this scientific description is to prove the correctness *The Spline Method* created by author, by verifying the basic *Splajn* procedure of this method, using four verification methods in addition created by the author, and demonstrating the universal *Splajn* procedure application for curved objects without classification in the fields of science. The procedures apply to creating objects with a free, irregular soft form. The package of computational procedures called *The Spline Method* contains the following procedures: *Splajn, Splajn1, SplajnDluku, SplajnObjPow, SplajnDzialkaSciana, SplajnRurociąg, Splajn 4G* combine components graphic and mathematical.

The *Splajn* procedure is the basic procedure of this method. It is necessary to perform a thorough verification of the method. Four verification methods were created for thoroughly checking the calculation results of the *Splajn* procedure.

The basis calculating is the exact calculation of the length, the surface area and the volume. The exact calculation of the surface and the volume of rectilinear forms is easy; however, in the case of curvilinear objects, it is harder.

An attempt to describe curvilinear objects made with the application of cubic spline interpolation is presented in this paper.

2. SOME METHODS OF INTERPOLATION

Based on literature [5], the basic concepts of the spline theory are presented in this section.

2.1. Elementary theory of cubic spline

We have got n+1 points in the interval <a, b> : $a = x_0, x_1, \ldots, x_n = b$, we call nodes and value function $y = f(x)$ in the points: $f(x_0) = y_0$, $f(x_1)$, …,$f(x_n)$. Pair (x_k, y_k) we call nodal points. We seek estimate values of function $f(x)$ class C2 between nodes using third degree polynomial for $x \in$ <xi-1, xi>.

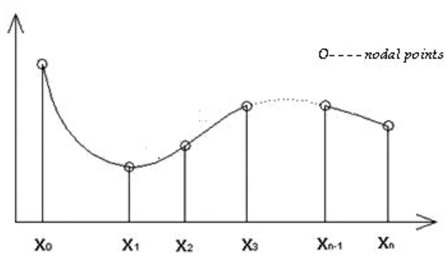


Fig. 1. Illustrative figure (prepared by author)

Let us mark *Mi* as second derivative in point *x*i:

$$M_i = f''(x_i) \quad \text{for} \quad i = 0,1,2,\ldots,n \qquad (1)$$

from definition of function f(x), it is known that f ”(x) is the continuous function in interval <a ,b> and linear for $x \in <x_{i-1}, x_i>$, so:

$$f''(x_i) = M_{i-1}\frac{x_i - x}{h_i} + M_i\frac{x - x_{i-1}}{h_i} \tag{2}$$

where: $x \in (x_{i-1}, x)$, $h = x_i - x_{i-1}$.

Integrating twice (2) and evaluating the constants of integration:

$$f'(x) = -M_{i-1}\frac{(x_i - x)^2}{2h_i} + M_i\frac{(x - x_{i-1})^2}{2h_i} + A_i \tag{3}$$

$$f(x) = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + A_i(x - x_{i-1}) + B_i \tag{4}$$

where :

$$B_i = y_{i-1} - M_{i-1}\frac{h_i^2}{6};\quad A_i = \frac{y_i - y_{i-1}}{h_i} - \frac{h_i}{6}(M_i - M_{i-1}) \tag{5}$$

Using the condition of continuity of function and the first derivative by algebraic conversion we obtain linear system of equations:

$$\mu_i M_{i-1} + 2M_i + \lambda_i M_{i+1} = d_i \quad \text{for} \quad i = 1,2,\ldots,n-1 \tag{6}$$

where:

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}},\quad \mu_i = 1 - \lambda_i$$

$$d_i = \frac{6}{h_i + h_{i+1}}\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right) \quad \text{for} \quad i = 1,2,\ldots,n-1$$

System (6) has n–1 equations and n+1 unknown coefficients: $M_0, M_1, \ldots, M_n$.

We often accept two additional conditions:

$M_0 = 0$, $M_n = 0$ ($f''(x_0) = 0, f''(x_n) = 0$: Natural cubic Spline), system (6) can be written in matri*x* form as:

$$\begin{bmatrix} 2 & \lambda_1 & 0 & \ldots & 0 \\ \mu_2 & 2 & \lambda_2 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \mu_{n-2} & 2 & \lambda_{n-2} \\ 0 & \ldots & \ldots & \mu_{n-1} & 2 \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \ldots \\ M_{n-2} \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \ldots \\ d_{n-2} \\ d_{n-1} \end{bmatrix} \tag{7}$$

This is a special three-diagonal linear system;
on this Fortuna Z. [5] finished the description of the method, the author continues to solve the system of equations and describes the function $f(x)$

The elementary method of solving system (6):

System (6) in the explicity form is:

$$\mu_1 M_0 + 2M_1 + \lambda_1 M_2 = d_1$$

$$\mu_2 M_1 + 2M_2 + \lambda_2 M_3 = d_2$$

$$\mu_3 M_2 + 2M_3 + \lambda_3 M_4 = d_3$$

- -

$$\mu_{n-1} M_{n-2} + 2M_{n-1} + \lambda_{n-1} M_n = d_{n-1} \tag{8}$$

where:

$$\lambda_i = \frac{h_{i+1}}{h_i + h_{i+1}}, \mu_i = 1 - \lambda_i \quad i = 1,2,\ldots,n-1$$

$$h_i = x_i - x_{i-1}, \qquad i = 1,2,\ldots,n.$$

Let us mark u_i and z_i as:

$$u_i = 2 - \frac{\mu_i}{u_{i-1}} \lambda_{i-1} \quad \text{for} \quad i = 2,3,\ldots,n-1 \tag{9}$$

$$z_i = d_i - \frac{\mu_i}{u_{i-1}} z_{i-1} \quad \text{for} \quad i = 2,3,\ldots,n-1 \tag{10}$$

and for symmetrical expression:

$$u_1 = 2, \; z_1 = d_1 \tag{11}$$

From the first equation of system (8), we obtain:

$$2M_1 + \lambda_1 M_2 = d_1$$

from (9), (10), (11) :

$$M_1 = \frac{z_1 - \lambda_1 M_2}{u_1}$$

we calculate the next coefficients:

$$M_2 = \frac{z_2 - \lambda_2 M_3}{u_2}; \quad M_3 = \frac{z_3 - \lambda_3 M_4}{u_3}$$

by recurrence and algebraic conversion, we obtain:

$$M_k = \frac{z_k - \lambda_k M_{k+1}}{u_k} \quad \text{for} \quad k = 1,\ldots,n-2 \tag{12}$$

Using principle of mathematical induction easily proof the truth of this expression (12)
From the last equation of system (8), by algebraic conversion we obtain:

$$M_{n-1}=\frac{z_{n-1}}{u_{n-1}}$$

after calculation coefficients M_k, we construct function *f(x)*:

$$f(x)=\begin{cases} f_0 \; dla < x_0, x_1 > \\ f_1 \; dla < x_1, x_2 > \\ ---------- \\ f_{n-1} \; dla < x_{n-1}, x_n > \end{cases}$$

where f_i is the right side of expression (4).

We define Heaviside's function [11]:

$$H(x-a)=\begin{cases} 0 \; dla \; x < a \\ 1 \; dla \; x \geq a, \end{cases}$$

and then function *f(x)* is expressed as one formula:

$$f(x)=\sum_{i=1}^{n-1} f_i \cdot \left[H(x-x_{i-1}) - H(x-x_i) \right] + f_n \cdot H(x-x_{n-1}) \tag{13}$$

Now we can write in any programming language the basic procedure called *Splajn*, in which the input parameters are data points: (x_0,y_0), (x_1,y_1), ..., (x_n,y_n) and at the output of that procedure, we will get function *f(x)*.

```
Splajn[lista_] :=
 Module[{}, Clear[n, m, u, z, d, X, Y, h, a, b, λ, μ, f];
  n = Length[lista] - 1; m[0] = 0; m[n] = 0; u[1] = 2;
  z[1] = d[1]; For[i = 0, i ≤ n, i++, X[i] = lista[[i + 1, 1]];
   Y[i] = lista[[i + 1, 2]]];
  For[i = 1, i ≤ n, i++, h[i] = X[i] - X[i - 1];
   b[i] = Y[i - 1] - m[i - 1] * h[i]^2 / 6;
   a[i] = (Y[i] - Y[i - 1]) / h[i] - h[i] * (m[i] - m[i - 1]) / 6;
   f[x_, i_] := m[i - 1] * (X[i] - x)^3 / (6 * h[i]) +
     m[i] * (x - X[i - 1])^3 / (6 * h[i]) + a[i] * (x - X[i - 1]) +
     b[i]]; For[i = 1, i ≤ n - 1, i++,
   λ[i] = h[i + 1] / (h[i] + h[i + 1]); μ[i] = 1 - λ[i];
   d[i] = 6 / (h[i] + h[i + 1]) *
     ((Y[i + 1] - Y[i]) / h[i + 1] - (Y[i] - Y[i - 1]) / h[i])];
  For[i = 2, i ≤ n - 1, i++, u[i] = 2 - μ[i] / u[i - 1] * λ[i - 1];
   z[i] = d[i] - μ[i] / u[i - 1] * z[i - 1]];
  m[n - 1] = z[n - 1] / u[n - 1];
  For[i = n - 2, i ≥ 1, i--,
   m[i] = (z[i] - λ[i] * m[i + 1]) / u[i]]; H[x_] := 0 /; x < 0;
  H[x_] := 1 /; x ≥ 0;
            n-1
  f[x_] := Σ  f[x, i] * (H[x - X[i - 1]] - H[x - X[i]]) +
           i=1
    f[x, n] * H[x - X[n - 1]]]
```

Fig. 2. The *Splajn* procedure written in the *Mathematica* program (prepared by author)

Mathematica notebooks ensure a sophisticated environment for creating technical documents, particularly if we want to merge your work with existing material in *TeX*.

We export the notebook from the *Matematica* program to *Microsoft Word* as a *Metafile* or *Bitmap*.

2.2. Natural Spline

The *NaturalSpline* procedure written by John H. Mathews, Ph.D. Emeritus Prof. of Mathematics, California State University, Fullerton has an open source code on the internet site [13].

The *NaturalSpline* procedure solves (7) based on the tridiagonal linear system theory.

```
In[1]:= NaturalSpline1[XY0_] := Module[{XY = XY0},
    Differences := Module[{k}, n = Length[XY] - 1; X = Transpose[XY][[1]];
      Y = Transpose[XY][[2]];
      h = d = Table[0, {n}]; m = Table[0, {n + 1}]; a = b = c = v = Table[0, {n - 1}];
      s = Table[0, {n}, {4}]; h[[1]] = X[[2]] - X[[1]];
      d[[1]] = (Y[[2]] - Y[[1]])/h[[1]]; For[k = 2, k ≤ n, k++, h[[k]] = X[[k+1]] - X[[k]];
       d[[k]] = (Y[[k+1]] - Y[[k]])/h[[k]]; a[[k-1]] = h[[k]]; b[[k-1]] = 2 (h[[k-1]] + h[[k]]); c[[k-1]] = h[[k]];
       v[[k-1]] = 6 (d[[k]] - d[[k-1]])];];
    TriDiagonal := Module[{k, t}, m[[1]] = 0; m[[n+1]] = 0;
      For[k = 2, k ≤ n - 1, k++, t = a[[k-1]]/b[[k-1]]; b[[k]] = b[[k]] - t c[[k-1]];
       v[[k]] = v[[k]] - t v[[k-1]];]; m[[n]] = v[[n-1]]/b[[n-1]];
      For[k = n - 2, 1 ≤ k, k--, m[[k+1]] = (v[[k]] - c[[k]] m[[k+2]])/b[[k]];];];
    ComputeCoeff := Module[{k}, For[k = 1, k ≤ n, k++, s[[k,1]] = Y[[k]];
       s[[k,2]] = d[[k]] - 1/6 h[[k]] (2 m[[k]] + m[[k+1]]); s[[k,3]] = m[[k]]/2; s[[k,4]] = (m[[k+1]] - m[[k]])/(6 h[[k]]);];];
    CS[t_] := Module[{j}, For[j = 1, j ≤ n, j++, If[X[[j]] ≤ t && t < X[[j+1]], k = j];];
      If[t < X[[1]], k = 1]; If[X[[n+1]] ≤ t, k = n]; w = t - X[[k]];
      Return[((s[[k,4]] w + s[[k,3]]) w + s[[k,2]]) w + s[[k,1]]];];
    (*The main part of they program starts here*);
    Differences; TriDiagonal; ComputeCoeff];
```

Fig. 3. The *NaturalSpline* procedure written in the *Mathematica* program (source: Internet [13])

2.3. Basic concept of orthogonal polynomials

We will now discuss interpolation using orthogonal polynomials [5]

Def. Sequence of function $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x)$ is called the orthogonal on the set of points $x_0, \ldots, xn$ if:

$$\sum_{i=0}^{n} \varphi_j(x_i)\varphi_k(x_i) = 0 \, dla \, j \neq k$$

The following relations can be proven [5]:

$$\varphi_{j+1}(x) = (x - \alpha_{j+1})\varphi_j(x) - \beta_j \, \varphi_{j-1}(x) \quad dla \, j = 0,1,\ldots,n$$

$$\varphi_0(x)=1, \quad \varphi_{-1}(x)=0,$$

where the constants α_{j+1} i β_j are defined by the formulas:

$$\beta_j=\frac{\sum_{i=0}^{n}\varphi_j^2(x_i)}{\sum_{i=0}^{n}\varphi_{j-1}^2(x_i)}, \quad \alpha_{j+1}=\frac{\sum_{i=0}^{n}x_i\varphi_j^2(x_i)}{\sum_{i=0}^{n}\varphi_j^2(x_i)}$$

finally the function $f(x)$ has the form:

$$f(x)=\sum_{k=0}^{n}b_k\varphi_k(x)$$

where:

$$b_k=\frac{C_k}{S_k}, \; C_k=\sum_{i=1}^{n}y_i\phi_k(x_i), \; S_k=\sum_{i=0}^{n}\phi_k^2(x_i)$$

We can now write the procedure called *WielOrt1* where the input parameters are data points: (x_0,y_0), (x_1,y_1), ..., (x_n,y_n) and at the output of that procedure, we obtain the function $f(x)$ and its graph. The basic procedures we extend by adding the graphical and numerical instructions.

```
In[1]:= << WielOrt1
In[2]:= daneW = {{1, 3}, {2, 4}, {4, 6}, {5, 8}, {6, 4}};
In[3]:= WielOrt1[daneW]
```

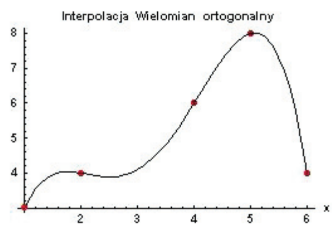


Fig. 4. The execution procedure *WielOrt1 [daneW]* (prepared by author)

3. VERIFICATION AND SELECTION OF THE INTERPOLATION METHODS

We will verify three procedures: *WielOrt1*, *Splajn1* and *NaturalSpline1* and the corresponding them errors we call as: *errW*, *errS* and *errNS*. A comparative analysis of these procedures was conducted by four author-prepared methods. The examples of verification were chosen in such a manner that it was possible to compare the identified interpolating function ***f(x)*** created by author with the known function *g(x)*.

As a measure of the method error, the following expression was applied.

$$err = \frac{1}{b-a}\int_{a}^{b}[f(x)-g(x)]^2\,dx$$

3.1. FTV – Function Table Verification Method

In the first method verification of procedure the nodal points were introduced as the coordinates from the formula of the function.

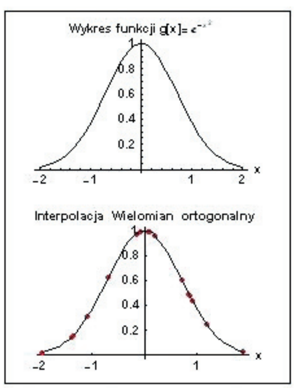
Example: The *LosujDane* procedure generating random input data was written for the verification of all procedures.

```
LosujDane[a_, b_, ld_] :=
 Module[{}, dane = Table[Random[Real, {a, b}], {ld}];
  << SortujListe; dane1 = SortujListe[dane];
  dane2 = Table[{dane1[[i]], g[dane1[[i]]]}, {i, 1, ld}];
  Print["dane2=", dane2];
  Print["dane2 wygenerowane losowo dla funkcji g[x]= ",
   g[x], " z przedziału <", a, ",", b, ">"]]

g[x_] = e^-x^2; LosujDane[-2, 2, 10]

dane2={{-1.73421, 0.0494152}, {-0.918374, 0.430241},
  {-0.905201, 0.440701}, {-0.649464, 0.655863},
  {-0.496048, 0.781873}, {0.444826, 0.820476},
  {0.494177, 0.783323}, {0.573378, 0.719814},
  {0.953878, 0.402571}, {1.87649, 0.0295634}}

dane2 wygenerowane losowo dla funkcji g[x]=
 e^-x^2 z przedziału <-2,2>
```

Fig. 5. Random values of the function *g(x)* (prepared by author)

The comparison graphs of procedures for [*dane2*] with function g[*x*]

```
In[6]:= Show[GraphicsArray[{{wykres}, {wielort}}], Frame → True];
```

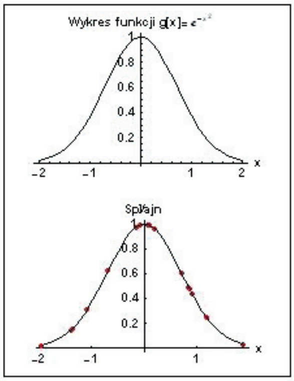



```
In[7]:= Print["errW=", 1/(X[n] - X[0]) * NIntegrate[(f[x] - g[x])^2, {x, X[0], X[n]}]]

errW=2.4299 × 10^-7
```

Fig. 6. The graphs and *errW* for the *WielOrt1*procedure *[dane2]* (prepared by author)

For the *Splajn1* procedure we have:

```
In[10]:= Show[GraphicsArray[{{wykres}, {splajn}}], Frame → True];
```



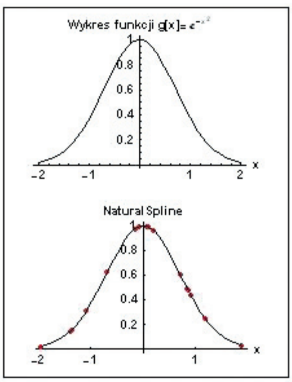
In[11]:= Print$\left[\text{"errS="}, \dfrac{1}{X[n]-X[0]} * \text{NIntegrate}\left[(f[x]-g[x])^2, \{x, X[0], X[n]\}\right]\right]$

errS=8.0908×10^{-6}

Fig. 7. The graphs and *errS* for the *Splajn[dane2]* procedure (prepared by author)

and for the *NaturalSpline1* procedure, we obtained:

```
In[14]:= Show[GraphicsArray[{{wykres}, {NS}}], Frame → True];
```



In[15]:= Print$\left[\text{"errNS="}, \dfrac{1}{X_{[[n+1]]}-X_{[[1]]}} * \text{NIntegrate}\left[(CS[x]-g[x])^2, \{x, X_{[[1]]}, X_{[[n+1]]}\}\right]\right]$

errNS=8.0908×10^{-6}

Fig. 8. Comparison of graphs and *errNS* for the *NaturalSpline1[dane2]* procedure (prepared by author)

Other functions have been tested (the data determined) and the results are shown in Table 1:

Table 1. Statement of errors for the **FTV** method

Function; Interval	Splajn	NaturalSpline	WielOrt
$e^{-x\hat{}2}$; <-2,2> random	$8.0908*10^{-6}$	$8.0908*10^{-6}$	$2.4299*10^{-7}$
$(1-x^2)^{1/2}$; <0,1>	0.000454465	0.000454465	0.000117545
Sin(x) ; <0,1.5>	$3.79068*10^{-6}$	$3.79068*10^{-6}$	$2.74535*10^{-15}$
x^2 for $x < 1$ $2 - x$ for $x >= 1$; <0,2>	0.00027388	0.00027388	0.0915378

The *WielOrt* procedure proved to be a better approximation than the *Splajn* and *Natural Splajn* procedures. In the last row, where the function is defined by two formulae, the *WielOrt* procedure gives an error that is approximately 300 times larger.

3.2. CCM – Coordinate Conversion Verification Method

For the second verification method, the *ZmienWsp* procedure was written; this transforms the graphic coordinates drawn to real coordinates. After delivering real coordinates to suitable procedures, we can formulate an analytical description of the curve and perform a verification.

Example:

In publications or websites we can often see a graph of function, but we do not know the value of this function. We will show in three steps, how we can find a formula of function.

Step. 1: We import a scanned graph of function to the *Mathematica* program

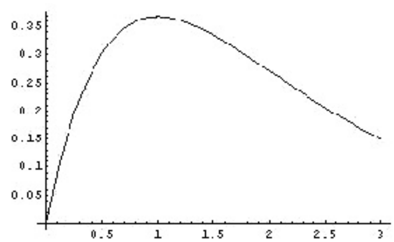


Fig. 9. Graph of unknown function *g[x]* (*prepared by author)*

Step. 2: We read and write coordinates of the screen with Fig. 9:

```
In[3]:= WspEkran = {{26.2174, 17.6913}, {39.9499, 73.6776}, {66.3586, 142.34},
         {105.443, 171.918}, {147.697, 161.354}, {183.613, 138.115},
         {236.43, 104.312}, {278.684, 81.072}};
```

Fig. 10. Coordinates of screen function *g[x]*

We read the coordinates: *(xB, yB)* of the beginning at the graph and coordinates: *(xE, yE)* of the end at the graph and corresponding them coordinates of screen are: *(xBS, yBS)* and *(xES, yES).* Now we create a procedure that uses linear interpolation and will transform all screen coordinates into real coordinates.

```
In[4]:= ZmienWsp[lista_] := Module[{xB = 0, yB = 0, xE = 3, yE = 0.15}, nz = Length[lista];
    xBS = 26.2174; yBS = 17.6913; xES = 278.684; yES = 81.072;
    fz[t_, u_] := (u[[4]] - u[[2]])/(u[[3]] - u[[1]]) * (t - u[[1]]) + u[[2]];
    ux = {xBS, xB, xES, xE}; uy = {yBS, yB, yES, yE};
    WspReal := Table[{fz[WspEkran[[i, 1]], ux], fz[WspEkran[[i, 2]], uy]}, {i, nz}];
    Print["WspReal=", WspReal]]
```

Fig. 11. The *ZmienWsp* procedure *(prepared by author)*

After executing this procedure with *WspEkran* parameters, we obtain real coordinates:

```
In[5]:= ZmienWsp[WspEkran]

    WspReal={{0., 0.}, {0.16318, 0.1325},
      {0.476988, 0.295}, {0.941419, 0.365001}, {1.44351, 0.339999},
      {1.87029, 0.285001}, {2.49791, 0.205001}, {3., 0.15}}
```

Fig. 12. Real coordinates

Step. 3: We execute procedure *WielOrt1[WspReal]* and we obtain function *f[x]* and its graph:

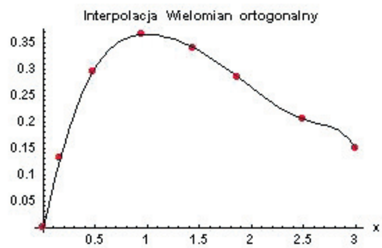
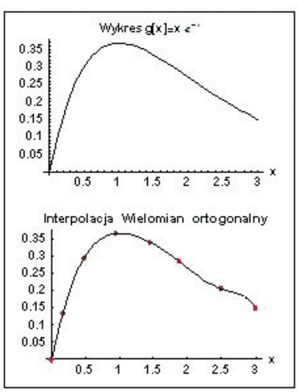


Fig. 13. Graph interpolation function (prepared by author)

Comparing Figs 13&9, we see that they are similar, author knows function: $g[x]=x\ e^{-x}$; therefore, we can find the estimated error using this method:

```
In[21]:= Show[GraphicsArray[{{funkcja}, {wielort}}], Frame → True];
```




```
In[22]:= Print["errW =", 1/3 * NIntegrate[(f[x] - g[x])^2, {x, 0, 3}]]

        errW =0.0000356719
```

Fig. 14. Comparison graph function *g[x]* with *WielOrt1* and the *errW* (prepared by author)

Following analogical steps we have for *Splajn1* and *NaturalSpline1*.

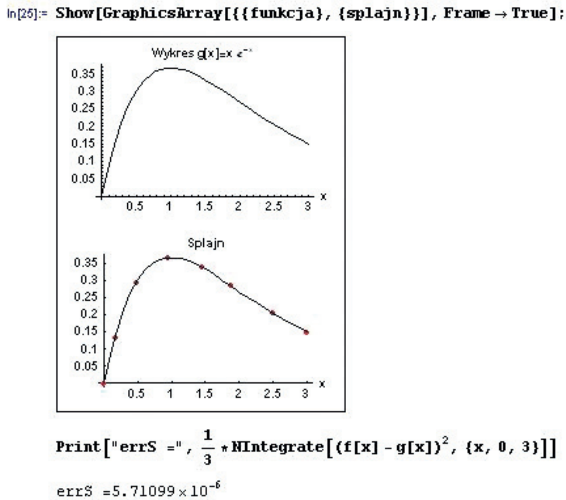


Fig. 15. Comparison graph function with *Splajn*1 and the *errS* (prepared by author)

The compatibility is near perfect.

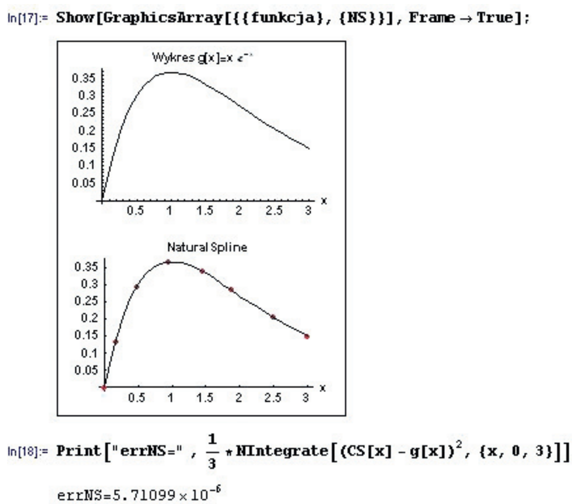


Fig. 16. Comparison graph function with *NaturalSpline1* and *errNS* (prepared by author)

In the same way, other functions have been tested and the results are presented in Table 2.

Table 2. Statement of errors for the **CCM** method

Function; Interval	Splajn1	NaturalSpline1	WielOrt1
$x\,e^{-x}$; <0,3>	$5.71099*10^{-6}$	$5.71099*10^{-6}$	$35.6798*10^{-6}$
Cos(x) ; <0,1>	0.000323145	0.000323145	0.00046454
x Log(x); <0.01,1.2>	0.0940895	0.0940895	0.0941283

The *NaturalSpline* and *Splajn* procedures give identical errors. Based on Tables 1&2, the *WielOrt1* procedure is rejected.

3.3. ALC – The Arc Length of the Curve -Verification Method

The first derivative is necessary for example to calculate the arc length of curve, the surface area, and to evaluate the line integral $\oint_C$, where C is the given curve.

Adding formulas (2) and (3) to the *Spline1* procedure, we obtain the new procedure: *SplajnPoch2.*

We now create the *SplajnPoch2* procedure. After executing *SplajnPoch2[dane]*, we obtain: dane = {{-3, 9}, {-1, 1}{0, 0.7}, {1, 1}, {2, 4}, {3, 3}}.

Example:

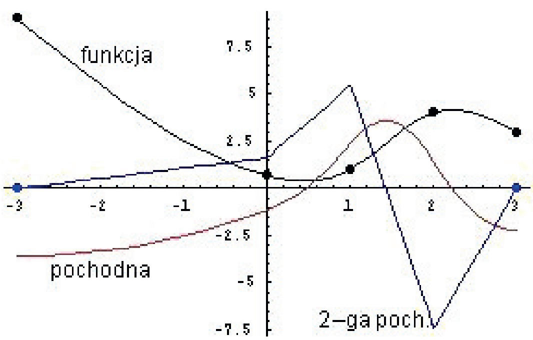
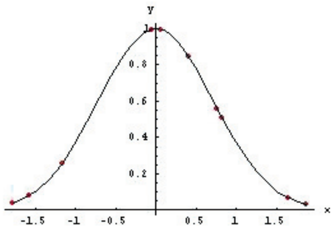


Fig. 17. Graph of the function, first and second derivatives (prepared by author)

The *NaturalSpline* procedure does not exactly calculate the first derivative, it requires a large modification to the inside of the procedure; however, this is not the purpose of this paper.

Now we can execute the *SplajnDluku* procedure and compare the result with the length calculated from the formula. Nodal points we get randomly.

```
In[1]:= << LosujDane

In[2]:= g[x_] = e^-x^2; LosujDane[-2, 2, 10]

    dane2={{-1.78761, 0.0409445}, {-1.57445, 0.0838358},
      {-1.16189, 0.25924}, {-0.0579191, 0.996651},
      {0.0575693, 0.996691}, {0.411726, 0.844071}, {0.761755, 0.559747},
      {0.820024, 0.510462}, {1.64186, 0.067495}, {1.87224, 0.0300384}}

    dane2 wygenerowane losowo dla funkcji g[x]= e^-x^2 z przedziału <-2,2>

In[3]:= << SplajnDluku

In[4]:= SplajnDluku[dane2]
```




```
    długość łuku=4.21745

In[5]:= Print["dł.ze wzoru=", NIntegrate[√(1 + g'[x]^2), {x, X[0], X[n]}]]

    dł.ze wzoru=4.2191

In[6]:= {X[0], X[n]}

Out[6]= {-1.78761, 1.87224}
```

Fig. 18. Comparison of arc length the curve (prepared by author)

Table 3. Comparison of the method results ALC

Function;Interval	SplajnDluku	Formula
$e^{-x^{\wedge}2}$; <-1.78761;1.87224> Random	4.21745	4.2191
$x\,e^{-x}$; <0,3>	3.10959	3.10978
Sin[x]; <0,6>	7.24261	7.24256

The compatibility is near perfect.

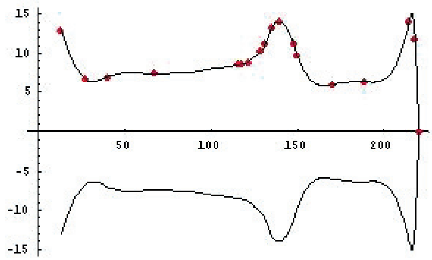
3.4. RO – Real Object Verification Method

The next method of verification was based on a real object for example, element of the bell, the heart of Zygmunt's Bell. The calculated weight of the heart was compared with its known weight.

Example:

In the *SplajnObjPow* procedure, the input parameters are data points, the output parameters is graphs and volume of the solid formed by the revolution of the curve y = *f[x]* around *x*-axis.

```
In[1]:= << Graphics`SurfaceOfRevolution`

In[2]:= << SplajnObjPow

In[3]:= DaneSerce = {{13.26, 12.94}, {27.37, 6.72}, {40, 6.98},
          {66.98, 7.41}, {115.51, 8.54}, {117, 8.58},
          {121.7, 8.9}, {128.4, 10.29}, {131.1, 11.18},
          {135, 13.25}, {139.11, 14}, {147.98, 11.18},
          {150, 9.82}, {170.37, 6}, {188.85, 6.36}, {214.54, 14},
          {218, 11.86}, {220, 0}};

In[4]:= SplajnObjPow[DaneSerce]
```




```
Pole pow. bryły obrotowej =11704.6[cm^2]

Objętość bryły obrotowej =47371.7[cm^3]
```

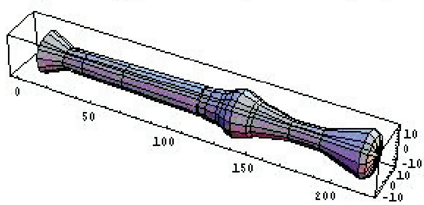


Fig. 19. Result of the *SplajnObjPow[DaneSerce]* procedure (prepared by author)

These calculations were made without taking into account the handle of the bell heart. The mass of the handle was estimated to be 20 kg. The density of the heart is unknown; there are some admi*x*tures: phosphorus, sulphur, etc. According to the accessible data, the heart mass is about 350 kg. The specific mass of the heart is estimated as 7.7 g/cm^3.

Calculated mass of the bell: mass = 47371.7[cm^3]*7.7[g/cm^3]/1000+20[kg] = 384.762 [kg], error of calculations = (384.762 –350)/350*100% = 9.9 %

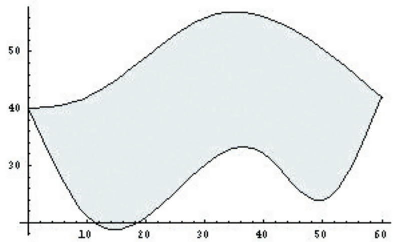
Inference:

The verification of three procedures was performed: Spline Fig. 2., *NaturalSpline* Fig. 3. and *WielOrt* (Table 1& 2). Procedure *Wielort1 (large calculated error),* procedure *NaturalSpline (it is not calculate the derivative)* were rejected.

4. APPLICATIONS OF *SPLINE METHOD*

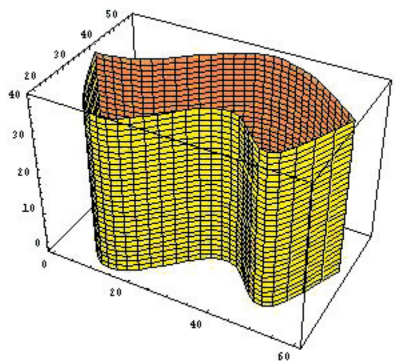
Example:

```
In[3]:= dane1 = {{0, 40}, {10, 42}, {30, 56}, {60, 42}};
        dane2 = {{0, 40}, {10, 21}, {40, 32}, {50, 24}, {60, 42}};

In[4]:= SplajnDzialkaSciana[dane1, dane2, 40, 0.2]
```



pole działki=1305.3 [m²]

obwód działki=161.347 [m]



pole powierzcni ściany = 6453.87 m²

objętość ściany = 1290.77 m³

Fig. 20. Result of the *SplajnDzialkaSciana[dane1, dane2, 40, 0.2]* procedure (prepared by author)

Example:

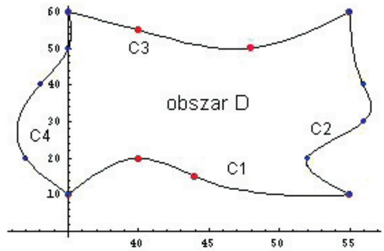
Let us consider a more general case in which the contour C of area D is described by the union of four curves: $C = C_1 \cup C_2 \cup C_3 \cup C_4$. We calculate the area field as a line integral:

$$\frac{1}{2}\oint_C x\,dy - y\,dx$$

where the direction of circulation of contour C is chosen as anticlockwise.

We create a new procedure: *Splajn4G* After execution of this procedure with parameters *[daneXd]*, *[daneXg]*, *[daneYl]*, *[daneYp]*, we obtain:

```
In[1]:= << Splajn4G

In[2]:= daneXd = {{35, 10}, {40, 20}, {44, 15}, {55, 10}};
        daneXg = {{35, 60}, {40, 55}, {48, 50}, {55, 60}};
        daneYl = {{10, 35}, {20, 32}, {40, 33}, {50, 35}, {60, 35}};
        daneYp = {{10, 55}, {20, 52}, {30, 56}, {40, 56}, {60, 55}};

In[3]:= Splajn4G[daneXd, daneXg, daneYl, daneYp]
```




```
obwód obszaru = 162.92 [m]

pole obszaru =  881.215 [m²]
```

Fig. 21. Perimeter and area of the region D (prepared by author)

Example:

We construct the circle with centre on a curve. The circle moves after curve in normal plane to curve. In the *SplajnRurociag* procedure, the input parameter are: the data points and the radius of the pipeline; the output parameters are: the graph and the surface area of the pipeline.

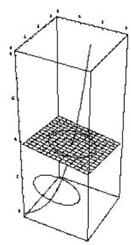


Fig. 22. *Illustrative figure* (prepared by author)

```
In[1]:= dane = {{0, 6}, {1, 2}, {3, 5}, {7, 9}, {8, 6}, {15, 8}, {17, 5}};

In[2]:= SplajnRurociag[lista_, promien_] :=
          Module[{}, << SplajnPR; SplajnPR[lista, promien]; << PowRurkowa;
           Show[PowRurkowa[{x, f[x], 0}, {x, X[0], X[n], 40}, 9, 0.7]]];

In[3]:= SplajnRurociag[dane, 0.7]

        Pole powierzhni rurkowej=148.399
```

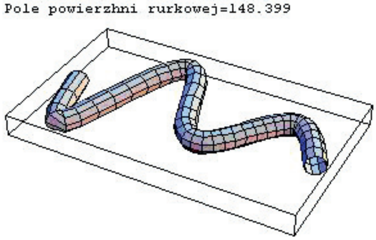


Fig. 23. Surface area of the pipeline (prepared by author)

5. APPLICATION IN MECHNICS

E*x*ample:

Moment of inertia relative to the *x* a*x*is of the heart

Changing to cylindrical coordinates, we have:

$$B_x = \mu \int_0^{2\pi} d\varphi \int_{X[0]}^{X[n]} dx \int_0^{f[x]} \rho^3 d\rho$$

and finally:

$$B_x = \frac{1}{2}\pi\mu \int_{X[0]}^{X[n]} f^4[x]dx$$

```
In[4]:= Print["Bx=", 1/2 π μ NIntegrate[f[x]^4, {x, X[0], X[n]}], " [g*cm^2]"]

        Bx=2.3285×10^6 μ [g*cm^2]
```

Fig. 24. Moment of inertia relative to the *x* a*x*is of the heart (prepared by author)

6. CONCLUSION

On the basis of studies of the available literature, the author has developed their own method for calculating parameters curvilinear obiects: *The Spline Method.*

The four verification methods presented in this study show the correctness of the calculations. With the support of the prepared procedures, can be calculate the needed parameters in the *Mathematica* program. *The Spline Method* enables the precise definition of the necessary object parameters and their optimisation. This creates the possibility of modelling objects and enables quick parameter changes in the design process. This can be implemented in various fields of science, also beyond the framework of engineering. In summary, the state of knowledge and research presented in this article, supplemented with the results of our own research, still needs to be developed through further research that will enable the systematic development of *The Spline Method* and the calculation of the parameters of curvilinear objects.

References

[1] Ahlberg J.H., Nilson E.N., Walsh J.L., *The Theory of Splines and Their Applications*, Academic press New York–London 1967.

[2] Drwal G., Grzymkowski R., Kapusta A., Słota D., *Mathematica 4,* Wydawnictwo Pracowni Komputerowej J. Skalmierskiego, Gliwice 2000.

[3] Drwal G., Grzymkowski R., Kapusta A., Słota D., *Mathematica programowanie i zastosowania*, Wydawnictwo Pracowni Komputerowej J. Skalmierskiego, Gliwice 1995.

[4] Dryja M., Jankowscy J. i M., *Przegląd metod i algorytmów numerycznych Część 2,* WNT, Warszawa 1998.

[5] Fortuna Z., Macukow D., Wasowski J., *Metody numeryczne*, WNT, Warszawa, 2005.

[6] Grzymkowski R., Kapusta A., Słota D., *Mathematica narzędzie inżyniera*, Wydawnictwo Pracowni Komputerowej J. Skalmierskiego, Gliwice 1994.

[7] Jankowscy J. i M., *Przegląd metod i algorytmów numerycznych Część 1,* WNT, Warszawa 1998.

[8] Knott Gary D., *Interpolating Cubic Splines*, Birkhauser, Boston Basel, Berlin 2000.

[9] Leja F., *Rachunek różniczkowy i całkowy ze wstępem do równań różniczkowych*, WNT, Warszawa 2008.

[10] Makowska A., *Metoda kalkulacji kosztów krzywoliniowych obiektów budowlanych z zastosowaniem współczynnika trudności*, praca doktorska, Kraków 2009.

[11] Trajdos-Wróbel T., *Matematyka dla Inżynierów,* WNT, Warszawa 1974.

[12] Wolfram S., *The Mathematica book,* Cambridge University Press, 1999.

[13] http://mathfaculty.fullerton.edu/mathews/n2003/CubicSplinesMod.html (access: 15.04.2018).

If you want to quote this article, its proper bibliographic entry is as follow: Makowska A., *Verification of the spline method and its application to curvilinear objects,* Technical Transactions, Vol. 1/2019, pp. 113–132.