www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 11 November 2017, Page No. 23107-23191
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i11.17

A Brief Review On The Application Of Polynomial Splines To Quadrature In One And Two Dimensions

H.T.Rathod[a*], K.V.Vijayakumar[b], A.S.Hariprasad[c], H. Y. Shrivalli[d]
[a*]Department of Mathematics, Central College Campus, Bangalore University,
Bangalore -560001, Karnataka State, India.
[b] Department of Mathematics,B.M.S.Institute of Technology,Avalahalli,
Bangalore-560064, Karnataka State, India.
[c] Department of Mathematics,Rajarajeswari College of Engineering,
Bangalore -560074, Karnataka State, India.
[d] Department of Mathematics,
B.M.S college of Engineering,
Basavanagudi, Bangalore -560019,India

Abstract

We first consider the integration of arbitrary functions over a linear polygon with *n*-edges say P_n in the Cartesian 2-space. A scheme based on the use of classical Green's theorem is proposed. In this scheme, the integration over P_n is obtained as a sum of *n*-one dimensional integrals over the oriented edges of P_n. Gauss Legendre quadrature rules are further applied to develop the numerical integration scheme.

We next consider the polynomial spline interpolatios.The term spline refers to an instrument used in drafting. It is a thin, flexible wooden or plastic tool that is passed through given data points and defines a smooth curve in between. Spline interpolation is a very powerful and widely used method and has many applications in numerical differentiation and integration, solution of boundary value problems, computer animation, computer graphics and robot path/ trajectory planning.

This study proposes a cubic spline which interpolates at first and the last knots and the two points located at the trisections between the knots. The performance of the new cubic spline is found to be superior to the existing cubic splines studied in the literature.

We have also presented the applications of proposed cubic spline to integral function approximations and the numerical integration over curved domains in 2-space. Applications to integral function approximations are illustrated for the indefinite integral of Runge function, logarithmic function and normal distribution. This application is very useful in the construction of table of integrals. As an application of numerical integration over curved domains the computation of some typical test integrals over a lunar model is considered. We also compare the numerical approximations of the proposed new cubic spline with the standard cubic splines (natural, clamped and not a knot)

Key Words: Runge's phenomina, Spline interpolations, Integral function Approximations ,Numerical Integrations over polygonal and curved domains,Green's Theorem,Gauss Legendre quadrature, Standard cubic splines,New cubic spline

1.0 integration of arbitrary functions over a linear polygon

Gaussian quadratures are considered as classical tool of numerical integration, and possess several desirable features such as positivity of weights and they can evaluate the (2*n*-1)th order polynomials with *n*-Gaussian integration points, mostly used in view of the accuracy and the efficiency of calculation. However, the integrands in practical situations are not always polynomials but rational or irrational expressions, which the Gaussian quadrature schemes cannot evaluate exactly. Quadrature theory in one dimension is relatively complete, with Gaussian quadrature being optimal and their construction is well understood [1-2]*.These references* can be found in **References-I.**

In higher dimensions, however the situation is more complicated. The domains of applications in R^2 or R^3 come in an infinite variety of shapes each with its own complexity. As would be expected one might try to generalize the one dimensional quadrature rules to higher dimensions using tensor products. However the considered opinion of researchers is that Gaussian quadrature in higher dimensions has to be studied separately for different geometries of integration regions, and each region should require a different set of rules.

The geometry that has been studied most extensively in two dimensions is the triangle [3-5] and the convex quadrilateral [6]. Further when elements with five vertices or more are used in polygonal finite element applications, one has to partition the element into triangular or quadrilateral elements for the sake of numerical integration. The literature on quadrature over polygons

is not very extensive, despite the fact that polygons (polyhedron) are at the core of computational geometry. However some recent studies have made remarkable progress on the construction of quadrature rules over regular polygons on the unit circle (see references [7-9]) and also over polygons and polyhedra [10, 11]. In a recent works [12, 13], the Green's integral formula is proposed to produce Gauss-like quadrature rules that can integrate the high order bivariate polynomials over polygonal domains. Historically speaking, the integration formulas for multiple integrals have always been of great interest in computer applications. Computation of mass properties of both plane and space objects is discussed by Wesley [14] and Mortenson [15]. A good overview of various methods for evaluating volume (triple) integrals in this context is given by Lee and Requicha [16]. They have further observed that most computational studies in multiple integrations deal with calculations over very simple domains, such as cube, sphere, square, regular polygons etc, while the integrating function is very complicated. On the contrary, in most engineering applications, the converse problem usually arises. In such problems the integration domain may have a nonconvex shape and the function inside the integral sign may also appear in various forms. In this context, several authors have contributed to the development of explicit integration formulas over a linear polyhedron when the integrand is a trivariate polynomial [17-19]. Timmer and Stern [17] discussed a theoretical approach to the evaluation of volume integral by transforming it into a sum of surface integrals over the boundary of the integration domain. Lie and Kajiya [18] presented an outline of a closed formula of volume integration for a tetrahedron and suggested that volume integration for a linear polyhedron can be obtained by decomposing it into a set of solid tetrahedrons. Cattani and Paoluzzi [19-20] have obtained explicit finite integration formulas for integrals of monomials over plane polygons and space polyhedra. Rathod and Govind Rao [21-22], Rathod and Hiremath [23-24] addressed some of these problems over plane polygons, tetrahedron and hexahedron. Bernardini [25] and Rathod and Govinda Rao [26] have further generalised these integration methods to compute integrals of *n*-variate polynomials over linear polyhedra in n-dimensional space.

In the present paper, it is our purpose to apply these recent studies to develop some numerical integration formulas over plane polygon. We have developed the numerical schemes using the MATLAB software [27-31] for all examples tested in this work. In the present study , we have developed the numerical integration formulas which mainly follow the concepts developed in our earlier work [20-24]. It is well known that one of the remarkable theorems of multivariate calculus, Greens theorem for the plane region is of immense value for the computation of multiple integrals over a plane polygon. This has been used by several authors referred in this article and also by Sommariva and Vianello [12] and Dasgupta [13] as

$$\iint_{\Omega} f(x,y)dxdy = \oint_{\partial\Omega} F(x,y)dy, \qquad F(x,y) = \int f(x,y)dx$$

(*f* being continuous on a domain Ω with piecewise smooth boundary $\partial\Omega$ described counterclockwise) gives us in principle an appealing tool for numerical cubature, since it transforms a two dimensional integration problem into a 1-dimensional integration problem. Its practical use, however, requires the knowledge of a primitive of the integrand. However, this restriction can be overcome by proper application of the available theoretical knowledge on integration [12]. In this paper we have shown by the application of Greens theorem that integration over the polygonal domain with *n*-edges can be obtained as a sum of n-one dimensional integrals over the oriented edges of P_n. We have illustrated this for the convex and non-convex polygonal domains.

2.0 Boundary Integration:

Let π_{xy} be a simple polygon in the *xy*-plane. We want to evaluate the following integral

$$II_{\pi_{xy}} \overset{def}{=} \iint_{\pi_{xy}} f(x,y)dxdy \quad \text{.....(1)}$$

Theorem 1: The integral $II_{\pi_{xy}}$ over a simple polygon with *N*-oriented edges $l_{ik}(k=i+1), i=1,2,3,...,N)$ each with end point (x_i , y_i) and (x_k, y_k) in the *xy*-plane is expressible as

$$II_{\pi_{xy}} = \sum_{i=1}^{N} \oint_{lik} \Phi(x,y)dy \,, \Phi(x,y) = \int f(x,y)dx \text{.....(2)}$$

with $(x_{N+1}, y_{N+1}) = (x_1, y_1)$.

Proof: We have from equation (1)

$$II_{\pi_{xy}} \overset{def}{=} \iint_{\pi_{xy}} f(x,y)dxdy$$

$$\overset{def}{=} \iint_{\pi_{xy}} \frac{\partial\Phi(x,y)}{\partial x}dxdy \quad \text{where} \quad \Phi(x,y) = \int f(x,y)dx$$

$$= \oint_{\partial\pi_{xy}} \Phi(x,y)dy \quad \text{.....(3)}$$

(On using Greens theorem with $\partial\pi_{xy}$ =boundary of π_{xy})

$$= \sum_{i=1}^{N} \oint_{lik} \Phi(x,y)dy \text{.....(4)}$$

This proves the theorem.

Lemma 1: The *x*- primitive is given by the integral

$$\int_{\alpha}^{x} f(x,y)\,dx \quad \text{.....(5)}$$

also satisfies the above theorem-1 of Green`s, when α is fixed and $x_i \neq \alpha, i = 1,2,...,(N+1)$ when x_i s are the *x*-coordinates of the vertices (x_i, y_i) for the polygonal boundary and thus it implies that we can also have

$$\Phi(x,y) = \int_{\alpha}^{x} f(u,y)\,du \quad \text{.....(6)}$$

Proof: Let us consider the sum

$$\sum_{i=1}^{N} \oint_{l_{ik}} \left(\int_{\alpha}^{x} f(u,y)\,du\right)dy = \sum_{i=1}^{N} \oint_{l_{ik}} \left(\left(\int f(u,y)\,du\right)_{u=x} - \left(\int f(u,y)\,du\right)_{u=\alpha}\right)dy$$

$$= \sum_{i=1}^{N} \oint_{l_{ik}} \left(\int f(x,y)\,dx + \psi(y)\right)dy \quad \text{.....(7)}$$

Where $\psi(y)$ is an arbitrary function of *y* and clearly

$$\sum_{i=1}^{N} \oint_{l_{ik}} \psi(y)\,dy = \sum_{i=1}^{N} \left[\chi(y)\right]_{y_i}^{y_{i+1}}, \quad \chi(y) = \int \psi\,dy \quad \because y_k = y_{i+1}$$

$$= \sum_{i=1}^{N} \left[\chi(y_{i+1}) - \chi(y_i)\right] = -\chi(y_1) + \chi(y_{N+1}) = 0 \quad \text{.....(8)}$$

Since $y_{N+1} = y_1$, for a closed boundary

From eqns (7) and (8), it is clear that

$$\sum_{i=1}^{N} \oint_{l_{ik}} \left(\int_{\alpha}^{x} f(u,y)\,du\right)dy = \sum_{i=1}^{N} \oint_{l_{ik}} \left(\int f(x,y)\,dx\right)dy \quad \text{.....(.9)}$$

$$\sum_{i=1}^{N} \oint_{l_{ik}} \Phi(x,y)\,dy = II_{\pi_{xy}}$$

This completes the proof of lemma 1.

Remark 1: Thus we have the alternative definition

$$II_{\pi_{xy}} \overset{def}{=} \sum_{i=1}^{N} \oint_{l_{ik}} \left(\int f(x,y)\,dx\right)dy$$

$$= \sum_{i=1}^{N} \oint_{l_{ik}} \left(\int_{\alpha}^{x} f(x,y)\,dx\right)dy \quad \text{..... (10)}$$

We may also note that if for some *i* (*i* =1,2,……*N*)

$$\left(-\int\left(\int f(u,y)\,du\right)dy\right)_{u=\alpha=x_i, y=y_i} + \left(\left(\int f(x,y)\,dx\right)\right)_{(x=x_i, y=y_i)} = 0 \quad \text{..... (11)}$$

and this leads to error in the computation of $II_{\pi_{xy}}$ and hence ,we should not have $\alpha = x_i$.

Eqn (10) also serves as a useful hint for the computation of *x*-primitive of the integrand, which is necessary for its practical application in the derivation of numerical integration formulas.

Remark 2: Numerical approximation to *x*-primitive of *f (x, y*) may be now found by application of Gauss Legendre quadrature rules as explained below;

We have proved in Lemma 1 that

$$\Phi(x,y) = \int_{\alpha}^{x} f(u,y)\,du \text{ , ... (12) for a closed boundary}$$

(Where $\alpha \neq x_i$, the *x*-coordinate of vertex (x_i, y_i) for all *i* =1(1) (*N*+1))

Let us choose the substitution $s = \left(u - \frac{(x+\alpha)}{2}\right) \div \frac{(x-\alpha)}{2}$ ..... (13)

$$u = \left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right) \quad \text{.....(14)}$$

Substitution of equation (13) in (12) gives us

$$\Phi(x,y) = \left(\frac{x-\alpha}{2}\right)\int_{-1}^{1} f\left(\left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right), y\right)ds \quad \text{..... (15)}$$3.0 3.0 3.0 3.0 **Numerical Integration Rules:**

It is well known that the *n*-point Gauss Legendre quadrature rule

$$\int_{-1}^{1} F(x)dx = \sum_{i=1}^{n} W_i^n \, F(X_i^n) \quad \text{.....(16)}$$

has the highest possible precision degree *(2n-1)* and is analytically exact for polynomials of degree *(2n-1)*, where $X_i^n s$ are the zeros of Legendre polynomial $P_n(X)$ and W_i^n are the corresponding weight coefficients.

We also note that the parametric equation of a straight line joining the points (X_i, Y_i) and (X_k, Y_k) in *XY*-space can be written as

$$X(t) = X_i + (X_k - X_i)t \;,\; Y(t) = Y_i + (Y_k - Y_i)t, \quad (0 \le t \le 1) \text{.....(17)}$$

3.1 **Boundary integration along straight edges:**

Application of the above result in eqn (17) to eqn (4) gives us for the *(x, y)* space.

$$I\,I_{\pi_{xy}} = \sum_{i=1}^{N} (y_k - y_i) \int_0^1 \Phi(x(t), y(t))dt \quad \text{.....(18)}$$

where $x(t) = x_i + x_{ki}t \;,\; y(t) = y_i + y_{ki}t, 0 \le t \le 1$ and the substitution $t = \frac{1+T}{2}$ in the eqn (18) gives;

$$II_{\pi_{xy}} = \sum_{i=1}^{N} \frac{(y_k - y_i)}{2} \int_{-1}^{1} \Phi\left(x\left(\frac{1+T}{2}\right), y\left(\frac{1+T}{2}\right)\right) dT \text{.....(19)}$$

We also have from eqn (15)

$$\Phi(x, y) = \left(\frac{x-\alpha}{2}\right) \int_{-1}^{1} f\left(\left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right), y\right) ds \text{.....(20)}$$

Where, x=x(t) and y=y(t)

$$II_{\pi_{xy}} = \sum_{i=1}^{N} \frac{(y_k - y_i)}{4} \int_{-1}^{1} \left(\int_{-1}^{1} (x-\alpha) f\left(\left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right), y\right) ds \right) dT \text{.....(21)}$$

where,

$$x(t) = x\left(\frac{1+T}{2}\right) = x_i + x_{ki}\left(\frac{1+T}{2}\right)$$

$$y(t) = y\left(\frac{1+T}{2}\right) = y_i + y_{ki}\left(\frac{1+T}{2}\right) \text{.....(22)}$$

Now on application of the Gauss Legendre rule of order *(n+1)* in S direction and of order in the direction, we can write

$$I\,I_{\pi_{xy}} \approx \sum_{i=1}^{N} \frac{(y_k - y_i)}{4} \sum_{p=1}^{n+1} \sum_{q=1}^{n} \left(x\left(t_p^{n+1}\right) - \alpha\right) W_p^{n+1} W_q^n \times f\left(\left(\frac{x\left(t_p^{n+1}\right) - \alpha}{2}\right) S_q^n + \left(\frac{x\left(t_p^{n+1}\right) + \alpha}{2}\right), y\left(t_p^{n+1}\right)\right) \text{...(23)}$$

$$t_p^{n+1} = \frac{1 + T_p^{n+1}}{2} \text{.....(24)}$$

where $\left(T_p^{n+1}, S_q^n\right), \left(W_p^{n+1}, W_q^n\right)$ are the sampling points (zeros of Legendre Polynomial) and the weight coefficients of order *(n+1)* and *n* respectively.

4. **Numerical examples:**

In this section, we consider several examples to show that the present formulation may be applied to integrals which find applications in practical situations.

We could find out from the drawing of polygon in [12] that the coordinates of vertices of the polygon P_6 must be:

{(0.1, 0), (0.7, 0.2), (1, 0.5), (0.75, 0.85), (0.5, 1), (0, 0.25)}

and the computer generated figure is as shown in fig1.

We call the polygon spanned by above six vertices as P_6. We could also find out from the drawing of nonconvex polygon in [12] that the coordinates of the vertices of the polygon must be:

{(0.25, 0), (0.75, 0.5), (0.75, 0), (1, 0.5), (0.75, 0.75), (0.75, 0.85), (0.5, 1), (0, 0.75), (0.25, 0.5)}

We call the polygon spanned by these vertices as P_9 and it is as shown fig.2

The integrals over the region P ($P = P_6, P_9$) are [12]:

$$I\ I_1 = \iint_P (x+y)^{19}\, dxdy$$

$$I\ I_2 = \iint_P \cos 30(x+y)\ dxdy$$

$$I\ I_3 = \iint_P \sqrt{(x-0.5)^2 + (y-0.5)^2}\ dxdy$$

$$I\ I_4 = \iint_P \exp\left\{-(x-0.5)^2 - (y-0.5)^2\right\}\ dxdy$$

$$I\ I_5 = \iint_P \exp\left\{-100(x-0.5)^2 - 100(y-0.5)^2\right\}\ dxdy$$

$$I\ I_6 = \iint_P \left[\begin{array}{l} \frac{3}{4}\exp\left(-\frac{1}{4}\left\{(9x-2)^2 + (9y-2)^2\right\}\right) \\ +\frac{3}{4}\exp\left(-\frac{1}{49}(9x+1)^2 - \frac{1}{10}(9y+1)\right) \\ +\frac{1}{2}\exp\left(-\frac{1}{4}\left\{(9x-7)^2 + (9y-3)^2\right\}\right) \\ -\frac{1}{5}\exp\left(-\left\{(9y-4)^2 + (9y-7)^2\right\}\right) \end{array}\right] dxdy$$

The results are obtained from the appended MATLAB programme based on symbolic maths: newpolygon _boundary _*n.m* and newnonconvexpolygon_boundary_*n.m* for the above six integrals. The solutions are found as in the Table 1.

By application of lemmas 1, it can be shown that the integrals II_1 and II_3 are also reducible to single integrals along the edges of the polygon P_n.

We have also written MATLAB programmes greens_integration_*n.m* based on scheme of section 3.1 The solution of integrals II_i (i=1, 2, 3, 4, 5, 6) are displayed in Tabls 2-7 for convex polygon of six sides and nonconvexpolygon of nine sides.

References-I

1. A.H.Stroud, D.Secrest, *Gaussian Quadrature Formulas*, Prentice Hall, Inc Englewood Cliffs, N.J., 1996.
2. A.H.Stroud, *Approximate calculation of multiple integrals*, PrenticeHall Series in Automatic computation, Prentice Hall, Inc Englewood Cliffs, N.J. ,1971.
3. C.T.Reddy and D.J.Shippy, *Alternative integration formula for triangular finite elements*, Int. J.Num.Methods in Engrg 17 (1981) 133-139.
4. J.N.Lyness, D.Jespersen, *Moderate degree symmetric quadrature rules for triangle*, J.Inst.Math.and Appl.15 (1975) 19-32.
5. D.Dunavant, *High degree efficient symmetrical quadrature rules for the triangle*, Int.J.Numer.Meth.Engrg 21 (1985) 1129-1148.
6. H.T.Rathod, *Some analytical integration formulae for a four node isoparametric finite element*, Comput. Struct. 30(1988) 1101-1109.
7. M.Nooijen, G.T.Velde, E.J.Baerends, *Symmetrical numerical integration formulas for regular polygons*, SIAM J. Numer. An.27 (1990) 198-218.
8. J.N.Lyness, G.Monegato, *Quadrature rules for regions having regular hexagonal symmetry*, SIAM J. Numer. An 14(2) (1977) 283-295.
9. J.A.Legget, *Exact formulae for areas, volumes and moments of polygons and polyhedra*, Comm.Appl.Numer.Methods, 4(6)(1988) 815-820.
10. H.T.Rathod and H.S.Govinda Rao, *Integration of trivariate polynomials over linear polyhedra in Euclidean three-dimensional space*, J.Aust.Math.Soc.Ser- B 39(3) (1998) 355-385.
11. A.Sommariva and M.Vianello, *Product Gauss Cubature over polygons based on Green's integration formula*, BIT Num. Math. 47(2007) 441-453.
12. G.Dasagupta, *Integration within polygonal finite elements*, J.Aero.Sp. Engrg.16 (2003) 9-18.
13. M.A.Wesley, *Construction and use of geometric modules in computer aided design*, Springer Verlag, New York, 79-136 (1980).
14. M.E.Mortenson, *Geometric modeling*, John Wiley, New York, 1985.
15. Y.T. Lee and A.A.G.Requicha, *Algorithm for computing the volume and other integral properties of solids I know methods and open issues*, communication A.C.M.25 (1982) 635- 641.
16. H.G.Timmer and J.M.Stern, *Computation of Global geometric properties of solid objects*, Comput. Aided Des. 12(1980), 301-304.
17. S.Lean and J.T.Kajiya, *A symbolic method for computing the integral properties of arbitrary nonconvex polyhedra*, IEEE Comput. Graph.Applic.4 (1984) 35-41.
18. C.Cattani and A.Paoluzzi, *Boundary integration over linear polyhedra*, Comput. Aided Des. 22, (1990) 130-135.
19. C.Cattani and A.Paoluzzi, *Symbolic analysis of linear polyhedra*, Engrg.Comput.6(1990)17-29.

20. H.T.Rathod and H.S.Govinda Rao, *Integration of polynomials over linear polyhedra in Euclidean three- dimensional space*, Comput. Methods Appl.Mech.Engrg.126 (1993)373-392.
21. H.T.Rathod and H.S.Govinda Rao, *Integration of polynomials over arbitrary tetrahedron in Euclidean three-dimensional space*, Comput.Struct.59 (1996), 35-65.
22. [23] H.T.Rathod and S.V. Hiremath, *Boundary Integration of polynomials over an arbitrary linear tetrahedron in Euclidean three dimensional space*, Comput. Methods Appl.Mech. Engrg. 153(1998) 81-106.

[24] H.T.Rathod and S.V. Hiremath, *Boundary Integration of polynomials over an arbitrary linear hexahedron in Euclidean three dimensional space*, Comput. Methods Appl.Mech. Engrg 161(1998)155-193.

[25] F.Bernardini, *Integration of polynomials over n-dimensional polyhedra*, Comput. Aided Des,(1991) 51-58.

[26] H.T.Rathod and H.S.Govinda Rao, *Integration of polynomials over n-dimensional linear polyhedra,*, Comput. Struct. 65(1997) 829-847.

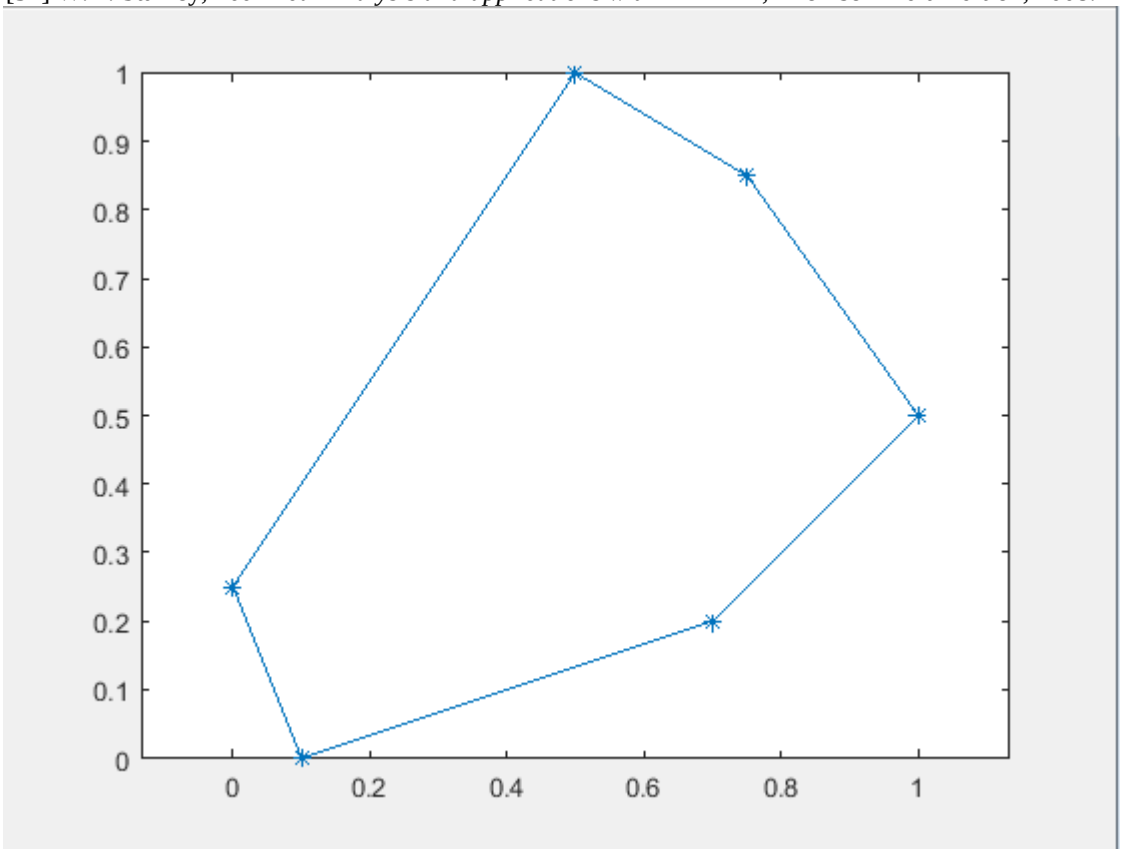
[27] A.Gilat, MATLAB, *An introduction with applications*, John Wiley and Sons.Inc, 2004.

[28] F.Gustafssan and N.Bergman, *MATLAB for engineers explained*, Springer International Edition, 2003.

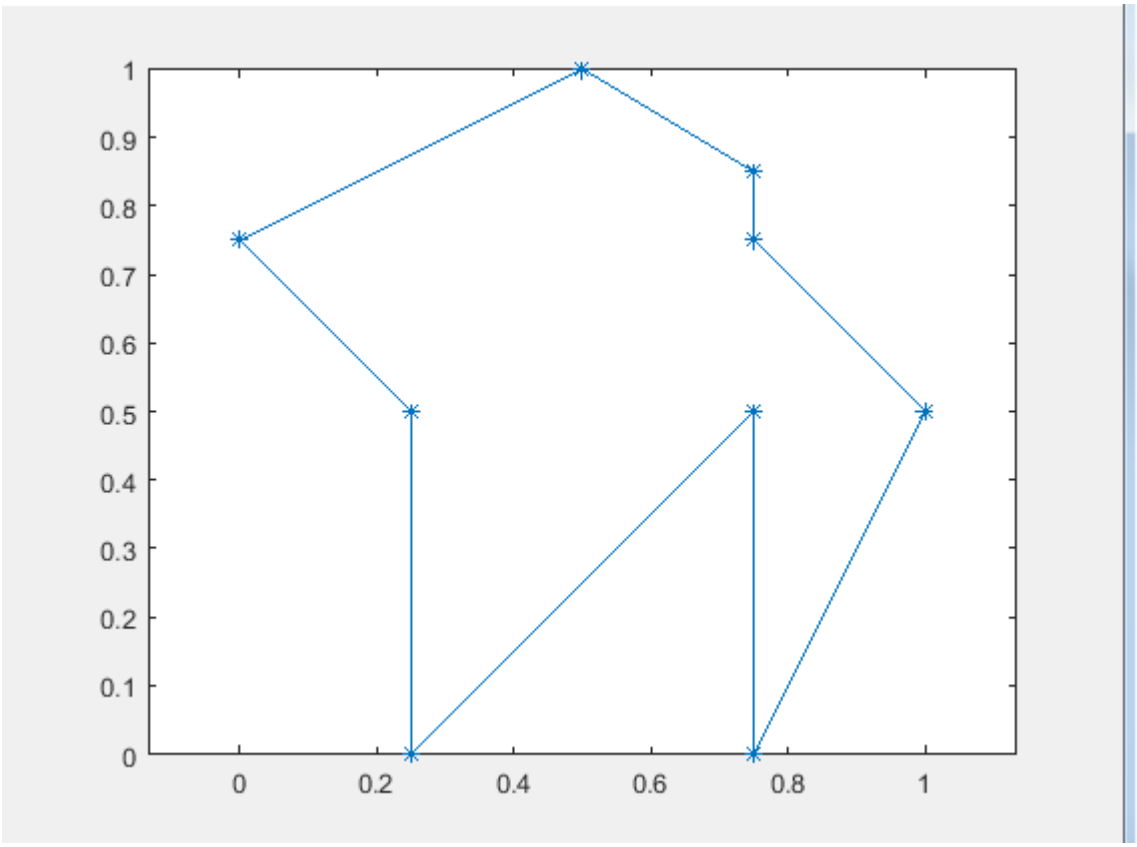
[29] S.R.Otto and J.P.Denier, *An Introduction to programming and numerical methods in MATLAB for beginners and experienced users*, Cambridge University Press, 2000.

[30] B.R.Hunt, R.L.Lipsman and J.M.Rosenberg, *A guide to MATLAB for beginners and experienced users*, Cambridge University Press, 2000.

[31] W.D. Stanley, *Technical Analysis and applications with MATLAB*, Thomson India Edition, 2008.



Fig, A CONVEX POLYGON WITH SIX SIDES:P_6



Fig, A NONCONVEX POLYGON WITH NINE SIDES:P_9

Table 1:Exact solution of integrals II_n (n=1(1)6)

Integral	Solutions by Numerical scheme based on boundary integration along n-sides & MATLAB command. >>newpolygon_boundary_*n* (2,6) **Convex Polygon with six sides**	Solutions by Numerical scheme based on boundary integration along *n*-sid MATLAB command. <<newnonconvexpolygon_boundary_n (2,9) **Nonconvex Polygon with nine sides**
I I_1	169.704343403127908648	130.84123498679649881 21
I I_2	$0.8421180941489947763966e^{-2}$	$0.14222050981512028804 1064 X(e^{-1})$
I I_3	0.15682512558608853742899	0.13938145677145110863 0493
I I_4	0.485060147024711349548	0.43740933669381119216958
I I_5	$0.3141452863239333773 6e^{-1}$	$0.3122083897153926882478 11X(e^{-1})$
I I_6	0.266330741912515235907 69634215953	0.18297132391896876545 6362

Table 2 : Numerical solutions for integral II_1

$$I\ I_1 = \iint_P (x+y)^{19}\, dxdy = \begin{cases} 169.704343403127908648 & \text{for convex polygonal domain } P_6 \\ 130.84123498679649881 21 & \text{for non-convex polygonal domain } P_9 \end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-2}$)	Non-convexpolygonal domain P_9 ($X\ e^{-2}$)	(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-2}$)	Non-convexpolygonal domain P_9 ($X\ e^{-2}$)
(1,2)	8.584703189850083	7.279838410567851	(25,26)	1.697043434031280	1.308412349867965

(2,3)	9.919055034429185	7.792785344352122	(26,27)	1.697043434031279	1.308412349867965
(3,4)	1.578298213050526	1.219911392705870	(27,28)	1.697043434031279	1.308412349867964
(4,5)	1.687278291040842	1.300906480467153	(28,29)	1.697043434031280	1.308412349867966
(5,6)	1.696625547922856	1.308065219985507	(29,30)	1.697043434031279	1.308412349867965
(6,7)	1.697034271854657	1.308403812322499	(30,31)	1.697043434031279	1.308412349867965
(7,8)	1.697043338749700	1.308412248317456	(31,32)	1.697043434031276	1.308412349867963
(8,9)	1.697043433631424	1.308412349385096	(32.33)	1.697043434031278	1.308412349867965
(9,10)	1.697043434030820	1.308412349867379	(33,34)	1.697043434031279	1.308412349867963
(10,11)	1.697043434031280	1.308412349867966	(34.35)	1.697043434031285	1.308412349867966
(11,12)	1.697043434031280	1.308412349867966	(35,36)	1.697043434031282	1.308412349867965
(12, 13)	1.697043434031279	1.308412349867965	(36.37)	1.697043434031281	1.308412349867964
(13,14)	1.697043434031279	1.308412349867965	(37,38)	1.697043434031276	1.308412349867967
(14,15)	1.697043434031280	1.308412349867965	(38,39)	1.697043434031279	1.308412349867966
(15,16)	1.697043434031281	1.308412349867966	(39,40)	1.697043434031280	1.308412349867967
(16,17)	1.697043434031281	1.308412349867966	(40,41)	1.697043434031281	1.308412349867966
(17,18)	1.697043434031279	1.308412349867965	(41,42)	1.697043434031280	1.308412349867965
(18,19)	1.697043434031277	1.308412349867965	(42,43)	1.697043434031280	1.308412349867966
(19,20)	1.697043434031282	1.308412349867966	(43,44)	1.697043434031282	1.308412349867965
(20,21)	1.697043434031280	1.308412349867966	(44,45)	1.697043434031286	1.308412349867965
(21,22)	1.697043434031280	1.308412349867966	(45,46)	1.697043434031281	1.308412349867963
(22,23)	1.697043434031277	1.308412349867964	(46,47)	1.697043434031278	1.308412349867964
(23,24)	1.697043434031280	1.308412349867965	(47,48)	1.697043434031275	1.308412349867965
(24,25)	1.697043434031280	1.308412349867965			

Table 3: Numerical solutions for integral II_2

$$I\ I_{\ 2}=\iint_{P}\cos\ 30(x+y)\,dxdy=\begin{cases}0.842118094148994776396 6e\text{-}2 \ \text{ for convex polygonal domain } P_6\\ 0.142220509815120288041064\ \text{X(e-1) for nonconvex polygonal domain } P_9\end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

(n, n+1)	Convexpolygonal domain P_6	Non-convexpolygonal domain P_9	(n, n+1)	Convexpolygonal domain P_6	Non-convexpolygonal domain P_9
(1,2)	7.242193614147435	8.642670515264378	(25,26)	1.697043434031280	1.422205098151206
(2,3)	-4.688436207395765	2.035590692846662	(26,27)	1.697043434031279	1.422205098151204
(3,4)	1.376458391696220	1.367930814406934	(27,28)	1.697043434031279	1.422205098151215
(4,5)	-5.613578891885893	1.230852677710495	(28,29)	1.697043434031280	1.422205098151199
(5,6)	9.447853050704561	7.066442629843972	(29,30)	1.697043434031279	1.422205098151205
(6,7)	-2.367390780180041	4.102426069657992	(30,31)	1.697043434031279	1.422205098151202
(7,8)	4.129544721048446	3.758075507869269	(31,32)	1.697043434031276	1.422205098151202
(8,9)	-9.019308621116063	7.767993692253883	(32.33)	1.697043434031278	1.422205098151206
(9,10)	1.467950958233120	1.542166724038094	(33,34)	1.697043434031279	1.422205098151198
(10,11)	6.736787265839745	1.406321250037825	(34.35)	1.697043434031285	1.422205098151209
(11,12)	8.779657688976764	1.423709691837702	(35,36)	1.697043434031282	1.422205098151203
(12, 13)	8.358926145630427	1.422110401722334	(36.37)	1.697043434031281	1.422205098151200
(13,14)	8.430193369471746	1.422207372949919	(37,38)	1.697043434031276	1.422205098151202
(14,15)	8.420075679761984	1.422205405054214	(38,39)	1.697043434031279	1.422205098151199
(15,16)	8.421297283588066	1.422205046615198	(39,40)	1.697043434031280	1.422205098151205
(16,17)	8.421170312443691	1.422205102861535	(40,41)	1.697043434031281	1.422205098151205
(17,18)	8.421181792509978	1.422205097828092	(41,42)	1.697043434031280	1.422205098151206
(18,19)	8.421180881266224	1.422205098169315	(42,43)	1.697043434031280	1.422205098151201
(19,20)	8.421180945285234	1.422205098150341	(43,44)	1.697043434031282	1.422205098151205
(20,21)	8.421180941275416	1.422205098151226	(44,45)	1.697043434031286	1.422205098151206
(21,22)	8.421180941500940	1.422205098151197	(45,46)	1.697043434031281	1.422205098151203
(22,23)	8.421180941489439	1.422205098151201	(46,47)	1.697043434031278	1.422205098151192
(23,24)	8.421180941489938	1.422205098151198	(47,48)	1.697043434031275	1.422205098151200
(24,25)	8.421180941489997	1.422205098151200			

Table 4: Numerical solutions for integral II_3

$$I\ I\ _3=\iint_P \sqrt{\left(x-0.5\right)^2+\left(y-0.5\right)^2}\,dxdy=\begin{cases}0.156825125586088537428 99 \text{ for convex polygonal domain } P_6\\ 0.139381456771451108630493 \text{ for non-convex polygonal domain } P_9\end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

.

(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)	(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)
(1,2)	8.071359265816813	6.872793814395499	(25,26)	1.568143063992179	1.393689638482703
(2,3)	1.662692657818633	1.480660935881813	(26,27)	1.568323394686016	1.393891520393837
(3,4)	1.526719737998170	1.349725702859312	(27,28)	1.568164884727789	1.393729603426915
(4,5)	1.582705171430963	1.410880266251929	(28,29)	1.568309326524392	1.393867470126955
(5,6)	1.557779761936418	1.381532157245348	(29,30)	1.568181209256887	1.393749089217504
(6,7)	1.572998617235859	1.399578442311508	(30,31)	1.568298692162079	1.393859179525478
(7,8)	1.564111442685137	1.389126746100178	(31,32)	1.568193665009588	1.393753000869676
(8,9)	1.570380351946265	1.395995807826128	(32.33)	1.568290504423578	1.393861656390401
(9,10)	1.566204068558331	1.391899042893934	(33,34)	1.568203333646477	1.393758639797320
(10,11)	1.569386030898827	1.394832496926622	(34.35)	1.568284098108929	1.393853388176169
(11,12)	1.567091832561209	1.392696738979680	(35,36)	1.568210953234467	1.393769829430239
(12, 13)	1.568926790769309	1.394484396009525	(36.37)	1.568279013815099	1.393842176289614
(13,14)	1.567531756515253	1.393019989313910	(37,38)	1.568217039150207	1.393783096124025
(14,15)	1.568685681059854	1.394351701115674	(38,39)	1.568274927310428	1.393835716593303
(15,16)	1.567774333337189	1.393255163945495	(39,40)	1.568221958699526	1.393785896189931
(16,17)	1.568547031299566	1.394152365829369	(40,41)	1.568271605310190	1.393835266446194
(17,18)	1.567918973667616	1.393459760670709	(41,42)	1.568225978472039	1.393786307454612
(18,19)	1.568461674594501	1.394015874451872	(42,43)	1.568268877047609	1.393836371346837
(19,20)	1.568010537051327	1.393594426815623	(43,44)	1.568229295146975	1.393788931970977
(20,21)	1.568406262880231	1.393954261573208	(44,45)	1.568266615565502	1.393831672672055
(21,22)	1.568071309398972	1.393632012206391	(45,46)	1.568232055982869	1.393794620456753
(22,23)	1.568368728884251	1.393942279710519	(46,47)	1.568264725143118	1.393827177115352
(23,24)	1.568113223355055	1.393656641249231	(47,48)	1.568234372721042	1.393799092545811
(24,25)	1.568342403462065	1.393926616550696			

Table 5: Numerical solutions for integral II_4

$$I\ I\ _4=\iint_P \exp\left\{-\left(x-0.5\right)^2-\left(y-0.5\right)^2\right\}dxdy=\begin{cases}0.4850601470247113495 48 \text{ for convex polygonal domain } P_6\\ 0.43740933669381119216958 \text{ for non-convex polygonal domain } P_9\end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

. .

(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)	(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)
(1,2)	5.199586493629421	4.705311467593615	(25,26)	4.850601470247119	4.374093366938111
(2,3)	4.841857989485471	4.365323836676870	(26,27)	4.850601470247111	4.374093366938107
(3,4)	4.850749301028626	4.374251327474454	(27,28)	4.850601470247117	4.374093366938106
(4,5)	4.850599546374505	4.374091204442674	(28,29)	4.850601470247109	4.374093366938119
(5,6)	4.850601490755657	4.374093390833759	(29,30)	4.850601470247108	4.374093366938118
(6,7)	4.850601470061354	4.374093366716373	(30,31)	4.850601470247113	4.374093366938118
(7,8)	4.850601470248577	4.374093366939887	(31,32)	4.850601470247107	4.374093366938107
(8,9)	4.850601470247103	4.374093366938098	(32.33)	4.850601470247105	4.374093366938116
(9,10)	4.850601470247113	4.374093366938112	(33,34)	4.850601470247115	4.374093366938121
(10,11)	4.850601470247113	4.374093366938109	(34.35)	4.850601470247103	4.374093366938106
(11,12)	4.850601470247116	4.374093366938110	(35,36)	4.850601470247105	4.374093366938102
(12, 13)	4.850601470247112	4.374093366938114	(36.37)	4.850601470247111	4.374093366938116
(13,14)	4.850601470247113	4.374093366938114	(37,38)	4.850601470247102	4.374093366938116
(14,15)	4.850601470247115	4.374093366938113	(38,39)	4.850601470247109	4.374093366938094
(15,16)	4.850601470247115	4.374093366938114	(39,40)	4.850601470247112	4.374093366938111
(16,17)	4.850601470247115	4.374093366938111	(40,41)	4.850601470247111	4.374093366938093
(17,18)	4.850601470247111	4.374093366938112	(41,42)	4.850601470247103	4.374093366938114
(18,19)	4.850601470247109	4.374093366938114	(42,43)	4.850601470247109	4.374093366938114
(19,20)	4.850601470247112	4.374093366938106	(43,44)	4.850601470247110	4.374093366938116
(20,21)	4.850601470247114	4.374093366938109	(44,45)	4.850601470247109	4.374093366938129
(21,22)	4.850601470247116	4.374093366938101	(45,46)	4.850601470247111	4.374093366938121
(22,23)	4.850601470247112	4.374093366938110	(46,47)	4.850601470247118	4.374093366938112
(23,24)	4.850601470247112	4.374093366938112	(47,48)	4.850601470247110	4.374093366938117
(24,25)	4.850601470247117	4.374093366938111			

Table 6: Numerical solutions for integral II_5

$$I\ I_5 = \iint_P \exp\left\{-100\left(x-0.5\right)^2 - 100\left(y-0.5\right)^2\right\} dxdy = \begin{cases} 0.31414528632393337736\text{e-}1 \text{ for convex polygonal domain } P_6 \\ 0.31220838971539268824781 1(\text{e-}1) \text{for non-convex polygonal domain } P_9 \end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

. .

(n, n+1)	Convexpolygonal domain P_6	Non-convexpolygonal domain P_9	(n, n+1)	Convexpolygonal domain P_6	Non-convexpolygonal domain P_9
(1,2)	1.745866441895653	3.141452863759899	(25,26)	3.141452863759899	3.122083897816000
(2,3)	5.113926986729511	3.141452863125061	(26,27)	3.141452863125061	3.122083897008633
(3,4)	6.854272031720735	3.141452863263612	(27,28)	3.141452863263612	3.122083897184786
(4,5)	1.127854825511277	3.141452863234330	(28,29)	3.141452863234330	3.122083897147569
(5,6)	4.469567863078354	3.141452863240328	(29,30)	3.141452863240328	3.122083897155193
(6,7)	2.409163107710347	3.141452863239138	(30,31)	3.141452863239138	3.122083897153674
(7,8)	3.528384609168796	3.141452863239367	(31,32)	3.141452863239367	3.122083897153969
(8,9)	2.951704923252380	3.141452863239322	(32.33)	3.141452863239322	3.122083897153916
(9,10)	3.229006641004684	3.141452863239334	(33,34)	3.141452863239334	3.122083897153927
(10,11)	3.103379561515891	3.141452863239330	(34.35)	3.141452863239330	3.122083897153923
(11,12)	3.157111161633770	3.141452863239332	(35,36)	3.141452863239332	3.122083897153925
(12, 13)	3.135346691447288	3.141452863239335	(36.37)	3.141452863239335	3.122083897153923
(13,14)	3.143715944313232	3.141452863239327	(37,38)	3.141452863239327	3.122083897153916
(14,15)	3.140654098289457	3.141452863239334	(38,39)	3.141452863239334	3.122083897153921
(15,16)	3.141721840788575	3.141452863239326	(39,40)	3.141452863239326	3.122083897153925
(16,17)	3.141366304611562	3.141452863239333	(40,41)	3.141452863239333	3.122083897153920
(17,18)	3.141479523557750	3.141452863239329	(41,42)	3.141452863239329	3.122083897153924
(18,19)	3.141444992790667	3.141452863239333	(42,43)	3.141452863239333	3.122083897153925
(19,20)	3.141455093185982	3.141452863239331	(43,44)	3.141452863239331	3.122083897153930
(20,21)	3.141452256082952	3.141452863239323	(44,45)	3.141452863239323	3.122083897153921
(21,22)	3.141453022291878	3.141452863239329	(45,46)	3.141452863239329	3.122083897153925
(22,23)	3.141452823105286	3.141452863759899	(46,47)	3.141452863239327	3.122083897153923
(23,24)	3.141452873004800	3.141452863125061	(47,48)	3.141452863239327	3.122083897153924
(24,25)	3.141452860945633	3.141452863263612			

Table 7: Numerical solutions for integral II_6

$$I\ I_6 = \iint_P \left[\begin{array}{l} \frac{3}{4}\exp\left(-\frac{1}{4}\left\{\left(9x-2\right)^2 + \left(9y-2\right)^2\right\}\right) \\ +\frac{3}{4}\exp\left(-\frac{1}{49}\left(9x+1\right)^2 - \frac{1}{10}\left(9y+1\right)\right) \\ +\frac{1}{2}\exp\left(-\frac{1}{4}\left\{\left(9x-7\right)^2 + \left(9y-3\right)^2\right\}\right) \\ -\frac{1}{5}\exp\left(-\left\{\left(9y-4\right)^2 + \left(9y-7\right)^2\right\}\right) \end{array}\right] dxdy = \begin{cases} 0.26633074191251523590769634215953 \text{ for convex polygon domain } P_6 \\ 0.18297132391896876545636 2 \text{ for non-convex polygon domain } P_9 \end{cases}$$

(n, n+1) = Gauss Legendre Quadrature rule of order n and n +1 used in the evaluation.

(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)	(n, n+1)	Convexpolygonal domain P_6 ($X\ e^{-1}$)	Non-convexpolygonal domain P_9 ($X\ e^{-1}$)
(1,2)	2.522766064492316	1.526466206323185	(25,26)	2.663307419125124	1.829713239189693
(2,3)	2.859505622231406	2.124274923568716	(26,27)	2.663307419125154	1.829713239189685
(3,4)	2.604908713479087	1.729889021067126	(27,28)	2.663307419125151	1.829713239189682
(4,5)	2.672127683792576	1.844260806823573	(28,29)	2.663307419125149	1.829713239189685
(5,6)	2.662757036319635	1.829468448109946	(29,30)	2.663307419125149	1.829713239189683
(6,7)	2.663289893626389	1.829359114195623	(30,31)	2.663307419125154	1.829713239189690
(7,8)	2.663281475431647	1.8298041985410 38	(31,32)	2.663307419125149	1.829713239189685
(8,9)	2.663333841839073	1.829699263023397	(32.33)	2.663307419125150	1.829713239189681
(9,10)	2.663293212933183	1.829714797861167	(33,34)	2.663307419125151	1.829713239189682
(10,11)	2.663313677900562	1.829713110532438	(34.35)	2.663307419125153	1.829713239189689
(11,12)	2.663304914132557	1.829713246063272	(35,36)	2.663307419125152	1.829713239189687
(12, 13)	2.663308356871179	1.829713239186423	(36.37)	2.663307419125152	1.829713239189683
(13,14)	2.663307087960897	1.829713239137059	(37,38)	2.663307419125157	1.829713239189686
(14,15)	2.663307529830036	1.829713239197468	(38,39)	2.663307419125152	1.829713239189691
(15,16)	2.663307384012941	1.829713239188913	(39,40)	2.663307419125149	1.829713239189694
(16,17)	2.663307429713036	1.829713239189754	(40,41)	2.663307419125149	1.829713239189684
(17,18)	2.663307416083859	1.829713239189683	(41,42)	2.663307419125148	1.829713239189688
(18,19)	2.663307419958824	1.829713239189684	(42,43)	2.663307419125155	1.829713239189687
(19,20)	2.663307418906696	1.829713239189693	(43,44)	2.663307419125151	1.829713239189686
(20,21)	2.663307419179962	1.829713239189685	(44,45)	2.663307419125153	1.829713239189679
(21,22)	2.663307419111964	1.829713239189694	(45,46)	2.663307419125153	1.829713239189681
(22,23)	2.663307419128201	1.829713239189690	(46,47)	2.663307419125153	1.829713239189683
(23,24)	2.663307419124480	1.829713239189682	(47,48)	2.663307419125154	1.829713239189685
(24,25)	2.663307419125296	1.829713239189688			

Computer Programs

```
function[II]=newpolygon_boundary_n(m,n)
%integration over polygon with n sides
%using symbolic maths
%fn=integrand function listed,fn=1,2,3,4,5,6
%n=number of sides of polygon
%m=1,for n-expanding triangles with respect to origin
%m=2,boundary integration along n-sides
syms t x y u v
format long e
A=[0.1;0.7;1;0.75;0.5;0;0.1];B=[0;0.2;0.5;0.85;1;0.25;0];
n1=length(A)-1;n2=length(B)-1;
if (n1~=n2)|(n~=n1)|(n~=n2)
   disp('mismatch of X,Y &number of sides')
   disp(n)
   disp(n1)
   disp(n2)
end

nn=n+1;
switch m
   case 1
    disp('indirect application of GREENS THEOREM')
    for fn=1:6

      switch fn
         case 1
        f=(x+y)^19
        X=A;Y=B;
        case 2
        f=cos(30*(x+y))
        X=A;Y=B;

        case 3
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=sqrt(x^2+y^2)
       case 4
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=exp(-(x^2+y^2))
       case 5
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=exp(-100*(x^2+y^2))
       case 6
      f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2)
      f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1))
      f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2)
      f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2)
      f=f1+f2+f3+f4
      X=A;Y=B;
     end
        ii=0;
    for N=1:n
     xi=X(N+1);yi=Y(N+1);xk=X(N);yk=Y(N);
     xx=(1-u)*xi+(xk-xi)*v;
     yy=(1-u)*yi+(yk-yi)*v;
     d=xk*yi-xi*yk;
     ff=subs(f,{x,y},{xx,yy});
     i=d*int(int(ff,v,0,1-u),u,0,1);
     ii=ii+i;
    end%end for N-LOOP%case 1
    iii=vpa(ii)
    II(fn)=double(iii);
end%end for fn
  case 2
     disp('direct application of GREENS THEOREM')
     for fn=1:6
```

```
      switch fn
      case 1
        f=(x+y)^19
        ff=int(f,x)
        X=A;Y=B;

      case 2
        f=cos(30*(x+y))
        ff=int(f,x)
        X=A;Y=B;

     case 3
        f=sqrt((x-0.5)^2+(y-0.5)^2)
        ff=int(f,x)
        X=A;Y=B;
    case 4
        f=exp(-((x-0.5)^2+(y-0.5)^2))
        ff=int(f,x)
         X=A-0.5*ones(nn,1)
         Y=B-0.5*ones(nn,1)
         ff=int(exp(-(x^2+y^2)),x)
     case 5
        f=exp(-100*((x-0.5)^2+(y-0.5)^2))
        ff=int(f,x)
         X=A;Y=B;

     case 6
        f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2)
        f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1))
        f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2)
        f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2)
        f=f1+f2+f3+f4
        ff=int(f,x)
        X=A;Y=B;

      end
       ii=0;
  for N=1:n
      xi=X(N+1);yi=Y(N+1);xk=X(N);yk=Y(N);
      xx=xk+(xi-xk)*t;yy=yk+(yi-yk)*t;
      d=(yi-yk);
      fff=subs(ff,{x,y},{xx,yy});
      i=d*int(fff,t,0,1);
      ii=ii+i;
  end%end for N-LOOP%case 2
  iii=vpa(ii)
  II(fn)=double(iii);

   end%end for fn
end%end for switch

function[II]=newnonconvexpolygon_boundary_n(m,n)
%integration over nonconvexpolygonal boundary
%using symbolic maths
%fn=integrand function listed,fn=1,2,3,4,5,6
%n=number of sides of polygon
%m=1,for n-expanding triangles with respect to origin
%m=2,boundary integration along n-sides
syms t x y u v
format long e
%  1   2   3   4  5   6   7  8  9    10
A=[0.25;0.75;0.75;1.0;0.75;0.75;0.5;0;0.25;0.25];
%  1  2  3  4   5   6  7   8   9  10
B=[0;0.5;0;0.5;0.75;0.85;1;0.75;0.5;0];

n1=length(A)-1;n2=length(B)-1;
if (n1~=n2)|(n~=n1)|(n~=n2)
   disp('mismatch of X,Y &number of sides')
```

```
   disp(n)
   disp(n1)
   disp(n2)
end
ii=0;nn=n+1;
switch m
   case 1
      for fn=1:6
      switch fn
         case 1
        f=(x+y)^19
        X=A;Y=B;
        case 2
        f=cos(30*(x+y))
        X=A;Y=B;
        case 3
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=sqrt(x^2+y^2)
       case 4
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=exp(-(x^2+y^2))
       case 5
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       f=exp(-100*(x^2+y^2))
       case 6
      f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2)
      f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1))
      f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2)
      f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2)
      f=f1+f2+f3+f4
      X=A;Y=B;
     end
        ii=0;
    for N=1:n
     xi=X(N+1);yi=Y(N+1);xk=X(N);yk=Y(N);
     xx=(1-u)*xi+(xk-xi)*v;
     yy=(1-u)*yi+(yk-yi)*v;
     d=xk*yi-xi*yk;
     ff=subs(f,{x,y},{xx,yy});
     warning off
     i=d*int(int(ff,v,0,1-u),u,0,1);
     ii=ii+i;
    end%end for N-LOOP%case 1
 ii=vpa(ii)
 II(fn)=double(ii);
end%end for fn
  case 2
     for fn=1:6
     switch fn
     case 1
      f=(x+y)^19
      ff=int(f,x)
      X=A;Y=B;
     case 2
      f=cos(30*(x+y))
      ff=int(f,x)
      X=A;Y=B;
    case 3
      f=sqrt((x-0.5)^2+(y-0.5)^2)
      ff=int(f,x)
       X=A-0.5*ones(nn,1);
       Y=B-0.5*ones(nn,1);
       ff=int(sqrt((x^2+y^2)),x)

   case 4
```

```
        f=exp(-((x-0.5)^2+(y-0.5)^2))
        ff=int(f,x)
         X=A-0.5*ones(nn,1);
         Y=B-0.5*ones(nn,1);
         ff=int(exp(-(x^2+y^2)),x)

    case 5
        f=exp(-100*((x-0.5)^2+(y-0.5)^2))
        ff=int(f,x)
        X=A-0.5*ones(nn,1);
        Y=B-0.5*ones(nn,1);
        ff=int(exp(-100*(x^2+y^2)),x)

     case 6
        f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2)
        f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1))
        f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2)
        f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2)
        f=f1+f2+f3+f4
        ff=int(f,x)
       X=A;Y=B;
      end
       ii=0;
  for N=1:n
      xi=X(N+1);yi=Y(N+1);xk=X(N);yk=Y(N);
      xx=xk+(xi-xk)*t;yy=yk+(yi-yk)*t;
      d=(yi-yk);
      fff=subs(ff,{x,y},{xx,yy});
      i=d*int(fff,t,0,1);
      ii=ii+i;
  end%end for N-LOOP%case 2
     ii=vpa(ii)
     II(fn)=double(ii);

end%end for fn
end%end for switch


function[]=greens_integration_n(ns,ng,fn,X,Y)
syms t x y rs
%coordinates for the vertices of polygon
%1:convex polygon 6-sides
%X=[0.1;0.7;1;0.75;0.5;0;0.1];Y=[0;0.2;0.5;0.85;1;0.25;0];
%2  nonconvex polygon 9-sides
%X=[0.25;0.75;0.75;1.0;0.75;0.75;0.5;0;0.25;0.25];
%  1  2  3  4   5    6   7   8   9  10
%Y=[0;0.5;0;0.5;0.75;0.85;1;0.75;0.5;0];

format long e
switch fn
   case 16
    f=(x+y)^19
   case 17
   f=cos(30*(x+y))
case 18
   f=sqrt((x-0.5)^2+(y-0.5)^2)
case 19
   f=exp(-((x-0.5)^2+(y-0.5)^2))
case 20
   f=exp(-100*((x-0.5)^2+(y-0.5)^2))
case 21
       f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2)
       f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1))
       f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2)
       f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2)
       f=f1+f2+f3+f4

end
```

```
    cc=0;
 for n=1:1:ng
    cc=cc+1;
   np0(cc,1)=n;
   [ss,ww]=glsampleptsweights(n);
   [sss,www]=glsampleptsweights(n+1);
   ii(cc,1)=0;
 for N=1:ns
       II=0;k=0;
       xi=X(N+1);yi=Y(N+1);xk=X(N);yk=Y(N);
       d=(yi-yk);

     if d~=0
       for jj=1:(n+1)
         for kk=1:n
            k=k+1;
            skk=(1+ss(kk))/2;

            t=sss(jj);wt=ww(kk)*www(jj)/(4);
            xt=((xi+xk)/2+(xk-xi)*t/2);yt=((yi+yk)/2+(yk-yi)*t/2);
             fff=xt*wt*fnxy(fn,skk*xt,yt);
             II=II+fff;
          end
       end
     end
     ii(cc,1)=ii(cc,1)+II*d;
  end
end
ii=double(ii);
table8(:,1)=np0;
table8(:,2)=ii;
disp(table8)
```

POLYNOMIAL SPLINE INTERPOLATIONS

1.0 Introduction

The term spline refers to an instrument used in drafting. It is a thin, flexible wooden or plastic tool that is passed through given data points and defines a smooth curve in between. Spline interpolation is very powerful and widely used method and has many applications in numerical differentiation, integration, solution of boundary value problems, computer animation, computer graphics and robot path/ trajectory planning.

Higher order polynomials were found to give erroneous results in certain cases, particularly when the function undergoes sudden changes in the vicinity of a point in its range. Further, it was found that a low order polynomial approximation in each subinterval provides a better approximation to the tabulated function than fitting a single high-order polynomial. Indeed for various functions f the corresponding interpolation polynomials may tend to oscillate more and more between nodes as n increases. Such oscillations are avoided by the method of splines, which was initiated by I.J. Schoenberg in 1946 and is now widely used in practice. Spline functions are named after the draftsman's device of using a thin flexible strip (called a spline) to draw a smooth curve through given points. The points at which two connecting splines meet are called knots. The connecting polynomials could be of any degree and therefore we have different types of spline functions, viz, linear, quadratic, cubic, quartic, quintic, etc. Of all these, the cubic spline (spline of degree three) has been found to be the most popular in engineering applications.

Historically speaking, long thin strips of wood or some other material have been used much like French curves by draftsmen to fair in a smooth curve between specified points. These strips or splines are anchored in place by attaching lead weights called "ducks" at the points along the spline. By varying the points where the ducks are attached to the spline itself and the position of both the spline and the duck relative to the drafting surface, the spline can be made to pass through the specified points provided a sufficient number of ducks are used. If we regard the draftsman's spline as a thin beam, then the Bernoulli Euler law,

$$M(x) = EI\left[\frac{1}{R(x)}\right]$$

is satisfied. Here $M(x)$ is the bending moment, E is Young's modulus, I is the geometric moment of inertia, and $R(x)$ is the radius of curvature of the elastica, i.e., the curve assumed by the deformed axis of the beam. For small deflections, $R(x)$ is replaced by $\frac{1}{y''(x)}$, where $y(x)$ denotes the elastica. Thus we have

$$y''(x) = \left(\frac{1}{EI}\right)M(x).$$

Since the ducks act efficiently as simple supports, the variation of $M(x)$ between duck position is linear. The mathematical spline is the result of replacing the draftsman's spline by its elastica and then approximating the latter by a piecewise cubic (normally a different cubic between each pair of adjacent ducks) with certain discontinuities of derivatives permitted at the junction points (the ducks) where two cubics join.

In its simple form, the mathematical spline is continuous and has both a continuous first derivative and a continuous second derivative. Normally, however, there is a jump discontinuity in its third derivative at the junction points. This corresponds to the draftsman's spline having continuous curvature with jumps occurring in the rate of change of curvature at the ducks. For many important applications, this mathematical model of the draftsman's spline is highly realistic.

In practice, the draftsman does not place the ducks at the specified points through which his spline must pass. Moreover, there is not usually a one-to-one correspondence between the specified points and the ducks. On the other hand, when the mathematical analog is used, it is common practice to interpolate to the specified points at the junction points and to keep the number of specified points and junction points (including the endpoints) the same.

Objective and Scope:

Spline functions play a vital role in the research and development endeavours of scientists and engineers. They constitute the main tool in computer aided geometric design (CAGD) which is concerned with the approximation and representation of curves and surfaces that arises when these objects have to be processed by a computer. The design of curves and surfaces plays an important role not only in the construction of different products as car bodies, ship hulls, airplane fuselages and wings, propellers blades, etc. but also in the description of geological, physical and even medical phenomena. New areas of CAGD applications include computer vision and inspection of manufactured parts, medical research (software for digital diagnostic equipment), image analysis, high resolution TV systems, cartography, the film industry, computer animation etc. The object of this thesis is to apply polynomial spline functions to obtain function approximations, integral function approximations, integration of functions along space curves and multiple integrals of functions over curved domains in two-dimensional Cartesian spaces.

Literature Review:**

The use of splines dates back at least to the beginning of previous century. Piecewise linear functions had been used in connection with Peano's existence proof for the solution to the initial value problems of the ordinary differential equations, although these functions were not called splines. Splines were first identified in the work of Schoenberg [47], usually a spline is a piecewise polynomial function defined in a region D, such that there exists a decomposition of D into sub regions, in each of which the function is a polynomial of some degree m. Also the function, as rule, is continuous in D, together with its derivatives of order up to (m-k) [33]. In other words spline function is a piecewise polynomial satisfying certain conditions of continuity of the function and its derivatives. The applications of spline as approximating, interpolating and curve fitting functions have been very successful [2, 25, 41]. It is also interesting to note that the cubic spline is a close mathematical approximation to the draftsman's spline, which is a widely used manual curve drawing tool. It has been shown by Schoenberg [47] that a curve drawn by a mechanical spline to a first order approximation is a cubic spline function. These reference*s*** can be found in **References-II.**

A number of authors have attempted polynomial and non polynomial spline approximations for the solution of differential equations [8, 10]. In this thesis, we shall confine to polynomial spline function approximations for numerical integration which is not rigorously pursued in the literature. Spline functions of maximum smoothness were first considered in the numerical solution of initial value problems in ordinary differential equations [9] and many interesting connections with standard numerical integration techniques have been established.

A spline function is a function that consists of polynomial pieces joined together with certain smoothness conditions. We may note that in spline theory knots are defined as the points where the spline function is permitted to change in form from one polynomial to another. The nodes are the points where values of the spline are specified. The first and second degree splines, though useful in certain applications, suffer an obvious imperfection: Their lower order derivatives are discontinuous. In case of

the first degree spline (or polygonal line) this lack of smoothness is immediately evident because the slope of the spline may change abruptly from one value to another at each knot. For the quadratic spline, the discontinuity is in the second derivative and therefore the lack of smoothness is not so evident. But the curvature of the quadratic spline changes abruptly at each knot, and the curve may not be pleasing to the eye. To achieve more smoothness (and greater accuracy) from the interpolating function, high degree polynomial pieces must be used. The most common choice is cubic polynomials. These cubic polynomial pieces can be combined in different ways to produce the overall interpolating function. The development of cubic splines as natural splines, clamped spline, not a knot spline is well known [12]. Cubic spline evaluation is possible only if the boundary end conditions are specified. When no information other than the function values at each interpolating point is available, the natural cubic spline and the not a knot cubic spline could be used. If the first derivatives are finite and known at the end points, then it is also well known that clamped spline can be used. This is extensively discussed in many texts on Numerical Analysis [4, 40, 45]. When no information other than the function values at each interpolating point is available, it is recommended that not a knot spline be applied. Thus, we have to choose between the natural spline and not a knot spline. This review presents a new cubic spline which can be applied when the function values are available at the end points of the spline. This was probably not attempted earlier as it was thought that with this form (four point form) it is difficult to join segments with C^1 continuity [39]. That is, we have another choice in the new cubic spline besides the natural spline and the not a knot spline when no information other than functional values at each interpolating point is available. The new cubic spline presented in this study is akin to the Subbotin quadratic spline [13, 14, 30, 53].

2.0 Interpolation

Let $y_0, y_1, ...y_n$ be a set of value of an unknown function $y = f(x)$ corresponding to the values of x: $x_0, x_1, ...x_n$. Then the process of estimating the value of y for any given value of x between x_0 *and* x_n is called interpolation. When we want to pass a smooth curve through these points or find some intermediate points, we use the technique of interpolation.

Any function $p(x)$ satisfying the conditions, $p(x_i) = f(x_i), \quad i = 0,1,.....n$ is called an interpolation function. Following are few methods of interpolation.

Lagrange Interpolation Formula

Consider the interpolating polynomial p associated with a table of data points (x_i, y_i) for $0 \le i \le n$. It is important to understand that there is one and only one interpolating polynomial of degree $\le n$ associated with the data (assuming that the $n+1$ abscissas x_i are distinct). However, the possibility certainly exists for expressing this polynomial in different forms and for arriving at it by different algorithms.

The alternative method will express p in the form

$$p(x) = y_0 l_0(x) + y_1 l_1(x) + ... + y_n l_n(x) = \sum_{k=0}^{n} y_k l_k(x) \quad \text{....(2.2.1)}$$

Here $l_0, l_1, ..., l_n$ are polynomials that depend on the nodes $x_0, x_1, ..., x_n$ but not on the ordinates $y_0, y_1, ..., y_n$. Since all the ordinates could be 0 except for a 1 occupying the i^{th} position, we see that

$$\delta_{ij} = p_n(x_j) = \sum_{k=0}^{n} y_k l_k(x_j) = \sum_{k=0}^{n} \delta_{ki} l_k(x_j) = l_i(x_j) \quad \text{...(2.2.2)}$$

Let us consider l_0. It is to be a polynomial of degree n that takes the value 0 at $x_1, x_2, ...x_n$ and the value 1 at x_0.

$$l_0(x) = c(x - x_1)(x - x_2).....(x - x_n) = c\prod_{j=1}^{n}(x - x_j)$$

Clearly, l_0 must be of the form

The value of c is obtained by putting $x = x_0$, so that

$$1 = c\prod_{j=1}^{n}(x_0 - x_j)$$

and

$$c = \prod_{j=1}^{n}(x_0 - x_j)^{-1}$$

Hence, we have

$$l_0(x) = \prod_{j=1}^{n} \frac{x - x_j}{x_0 - x_j}$$

Each l_i is obtained by similar reasoning, and the general formula is then

$$l_i(x) = \prod_{\substack{j=0 \\ j \ne i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad (0 \le i \le n) \qquad \text{.....(2.2.3)}$$

For the set of nodes $x_0, x_1, ..., x_n$, these polynomials are known as the cardinal functions. With the cardinal polynomials, eqn (2.2.1) gives the Lagrange form of the interpolating polynomials.

observe that

$$l_i(x_j) = \begin{cases} 1, & if \ \ i = j \\ 0, & otherwise \end{cases} \quad \text{..... (2.2.4)}$$

The polynomial $l_i(x)$ satisfying the conditions (2.2.4) can be written as

$$l_i(x_j)=\frac{(x-x_0).....(x-x_{i-1})(x-x_{i+1}).....(x-x_n)}{(x_i-x_0).....(x_i-x_{i-1})(x_i-x_{i+1}).....(x_i-x_n)}$$

Or

$$l_i(x_j)=\frac{w(x)}{(x-x_i)w'(x_i)} \quad i=0,1,.....n$$

Where,

$w(x)=(x-x_0)(x-x_1).....(x-x_n).$

Theorem 1: Theorem on polynomial Interpolation Error.

Let f be a function in $C^{n+1}[a,b]$, and let p be the polynomial of degree $\le n$ which interpolates the function f at $n+1$ distinct points $x_0,x_1,.....,x_n$ in the interval $[a,b]$. Then, for each x in $[a,b]$, there exists a number ξ_x in $[a,b]$ such that

$$f(x)-p(x)=\frac{1}{(n+1)!}f^{(n+1)}(\xi_x)w(x).$$

Hermite lnterpolation Formula

The Hermite interpolating polynomial interpolates not only the function $f(x)$ but also its derivatives at a given set of tabular points. Given the values of $f(x)$ and $f'(x)$ of the distinct points x_i, $i=0,1,2,.....n$. We determine a unique polynomial of degree $\le 2n+1$ which satisfies the conditions $p(x_i)=f(x_i), p'(x_i)=f'(x_i), \ i=0,1,....n$. Then required polynomial is given by $p(x)=\sum_{i=0}^{n}A_i(x)f(x_i)+\sum_{i=0}^{n}B_i(x)f'(x_i)$

Where $A_i(x)$ *and* $B_i(x)$ are polynomials of degree $\le 2n+1$

$A_i(x)=\left[1-2(x-x_i)l_i'(x_i)\right]l_i^2(x)$

and $B_i(x)=(x-x_i)l_i^2(x)$

where, $l_i(x)=\frac{(x-x_0)(x-x_1).....(x-x_{i-1})(x-x_{i+1}).....(x-x_n)}{(x_i-x_0)(x_i-x_1).....(x_i-x_{i-1})(x_i-x_{i+1}).....(x_i-x_n)}$

Theorem 2. : Theorem on Hermite Interpolation Error Estimate.

Let $x,x_0,x_1,.....,x_n$, be distinct nodes in $[a,b]$ and let $f\in C^{2n+2}[a,b]$. If p is the polynomial of degree at most $2n+1$ such that

$p(x_i)=f(x_i) \quad p'(x_i)=f'(x_i) \quad (i=0\le i\le n)$

then to each x in $[a,b]$ there corresponds a a point ξ in (a,b) such that

$$f(x)-p(x)=\frac{f^{(2n+2)}(\xi)}{(2n+2)!}\prod_{i=0}^{n}(x=x_i)^2$$

Runge's Phenomenon

Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when polynomials of high degree is used in the polynomial interpolation. It was discovered by Carl David Tolme Runge when exploring the behaviour of errors when using polynomial interpolation to approximate certain functions.

Consider the function $f(x)=\frac{1}{1+25x^2}$

Runge found that if this function is interpolated at equidistant points x_i between

-1 and 1 such that $x_i=-1+(i-1)\frac{2}{n}, \quad i\in\{1,2,.....n+1\}$

by a polynomial $p_n(x)$ of degree $\leq n$, the resulting interpolation oscillates toward the end of the interval, i.e. close to -1 and 1. It can even be proven that the interpolation error tends toward infinity when the degree of the polynomial increases:

$$\lim_{n\to\infty}\left(\max_{-1\leq x\leq 1}\left|f(x)-p_n(x)\right|\right)=+\infty$$

The error between the generating function and the interpolating polynomial of order N is given by

$$f(x)-P_N(x)=\frac{f^{(N+1)}(\xi)}{(N+1)!}\prod_{i=0}^{N}(x-x_i) \qquad \text{for some } \xi \text{ in } [-1,1]$$

For the case of the Runge function $f(x)=\frac{1}{1+25x^2}$,

The first two derivatives are;

$$f'(x)=-\frac{50x}{\left(1+25x^2\right)^2} \qquad so \ \left|f'(1)\right|=\frac{50}{26^2}\approx 0.0740;$$

$$f''(x)=\frac{5000x^2-50\left(1+25x^2\right)}{\left(1+25x^2\right)^3} \qquad so \ \left|f''(1)\right|=\frac{3700}{26^3}\approx 0.2105.$$

The magnitude of higher order derivatives of the Runge function get even larger. Therefore, the bound for the error behaviour (between the interpolating points) when using higher order interpolating polynomials becomes larger.
The oscillations can be minimized by using nodes that are distributed more densely towards the edges of the interval, specifically, with asymptotic density (on the interval $[-1,1]$) given by the formula $\frac{1}{\sqrt{1-x^2}}$. A standard example of such a set of nodes is Chebyshev nodes, for which the maximum error is guaranteed to diminish with increasing polynomial order. The phenomenon demonstrates that high degree polynomials are generally unsuitable for interpolation with equidistant nodes. The problem can be avoided by using spline curves which are piecewise polynomials. When trying to decrease the interpolation error one can increase the number of polynomial pieces which are used to construct the spline instead of increasing the degree of the polynomials used.
Runge's phenomenon [55] demonstrates that lower-order polynomials are generally to be preferred instead of raising the degree of the interpolation polynomial. Runge's function is nicely interpolated using splines however, and cubic splines are the most common interpolation method in this family.

Runge's Phenomena

(RED)- Runge Function
(BLUE)- 5th Degree Interpolating Polynomial
(GREEN)- 9th Degree Interpolating Polynomial

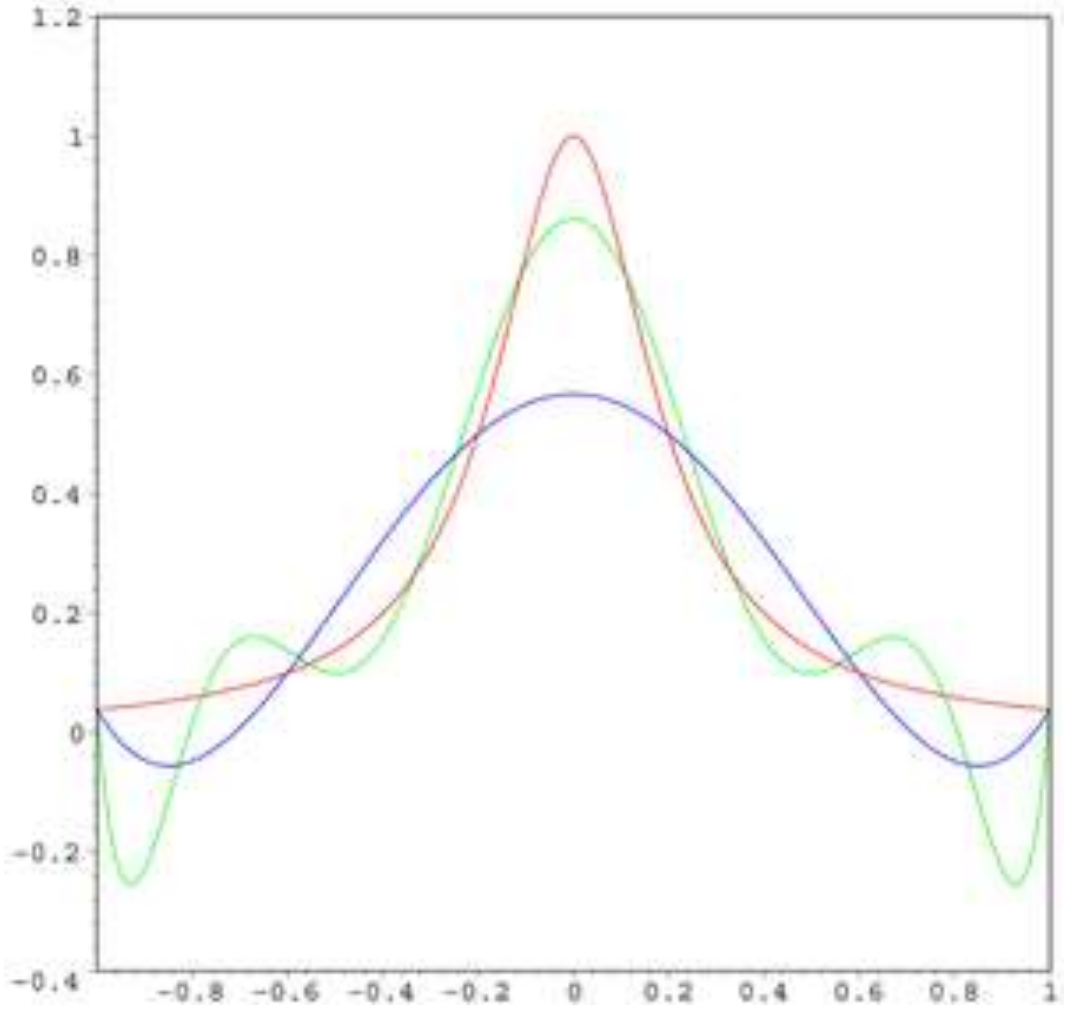


FIG 1: Error between the fuction and the interpolating polynomials.

The red curve is the Runge function.The blue curve is a 5^{th}-degree interpolating polynomial (using six equally- spaced interpolating points). The green curve is a 9^{th} degree interpolating polynomial (using ten equally- spaced interpolating points). At the interpolating points, the error between the function and the interpolating polynomial is (by definition) zero. Between the interpolating points (especially in the region close to the endpoints 1 and -1), the error between the fuction and the interpolating polynomial gets worse for higher-order polynomials.

2.1 Linear and Quadratic Splines

Linear Splines (Spline Of Degree 1)

Definition: A function S is called a spline of degree 1 if:

1.The domain of S is an interval $[a,b]$.

2. S is continuous on $[a,b]$.

3.There is a partitioning of the interval $a=t_0<t_1<.....<t_n=b$ such that S is a linear polynomial on each subinterval $[t_i,t_{i+1}]$.

A spline function is a function that consists of polynomial pieces joined togeter with certain smoothness conditions. A simple example is the polygonal function (or spline degrre of 1), whose pieces are linear polynomials joined together to achieve continuity. The points $t_{0,}t_1.....t_n$ at which the function changes its character are termed knots in the theory of splines. Such a function appears some what complicated when defined in explicit terms, and is written as

$$S(x)=\begin{cases} S_0(x) & x\in[t_0,t_1] \\ S_1(x) & x\in[t_1,t_2] \\ . & \\ . & \\ . & \\ S_{n-1}(x) & x\in[t_{n-1},t_n] \end{cases} \quad . \quad \text{.....(2.5.1)}$$

Where, $S_i(x)=a_i x+b_i \quad . \quad \text{.....}(2.5.2)$

because each piece of $S(x)$ is a linear polynomial. Such a function $S(x)$ is piece-wise linear. If the knots $t_0 t_1 \text{.....} t_n$ were given and if the coefficients $\left((a_i,b_i),\ i=0,(1)n-1\right)$ were all known, then the evaluation of $S(x)$ at a specific x would proceed by first determining the interval that contains x and then using the appropriate linear function for that interval.

If the function S defined by equation(2.5.1) is continuous, we call it a first-degree spline. It is characterized by the above three properties.

Quadratic splines (spline of degree 2)

Definition: A function Q is called a spline of degree 2 if:

1. The domain of Q is an interval $[a,b]$

2. Q and Q' are continuous on $[a,b]$

3. There are points t_i (called knots) such that $a=t_0<t_1<\text{.....}<t_n=b$ and Q is a polynomial of degree at most 2 on each subinterval $[t_i,t_{i+1}]$.

In brief, a quadratic spline is a continuously differentiable piecewise quadratic function, where quadratic includes all linear combination of the basic functions $x\to 1,x,x^2$.

Interpolating Quadratic Spline $Q(x)$

Quadratic splines are not used in applications as often as are natural cubic splines, however , the derivations of interpolating quadratic and cubic splines are similar enough that an understanding of the simpler second-degree spline theory will allow one to grasp easily the more complicated third-degree spline theory. Proceeding to the interpolation problem, Consider the table of values as:

x	t_0	t_1	t_2	.	.	.	t_n
y	y_0	y_1	y_2	.	.	.	y_n

Where, y(t_i) =y_i ,i=0(1)n

Suppose the points $t_0 t_1 \text{.....} t_n$ are the nodes for the interpolation problem, which are also the knots for the spline function to be constructed. Later, another quadratic spline interpolant is discussed in which the nodes for interpolation are different from the knots.

A quadratic spline consists of n separate quadratic functions $Q_i(x)=a_i x^2+b_i x+c_i$, one for each subinterval created by the n+1 knots. Thus, we start with 3n coefficients. On each subinterval $[t_i,t_{i+1}]$, the quadratic spline function Q_i must satisfy the interpolation conditions $Q_i(t_i)=y_i$ and $Q_i(t_{i+1})=y_{i+1}$. Since there are n such subintervals, this imposes 2n conditions. The continuity of Q does not add any additional conditions, however, the continuity of Q' at each of the interior knots gives n-1 more conditions. Thus, we have 2n+n-1=3n-1 conditions, or one condition short of the 3n conditions required. There are variety of ways to impose this additional condition; for example, $Q'(t_0)=0 \quad or \quad Q_0''=0$.

we now derive the equations for the interpolating quadratic spline, $Q(x)$. The value of $Q'(t_0)$ is prescribed as the additional condition. We seek a piecewise quadratic function

$$Q(x)=\begin{cases} Q_0(x) & t_0 \le x \le t_1 \\ Q_1(x) & t_1 \le x \le t_2 \\ . & . \\ . & . \\ . & . \\ Q_{n-1}(x) & t_{n-1} \le x \le t_n \end{cases} \qquad \text{.....(2.5.3)}$$

which is continuously differentiable on the entire interval $[t_0, t_n]$ and which interpolates the table; that is, $Q(t_i) = y_i \quad for\ 0 \le i \le n$.

Since Q' is continuous, we can put $Z_i = Q'(t_i)$. At present, we do not know the correct values of Z_i, but nevertheless the following must be the formula for Q_i.

$$Q_i(x) = \frac{Z_{i+1} - Z_i}{2(t_{i+1} - t_i)}(x - t_i)^2 + Z_i(x - t_i) + y_i \qquad \text{.....(2.5.4)}$$

To see that this is correct, verify that

$Q_i(t_i) = y_i,\ Q_i'(t_i) = Z_i,\ \ and\ \ Q_i'(t_{i+1}) = Z_{i+1.}$

These three conditions define the function Q_i uniquely on $[t_i, t_{i+1}]$ as given in Equation (2.5.4).

Now, in order for the quadratic spline function Q to be continuous and to interpolate the table of data, it is necessary and sufficient that $Q_i(t_{i+1}) = y_{i+1}$ for $i = 0,1,.....n-1$ in Equation (2.5.4). When this equation is written out in detail and simplified, the result is this:

$$Z_{i+1} = -Z_i + 2\left(\frac{y_{i+1} - y_i}{t_{i+1} - t_i}\right) \qquad (0 \le i \le n-1) \qquad \text{.....(2.5.5)}$$

This equation can be used to obtain the vector $[Z_0, Z_1, ...Z_n]^T$, starting with an arbitrary value for Z_0, It can be summarize with an algorithm:

ALGORITHM : **Quadratic Spline Interpolation at the knots**

Determine $[Z_0, Z_{1,}.....Z_n]^T$ by selecting Z_0 arbitrarily and computing $Z_{1,}Z_{2,}.....Z_n$ recursively by using Formula (2.5.5).

The quadratic spline interpolating function Q is given by formulas (2.5.3) and (2.5.4).

Subbotin Quadratic Spline

A useful approximation process, first proposed by Subbotin [1967], consists of interpolation with quadratic splines, where the nodes for interpolation are chosen to be the first and last knots and the midpoints between the knots. The knots are defined as the points where the spline function is permitted to change in form from one polynomial to another. The nodes are the points where values of the spline are specified. In the Subbotin quadratic spline function, there are $n+2$ interpolation conditions and $2(n-1)$ conditions from the continuity of Q and Q'. Hence, we have the exact number of conditions needed, $3n$, in order to define the quadratic spline function completely.

Suppose that knots $a = t_0 < t_1 < ..... < t_n = b$ have been specified; let the nodes be the points

$$\begin{cases} \tau_0 = t_0 & \tau_{n+1} = t_n \\ \tau_i = \frac{1}{2}(t_i + t_{i-1}) & (1 \le i \le n) \end{cases}$$

We seek a quadratic spline function Q that has the given knots and takes prescribed values at the nodes:

$Q(t_i) = y_i \qquad (0 \le i \le n+1)$

The knots create n subintervals, and in each of them Q can be a different quadratic polynomial. Let us say that on $[t_i, t_{i+1}]$, Q is equal to the quadratic polynomial Q_i. Since Q is a quadratic spline, it and its first derivative should be continuous. Thus,

$Z_i = Q'(t_i)$ is well defined, although its values are not known. It is easy to see that on $[t_i, t_{i+1}]$, quadratic polynomial can be represented in the form

$$Q_i(x) = y_{i+1} + \frac{1}{2}(Z_{i+1} + Z_i)(x - \tau_{i+1}) + \frac{1}{2h_i}(Z_{i+1} - Z_i)(x - \tau_{i+1})^2 \quad \text{.....(2.5.6)}$$

In which $h_i = t_{i+1} - t_i$. To verify the correctness of equation (2.5.6), we must check that $Q_i(\tau_{i+1}) = y_{i+1}$, $Q_i'(t_i) = Z_i$, *and* $Q_i'(t_{i+1}) = Z_{i+1.}$.When the polynomial pieces $Q_{0,}Q_1, \ldots\ldots\ldots Q_{n-1}$ are joined together to form Q, the result may be discontinuous. Hence, we impose continuity conditions at the interior knots:

$$\lim_{x \to t_i^-} Q_{i-1}(x) = \lim_{x \to t_i^+} Q_i(x) \qquad (1 \le i \le n-1)$$

which leads to

$$h_{i-1}Z_{i-1} + 3(h_{i-1} + h_i)Z_i + h_iZ_{i+1} = 8(y_{i+1} - y_i) \quad (1 \le i \le n-1) \quad \text{…(2.5.7)}$$

The first and last interpolation conditions must also be imposed:

$$Q(\tau_0) = y_0 \qquad Q(\tau_{n+`}) = y_{n+1}$$

These two equations lead to

$$3h_0Z_0 + h_0Z_1 = 8(y_1 - y_0)$$

$$h_{n-1}Z_{n-1} + 3h_{n-1}Z_n = 8(y_{n+1} - y_n)$$

The system of equations governing the vector $Z = [Z_0, Z_{1,} \ldots\ldots Z_n]^T$ then it can be written in the matrix form as

$$\begin{bmatrix} 3h_0 & h_0 & & & & & \\ h_0 & 3(h_0+h_1) & h_1 & & & & \\ & h_1 & 3(h_1+h_2) & h_2 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & & \\ & & & h_{n-2} & 3(h_{n-2}+h_{n-1}) & h_{n-1} & \\ & & & & h_{n-1} & 3h_{n-1} & \end{bmatrix} \begin{bmatrix} Z_0 \\ Z_1 \\ Z_2 \\ \cdot \\ \cdot \\ \cdot \\ Z_{n-1} \\ Z_n \end{bmatrix} = 8 \begin{bmatrix} y_1 - y_0 \\ y_2 - y_1 \\ y_3 - y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_n - y_{n-1} \\ y_{n+1} - y_n \end{bmatrix}$$

This system of $n+1$ equations in $n+1$ unknowns can be conveniently solved. After the Z vector has been obtained, values of $Q(x)$ can be computed from Equation (2.5.6).

ALGORITHMS:

Linear and Quadratic Splines

(1) We are given $n+1$ pairs of points (t_i, y_i) with distinct knots $a = t_0 < t_1 < \ldots\ldots\ldots\ldots < t_n = b$ over the interval $[a, b]$. A first degree spline function S is a piecewise linear polynomial defined on the interval [a.b] so that it is continuous. It has the form

$$S(x) = \begin{cases} S_0(x) & x \in [t_0, t_1] \\ S_1(x) & x \in [t_1, t_2] \\ \cdot & \\ \cdot & \\ \cdot & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases}$$

Where,

$$S_i(x) = y_i + \left(\frac{y_{i+1} - y_i}{t_{i+1} - t_i}\right)(x - t_i)$$

on the interval $[t_i, t_{i+1}]$. Clearly, $S(x)$ is continuous since,

$$S_{i-1}(t_i) = S_i(t_i) = y_i \qquad for \quad 1 \le i \le n.$$

(2). A second-degree spline function Q is a piecewise quadratic polynomial with Q and Q' continuous on the interval $[a,b]$. It has the form

$$Q(x)=\begin{cases} Q_0(x) & x\in[t_0,t_1] \\ Q_1(x) & x\in[t_1,t_2] \\ . & \\ . & \\ . & \\ Q_{n-1}(x) & x\in[t_{n-1},t_n] \end{cases}$$

Where, $Q_i(x)=\dfrac{Z_{i+1}-Z_i}{2(t_{i+1}-t_i)}(x-t_i)^2+Z_i(x-t_i)+y_i$

On the interval $[t_i,t_{i+1}]$. The coefficients $Z_0,Z_1,.....Z_n$ are obtained by selecting Z_0 and then using the recurrence relation

$$Z_{i+1}=-Z_i+2\left(\frac{y_{i+1}-y_i}{t_{i+1}-t_i}\right) \qquad (0\le i\le n-1)$$

(3) A Subbotin quadratic spline function Q is a piecewise quadratic polynomial with Q and Q' continuous on the interval $[a,b]$ and with interpolation condition at the end points of the interval $[a,b]$ and at the midpoints of the subintervals; namely,

$Q(\tau_i)=y_i \quad for \quad 0\le i\le n+1$

Where, $\tau_0=t_0, \qquad \tau_i=\dfrac{1}{2}(t_i+t_{i-1}) \qquad (1\le i\le n), \qquad \tau_{n+1}=t_n$

It has the form

$$Q_i(x)=y_{i+1}+\frac{1}{2}(Z_{i+1}+Z_i)(x-\tau_{i+1})+\frac{1}{2h_i}(Z_{i+1}-Z_i)(x-\tau_{i+1})^2$$

Where $h_i=t_{i+1}-t_i$. The coefficients Z_i are found by solving the tridiagonal system

$$\begin{cases} 3h_0Z_0+h_0Z_1=8(y_1-y_0) & \\ h_{i-1}Z_{i-1}+3(h_{i-1}+h_i)Z_i+h_iZ_{i+1}=8(y_{i+1}-y_i) & (1\le i\le n-1) \\ h_{n-1}Z_{n-1}+3h_{n-1}Z_n=8(y_{n+1}-y_n) & \end{cases}$$

2.2 Spline Integration

In many situations, it is desirable to know the area under a function described by piecewise polynomials as a function of the independent variable x. That is, if the piecewise polynomials are denoted $y=s(x)$, we are interested in computing

$$S(x)=\int_{x_1}^{x} s(x)dx+C$$

Where x_1 is the first breakpoint and C is the integration constant. Since $s(x)$ is composed of connected cubic polynomials, with the k^{th} cubic polynomial being

$$s_k(x)=a_k(x-x_k)^3+b_k(x-x_k)^2+c_k(x-x_k)+d_k, \qquad x_k\le x\le x_{k+1}$$

and whose integral or area over the range $[x_k,x]$, where $x_k\le x\le x_{k+1}$, is

$$S_k(x)=\int_{x_k}^{x} s_k(x)dx=\frac{a_k}{4}(x-x_k)^4+\frac{b_k}{3}(x-x_k)^3+\frac{c_k}{2}(x-x_k)^2+d_k(x-x_k)$$

the area under a piecewise polynomial is easily computed as

$$S(x) = \sum_{i=1}^{k-1} S_i(x_{i+1}) + S_k(x)$$

where $x_k \le x \le x_{k+1}$. The summation term is the cumulative sum of the areas under all preceding cubic polynomials. As such, it is readily computed and forms the constant term in the polynomial describing $S(x)$, since $S_k(x)$ is a polynomial. With this understanding, the integral itself can be written as a piecewise polynomial. In this case, it is a quartic piecewise polynomial, because the individual polynomials are of order four.

Smoothness Property

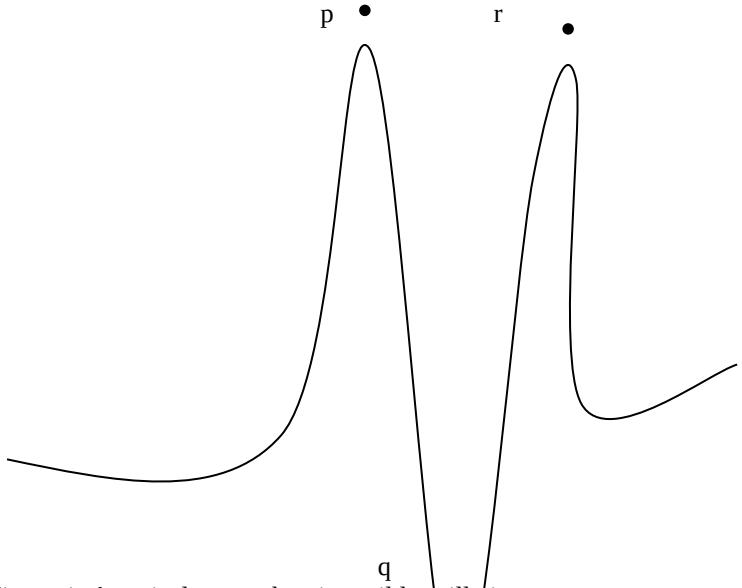


Figure 1: A typical curve showing wild oscillations.

Spline functions serve the needs of data fitting better than ordinary polynomials because, the interpolation by polynomials of high degree is often unsatisfactory due to the wild oscillations of polynomials. Polynomials are smooth in the technical sense of possessing continuous derivatives of all orders, where as in this sense spline functions are not smooth.

Wild oscillations in a function can be attributed to its derivatives being very large. Consider the function whose graph is shown in Fig.1. The slope of the chord that joins the Points $p \text{ and } q$ is very large in magnitude. By the Mean-value Theorem, the slope of that chord is the value of the derivative at some point between $p \text{ and } q$. Thus, the derivative must attain large values. Indeed, somewhere on the curve between $p \text{ and } q$ there is a point where $f'(x)$ is large and negative. Similarly, between q and r there is a point where $f'(x)$ is large and positive. Hence, there is a point on the curve between $p \text{ and } r$ where $f''(x)$ is large. This reasoning can be continued to higher derivatives if there are more oscillations. This is the behaviour that spline functions do not exhibit.

Theorem 4: Cubic spline smoothness theorem.

If S is the natural cubic spline function that interpolates a twice-continuously differentiable function f at knots $a = t_0 < t_1 <, \ldots\ldots\ldots\ldots < t_n = b$,

then

$$\int_a^b \left[S''(x)\right]^2 dx \le \int_a^b \left[f''(x)\right]^2 dx.$$

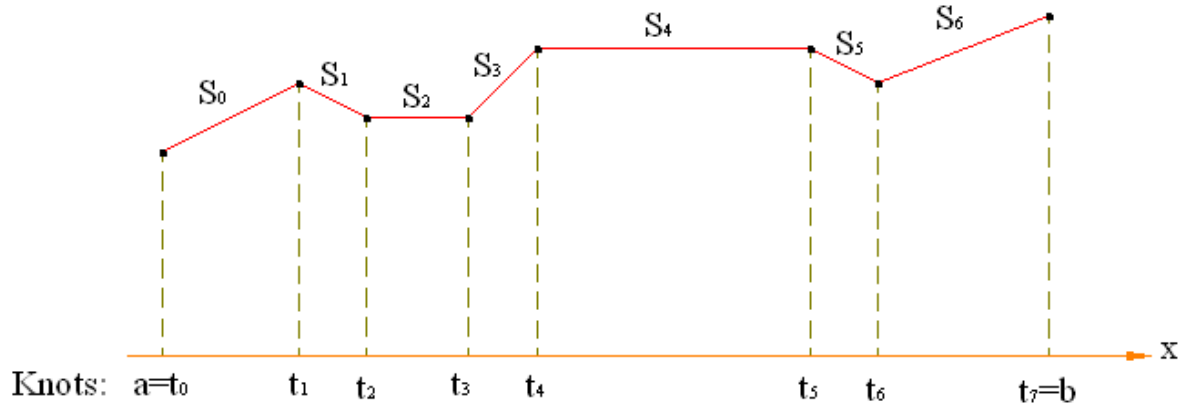
2.3 SPLINE INTERPOLATIONS: SOME BASIC DEFINITIONS

The spline is a flexible strip used to produce a smooth curve through a designated set of points. In computer graphics the term spline curve refers any composite curve forms with polynomial sections satisfying specified continuity conditions at the boundary of the pieces. Splines are used in graphics applications to design curve and surface shapes to digitize drawings for computer storage and to specify animation paths for the objects or the camera in a scene. A typical CAD application for splines includes the design of automobiles bodies, aircraft and spacecraft surfaces and ship hulls.

First degree Spline

A spline function is a function that consists of polynomial pieces joined together with certain smoothness conditions. A simple example is the **polygonal** function (or spline of degree 1), whose pieces are linear polynomials joined together to achieve continuity, as in figure below. The points $t_0, t_1, t_2, \ldots\ldots, t_n$ at which the function changes its character are turned knots in the theory of splines. Thus, the spline function shown in below figure has eight knots.

Figure First-degree spline function



Such a function appears complicated when defined in explicit terms. We are forced to write

$$S(x) = \begin{cases} S_0(x) & x \in [t_0, t_1] \\ S_1(x) & x \in [t_1, t_2] \\ \quad : & \\ \quad : & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases} \qquad (2.3.1.1)$$

where $S_i(x) = a_i x + b_i$ (2.3.1.2)

because each piece of $S(x)$ is a linear polynomial. Such a function $S(x)$ is piecewise linear.

Definition: Spline of degree 1

A function S is called a spline of degree 1 if:

The domain of S is an interval [a, b].

S is continuous on [a, b].

There is a partitioning of the interval $a = t_0 < t_1 < t_2 < \ldots\ldots, < t_n = b$ such that S is a linear polynomial on each subinterval $[t_i, t_{i+1}]$.

Consider the interval [a, b], S(x) is usually defined to be the same function left of a as it is on the leftmost subinterval $[t_0, t_1]$ and the same on the right it is on the rightmost subinterval $[t_{n-1}, t_n]$; namely, $S(x) = S_0(x)$ when $S(x) = S_{n-1}(x)$ when $x > b$.

Continuity of a function f at a point s can be defined by the condition

$$\lim_{x \to s^+} f(x) = \lim_{x \to s^-} f(x) = f(s)$$

Here $\lim_{x \to s^+}$ means that the limit is taken over x values that converges above s; that is, (x - s) is positive for all x values. Similarly $\lim_{x \to s^-}$ over the x values converges to s .

Spline of degree 2

A function Q is called a spline of degree 2 if:

The domain of Q is an interval [a, b].

Q and Q' are continuous on [a, b].

There are points t_i(called knots) such that $a = t_0 < t_1 < t_2 < \ldots\ldots, < t_n = b$ and Q is a polynomial of degree at most 2 on each subinterval $[t_i, t_{i+1}]$.

2.4 CUBIC SPLINE INTERPOLATION

This class of splines is most often used to setup paths for object motions or to provide a representation for an existing object or drawing, but interpolation splines are also used sometimes to design object shapes. Cubic polynomials offer a reasonable compromise between flexibility and speed of computation. Compared to higher-order polynomials, cubic splines require less calculations and memory and they are more stable. Compared to lower – order polynomials , cubic splines are more flexible for modeling arbitrary curve shapes.

2.4.1 The Cubic Spline Interpolant

Let f be a function defined on the interval [a, b], and let

$a = x_0 < x_1 < x_2 < \cdots \ldots < x_{n-1} < x_n = b$

Be the $n + 1$ distint points at which f is to be interpolated. Recall that the x_i divide [a, b] into n subintervals, referred to as a partition of [a, b].

Definition. A CUBIC SPLINE INTERPOLANT of f relative to the partition

$$a = x_0 < x_1 < x_2 < \cdots \ldots < x_{n-1} < x_n = b$$

is a function s that satisfies
On each subinterval $[x_j, x_{j+1}], j = 0, 1, 2, \ldots . n-1$, s coincides with cubic polynomial $s(x) = s_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$
s interpolates f at $x_0, x_1, x_2, \ldots \ldots, x_{n-1}, x_n$;
s is continuous on [a, b];
s' is continuous on [a, b];
s'' is continuous on [a, b];
The function s is composed of n different cubic polynomials, each with four coefficients, so there are a total of 4n unknowns Interpolation provides $n + 1$ equations. Continuity of the spline and its first two derivatives contribute an additional $3(n-1) = 3n - 3$ equations – remember that continuity applies at the interior points $x_0, x_1, x_2, \ldots \ldots, x_{n-1}$ only. The definition of a cubic spline therefore provides $n + 1 + 3(n-1) = 4n - 2$ equations. To completely determine the interpolating function, two or more equations will to be specified.
First, we will discuss two different types of additional constraints, or boundary conditions: the not-a-knot boundary conditions and clamped (or complete) boundary conditions.
Let us explain the conditions to be satisfied.
Interpolation:

$$s_j(x_j) = a_j = f(x_j), \qquad j = 0, 1, 2, \ldots . n$$

Continuity of spline:

$$s_{j+1}(x_{j+1}) = s_j(x_{j+1})$$

$a_{j+1} = a_j + b_j h_j + c_j {h_j}^2 + d_j {h_j}^3, \quad j = 0, 1, 2, \ldots \ldots, n-2$
Continuity of spline derivative:
$b_{j+1} = b_j + 2c_j h_j + 3d_j {h_j}^2, \qquad j = 0, 1, 2, \ldots \ldots, n-2$
Continuity of spline second derivative:
$c_{j+1} = c_j + 3d_j h_j \qquad j = 0, 1, 2, \ldots \ldots, n-2$
When, we have defined $h_j = x_{j+1} - x_j$, $a_n = f(x_n) = f(b)$. The interpolation conditions directly provide the values for the a_j, thereby removing one – quarter of the unknowns. Solve the equation for the continuity of the spline second derivative for d_j:

$$d_j = \frac{c_{j+1} - c_j}{3h_j} \qquad (2.4.1.1)$$

Substituting this expression into the equations for the continuity of the spline and its first derivative gives

$$a_{j+1} = a_j + b_j h_j + c_j {h_j}^2 + \frac{c_{j+1} - c_j}{3} {h_j}^2$$

$$= a_j + b_j h_j + \frac{c_{j+1} + 2c_j}{3} {h_j}^2 \qquad (2.4.1.2)$$

and

$$b_{j+1} = b_j + 2c_j h_j + (c_{j+1} - c_j) h_j$$

$$= b_j + (c_{j+1} + c_j) h_j \qquad (2.4.1.3)$$

Finally, solve equation (2.4.1.2) for b_j:

$$b_j = \frac{a_{j+1} - a_j}{h_j} - \frac{2c_j + c_{j+1}}{3} h_j \qquad (2.4.1.4)$$

*and substitute the result the result into equation (*2.4.1.*3). After performing some* algebraic manipulation and shifting the subscripts down by one, we get

$$h_{j-1} c_{j-1} + 2(h_{j-1} + h_j) c_{j-1} + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}). \quad (2.4.1.5)$$

This equation holds for $j = 1, 2, 3, \ldots, n-1$ and forms the basis for a tridiagonal system of equations for determining the c_j. The equations for $j = 0 \; and \; j = n$ depend on the type of boundary conditions which are being applied.
Regardless of the choice of boundary conditions, computing the coefficients of a cubic spline interpolant is a two-step process. First, the linear system for the c_j must be solved. Recall that an efficient algorithm for solving tridiagonal systems was developed in chapter 3 and is listed in Appendix B for convenience. Once this has been done, equation (2.4.1.1) is used to compute the d_j and equation (2.4.1.4) is used to compute b_j. Remember that the a_j are given by the function values, $f(x_j)$. Evaluation of a cubic spline interpolant, as with any piecewise function, is also a two–step process. Based on the value of the independent variable, the polynomial piece which needs to be evaluated must first be determined. Once the polynomial piece has been selected, the value of the interpolant at the given value of the independent variable can then be calculated.

2.4.2 Not-a-Knot Boundary Conditions

When no information other that the value of f at each interpolating point is available, it is recommended that the not-a-knot boundary conditions be applied. These conditions require that s''' be continuous at $x = x_1$ and $x = x_{n-1}$. In terms of the spline coefficients, this translates to
$d_0 = d_1$ and $d_{n-2} = d_{n-1}$.
Using (1), and rearranging terms, these equations can be expressed in terms of the c_j as

$$h_1 c_0 - (h_0 + h_1) c_1 + h_0 c_2 = 0 \qquad (2.4.2.1)$$

$$h_{n-1} c_{n-2} - (h_{n-2} + h_{n-1}) c_{n-1} + h_{n-2} c_n = 0 \qquad (2.4.2.2)$$

Unfortunately, (2.4.2.1) and (2.4.2.2) do not preserve the tridiagonal structure of (2.4.1.5). This situation, however, can be remedied as follows. Solve equations (2.4.2.1) for c_0 and equation (2.4.2.2) for c_n. This gives

$$c_0 = c_1 - \frac{h_0}{h_1}c_2 \qquad (2.4.2.3)$$

$$c_n = -\frac{h_{n-1}}{h_{n-2}}c_{n-2} + \left(1 + \frac{h_{n-1}}{h_{n-1}}\right)c_{n-1} \qquad (2.4.2.4)$$

Now, substitute c_0 from (2.4.2.3) into (2.4.1.5) for j = 1, and group terms to obtain,

$$\left(3h_0 + 2h_1 + \frac{h_0^2}{h_1}\right)c_1 + \left(h_1 - \frac{h_0^2}{h_1}\right)c_2 = \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \qquad (2.4.2.5)$$

Proceed in similar manner with the expression for c_n from (2.4.2.5) substituted into (2.4.1.5) for $j = n - 1$ to produce

$$\left(h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-2} + \left(3h_{n-1} + 2h_{n-2} + \frac{h_{n-1}^2}{h_{n-2}}\right)c_{n-1} = \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \qquad (2.4.2.6)$$

Equation (2.4.1.5) for $j = 2, 3, 4, \ldots\ n - 2$ together with equations (2.4.2.5) and (2.4.2.6) constitute a complete tridiagonal system for the coefficients $c_1, c_2, c_3 \ldots\ldots c_{n-1}$. Since each h_j is positive by definition, the coefficient matrix for this linear system is strictly diagonally dominant. Hence, there is always a unique solution for c_j

2.4.3 Clamped Boundary Conditions

If the values $f'(a)$ and $f'(b)$ are known, then it is better to apply the clamped (or complete) boundary conditions $s'(a) = f'(a)$ and $s'(b) = f'(b)$. Starting with $x = a$, we find $f'(a) = s'(a) = s_0'(a) = b_0$. Equation (2.4.1.4) with $j = 0$ allows us to write this condition in terms of the c_j

$$f'(a) = \frac{a_1 - a_0}{h_0} - \frac{2c_0 + c_1}{3}h_0$$

$$or, \qquad 2h_0c_0 + h_0c_1 = \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \qquad (2.4.3.1)$$

At $x = b$, $f'(b) = s'(b) = s_n'(b) = b_n$. Using the condition (2.4.1.3) to express b_n in terms of $b_{n-1}, c_{n-1}\ and\ c_n$, followed by equation (2.4.1.4) to rewrite b_{n-1} in terms of $a_{n-1}, a_n, c_{n-1}\ and\ c_n$, we obtain the equation

$$h_{n-1}c_{n-1} + 2h_{n-1}c_n = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \qquad (2.4.3.2)$$

Combining equation (2.4.1.5) for $j = 1, 2, 3 \ldots, n - 1$ with equations (2.4.2.6) and (2.4.3.1) produces a complete tridiagonal linear system for determining the c_j. As with not – a knot boundary conditions, the coefficient matrix associated with clamped boundary conditions is strictly diagonally dominant; hence, there is again always a unique solution for the c_j.

The clamped cubic spline satisfies an interesting property related to the curvature of a function. For a general function, the curvature at a point is dedfined by

$$\kappa(x) = \frac{|f''(x)|}{(1 + |f'(x)|^2)^{3/2}}$$

which is commonly linearized to $\kappa(x) \approx |f''(x)|$. The quantity $\int_a^b [f''(x)]^2\,dx$ can therefore be viewed as a crude measure of the total curvature over an interval. We will now prove that, in this measure, any smooth interpolating function which satisfies clamped boundary conditions must have a total curvature at least as large as that of the clamped cubic spline.

2.4.4 Natural Cubic Spline:

One of the first spline curves to be developed for graphic applications is the natural cubic spline . This interpolation curve is a mathematical representation of the original drafting spline. We formulate a natural cubic spline by requiring that two adjacent curve sections have the same first and second parametric derivatives at their common boundary. Thus, natural cubic splines have c^2 continuity. In this case the end conditions are $s''(x_0) = s''(x_n) = 0$ and this implies $c_0 = 0 = c_n$.

Error in Cubic Spline Interpolation

Theorem. Let f be continuous, with four continuous derivatives, on the interval [a, b], and let s be the clamped cubic spline interpolant of f relative to the partition $a = x_0 < x_1 < x_2 < \cdots \ldots < x_{n-1} < x_n = b\ then$

$$\max_{x\in(a,b)} |f(x) - s(x)| \le \frac{5}{384}h^4 \max_{x\in(a,b)} |f^4(x)|$$

$where\ h = \max_{0\le i\le n-1}(x_{i+1} - x_i)$

3.0 ON A NEW CUBIC SPLINE INTERPOLATION

3.1. Introduction

The fitting of a polynomial curve to a set of data points has applications in CAD (Computer assisted design), CAM (Computer assisted manufacturing) and Computer graphic systems and robot path/trajectory planning. The interpolating methods lead to higher order polynomials, in general of order equal to one less than the number of data points. In addition to being laborious computationally, these high order polynomials does indeed pass through the data points as required, it is quite well known that it does so by oscillating wildly between the given data points, particularly near the ends or middle of the data range. This is a characteristic feature of all high order polynomials, a feature which makes them unattractive for interpolation applications. Spline

methods involve interpolation between the given data points by attaching together several low order polynomials. Various continuity requirements can be specified at the data points to impose various degrees of smoothness of the resulting curve. The order of continuity becomes important when a complex curve is modeled by several curve segments pieced together end to end. A cubic polynomial is the minimum order polynomial that guarantees the generation of C^0, C^1 *or* C^2 curves. The best known methods resort to piecewise cubic curve constructed by the individual third degree polynomials assigned to each subinterval, which is called cubic spline interpolation.

In this chapter, we present a new cubic spline, where the nodes for interpolation are chosen to be the first and the last knots and the two points at the trisections between the knots. We shall note here that knots are defined as the points where the spline function is permitted to change in form from one polynomial to another and the nodes are the points where the values of the splines are specified. The new cubic spline presented in this chapter is akin to the Subbotin quadratic spline [13,14,30,53]. We mention here that in Subbotin quadratic spline the nodes for interpolation are chosen to be the first and last knots and the midpoint between the knots, this gives $(n+2)$ interpolation conditions over the n-subintervals (n-midpoints and 2- first and last points), and $(n-1)$ conditions from the continuity of spline functions. However, we note that since the first derivatives are defined at the nodes, the continuity of the first derivatives add no additional conditions. This determines the 3n parameters for the n-quadratic splines over the n-subintervals. In a similar manner, for the proposed new cubic spline there are $(2n+2)$ interpolation conditions over n-subintervals at the interpolation nodes as mentioned earlier and $2(n-1)$ conditions from the continuity of the spline functions and their first derivatives over n-subintervals. However, as in case of Subbotin quadratic splines, for the new cubic splines, since the values of the second derivatives at the nodes are defined and hence in this instance also continuity adds no additional conditions. Thus we can determine the 4n parameters of the n-spline functions over n-subintervals for the proposed new cubic spline function. In section-3.2, of this chapter, we present the derivation of the proposed new cubic spline. In section-3.3 of this chapter we have presented two improvements. The first improvement is regarding a simple and standard form of expression for the spline function valid over any subinterval. The second improvement is regarding the solution of the two band linear system obtained in section-3.2. We have shown that the solution to the two band linear system of section-3.2 can be obtained by simple assignments and recurrence relations. Finally, in section-3.4 of the chapter, we have presented some numerical illustrations. We have then used these illustrations to compare the new cubic spline function with the natural cubic spline, clamped cubic spline and the not a knot cubic spline which are generally discussed in standared texts [4,9,12,17,23,26,30,35,40,43,45,53,56].

3.2. Cubic Spline

We shall apply a useful approximation process, first proposed by Subbotin [1967] and later by Kammerer, Reddien and Varga [1974]. We propose a cubic spline, where the nodes for interpolation are chosen to be the first and the last and two points between the knots. It is assumed that the knots are defined as the points where the spline function is permitted to change in form from one polynomial to another and the nodes are the points where the values of splines are specified.

We suppose that knots $a = t_0 < t_1 < \ldots\ldots\ldots < t_n = b$ have been specified, let the nodes be the points

$$T_1 = t_1, \quad T_{2n+2} = t_n, \quad h_i = t_{i+1} - t_i$$

$$\left.\begin{aligned} T_{2i} &= t_i + h_i/3 \\ T_{2i+1} &= t_i + 2h_i/3 \end{aligned}\right\} \quad (i = 0,1,2,\ldots\ldots\ldots,n-1)$$

We seek a cubic spline Q that has the $(n+1)$ given knots and take the prescribed values at the nodes.

$$Q(T_i) = y_i \quad (1 \le i \le (2n+2))$$

The knots create n subintervals, and in each of them Q can be a different cubic polynomial. Let us say that on $[t_i, t_{i+1}]$, Q is equal to a cubic polynomial Q_i. Since Q is a cubic polynomial, Q, Q' *and* Q'' should be continuous. Thus $Z_i = Q''(t_i)$ is well defined. We do not yet know the values of $Z_1, Z_2, \ldots\ldots\ldots, Z_{n+1}$.

If the Z_i's were known, we could construct Q as now described. On the interval $[t_i, t_{i+1}]$, Q'' is a linear polynomial that takes the values Z_i *and* Z_{i+1} at the end points.

Thus, we have,

$$Q_i''(x) = \frac{Z_{i+1}}{h_i}(x - t_i) + \frac{Z_i}{h_i}(t_{i+1} - x)$$

$$\textit{with } h_i = t_{i+1} - t_i, \quad \textit{for } 1 \le i \le n \qquad \ldots\ldots(3.2.1)$$

Rewriting equation (3.2.1) as

$$Q_i^{//}(x) = \left(\frac{Z_{i+1} + Z_i}{3}\right) + \frac{Z_{i+1}}{h_i}(x - T_k) - \frac{Z_i}{h_i}(x - T_{k+1}) \quad \text{.....(3.2}a\text{)}$$

where

$$T_K = t_i + h_i / 3, \quad T_{k+1} = t_i + 2h_i / 3, \quad with \ \ k = 2i \quad \text{.....(3.2}b\text{)}$$

Integrating (3.2a) twice, and setting $x = T_k$ *and* $x = T_{k+1}$, the constants of integration are determined by the conditions :

$$Q_i(T_k) = y_k \qquad and \qquad Q_i(T_{k+1}) = y_{k+1.}$$

We finally obtain,

$$Q_i(x)=\frac{Z_{i+1}}{6h_i}\left[(x-T_k)^3+h_i(x-T_k)^2\right]-\frac{Z_i}{6h_i}\left[(x-T_{k+1})^3-h_i(x-T_{k+1})^2\right]$$
$$+\frac{\left(81y_{k+1}-2h_i^{\,2}Z_{i+1}\right)(x-T_k)}{27h_i}+\frac{\left(-81y_k+2h_i^{\,2}Z_i\right)}{27h_i}(x-T_{k+1})$$
.....(3.3)

We shall now show how to determine the unknowns Z_i's. The cubic spline $Q_i(x)$ of eqn (3.3) must also satisfy the following continuity relations.

$$Q_{i-1}(t_i)=Q_i(t_i) \quad \text{.....(3.4)}$$
$$Q'_{i-1}(t_i)=Q'_i(t_i) \quad \text{.....(3.5)}$$

From equations (3.3) and (3.4), the continuity of the spline function implies

$$-h_{i-1}^{\,2}Z_{i-1}+2\left(h_i^2-h_{i-1}^{\,2}\right)Z_i+h_i^2Z_{i+1}=27\left[(2y_{2i-1}-y_{2i-2})-(2y_{2i}-y_{2i+1})\right]$$
$$(i=2,3,.....,n) \quad \text{.....(3.6)}$$

and from equations (3.3) and (3.5), the continuity of the first derivative of spline function implies

$$7h_{i-1}Z_{i-1}+20(h_i+h_{i-1})Z_i+7h_iZ_{i+1}=162\left[(y_{2i+1}-y_{2i})/h_i-(y_{2i-1}-y_{2i-2})\right]$$
$$(i=2,3,.....,n) \quad \text{.....(3.7)}$$

Now eliminating Z_{i+1} between equations (3.6) and (3.7), we obtain

$$7h^2_{i-1}(h_i+h_{i-1})Z_{i-1}+2h_{i-1}(h_i+h_{i-1})(3h_i+7h_{i-1})Z_i=162h_i(y_{2i-2}-y_{2i-1})$$
$$+27h_{i-1}(-14y_{2i-1}+7y_{2i-2}+8y_{2i}-y_{2i+1})$$
$$(i=2,3,.....,n) \quad \text{.....(3.8)}$$

Finally imposing the first and the last interpolation conditions:

$$Q(T_1)=Q(t_1)=y_1 \quad \text{.....(3.9)}$$
$$Q(T_{2n+2})=Q(t_{n+1})=y_{2n+2} \quad \text{....(3.10)}$$

we obtain

$$h_1^{\,2}(2Z_1+Z_2)=27(y_3-2y_2+y_1) \quad \text{....(3.11)}$$
$$h_n^{\,2}(Z_n+2Z_{n+1})=27(y_{2n+2}-y_{2n+1}+y_{2n}) \quad \text{....(3.12)}$$

We can express equations (3.8), (3.11) and (3.12) as a two band matrix

$$\begin{bmatrix} 2h_1 & h_1 & 0 & 0 & 0 & & & & \\ \alpha_2 & \beta_2 & 0 & 0 & 0 & & & & \\ 0 & \alpha_3 & \beta_3 & 0 & 0 & & & & \\ 0 & 0 & \alpha_4 & \beta_4 & 0 & & & & \\ - & - & - & - & - & - & - & - & \\ - & - & - & - & - & - & - & - & \\ & & & \alpha_{n-2} & \beta_{n-2} & 0 & 0 & 0 & \\ & & & 0 & \alpha_{n-1} & \beta_{n-1} & 0 & 0 & \\ & & & 0 & 0 & \alpha_n & \beta_n & 0 & \\ & & & 0 & 0 & 0 & h_n & 2h_n & \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ - \\ Z_{n-3} \\ Z_{n-2} \\ Z_{n-1} \\ Z_n \\ Z_{n+1} \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ - \\ r_{n-3} \\ r_{n-2} \\ r_{n-1} \\ r_n \\ r_{n+1} \end{bmatrix}$$

…. (3.13)

Where,

$$r_1 = 27\left(y_3 - 2y_2 + y_1\right)/h_1$$

$$r_{n+1} = 27\left(y_{2n+2} - y_{2n+1} + y_{2n}\right)/h_n$$

$$\alpha_i = 7h_{i-1}^{\ 2}\left(h_i + h_{i-1}\right)\left(3h_i + 7h_{i-1}\right)$$

$$\beta_i = 2h_{i-1}\left(h_i + h_{i-1}\right)\left(3h_i + 7h_{i-1}\right)$$

$$r_i = 162\left(y_{2i-2} - y_{2i-1}\right) + 27h_{i-1}\left(-14y_{2i-1} + 7y_{2i-2} + 8y_{2i} - y_{2i+1}\right)$$

$$\left(i = 2,3,.....,n\right) \qquad .....(3.14)$$

The coefficient matrix of eqn (3.13) is a two band matrix and diagonally dominant. It can be shown that such a matrix is nonsingular, which means that the system has a unique solution.

3.3. Some Improvements

(1) For the computational efficiency of spline functions, we can rewrite the eqn (3.3). We first introduce a new variable u as

$$\frac{x - t_i}{h_i} = u \qquad .....(3.15)$$

Now from equations (3.2b) and (3.15), we have

$$x - T_k = x - t_i + T_k - t_i = h_i\left(u - \frac{1}{3}\right) \qquad \left(k = 2i\right)$$

$$x - T_{k+1} = x - t_i + T_{k+1} - t_i = h_i\left(u - \frac{2}{3}\right)$$

.....(3.16)

Substitutiing from eqn (3.16) into eqn (3.3) gives

$$Q_i\left(x\right) = \left(\frac{Z_{i+1} - Z_i}{6}\right)h_i^{\,2}u^3 + \left(\frac{Z_i h_i^{\,2}}{2}\right)u^2 + \left[3\left(y_{k+1} - y_k\right) - \frac{7}{54}h_i^{\,2}Z_{i+1} - \frac{10}{27}Z_i h_i^{\,2}\right]u + \left[\left(\frac{Z_{i+1} + 2Z_i}{27}\right)h_i^{\,2} + \left(2y_k - y_{k+1}\right)\right],$$

$$\left(k = 2i,\ \ t_i \le x \le t_{i+1,}\ 0 \le u \le 1\right) \qquad .....\left(3.17\right)$$

(2) For the computational efficiency of $Z_i\left(i = 1,2,3,.....n+1\right)$ values, we further suggest the following.

We have from the first two equations of the linear system of eqn (3.13)

$$2h_1Z_1 + h_1Z_1 = r_1$$

$$\alpha_2Z_1 + \beta_2Z_2 = r_2 \qquad ……(3.18)$$

Solving the above linear system of eqn (3.18)

$$Z_1 = \frac{(r_1\beta_2 - r_2 h_1)}{(2h_1\beta_2 - h_1\alpha_2)}$$

$$Z_2 = \frac{(2h_1 r_2 - r_1\alpha_2)}{(2h_1\beta_2 - h_1\alpha_2)} \quad . \qquad \text{.....(3.19)}$$

Further, again from the linear system of eqn (3.13),

$$\alpha_i Z_{i-1} + \beta_i Z_i = r_i \qquad (i = 3,4,,.....,n.) \qquad \text{….(3.20)}$$

This gives the recurrence relation

$$Z_i = \frac{r_i - \alpha_i Z_{i-1}}{\beta_i}, \qquad (i = 3,4,.....n) \qquad \text{….(3.21)}$$

We have from the last equation of the linear system of eqn (3.13)

$$Z_n + 2Z_{n+1} = \frac{r_{n+1}}{h_n} \qquad \text{….(3.22)}$$

$$\therefore\ Z_{n+1} = (r_{n+1} / h_n - Z_n)/2$$

Thus, we find that the values $Z_i \ (i = 1,2,3,.....n+1)$ can be determined by the use of eqns (3.19),(3.21) and (3.22).

3.4. Numerical Results and Discussion:

We consider four typical examples for illustrating the performance of the present new cubic spline with standard cubic splines, namely, the natural spline, clamped spline and not a knot spline. All the computations were carried out by using the MATLAB programs which uses double precision arithmetic.

Example 1.Runge's function

$$f(x) = \frac{1}{(1+25x^2)}, x \in [-1,1]$$

Example 2.Humps function

$$f(x) = \frac{1}{(x-0.3)^2 + 0.01} + \frac{1}{(x-0.9)^2 + 0.04} - 6 \quad , x \in [-1,2]$$

Example 3.

$$f(x) = (\cos(x))^{10}, x \in [-2,2]$$

Example 4.

$$f(x) = \frac{1}{1 + \frac{1}{(1+\exp(20x))}}, \quad x \in [-1,1]$$

We have compared the new cubic spline with the cubic spline studied in the standard texts, which are classified according to the end conditions. Some of the prominent cubic splines are natural spline. Clamped spline and not-a-knot spline. We have also included the error curves corresponding to all the three cubic splines studied in the standard text books and compared them with the error plot of the new cubic spline. The graphs are displayed in Fig1 - Fig4. The following MATLAB programs

(1) new_spline_cubic_subbotin.m
(2) splint1.m and splint2.m taken from text in reference [17]
(3) fspline.m
(4) cubic_subbotin_new.m.
(5) cubic_subbotin1_new.m
(6) spline.m % MATLAB supplied program

are developed and some available programs are either directly used or slightly modified for our purpose. We have chosen example 1 for disply in the present section.

Table-1a

Computed function values of natural (sn), clamped (sc) and not a knot (ss) splines and exact function (f) at the data points (t_i= -1+(i-1)/100, i= 1 (1) 201)

i	ss (t_i)	sn (t_i)	sc (t_i)	f (t_i)

1	0.03846153846154	0.03846153846154	0.03846153846154	0.03846153846154
2	0.03921214501271	0.03922668640607	0.03921182774660	0.03921184197628
3	0.03998430978779	0.03999935510368	0.03998398152702	0.03998400639744
4	0.04077903872695	0.04078706530742	0.04077886360191	0.04077887654195
5	0.04159733777038	0.04159733777038	0.04159733777038	0.04159733777038
6	0.04244021285828	0.04243631650367	0.04244029786948	0.04244031830239
7	0.04330866993082	0.04330463855058	0.04330875788803	0.04330879168471
8	0.04420371492821	0.04420156421245	0.04420376185282	0.04420377942314
9	0.04512635379061	0.04512635379061	0.04512635379061	0.04512635379061
10	0.04607763850295	0.04607868252802	0.04607761572427	0.04607764082479
11	0.04705880522897	0.04705988543405	0.04705878166091	0.04705882352941
12	0.04807113617716	0.04807171245971	0.04807112360375	0.04807114529504
13	0.04911591355599	0.04911591355599	0.04911591355599	0.04911591355599
14	0.05019446546648	0.05019418572080	0.05019447157001	0.05019450370184
15	0.05130830357977	0.05130801413969	0.05130830989481	0.05130836326321
16	0.05245898545956	0.05245883104511	0.05245898882859	0.05245901639344
17	0.05364806866953	0.05364806866953	0.05364806866953	0.05364806866953
18	0.05487717270035	0.05487724765798	0.05487717106492	0.05487721223762
19	0.05614816475062	0.05614824230586	0.05614816305851	0.05614823133071
20	0.05746297394590	0.05746301532113	0.05746297304317	0.05746300818848
21	0.05882352941176	0.05882352941176	0.05882352941176	0.05882352941176
22	0.06023183906887	0.06023181898404	0.06023183950709	0.06023189278723
23	0.06169022601824	0.06169020523738	0.06169022647164	0.06169031462060
24	0.06320109215600	0.06320108106954	0.06320109239788	0.06320113762048
25	0.06476683937824	0.06476683937824	0.06476683937824	0.06476683937824
26	0.06638997209340	0.06638997747511	0.06638997197598	0.06639004149378
27	0.06807340475915	0.06807341032736	0.06807340463766	0.06807351940095
28	0.06982015434547	0.06982015731608	0.06982015428066	0.06982021295165
29	0.07163323782235	0.07163323782235	0.07163323782235	0.07163323782235
30	0.07351580629039	0.07351580484837	0.07351580632185	0.07351589781290
31	0.07547154737269	0.07547154588069	0.07547154740524	0.07547169811321
32	0.07750428282296	0.07750428202698	0.07750428284032	0.07750435962023
33	0.07961783439490	0.07961783439490	0.07961783439490	0.07961783439490
34	0.08181620094469	0.08181620133108	0.08181620093626	0.08181632235631
35	0.08410408973826	0.08410409013804	0.08410408972953	0.08410428931876
36	0.08648638514398	0.08648638535726	0.08648638513933	0.08648648648649
37	0.08896797153025	0.08896797153025	0.08896797153025	0.08896797153025
38	0.09155396898031	0.09155396887678	0.09155396898257	0.09155413138018
39	0.09425044043689	0.09425044032977	0.09425044043923	0.09425070688030
40	0.09706368455759	0.09706368450044	0.09706368455884	0.09706381946130
41	0.10000000000000	0.10000000000000	0.10000000000000	0.10000000000000
42	0.10306600141870	0.10306600144645	0.10306600141810	0.10306622004638
43	0.10626956745622	0.10626956748492	0.10626956745559	0.10626992561105
44	0.10961889275205	0.10961889276736	0.10961889275172	0.10961907371883
45	0.11312217194570	0.11312217194570	0.11312217194570	0.11312217194570
46	0.11678802559688	0.11678802558944	0.11678802559704	0.11678832116788
47	0.12062677794608	0.12062677793839	0.12062677794625	0.12062726176116
48	0.12464917915404	0.12464917914993	0.12464917915413	0.12464942349642
49	0.12886597938144	0.12886597938144	0.12886597938144	30.12886597938144
50	0.13328850410750	0.13328850410949	0.13328850410746	054.13328890369877
51	0.13793038008534	0.13793038008740	0.13793038008529	0.1553793103448276
52	0.14280580938658	0.14280580938768	0.14280580938655	0.1425680614066405
53	0.14792899408284	0.14792899408284	0.14792899408284	0.1479257899408284
54	0.15331491044046	0.15331491043993	0.15331491044047	0.1533154584653124
55	0.15898163150463	0.15898163150408	0.15898163150465	0.1589825115992369
56	0.16494800451529	0.16494800451499	0.16494800451529	0.1649484536060825
57	0.17123287671233	0.17123287671233	0.17123287671233	0.17123287671233
58	0.17785612186439	0.17785612186453	0.17785612186438	0.17785682525567
59	0.18484171985491	0.18484171985506	0.18484171985491	0.18484288354898
60	0.19221467709607	0.19221467709614	0.19221467709606	0.19221528111485
61	0.20000000000000	0.20000000000000	0.20000000000000	0.20000000000000

62	0.20822400716820	0.20822400716817	0.20822400716820	0.20822488287350
63	0.21691826595948	0.21691826595944	0.21691826595948	0.21691973969631
64	0.22611565592196	0.22611565592194	0.22611565592196	0.22611644997174
65	0.23584905660377	0.23584905660377	0.23584905660377	0.23584905660377
66	0.24615288691114	0.24615288691115	0.24615288691114	0.24615384615385
67	0.25706772318262	0.25706772318263	0.25706772318262	0.25706940874036
68	0.26863568111486	0.26863568111486	0.26863568111486	0.26863666890531
69	0.28089887640449	0.28089887640449	0.28089887640449	0.28089887640449
70	0.29390085432267	0.29390085432267	0.29390085432267	0.29390154298310
71	0.30769087843854	0.30769087843853	0.30769087843854	0.30769230769231
72	0.32231964189571	0.32231964189571	0.32231964189571	0.32232070910556
73	0.33783783783784	0.33783783783784	0.33783783783784	0.33783783783784
74	0.35429642903738	0.35429642903738	0.35429642903738	0.35429583702391
75	0.37174745678214	0.37174745678214	0.37174745678214	0.37174721189591
76	0.39024323198872	0.39024323198872	0.39024323198872	0.39024390243902
77	0.40983606557377	0.40983606557377	0.40983606557377	0.40983606557377
78	0.43057476529717	0.43057476529717	0.43057476529717	0.43057050592034
79	0.45249412629186	0.45249412629186	0.45249412629186	0.45248868778281
80	0.47562544053407	0.47562544053407	0.47562544053407	0.47562425683710
81	0.50000000000000	0.50000000000000	0.50000000000000	0.50000000000000
82	0.52563658991487	0.52563658991487	0.52563658991487	0.52562417871222
83	0.55250396849987	0.55250396849987	0.55250396849987	0.55248618784530
84	0.58055838722518	0.58055838722518	0.58055838722518	0.58055152394775
85	0.60975609756098	0.60975609756098	0.60975609756098	0.60975609756098
86	0.64002505774674	0.64002505774674	0.64002505774674	0.64000000000000
87	0.67118005309907	0.67118005309907	0.67118005309907	0.67114093959732
88	0.70300757570388	0.70300757570388	0.70300757570388	0.70298769771529
89	0.73529411764706	0.73529411764706	0.73529411764706	0.73529411764706
90	0.76778411859257	0.76778411859257	0.76778411859257	0.76775431861804
91	0.80005380851660	0.80005380851660	0.80005380851660	0.80000000000000
92	0.83163736497341	0.83163736497341	0.83163736497341	0.83160083160083
93	0.86206896551724	0.86206896551724	0.86206896551724	0.86206896551724
94	0.89086058616940	0.89086058616940	0.89086058616940	0.89086859688196
95	0.91743539681932	0.91743539681932	0.91743539681932	0.91743119266055
96	0.94119436582351	0.94119436582351	0.94119436582351	0.94117647058824
97	0.96153846153846	0.96153846153846	0.96153846153846	0.96153846153846
98	0.97791887921698	0.97791887921698	0.97791887921698	0.97799511002445
99	0.98998772169714	0.98998772169714	0.98998772169714	0.99009900990099
100	0.99744731871335	0.99744731871335	0.99744731871335	0.99750623441397
101	1.00000000000000	1.00000000000000	1.00000000000000	1.00000000000000
102	0.99744731871335	0.99744731871335	0.99744731871335	0.99750623441397
103	0.98998772169714	0.98998772169714	0.98998772169714	0.99009900990099
104	0.97791887921698	0.97791887921698	0.97791887921698	0.97799511002445
105	0.96153846153846	0.96153846153846	0.96153846153846	0.96153846153846
106	0.94119436582351	0.94119436582351	0.94119436582351	0.94117647058824
107	0.91743539681932	0.91743539681932	0.91743539681932	0.91743119266055
108	0.89086058616939	0.89086058616939	0.89086058616939	0.89086859688196
109	0.86206896551724	0.86206896551724	0.86206896551724	0.86206896551724
110	0.83163736497341	0.83163736497341	0.83163736497341	0.83160083160083
111	0.80005380851660	0.80005380851660	0.80005380851660	0.80000000000000
112	0.76778411859257	0.76778411859257	0.76778411859257	0.76775431861804
113	0.73529411764706	0.73529411764706	0.73529411764706	0.73529411764706
114	0.70300757570388	0.70300757570388	0.70300757570388	0.70298769771529
115	0.67118005309907	0.67118005309907	0.67118005309907	0.67114093959732
116	0.64002505774674	0.64002505774674	0.64002505774674	0.64000000000000
117	0.60975609756098	0.60975609756098	0.60975609756098	0.60975609756098
118	0.58055838722518	0.58055838722518	0.58055838722518	0.58055152394775
119	0.55250396849987	0.55250396849987	0.55250396849987	0.55248618784530
120	0.52563658991487	0.52563658991487	0.52563658991487	0.52562417871222
121	0.50000000000000	0.50000000000000	0.50000000000000	0.50000000000000
122	0.47562544053407	0.47562544053407	0.47562544053407	0.47562425683710

123	0.45249412629186	0.45249412629186	0.45249412629186	0.45248868778281
124	0.43057476529717	0.43057476529717	0.43057476529717	0.43057050592034
125	0.40983606557377	0.40983606557377	0.40983606557377	0.40983606557377
126	0.39024323198872	0.39024323198872	0.39024323198872	0.39024390243902
127	0.37174745678214	0.37174745678214	0.37174745678214	0.37174721189591
128	0.35429642903738	0.35429642903738	0.35429642903738	0.35429583702391
129	0.33783783783784	0.33783783783784	0.33783783783784	0.33783783783784
130	0.32231964189571	0.32231964189571	0.32231964189571	0.32232070910556
131	0.30769087843854	0.30769087843853	0.30769087843854	0.30769230769231
132	0.29390085432267	0.29390085432267	0.29390085432267	0.29390154298310
133	0.28089887640449	0.28089887640449	0.28089887640449	0.28089887640449
134	0.26863568111486	0.26863568111486	0.26863568111486	0.26863666890531
135	0.25706772318262	0.25706772318263	0.25706772318262	0.25706940874036
136	0.24615288691114	0.24615288691115	0.24615288691114	0.24615384615385
137	0.23584905660377	0.23584905660377	0.23584905660377	0.23584905660377
138	0.22611565592196	0.22611565592194	0.22611565592196	0.22611644997174
139	0.21691826595948	0.21691826595944	0.21691826595948	0.21691973969631
140	0.20822400716820	0.20822400716817	0.20822400716820	0.20822488287350
141	0.20000000000000	0.20000000000000	0.20000000000000	0.20000000000000
142	0.19221467709607	0.19221467709614	0.19221467709606	0.19221528111485
143	0.18484171985491	0.18484171985506	0.18484171985491	0.18484288354898
144	0.17785612186439	0.17785612186453	0.17785612186438	0.17785682525567
145	0.17123287671233	0.17123287671233	0.17123287671233	0.17123287671233
146	0.16494800451529	0.16494800451499	0.16494800451529	0.16494845360825
147	0.15898163150463	0.15898163150408	0.15898163150465	0.15898251192369
148	0.15331491044046	0.15331491043993	0.15331491044047	0.15331544653124
149	0.14792899408284	0.14792899408284	0.14792899408284	0.14792899408284
150	0.14280580938658	0.14280580938768	0.14280580938655	0.14280614066405
151	0.13793038008534	0.13793038008740	0.13793038008529	0.13793103448276
152	0.13328850410750	0.13328850410949	0.13328850410746	0.13328890369877
153	0.12886597938144	0.12886597938144	0.12886597938144	0.12886597938144
154	0.12464917915404	0.12464917914993	0.12464917915413	0.12464942349642
155	0.12062677794608	0.12062677793839	0.12062677794625	0.12062726176116
156	0.11678802559688	0.11678802558944	0.11678802559704	0.11678832116788
157	0.11312217194570	0.11312217194570	0.11312217194570	0.11312217194570
158	0.10961889275205	0.10961889276736	0.10961889275172	0.10961907371883
159	0.10626956745622	0.10626956748492	0.10626956745559	0.10626992561105
160	0.10306600141870	0.10306600144645	0.10306600141810	0.10306622004638
161	0.10000000000000	0.10000000000000	0.10000000000000	0.10000000000000
162	0.09706368455759	0.09706368450044	0.09706368455884	0.09706381946130
163	0.09425044043689	0.09425044032977	0.09425044043923	0.09425070688030
164	0.09155396898031	0.09155396887678	0.09155396898257	0.09155413138018
165	0.08896797153025	0.08896797153025	0.08896797153025	0.08896797153025
166	0.08648638514398	0.08648638535726	0.08648638513933	0.08648648648649
167	0.08410408973826	0.08410409013804	0.08410408972953	0.08410428931876
168	0.08181620094469	0.08181620133108	0.08181620093626	0.08181632235631
169	0.07961783439490	0.07961783439490	0.07961783439490	0.07961783439490
170	0.07750428282296	0.07750428202698	0.07750428284032	0.07750435962023
171	0.07547154737269	0.07547154588069	0.07547154740524	0.07547169811321
172	0.07351580629039	0.07351580484837	0.07351580632185	0.07351589781290
173	0.07163323782235	0.07163323782235	0.07163323782235	0.07163323782235
174	0.06982015434547	0.06982015731608	0.06982015428066	0.06982021295165
175	0.06807340475915	0.06807341032736	0.06807340463766	0.06807351940095
176	0.06638997209340	0.06638997747511	0.06638997197598	0.06639004149378
177	0.06476683937824	0.06476683937824	0.06476683937824	0.06476683937824
178	0.06320109215600	0.06320108106954	0.06320109239788	0.06320113762048
179	0.06169022601824	0.06169020523738	0.06169022647164	0.06169031462060
180	0.06023183906887	0.06023181898404	0.06023183950709	0.06023189278723
181	0.05882352941176	0.05882352941176	0.05882352941176	0.05882352941176
182	0.05746297394590	0.05746301532113	0.05746297304317	0.05746300818848
183	0.05614816475062	0.05614824230586	0.05614816305851	0.05614823133071

184	0.05487717270035	0.05487724765798	0.05487717106492	0.05487721223762
185	0.05364806866953	0.05364806866953	0.05364806866953	0.05364806866953
186	0.05245898545956	0.05245883104511	0.05245898882859	0.05245901639344
187	0.05130830357977	0.05130801413969	0.05130830989481	0.05130836326321
188	0.05019446546648	0.05019418572080	0.05019447157001	0.05019450370184
189	0.04911591355599	0.04911591355599	0.04911591355599	0.04911591355599
190	0.04807113617716	0.04807171245971	0.04807112360375	0.04807114529504
191	0.04705880522897	0.04705988543405	0.04705878166091	0.04705882352941
192	0.04607763850295	0.04607868252802	0.04607761572427	0.04607764082479
193	0.04512635379061	0.04512635379061	0.04512635379061	0.04512635379061
194	0.04420371492821	0.04420156421245	0.04420376185282	0.04420377942314
195	0.04330866993082	0.04330463855058	0.04330875788803	0.04330879168471
196	0.04244021285828	0.04243631650367	0.04244029786948	0.04244031830239
197	0.04159733777038	0.04159733777038	0.04159733777038	0.04159733777038
198	0.04077903872695	0.04078706530742	0.04077886360191	0.04077887654195
199	0.03998430978779	0.03999935510368	0.03998398152702	0.03998400639744
200	0.03921214501271	0.03922668640607	0.03921182774660	0.03921184197628
201	0.03846153846154	0.03846153846154	0.03846153846154	0.03846153846154

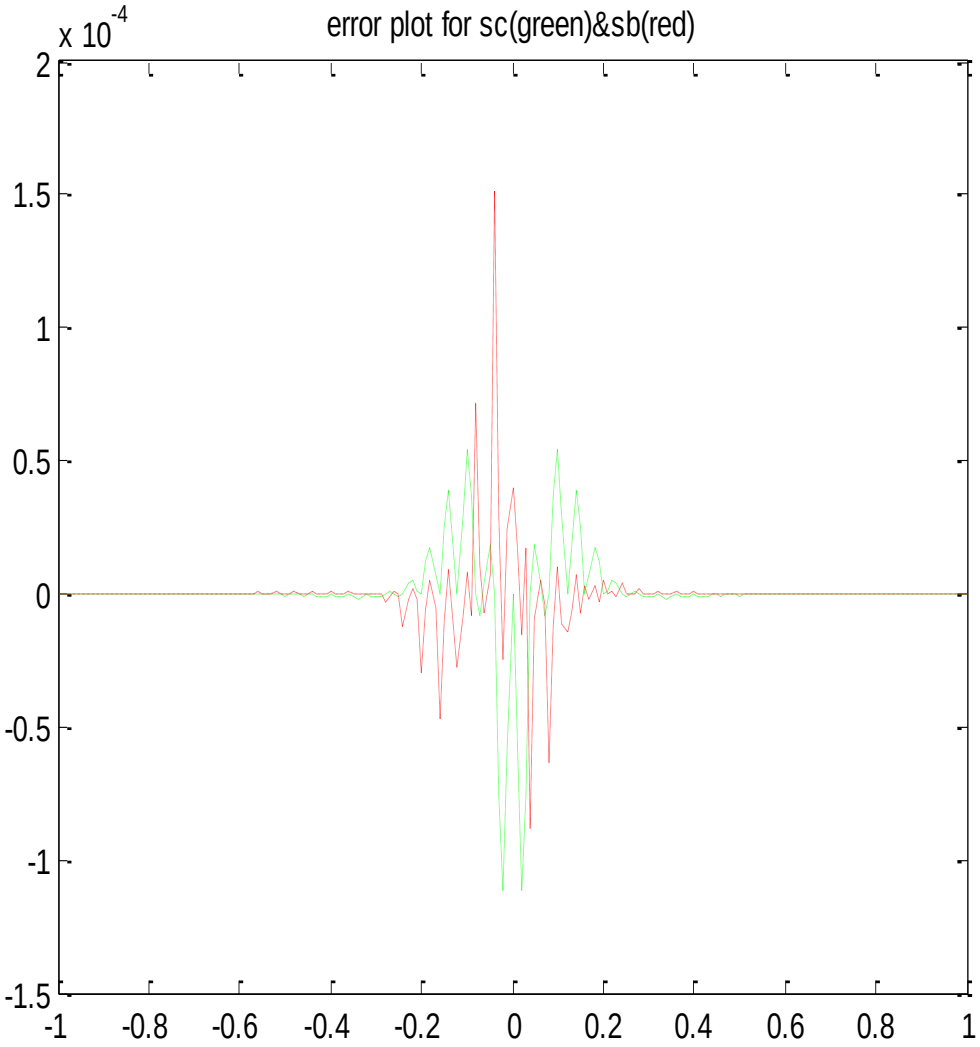


FIGURE-1
sc: clamped spline, sb: cubic subbotin spline (present study)

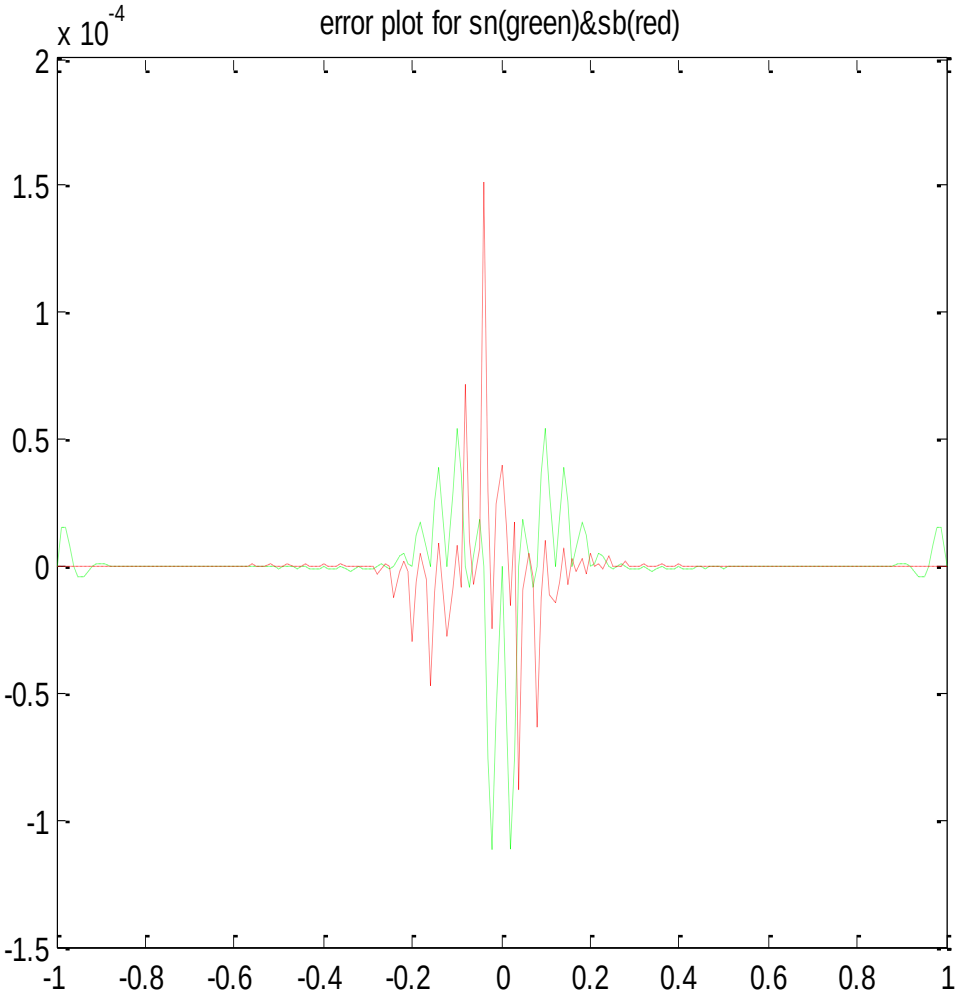


FIGURE-2
sn: natural spline, sb: cubic subbotin spline (present study)

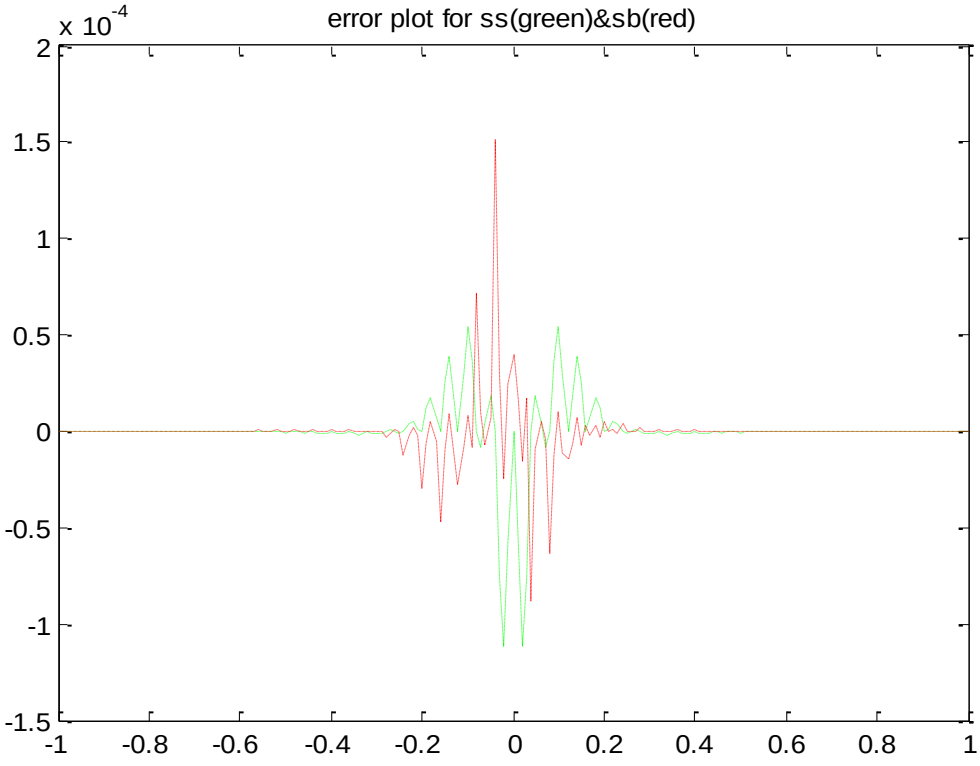


FIGURE-3
ss: not a knot spline, sb: cubic subbotin spline (present study)

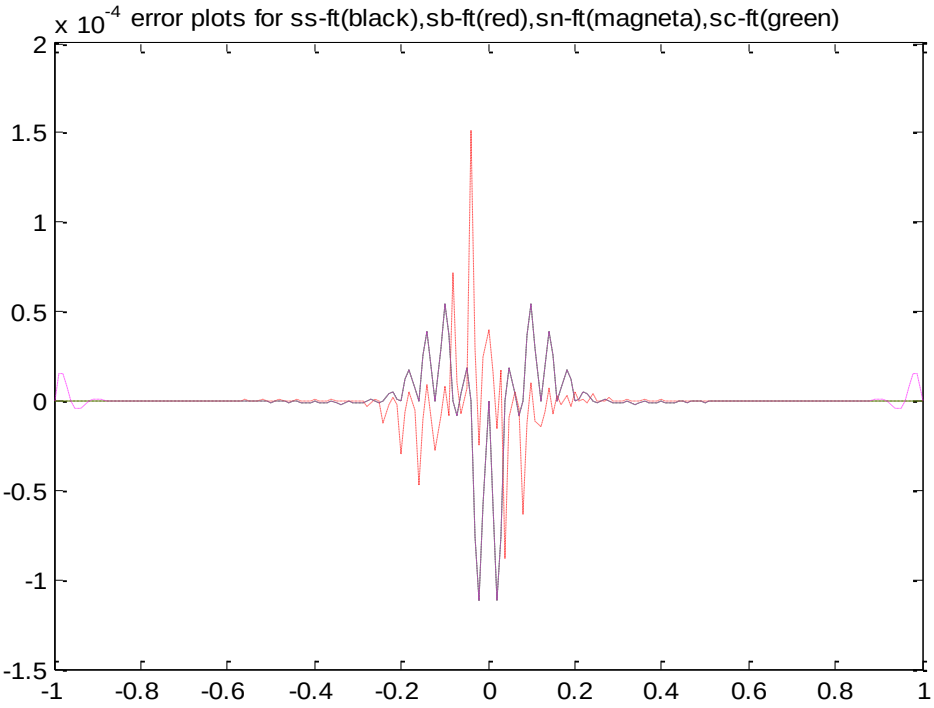


FIGURE-4
ss: not a knot spline, sb: cubic subbotin spline (present study)
sn: natural spline, sc: clamped spline.

4.0 ON A NEW CUBIC SPLINE INTERPOLATION WITH APPLICATION TO QUADRATURE

4.1. Introduction

In this study, we present a new cubic spline, where the nodes for interpolation are chosen to be the first and the last knots and the two points at the trisections between the knots. We shall note here that the knots are defined as the points where the spline function is permitted to change in form from one polynomial to another and the nodes are the points where the values of the splines are specified. The new cubic spline presented in this chapter is akin to the Subbotin quadratic spline [13,14,17,23,26,30,35,36,40,43,45,51,53]. We mention here that in Subbotin quadratic spline the nodes for interpolation are chosen to be the first and last knots and the midpoint between the knots, this gives $(n+2)$ interpolation condition over the n-subinterval (n-midpoints and 2-first and last points), and $(n-1)$ conditions from the continuity of spline functions. However, we note that since the first derivatives are defined at the nodes, the continuity of the first derivatives add no additional conditions. This determines the 3n parameters for the n-quadratic splines over the n-subintervals. In a similar manner, for the proposed new cubic spline there are $(2n+2)$ interpolation conditions over n-subintervals at the interpolation nodes as mentioned earlier and $2(n-1)$ conditions from the continuity of the spline functions and their first derivatives over n-subintervals. However, as in case of Subbotin quadratic splines, for the new cubic splines, since the values of the second derivatives at the nodes are defined and hence in this instance also continuity adds no additional conditions. Thus we can determine the 4n parameters of the n-spline functions over n-subintervals for the proposed new cubic spline function. In section-4.2, of this study, we present the derivation of the proposed new cubic spline. In section-4.3 of this study, we present the applications of proposed cubic spline to integral function approximations and to the numerical integration over curved domains in 2-space. Applications to integral function approximations are illustrated for the indefinite integral of Runge function, logarithmic function and normal distribution. This application is useful in the construction of table of integrals. As an application of numerical integration over curved domains the computation of some typical test integra,s over a lunar model considered. We also compare the function approximations of the proposed new spline with the standard cubic spline (natural, clamped and not a knot) [4,5,9,12,13,14,17,23,26,30,35,36,40,43,45,51,53,56].

4.2. The Interpolation Problem

We shall apply a useful approximation process, first proposed by Subbotin [1967] and latter by Kammerer, Reddien and Varga [30] see the texts [13,14]. We propose a cubic spline, where the nodes for the interpolation are choosen to be the first and the last

and two points between the knots. It is assumed that the knots are defined as the points where the spline function is permitted to change in form from one polynomial to another and the nodes are the points where the values of splines are specified.

4.2.1 THEOREM :

Let $\{t_i\}_{i=1}^{n+1}$ be a nonuniform partition of the interval $\left[t_{1,}\, t_{n+1}\right]$. Denote by $Q_i(x)$, the cubic polynomial over the interval $t_i \le x \le t_{i+1}$ *and* $h_i = t_{i+1} - t_i$ the length of the interval. Let us further assume that we are given the real numbers $\{y_i\}_{i=1}^{2n+2}$ such that the cubic spline interpolant $Q(x)$ over the interval $\left[t_{1,}\, t_{n+1}\right]$ satisfies

$$T_1 = t_{1,}\;\; T_{2i} = t_i + \frac{h_i}{3},\; T_{2i+1} = t_i + \frac{2h_i}{3} \qquad \text{.....(4.1)}$$

$$Q(T_i) = y_i \qquad (i = 1,2,3,.....,2n+2) \qquad \text{....(4.2)}$$

then there exists a unique cubic polynomial

$$Q_i(x) = a_i + b_i u + c_i u^2 + d_i u^3 \qquad \text{.....(4.3a)}$$

Where, $d_i = \dfrac{(Z_{i+1} - Z_i)}{6}$,

$$c_i = \frac{Z_i h_i^{\,2}}{2},$$

$$b_i = 3(y_{k+1} - y_k) - \frac{7}{54} h_i^{\,2} Z_{i+1} - \frac{10}{27} Z_i h_i^2$$

$$a_i = \frac{(Z_{i+1} + 2Z_i) h_i^2}{27} + (2y_k - y_{k+1})$$

$$k = 2i, \quad t_i \le x \le t_{i+1,} \qquad u = \frac{(x - t_i)}{h_i} \qquad (0 \le u \le 1)$$

and $Z_i = Q^{//}(t_i), \qquad i = 1,2,.....n+1$

Proof: The constructive proof of the above statement can be easily obtained.

4.2.2 Determination of Z_i values

The knots create n subintervals $\left[t_{i,}\, t_{i+1}\right], \quad (i = 1,2,.....,n)$ and in each of them Q can be a different cubic polynomial, that is on $\left[t_{i,}\, t_{i+1}\right]$. Q is equal to a cubic polynomial Q_i. Since Q is a cubic polynomial, Q, $Q^{/}$ and $Q^{//}$ should be continuous. Thus $Z_i = Q^{//}(t_i)$ is well defined. However, we do not yet know the values of $Z_{1,}\, Z_{2,}.....Z_{n+1}$. We shall now show how to determine the unknowns Z_i's. The cubic spline over $\left[t_{i,}\;\; t_{i+1}\right]$, $Q_i(x)$ as given in eqns must also satisfy the following continuity relations.

$$Q_{i-1}(t_i) = Q_i(t_i) \qquad \text{....(4.4)}$$

$$Q'_{i-1}(t_i) = Q'_i(t_i) \qquad \text{....(4.5)}$$

From eqns (4.3) and (4.4), eqns (4.3) and (4.5), the continuity of the spline function and its first derivatives imply the following for $(i = 2,3,.....,n)$:

$$-h_{i-1}^{\,2} Z_{i-1} + 2\left(h_i^2 - h_{i-1}^{\,2}\right) Z_i + h_i^2 Z_{i+1} = 27\left[(2y_{2i-1} - y_{2i-2}) - (2y_{2i} - y_{2i+1})\right]$$

$$\text{.....}(4.6)$$

$$7h_{i-1}Z_{i-1}+20\left(h_i+h_{i-1}\right)Z_i+7h_iZ_{i+1}=162\left[\frac{y_{2i+1}-y_{2i}}{h_i-\left(y_{2i-1}-y_{2i-2}\right)}\right]$$

$$.....(4.7)$$

Now eliminating Z_{i+1} between eqns (4.6) and (4.7), we obtain

$$7h^2_{i-1}\left(h_i+h_{i-1}\right)Z_{i-1}+2h_{i-1}\left(h_i+h_{i-1}\right)\left(3h_i+7h_{i-1}\right)Z_i=162h_i\left(y_{2i-2}-y_{2i-1}\right)$$
$$+27h_{i-1}\left(-14y_{2i-1}+7y_{2i-2}+8y_{2i}-y_{2i+1}\right)$$
$$\left(i=2,3,.....,.n\right) \qquad .....(4.8)$$

Finally, imposing the first and the last interpolation conditions:

$$Q\left(T_1\right)=Q\left(t_1\right)=y_1 \qquad .....(4.9)$$

$$Q\left(T_{2n+2}\right)=Q\left(t_{n+1}\right)=y_{2n+2} \qquad .....(4.10)$$

We obtain

$$h_1^2\left(2Z_1+Z_2\right)=27\left(y_3-2y_2+y_1\right) \qquad ....(4.11)$$

$$h_n^2\left(Z_n+2Z_{n+1}\right)=27\left(y_{2n+2}-2y_{2n+1}+y_{2n}\right) \qquad ....(4.12)$$

Now eqns (4.8), (4.11) ands (4.12) can be expressed as a two band matrix:

$$\begin{bmatrix}
2h_1 & h_1 & 0 & 0 & 0 & & & & \\
\alpha_2 & \beta_2 & 0 & 0 & 0 & & & & \\
0 & \alpha_3 & \beta_3 & 0 & 0 & & & & \\
0 & 0 & \alpha_4 & \beta_4 & 0 & & & & \\
- & - & - & - & - & - & - & - & \\
- & - & - & - & - & - & - & - & \\
 & & & \alpha_{n-2} & \beta_{n-2} & 0 & 0 & 0 & \\
 & & & 0 & \alpha_{n-1} & \beta_{n-1} & 0 & 0 & \\
 & & & 0 & 0 & \alpha_n & \beta_n & 0 & \\
 & & & 0 & 0 & 0 & h_n & 2h_n &
\end{bmatrix}
\begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ - \\ Z_{n-3} \\ Z_{n-2} \\ Z_{n-1} \\ Z_n \\ Z_{n+1} \end{bmatrix}
=
\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ - \\ r_{n-3} \\ r_{n-2} \\ r_{n-1} \\ r_n \\ r_{n+1} \end{bmatrix}$$

$$.....(4.13)$$

$$r_1 = 27\left(y_3 - 2y_2 + y_1\right)/h_1$$
$$r_{n+1} = 27\left(y_{2n+2} - 2y_{2n+1} + y_{2n}\right)$$
$$\alpha_i = 7h_{i-1}{}^2\left(h_i + h_{i-1}\right)$$

Where, The coefficient

$$\beta_i = 2h_{i-1}\left(h_i + h_{i-1}\right)\left(3h_i + 7h_{i-1}\right)$$
$$r_i = 162h_i\left(y_{2i-2} - y_{2i-1}\right) + 27h_{i-1}\left(-14y_{2i-1} + 7y_{2i-2} + 8y_{2i} - y_{2i+1}\right)$$
$$\left(i = 2,3,.....n\right) \qquad .....(4.14)$$

matrix of eqn (4.13) is a two band matrix and diagonally dominant. It can be shown that such a matrix is nonsingular, which means that the system has a unique solution.
The two band matrix eqns (4.13) - (4.14) can be solved by the following recurrence relations.

$$Z_i = \frac{\left(r_i - \alpha_{i-1}\right)}{\beta_i}, \qquad \left(i = 3,4,.....,n\right) \qquad ....(4.15)$$

With $Z_1,\ Z_2$ *and* Z_{n+1} given by :

$$Z_1 = \frac{\left(r_1\beta_2 - r_2h_1\right)}{\left(2h_1\beta_2 - h_1\alpha_2\right)}$$
$$Z_2 = \frac{\left(2h_1r_2 - r_1\alpha_2\right)}{\left(2h_1\beta_2 - h_1\alpha_2\right)}$$
$$Z_{n+1} = \frac{\left(r_{n+1}/h_n - Z_n\right)}{2} \qquad .....(4.16)$$

4.3. APPLICATIONS TO QUADRATURE

4.3.1 Integral Function Approximations

Consider the indefinite integral

$$S\left(x\right) = \int_a^x f\left(t\right)dt, \qquad x \in \left[a,b\right] \qquad ....(4.17)$$

We can choose an appropriate step size h and approximate the right side of eqn (4.17) for $x = a + h,\ a + 2h,.....$ inturn. Each integral can be estimated by the spline integration.

Letting $f\left(t\right) = S\left(t\right)$, we can write

$$S\left(x\right) = \int_a^x S\left(t\right)dt, \qquad x \in \left[a,b\right] \qquad ....(4.18)$$

Where a is the first break point. Since $S\left(x\right)$ can be viewed as composed of connected Cubic polynomial with the k^{th} cubic polynomial being

$$S_k\left(u\right) = a_k + b_ku + c_ku^2 + d_ku^3,$$

$$u = \frac{\left(x - x_k\right)}{h}, \qquad h = x_{k+1} - x_k \quad \forall k$$
$$x_k \le x \le x_{k+1,} \qquad 0 \le u \le 1$$
$$a = x_1 < x_2 < ..... < x_{k+1} < x_k = x \qquad .....(4.19)$$

Letting, $x = x_{k,}$ we can write

$$S\left(x_k\right) = \int_{x_1}^{x_2} S_1\left(u\right)dx + \int_{x_2}^{x_3} S_2\left(u\right)dx + ... + \int_{x_{k-1}}^{x_k} S_{k-1}\left(u\right)du + \int_{x_k}^{x_{k+1}} S_k\left(u\right)du$$
$$= \sum_{x_k}^{x_{k+1}} S_i\left(1\right), \qquad \left(0 \le u \le 1\right) \qquad .....\left(4.20a\right)$$

Where,

$$S_i(1) = \int_{x_i}^{x_{i+1}} S_i(u)dx = h\int_0^1 S_i(u)du$$

$$= h\left(a_i + \frac{b_i}{2} + \frac{c_i}{3} + \frac{d_i}{4}\right) \quad \ldots.(4.20b)$$

4.3.2 Computation of Table of Integrals

Example 1: Indefinite integral of Runge Function

$$I(x) = \int_{-1}^{x} \frac{dt}{\left(1+25t^2\right)} \ , \qquad x \in (-1,1] \qquad \ldots.(4.21)$$

Example 2: Logarithmic Function

$$I(x) = \int_{1}^{x} \frac{dt}{(1+t)} \ , \qquad x \in (1,5] \qquad \ldots.(4.22)$$

Example 3: Normal Distribution

$$I(x) = \int_0^x \frac{dt}{\left(\sqrt{2\pi}\right)} \exp\left(\frac{-t^2}{2}\right) dt \ , \qquad x \in (0,4] \qquad \ldots..(4.23)$$

The approximations to $I(x)$ will produce table of integrals in each case.

4.3.3. Numerical Integration over Curved Domains in 2- Space

Consider the integral [5,51]

$$\iint_{\pi_{xy}} f(x,y)dx\,dy \qquad \ldots.....(4.24)$$

Where π_{xy} is the curved boundary.

Let the curved boundary be discritised into N curved arcs by points $(x_k, y_k), (k = 1,2,3,.....N+1)$ and the i^{th} arc is joined by the vertices $V_i = (x_i, y_i)$ *and* $V_{i+1} = (x_{i+1}, y_{i+1})$, let us denote this curved arc as $\partial\pi_{i,\,i+1}$. Thus we represent the closed curved boundary as

$$\partial\pi_{xy} = \partial\pi_{1,2} + \partial\pi_{2,3} + ...\partial\pi_{N,N+1}$$

We have by application of Green's theorem

$$\iint_{\pi_{xy}} f(x,y)dx\,dy = \oint \phi(x,y)\,dy$$

$$\partial\pi_{xy} = \sum_{i=1}^{N} \partial\pi_{i,i+1,} \qquad \ldots..(4.25a)$$

$$= \sum_{1}^{N} \int_{\partial\pi_{i,i+1}} \varphi(x,y)\,dy \qquad \ldots..(4.25b)$$

Where, $\phi(x,y) = \int_{\alpha}^{x} f(u,y)\,du \qquad \ldots..(4.26)$

In which α is fixed and $x_i \neq \alpha \quad (i = 1,2,.....,N+1)$

Let us now choose the substitution

$$u = \left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right) \qquad \ldots..(4.27)$$

Substituting for u from eqn (4.27) into eqn (4.26)

We obtain

$$\phi(x,y) = \left(\frac{x-\alpha}{2}\right)\int_{-1}^{1} f\left(\left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right), y\right)ds \qquad \ldots..(4.28)$$

Substituting the above expression for $\phi(x,y)$ from eqn (4.28) into eqn (4.25b), we obtain

$$\iint_{\pi_{xy}} f(x,y)\,dx\,dy = \sum_{i=1}^{N} \int_{\partial\pi_{i,i+1}} \phi(x.y)\,dy$$

$$= \sum_{i=1}^{N} \int_{\partial\pi_{i,i+1}} \left(\frac{x-\alpha}{2}\right)\left[\int_{-1}^{1} f\left(\left(\frac{x-\alpha}{2}\right)s + \left(\frac{x+\alpha}{2}\right), y\right) ds\right] dy \qquad \text{....(4.29)}$$

By using suitable parametric form of splines for x and y with parameter t, we can write

$$\iint_{\pi_{xy}} f(x,y)dxdy =$$

$$\sum_{i=0}^{N} \int_{-1}^{1} \left(\frac{x^i(t)-\alpha}{2}\right)\left[\int_{-1}^{1} f\left\langle\left(\frac{x^i(t)-\alpha}{2}\right)s + \left(\frac{x^i(t)+\alpha}{2}\right), y^i(t)\right\rangle ds\right] (y^i(t))\text{dt}$$

Where $x^i(t)$ *and* $y^i(t)$ refer to parametric splines of order higher than two over the curved arc joining the points (x_i, y_i) *and* (x_{i+1}, y_{i+1}).

5.4. Application Example

5.4.0 A Lunar Model

We shall apply the theoretical developments of the previous sections to compute the integral $I\,I$ over a typical domain in the shape of a lunar model.

We shall now consider the following curved domain in the shape of a lunar model whose boundary is composed of two circular arcs. The outer circular arc satisfies the equation $(x-1/2)^2 + (y-1/2)^2 = 1/4$ and the inner circular arc satisfies the equation $x^2 + y^2 = 1/4$.

This is shown in Fig 1.
Numerical scheme developed in section 5.3 of this chapter requires the suitable form of parametric equations to describe the curved boundary of the given domain π_{xy} as stated in equations (5.28) to (5.37). In some cases, the boundary of the curved domain is described by exact mathematical relations and it may be thus possible for us to transform this into explicit parametric equations. We could always generate parametric equations of the curved boundary in the form of spline interpolating polynomials of order 2,3,4,……. Etc. This latter technique is very useful even when the curved boundary is described by a set of discrete coordinate points. We shall now explain the above two methods of parametric equations for the above typical example of lunar model.

5.4.1 Explicit form of Parametric Equations. The following parametric relations can be immediately obtained for the lunar model described above in Fig 1.

(1) Outer Circular Arc:
In Fig1, we have shown that the outer circular arc is the boundary curve over the fourth, first and the second quadrants and this part of the boundary is described by the equation:

$$(x-1/2)^2 + (y-1/2)^2 = 1/4 \qquad \text{….(5.39)}$$

Hence the parametric equations are :

$$x = \frac{1}{2} + \left(\frac{1}{2}\right)\cos\theta, \quad y = \frac{1}{2} + \left(\frac{1}{2}\right)\sin\theta$$

$$\theta \in \left([0, \pi/2], [\pi/2, \pi], [3\pi/2, 2\pi]\right) \qquad \text{.....(5.40)}$$

(2) Inner Circular Arc :

Again referring to the Fig 1, we have shown that the inner circular arc is the boundary curve over the third quadrant and this part of the boundary is described by the relation:

$$x^2 + y^2 = 1/4 \qquad \ldots.(5.41)$$

Hence the parametric equations are :

$$x = \left(\frac{1}{2}\right)\cos\theta, \quad y = \left(\frac{1}{2}\right)\sin\theta, \qquad \left(\frac{\pi}{2} \le \theta \le 0\right) \qquad \ldots.(5.42)$$

5.4.2. Cubic Spline Interpolants as Parametric Equations

The parametric equations described above can be represented by cubic splines over each piece of curved boundary. One simple representation is to generate the boundary data for the entire model we shall first compute the boundary coordinates for the lunar model by numbering the nodes from the beginning of fourth quadrant to the end of the third quadrant. The model consists of four quadrants. We suppose that d- divisions of equal area in the form of circular sectors are created in each quadrant. This requires $(d+1)$ nodes for each quadrant. Thus, we require $(4d+1)$ nodes to cover the lunar model which is made up of four circular arcs. Outer circular arc covering the fourth, first and the second quadrant is described by the equation $(x-1/2)^2 + (y-1/2)^2 = 1/4$ and the inner circular arc covers the third quadrant and it is described by the equation $x^2 + y^2 = 1/4$. This is clear from the lunar model shown in Fig 1. Let the boundary values of the x *and* y coordinates for the lunar model be stored in Vectors a *and* b respectively. As said above, we have (4d+1) nodes covering the entire lunar model and the boundary nodes for the end points covering the quadrants are :

Fourth Quadrant End Points

$$a(1) = \frac{1}{2}, \quad b(1) = 1$$

$$a(d+1) = 1, \quad b(d+1) = \frac{1}{2} \qquad \ldots.(5.43a)$$

First Quadrant End Points

$$a(d+1) = 1, \quad b(d+1) = \frac{1}{2}$$

$$a(2d+1) = \frac{1}{2}, \quad b(2d+1) = 1 \qquad \ldots.(5.43b)$$

Second Quadrant End Points

$$a(2d+1) = \frac{1}{2}, \quad b(2d+1) = 1$$

$$a(3d+1) = 0, \quad b(2d+1) = \frac{1}{2} \qquad \ldots.(5.43c)$$

Third Quadrant End Points

$$a(3d+1) = 0, \quad b(2d+1) = \frac{1}{2}$$

$$a(4d+1) = a(1), \quad b(4d+1) = b(1) \qquad \ldots.(5.43d)$$

We can now generate the remaining boundary coordinates for the quadrants

Over the fourth quadrant spanned by nodes 2 to d, we have

$$a(k+1) = \frac{1}{2} + \left(\frac{1}{2}\right)\cos\left(\frac{3\pi}{2} + \frac{k\pi}{2d}\right),$$

$$b(k+1) = \frac{1}{2} + \left(\frac{1}{2}\right)\sin\left(\frac{3\pi}{2} + \frac{k\pi}{2d}\right)$$

$$(k = 1, 2, \ldots\ldots\ldots, d-1) \qquad \ldots.(5.43e)$$

Over the first quadrant spanned by the nodes $(d+2)\ to\ 2d$, we have

$$a(d+k+1)=\frac{1}{2}+\left(\frac{1}{2}\right)\cos\left(\frac{k\pi}{2d}\right),$$

$$b(d+k+1)=\frac{1}{2}+\left(\frac{1}{2}\right)\sin\left(\frac{k\pi}{2d}\right)$$

$$(k=1,2,.....,d-1) \quad \text{....(5.43f)}$$

(Vii) Over the second quadrant spanned by the nodes (2d+2) to 3d, we have

$$a(2d+k+1)=\frac{1}{2}+\frac{1}{2}\cos\left(\frac{\pi}{2}+\frac{k\pi}{2d}\right)$$

$$b(2d+k+1)=\frac{1}{2}+\frac{1}{2}\sin\left(\frac{\pi}{2}+\frac{k\pi}{2d}\right)$$

$$k=(1,2,.....d-1), \quad \text{.....(5.43g)}$$

Over the third quadrant spanned by the nodes $(3d+2)\ to\ 4d$, we have

$$a(n+1-k)=r\cos\left(\frac{k\pi}{2d}\right),$$

$$b(n+1-k)=r\sin\left(\frac{k\pi}{2d}\right)$$

$$(k=1,2,.....,d-1)(n=4d)$$

……(5.43h)

We have computed this boundary coordinate data in the MATLAB program buffer.m which returns the vectors $a\ and\ b$ for use in the computation of spline interpolants. Using the boundary coordinate data $(a\ ,\ b)$ the spline interpolants for $'4d'$ curved arcs can be determined. We note that the i-th spline interpolant have endpoints $(a_i\ ,b_i)$ and (a_{i+1},b_{i+1}). We have applied the data generated as above to well known cubic splines viz : the natural cubic spline, the clamped cubic spline and the not a knot cubic spline.

The above boundary data is not suitable for application of the new cubic spline. Because the new cubic spline chooses the first and the last points of the curved boundary and two points between the knots as nodes for interpolation. If this is applied to the lunar model for which the first and the last point is (1/2,0), we cannot get the solution to the integration of an arbitrary function over the lunar model. Hence, we have to apply the new cubic spline over pieces of curved arcs. This is achieved as follows:

(i) We have generated the boundary data over the four quadrants (see Fig 2) in a piecewise manner and named it as buffer_ quadrant.m.

(ii) We have generated the boundary data over the eight octants in a piecewise manner and named it as buffer_octant.m (see Fig 3)

(iii) We have generated the boundary data over the 12-sectors (obtained by division of a quadrant into three sectors of equal size) in a piecewise manner and named it as buffer_12tant.m. (see Fig 4).

(iv) We have generated the boundary data over the 16-sectors (obtained by division of a quadrant into four sectors of equal size) in a piecewise manner and named it as buffer_16tant.m. (see Fig 5) and finally

(iv) We have generated the boundary data over the 20-sectors (obtained by division of a quadrant into five sectors) in a piecewise maner and named it as buffer_20tant.m (see Fig 6).

5.4.3. Examples of Integrals

We shall now consider the following integrals

$$I I_i = \iint_{\pi_{xy}} f_i(x,y)dxdy, \qquad (i = 1,2,3,4,5,6,7) \qquad \text{.....(5.44)}$$ Where π_{xy} is the curved domain described in Fig 1. And the test integrands $f_i(x,y)$ are the following [51]

$$f_1(x,y) = (x+y)^{19},$$

$$f_2(x,y) = \sqrt{(x-0.5)^2 + (y-0.5)^2},$$

$$f_3(x,y) = \exp\left\{-\left((x-0.5)^2 + (y-0.5)^2\right)\right\},$$

$$f_4(x,y) = \exp\left\{-100\left((x-0.5)^2 + (y-0.5)^2\right)\right\}$$

$$f_5(x,y) = \frac{3}{4}\exp\left\{-\frac{1}{4}\left((9x-2)^2 + (9y-2)^2\right)\right\} + \frac{3}{4}\exp\left\{-\frac{1}{49}(9x+1)^2 - \frac{1}{10}(9y+1)\right\}$$
$$+ \frac{1}{2}\exp\left\{-\frac{1}{4}\left((9x-7)^2 + (9y-3)^2\right)\right\} - \frac{1}{5}\exp\left\{-\left((9y-4)^2 + (9y-7)^2\right)\right\},$$

$$f_6(x,y) = 1$$

$$f_7(x,y) = \cos\left(20(x+y)\right) \qquad \text{.....(5.45)}$$

5.4.4. Explicit Method

We have implemented the numerical scheme of section-5.3 by using the explicit form of parametric equations described in section-5.4.1 to compute the integrals stated in eqns (5.43 – 5.44). The following MATLAB programs are developed for this purpose.

(1) parametric_integration_lunar_model.m

(2) new_parametric_integration_lunar_model.m

We may note that in the above programs, the boundary curve of lunar model is exactly represented by the parametric equations given in eqns (40)-(42). In the above explicit representation of boundary curve, as few as four points on the boundary curve of the lunar model with Gauss Legendre Quadrature rules of the order (32,36) or (32,40) in the bivariates gives us the near exact results.

We shall next consider the method when the boundary curve is described by a set of discrete coordinate points. We can use this information to obtain parametric equations in the form of spline interpolating polynomials. This is further explained in the next section for the four cubic splines described in section-5.2.1 and5.2.2.

5.4.5. Spline Method

We have also implemented the numerical scheme of section 5.3 by using the cubic spline interpolating polynomials as parametric equations as explained in section 5.4.2 to compute the integrals of section-5.4.3, eqn (5.43 – 5.44). The following MATLAB programs are developed for this purpose.

(1) greens_natural_cubic_spline_integration.m

(2) greens_slope_cubic_spline_integration.m

(3) greens_notaknot_cubicspline_integration.m

(4) new_cubic_subbotin_spline_integration_.m

(5) new_cubic_subbotin_spline_integration_octant.m

(6) new_cubic_subbotin_spline_integration_12tant.m

(7) new_cubic_subbotin_spline_integration_16tant.m

(8) new_cubic_subbotin_spline_integration_20tant.m

(9) buffer.m

(10) buffer_quadrant.m

(11) buffer_octant.m

(12) buffer_12tant.m

(13) buffer_16tant.m

(14) buffer_20tant.m

The performance of new cubic spline when used as a continuous boundary data for the entire lunar model is unsatisfactiory, since in this case the end points (first and last) coincide. The performance improves when the boundary data is given separately for each of the four quadrants. The performance improves further when the boundary data is given separately for each octant. We have then divided the quadrants into three, four and five sectors. This gives 12, 16, and 20 sectors for the entire lunar model. We have computed the integral approximations by using the boundary data thus created in buffer_quadrant (lunar model with four quadrants),buffer_octant (lunar model with 8- octanants), buffer_12tant (lunar model with 12 sectors), buffer_16tant (lunar model with 16-sectors) and finally buffer_20tant(lunar model with 20-sectors). This method of subdivisions gave us improved results. These divisions of the lunar model are displayed in Fig 2-6.

5.5. CONCLUSION

This paper studies the performance of four cubic splines.Three of these are the interpolating cubic splines, viz, the natural cubic spline, clamped cubic spline and the not a knot cubic spline [12]. The fourth one is a new cubic spline which was recently proposed by the authors [42]. This new cubic spline interpolates at the first and the last knots and the two points located at the trisections between the knots. This chapter first presents brief derivation of these cubic splines . Then a numerical scheme based on Green's theorem, Gauss Legendre quadrature rules and parametric equations of the curved boundary is developed. In practical situations, parametric equations of the curved boundary may not be available, instead we may have a discrete set of points on the curved boundary . These difficulties may be overcome by use of splines. This is demonstrated here by the use of the cubic splines as parametric equations for the curved boundary.It is found that natural cubic spline and not a knot cubic spline perform very well. The performance of clamped spline is not not a very satisfactory. We also note that for a fixed number of spline, the performance of the new cubic spline is the best among the cubic splines studies .

Table-1b.

Natural cubic spline Interpolation
Output of the MATLAB commands:
n=32,
for div =20:20:100
greens_natural_cubic_spline_integration(div,n)
end
for div=100:100:500
greens_natural_cubic_spline_integration(div,n)
end

div	I	Integral II (i)	div	i	Integral II (i)
20	1	638.557127934958	40	1	638.557413717227
	2	0.206467625865532		2	0.206467698050052
	3	0.572637215563281		3	0.572637205025403
	4	0.0313718520466439		4	0.03133718519958398
	5	0.210503893101308		5	0.210503819542919
	6	0.642699040131155		6	0.64269907910361
	7	0.00628962625774288		**7**	0.00628958241771448
	1	638.557428988812		1	638.557431557984
	2	0.206467701914868		2	0.206467702565589

60	3	0.572637204463626	80	3	0.572637204369081
	4	0.0313718519931249		4	0.0313718519926681
	5	0.210503815625262		5	0.210503814967122
	6	0.642699081186213		6	0.642699081536574
	7	0.00628958139542727		7	0.00628958126821149
100	1	638.557432260005	200	1	638.55743271658
	2	0.206467702743479		2	0.206467702859223
	3	0.572637204343235		3	0.572637204326418
	4	0.03133718519925434		4	0.03133718519924622
	5	0.210503814787426		5	0.210503814670642
	6	0.642699081632309		6	0.642699081694574
	7	0.00628958123811101		7	0.00628958122059999
300	1	638.557432741006	400	1	638.557432745114
	2	0.206467702865418		2	0.206467702866461
	3	0.572637204325515		3	0.572637204325363
	4	0.0313718519924578		4	0.0313718519924572
	5	0.210503814664401		5	0.210503814663351
	6	0.642699081697903		6	0.642699081698464
	7	0.00628958121976457		7	0.00628958121962791

Output of the MATLAB commands:

>>greens_natural_cubic_spline_integration(750,32),
>>greens_natural_cubic_spline_integration(1000,32),
>>green_natural_cubic_spline_integration(1250,32),

div	Div	II(i)	div	i	II(i)
500	1	638.557432746239	750	1	638.557432746864
	2	0.206467702866746		2	0.206467702866904
	3	0.572637204325324		3	0.5726372043253
	4	0.03133718519924569		4	0.0313718519924567
	5	0.210503814663063		5	0.210503814662904
	6	0.642699081698617		6	0.642699081698699
	7	0.00628958121959102		7	0.00628958121957066
1000	1	638.557432746972	1250	1	638.557432746996
	2	0.206467702866932		2	0.206467702866938
	3	0.572637204325296		3	0.572637204325296
	4	0.0313718519924567		4	0.0313718519924568
	5	0.210503814662877		5	0.21050381466287
	6	0.642699081698717		6	0.64269908169872
	7	0.00628958121956724		7	0.00628958121956634

Output of the MATLAB command:

>> **greens_natural_cubic_spline_integration(1500,32)**

div	i	II(i)
1500	1	638.557432747008
	2	0.206467702866941
	3	0.572637204325295
	4	0.0313718519924567
	5	0.210503814662869
	6	0.642699081698721
	7	0.00628958121956598

Table-2.

Clamped cubic spline Interpolation

Output of the MATLAB command:
n=32,
for div=10:10:50
greens_slope_cubic_spline_integration(div,n)
end
for div=100:100:700
greens_slope_cubic_spline_integration(div,n)
end

div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
10	1	638.552534346567	20	1	638.557127935771
	2	0.206744877798084		2	0.206534998865954
	3	0.572978213291479		3	0.572723631862649
	4	0.0313719004837743		4	0.0313718520394426
	5	0.210693854665818		5	0.210551333293605
	6	0.643174363957289		6	0.642817844599741
	7	0.00535726607153629		**7**	0.00617575791127833
30	1	638.557372586678	40	1	638.557413717446
	2	0.20649732133615		2	0.206484282564622
	3	0.572675797184113		3	0.572658965359046
	4	0.0313718520030011		4	0.0313718519958174
	5	0.210524834616364		5	0.210515598882035
	6	0.642751852864198		6	0.642728763047937
	7	0.00625294350657109		7	0.00627228281234315
50	1	638.55742495359	100	1	638.557432260042
	2	0.206478283104479		2	0.206470332525949
	3	0.572651151909444		3	0.572640701668697
	4	0.0313718519938351		4	0.0313718519925428
	5	0.210511338853623		5	0.210505685358376
	6	0.64271807704191		6	0.6427038302944
	7	0.00627964747087298		7	0.00628758942361841
200	1	638.557432716589	300	1	638.55743274101
	2	0.206468358312474		2	0.206467993862576
	3	0.572638079989583		3	0.572637593707668
	4	0.0313718519924621		4	0.0313718519924578
	5	0.210504280849164		5	0.210504021623355
	6	0.642700268832636		6	0.642699609312559
	7	0.00628913498685772		7	0.00628938990684939
div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
400	1	638.557432745116	500	1	638.557432746241
	2	0.206467866521443		2	0.20646780757038
	3	0.572637423408787		3	0.57263734456023
	4	0.0313718519924572		4	0.0313718519924569
	5	0.210503931011104		5	0.210503889099711
	6	0.642699378481263		6	0.642699271639477
	7	0.00628947550962658		**7**	0.00628951428196982
600	1	638.557432746642	700	1	638.557432746816
	2	0.20646777552673		2	0.206467756231287
	3	0.572637301720628		3	0.572637275886278
	4	0.0313718519924568		4	0.0313718519924568
	5	0.210503866342987		5	0.210503852625538
	6	0.642699213601999		6	0.642699178607241
	7	0.00628953506285261		7	0.00628954747901685

Table-3.

Not a knot cubic spline Interpolation

Output of the MATLAB command:

n=32,

for div=10:10:50

greens_notaknot_cubic_spline_integration(div,n)

end

for div=100:100:500

greens_notaknot_cubic_spline_integration(div,n)

end

div	i	Integral II (i)	div	i	Integral II (i)
10	1	638.552534259336	20	1	638.557127935179
	2	0.206722048917279		2	0.206531614809103
	3	0.573040421930193		3	0.572737604866036
	4	0.0313718563095617		4	0.0313718520466205
	5	0.21074984737627		5	0.210565476911855
	6	0.64321371776262		6	0.642827629904008
	7	0.00591825500381323		**7**	0.00618633665511545
30	1	638.55737258642	40	1	638.557413717291
	2	0.2064961580339		2	0.20648372312569
	3	0.572681775741298		3	0.572662262285842
	4	0.031371852003152		4	0.0313718519958401
	5	0.210531249419807		5	0.210519255593792
	6	0.642756204921747		6	0.642731211677011
	7	0.00624317966543824		7	0.00626332570702456

Output of the MATLAB command:

n=32,

for div=1000:500:1500

greens_notaknot_cubic_spline_integration(div,n) end

div	i	Integral II (i)	div	i	Integral II (i)
50	1	638.557424953487	100	1	638.557432260015
	2	0.206477961338995		2	0.206470270263378
	3	0.572653236430019		3	0.572641209955162
	4	0.0313718519938425		4	0.0313718519925434
	5	0.210513700383575		5	0.210506287913633
	6	0.642719644341209		6	0.642704222178105
	7	0.00627270976329171		**7**	0.00628532210769751
200	1	638.557432716582	300	1	638.557432741007
	2	0.206468345066564		2	0.206467988341126
	3	0.572638205445348		3	0.572637649225745
	4	0.0313718519924622		4	0.0313718519924578
	5	0.210504433209714		5	0.210504089607663
	6	0.642700366807241		6	0.642699652857131
	7	0.00628851013726533		7	0.00628910416148747

400	1	638.557432745115	500	1	638.55743274624
	2	0.206467863460855		2	0.206467805652691
	3	0.572637454570126		3	0.572637364477523
	4	0.0313718519924572		4	0.0313718519924569
	5	0.210503969329222		5	0.210503913653146
	6	0.642699402975145		6	0.642699287315579
	7	0.0062893125773472		7	0.00628940917244047
1000	1	638.557432746973	1500	1	638.557432747009
	2	0.206467728566208		2	0.206467714289258
	3	0.572637244361128		3	0.572637222118671
	4	0.0313718519924568		4	0.0313718519924567
	5	0.210503839412346		5	0.210503825662919
	6	0.642699133102919		6	0.64269910454503
	7	0.00628953814856751		7	0.00628956206797184

Table-4a

New Cubic Spline (presented in this study)

Output of the MATLAB command:

n=32,

for i=3:3:12

for i=15:15:30

new_cubic_spline_integration(div,n) end

div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
3	1	6.391130288756544e+02	6	1	6.387111054609800e+02
	2	2.067899240998958e-001		2	2.063404376602289e-001
	3	5.731057208572563e-001		3	5.722717762215132e001
	4	3.137320817727273e-002		4	3.136713251470544e-002
	5	2.105788241527493e-001		5	2.103545799067718e-001
	6	6.433173788646684e-001		6	6.422943194152765e-001
	7	6.432506802333256e-003		**7**	6.306874627111190e-003
9	1	6.386640307844253e+02	12	1	6.385951681380394e+00
	2	2.064380684911331e-001		2	2.064580599947393e-001
	3	5.725350563918967e-001		3	5.726029548306322e-001
	4	3.137097644005311e-002		4	3.137172123275765e-002
	5	2.104601484499027e-001		5	2.104893459628005e-001
	6	6.425894688395121e-001		6	6.426624461784527e-001
	7	6.271743547502911e-003		7	6.288335552611027e-003
15	1	6.3857207135751e+002	30	1	6.385582797446807e+002
	2	2.064635815145924e-001		2	2.064674265211407e-001
	3	5.726223242392580e-001		3	5.726361937974155e-001
	4	3.137167202262090e-002		4	3.137181355517409e-002
	5	2.104975339088565e-001		5	2.105033886551050e-001
	6	6.426831790306097e-001		6	6.426980015960035e-001
	7	6.288253536647361e-003		7	6.289509037014241e-003

Table 4b

New Cubic Spline (presented in this study)

Output of the MATLAB command:

>>new_cubic_subbotin_spline_integration_octant(3,32) >>new_cubic_subbotin_spline_integration_octant(6,32)

>>new_cubic_subbotin_spline_integration_octant(9,32)

>>new_cubic_subbotin_spline_integration_octant(999,32)

end

div	i	Integral II (i)	div	i	Integral II (i)
3	1	6.388940596606644e+02	6	1	6.385774103036348e+02
	2	2.064933481696384e001		2	2.064614687945894e001
	3	5.726693943528192e001		3	5.72613941992794e-001
	4	3.137172169331359e002		4	3.137116438350909e002
	5	2.105064065468572e001		5	2.104937436621937e001
	6	6.427432657912181e001		6	6.426742969938228e001
	7	6.282173868529146e003		**7**	6.286538350795939e003
9	1	6.385626274347875e+02	999	1	6.385574327470600e+02
	2	2.064659481031991e001		2	2.064677028669286e001
	3	5.726308730888177e001		3	5.726372043252416e001
	4	3.137160736487444e002		4	3.137185199245444e002
	5	2.105010907895720e001		5	2.105038146628427e001
	6	6.426922896563760e001		6	6.426990816986677e001
	7	6.288780349470428e003		7	6.289581219562179e003

Table 4c

New Cubic Spline (presented in this study)

Output of the MATLAB command:

>>new_cubic_subbotin_spline_integration_12tant(3,32)

>>new_cubic_subbotin_spline_integration_12tant(6,32) etc…

div	i	Integral II (i)	div	i	Integral II (i)
3	1	6.386229822149559e+002	6	1	6.385611359968024e+002
	2	2.064729185480136e-001		2	2.064664913902128e-001
	3	5.726436782234794e-001		3	5.726326261390076e-001
	4	3.137184996632206e-002		4	3.137165406745945e-002
	5	2.105042889451878e-001		5	2.105018534770141e-001
	6	6.427080206016118e-001		6	6.426942025011199e-001
	7	6.284290868457155e-003		**7**	6.289258752274072e-003
9	1	6.385585969448636e+002	12	1	6.385578006925822e+002
	2	2.064673480992109e-001		2	2.064675860288256e-001
	3	5.726359250937346e-001		3	5.726367787664025e-001
	4	3.137177739846116e-002		4	3.137182757433277e-002
	5	2.105032571105280e-001		5	2.105036314145738e-001
	6	6.426977085328250e-001		6	6.426986257085657e-001
	7	6.289499709864270e-003		7	6.289553005355577e-003
15	1	6.385575915376644e+002	18	1	6.385575104652536e+002
	2	2.064676513374554e-001		2	2.064676773454989e-001
	3	5.726370176690186e-001		3	5.726371117882165e-001
	4	3.137184121527319e-002		4	3.137184670623126e-002
	5	2.105037343714198e-001		5	2.105037749626826e-001
	6	6.426988815361195e-001		6	6.426989824881576e-001
	7	6.289569448084290e-003		7	6.289575361473692e-003

24	1	6.385574579591390e+002	27	1	6.385574486077848e+002
	2	2.064676944430555e-001		2	2.064676975366959e-001
	3	5.726371738116616e-001		3	5.726371850226639e-001
	4	3.137185027117705e-002		4	3.137185090909232e-002
	5	2.105038016027627e-001		5	2.105038064078275e-001
	6	6.426990489821161e-001		6	6.426990610022468e-001
	7	6.289579316482529e-003		7	6.289580022453730e-003

Output of the MATLAB command:

>>new_cubic_subbotin_spline_integration_12tant(399,32) >>**new_cubic_subbotin_spline_integration_12tant(600,32)**

end

div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
30	1	6.385574432132142e+02	33	1	6.385574399290449e+02
	2	2.064676993336723e001		2	2.064677004332126e-001
	3	5.726371915316955001		3	5.726371955141064e-001
	4	3.137185127758350002		4	3.137185150192668e-002
	5	2.105038091951759e001		5	2.105038108992003e-001
	6	6.426990679813438e001		6	6.426990722513020e-001
	7	6.289580429624415e003		**7**	6.289580677796844e-003
399	1	6.385574327473703e+002	600	1	6.385574327470871e+002
	2	2.064677028668223e-001		2	2.064677028669208e-001
	3	5.726372043248555e-001		3	5.726372043252118e-001
	4	3.137185199243331e-002		4	3.137185199245245e-002
	5	2.105038146626782e-001		5	2.105038146628299e-001
	6	6.426990816982534e-001		6	6.426990816986353e-001
	7	6.289581219540130e-003		7	6.289581219561169e-003

Table 4d

New Cubic Spline (presented in this study)

Output of the MATLAB command:

n=32,

for div=3:3:30

>>new_cubic_subbotin_spline_integration_16tant(div,n)

end

>>new_cubic_subbotin_spline_integration_16tant(600,32) and (1002,32)

div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
3	1	6.385783468845478e+002	6	1	6.385586040908609e+002
	2	2.064693730110150e-001		2	2.064673208199806e-001
	3	5.726392656357869e-001		3	5.726357576264928e-001
	4	3.137184691872975e-002		4	3.137176607822508e-002
	5	2.105039585872179e-001		5	2.105031933278487e-001

	6	6.427019336136174e-001		6	6.426975396726528e-001
	7	6.289398893004205e-003		**7**	6.289407825638331e-003
div	**i**	**Integral II (i)**	**div**	**i**	**Integral II (i)**
9	1	6.385578292883993e+002	12	1	6.385575545343311e+002
	2	2.064675871370056e-001		2	2.064676651909758e-001
	3	5.726367885208132e-001		3	5.726370674075060e-001
	4	3.137182588707137e-002		4	3.137184378985729e-002
	5	2.105036314144340e-001		5	2.105037552743084e-001
	6	6.426986349954549e-001		6	6.426989349157353e-001
	7	6.289567130914111e-003		7	6.289575944846044e-003
div	i	**Integral II (i)**	**div**	**i**	**Integral II (i)**
15	1	6.385574857140819e+002	` 18	1	6.385574585808499e+002
	2	2.064676861991413e-001		2	2.064676946238377e-001
	3	5.726371441241691e-001		3	5.726371745198821e-001
	4	3.137184843017029e-002		4	3.137185025740071e-002
	5	2.105037885695144e-001		5	2.105038017867793e-001
	6	6.426990171034845e-001		6	6.426990497261756e-001
	7	6.289578900186174e-003		**7**	6.289579979823518e-003
21	1	6 .385574469248979e+002	24	1	6.385574411327430e+002
	2	2.064676982992334e-001		2	2.064677001461582e-001
	3	5.726371878262645e-001		3	5.726371944982452e-001
	4	3.137185104087244e-002		4	3.137185143021080e-002
	5	2.105038075447907e-001		5	2.105038104288848e-001
	6	6.426990639980924e-001		6	6.426990711562969e-001
	7	6.289580506585988e-003		7	6.289580779234485e-003
27	1	6.385574380234888e+002	30	1	6.385574362290178e+002
	2	2.064677011454128e-001		2	2.064677017258811e-001
	3	5.726371981091670e-001		3	5.726372002056817e-001
	4	3.137185163890405e-002		4	3.137185175928478e-002
	5	2.105038119870835e-001		5	2.105038128908974e-001
	6	6.426990750299816e-001		6	6.426990772791630e-001
	7	6.289580933765129e-003		7	6.289581026093169e-003
600	1	6.385574327470370e+002	1002	1	6.385574327470296e+002
	2	2.064677028669336e-001		2	2.064677028669422e-001
	3	5.726372043252623e-001		3	5.726372043252900e-001
	4	3.137185199245559e-002		4	3.137185199245645e-002
	5	2.105038146628509e-001		5	2.105038146628626e-001
	6	6.426990816986893e-001		6	6.426990816987196e-001
	7	6.289581219563993e-003		7	6.289581219566125e-003

Table 4e

New Cubic Spline (presented in this study)

Output of the program:

>>new_cubic_subbotin_spline_integration_20tant(3,32) >>new_cubic_subbotin_spline_integration_20tant(15,32)

>>new_cubic_subbotin_spline_integration_20tant(30,32) >>new_cubic_subbotin_spline_integration_20tant(300,32)

end

div	i	Integral II (i)	div	i	Integral II (i)
3	1	6.385660317821722e+002	15	1	6.385574556284504e+002
	2	2.064683905224022e-001		2	2.064676958825455e-001
	3	5.726380510997164e-001		3	5.726371791800592e-001
	4	3.137185066602721e-002		4	3.137185048248150e-002
	5	2.105038722989928e-001		5	2.105038036935046e-001
	6	6.427002543301441e-001		6	6.426990547013142e-001
	7	6.289311918415437e-003		**7**	6.289579946477400e-003
30	1	6.385574342458715e+002	300	1	6.385574327471832e+002
	2	2.064677023898656e-001		2	2.064677028668929e-001
	3	5.726372026080197e-001		3	5.726372043251121e-001
	4	3.137185189388298e-002		4	3.137185199244684e-002
	5	2.105038139199200e-001		5	2.105038146627865e-001
	6	6.426990798553596e-001		6	6.426990816985282e-001
	7	6.289581131440064e-003		7	6.289581219555243e-003

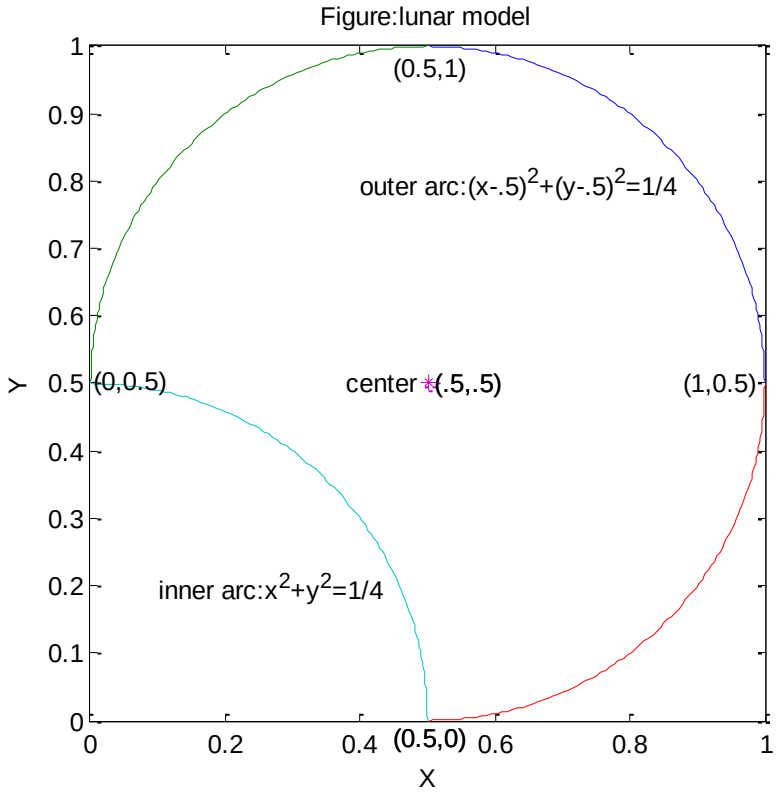


Figure:lunar model
(0.5,1)
outer arc:(x-.5)2+(y-.5)2=1/4
(0,0.5)
center *(.5,.5)
(1,0.5)
inner arc:x2+y2=1/4
(0.5,0)
X
Y

Fig-1

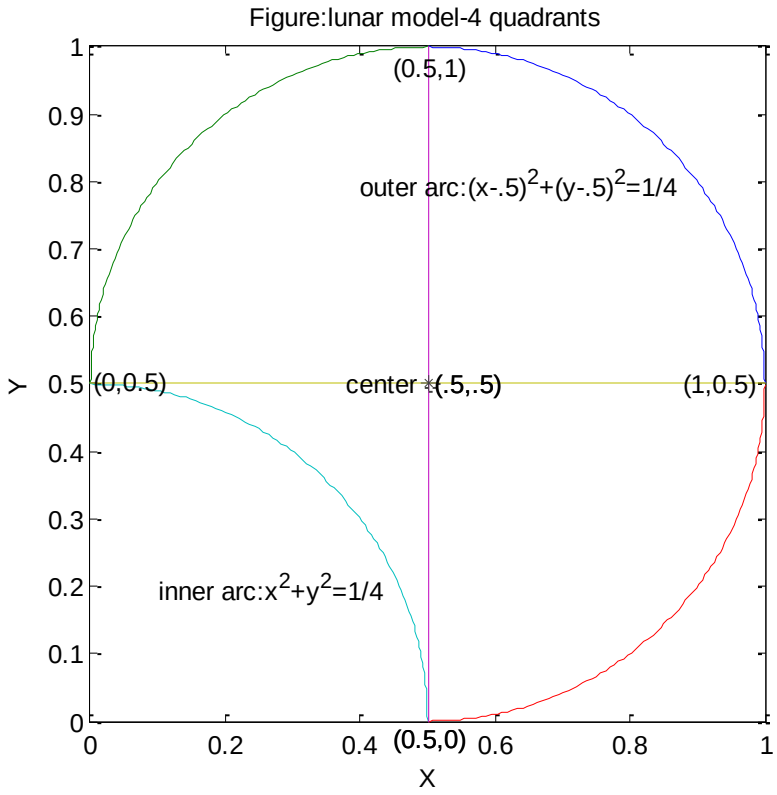


Figure:lunar model-4 quadrants
(0.5,1)
outer arc:(x-.5)2+(y-.5)2=1/4
(0,0.5)
center *(.5,.5)
(1,0.5)
inner arc:x2+y2=1/4
(0.5,0)
X
Y

FIG-2

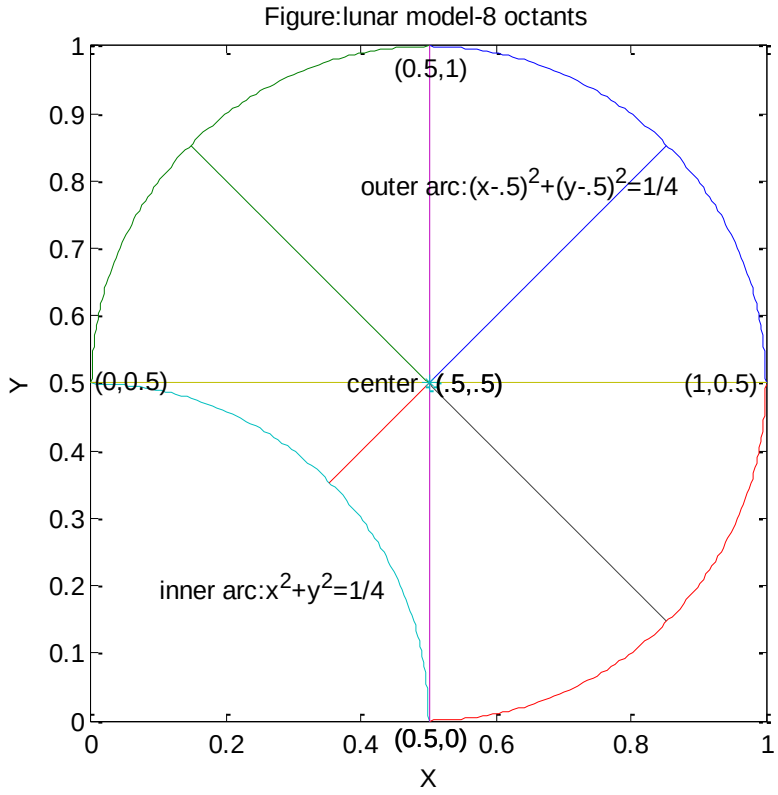


Figure:lunar model-8 octants
(0.5,1)
outer arc:(x-.5)2+(y-.5)2=1/4
(0,0.5)
center (.5,.5)
(1,0.5)
inner arc:x2+y2=1/4
(0.5,0)
X
Y

FIG-3

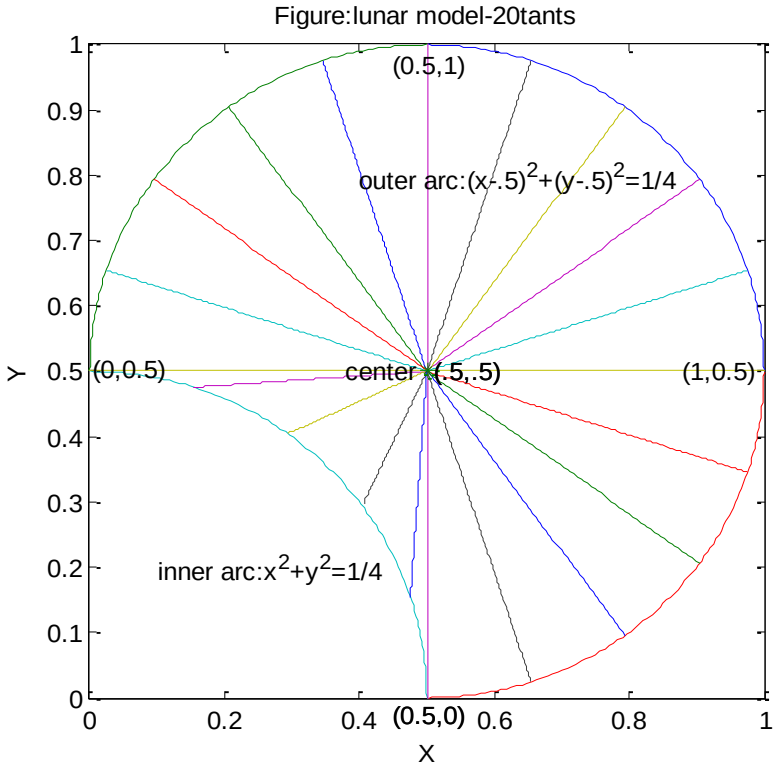


Figure:lunar model-20tants
(0.5,1)
outer arc:(x-.5)2+(y-.5)2=1/4
(0,0.5)
center (.5,.5)
(1,0.5)
inner arc:x2+y2=1/4
(0.5,0)
X
Y

FIG-5

FIG-6

SOME COMPUTER PROGRAMS

```
function[]=greens_natural_cubic_spline_integration(div,n)
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer(div);
%disp('A=')
%disp(A)
%disp('B=')
%disp(B)

%A=[0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;1/2-sqrt(1/8);0;sqrt(1/8);0.5]
%B=[0;1/2-sqrt(1/8);0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;sqrt(1/8);0]
for m=16:24
   if(m~=17)&(m~=22)
switch m
 case 16
    %disp(' fn=(x+y)^19')
 case 18
 %disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
  case 19
  %  disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
case 20
   %    disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
 case 21

    % disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')

     % disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
     % disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
     %   disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
      %  disp('fn=f1+f2+f3+f4;')
case 23
  % disp('fn=1')
   case 24
   %   disp('fn=cos(20*(x+y))')

otherwise
   disp('something wrong')
end
format long e
ccc=ccc+1;
   N=0;M=0;
   t0=0;x0=0.5;y0=0;
   for k=1:3*div
      t(k,1)=k;
   end
   x(1:3*div,1)=A(2:(3*div+1),1);y(1:3*div,1)=B(2:(3*div+1),1);

 [coefx]=cubic_natural_spline(t0,x0,t,x);
 [coefy]=cubic_natural_spline(t0,y0,t,y);
T0=0;X0=0;Y0=1/2;
for k=1:div
   T(k,1)=k;
end

X(1:div,1)=A(3*div+2:4*div+1,1);Y(1:div,1)=B(3*div+2:4*div+1,1);
[COEFX]=cubic_natural_spline(T0,X0,T,X);
[COEFY]=cubic_natural_spline(T0,Y0,T,Y);
for kk=1:3*div
          N=N+1; M=M+1;II(M,1)=0;
```

```
 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=coefx(N,4)+coefx(N,3)*ttp+coefx(N,2)*ttp^2+coefx(N,1)*ttp^3;
  yt=coefy(N,4)+coefy(N,3)*ttp+coefy(N,2)*ttp^2+coefy(N,1)*ttp^3;
   xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*coefy(N,1)*ttp^2+2*coefy(N,2)*ttp+coefy(N,3))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
N=0;
for kk=1:div
          N=N+1; M=M+1;II(M,1)=0;

   for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;


  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  XT=COEFX(N,4)+COEFX(N,3)*ttp+COEFX(N,2)*ttp^2+COEFX(N,1)*ttp^3;
  YT=COEFY(N,4)+COEFY(N,3)*ttp+COEFY(N,2)*ttp^2+COEFY(N,1)*ttp^3;
   XX=(XT-XC)*tq/2+(XT+XC)/2;
  WT=(XT-XC)*(3*COEFY(N,1)*ttp^2+2*COEFY(N,2)*ttp+COEFY(N,3))*wp*wq/4;
  II(M,1)=II(M,1)+WT*fnxy(m,XX,YT);
  end%end for q
end%end for p
end%end for kk
iii(ccc,1)=0;
for mm=1:M
   iii(ccc,1)=iii(ccc,1)+II(mm,1);
end
%disp(II)
%disp('----------------------------------')
%disp('X=')
%disp(X)
%disp('----------------------------------')
%disp('Y=')
%disp(Y)
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%end for m
%disp(iii)
format long g
for i=1:7
   III(i,1)=div;
   III(i,2)=i;
   III(i,3)=iii(i,1);
end
disp('-------------------------------------------------------------------------------------------')
disp('-------------------------------------------------------------------------------------------')
disp('                  div                    i          II(i)          ')
disp('-------------------------------------------------------------------------------------------')
disp(III)
disp('-------------------------------------------------------------------------------------------')
```

```
function[]=greens_slope_cubic_spline_integration(div,n)
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer(div);
%disp('A=')
%disp(A)
%disp('B=')
%disp(B)

%A=[0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;1/2-sqrt(1/8);0;sqrt(1/8);0.5]
%B=[0;1/2-sqrt(1/8);0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;sqrt(1/8);0]
for m=16:24
   if(m~=17)&(m~=22)
switch m
  case 16
     %disp(' fn=(x+y)^19')
  case 18
  %disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
   case 19
   %  disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
    %    disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
  case 21

     % disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
      % disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
       %disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
       %  disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
        % disp('fn=f1+f2+f3+f4;')
case 23
   %disp('fn=1')
   case 24
    %  disp('fn=cos(20*(x+y))')

otherwise
   disp('something wrong')
end
format long e
ccc=ccc+1;
   N=0;M=0;
   for k=1:3*div+1
      t(k,1)=k-1;
   end
   x(1:(3*div+1),1)=A(1:(3*div+1),1);y(1:(3*div+1),1)=B(1:(3*div+1),1);
 dx=[1/2 0];dy=[0 -1/2];
 [coefx]=spline_3(t,x,dx);
 [coefy]=spline_3(t,y,dy);
for k=1:div+1
  T(k,1)=k-1;
end
X(1:div+1,1)=A(3*div+1:4*div+1,1);Y(1:div+1,1)=B(3*div+1:4*div+1,1);
DX=[-1/2 0];DY=[0 1/2];
[COEFX]=spline_3(T,X,DX);
[COEFY]=spline_3(T,Y,DY);
for kk=1:3*div
          N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
```

```
  ttq=(1+tq)/2;
  xt=coefx(N,1)+coefx(N,2)*ttp+coefx(N,3)*ttp^2+coefx(N,4)*ttp^3;
  yt=coefy(N,1)+coefy(N,2)*ttp+coefy(N,3)*ttp^2+coefy(N,4)*ttp^3;
   xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*coefy(N,4)*ttp^2+2*coefy(N,3)*ttp+coefy(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
N=0;
for kk=1:div
          N=N+1; M=M+1;II(M,1)=0;

   for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  XT=COEFX(N,1)+COEFX(N,2)*ttp+COEFX(N,3)*ttp^2+COEFX(N,4)*ttp^3;
  YT=COEFY(N,1)+COEFY(N,2)*ttp+COEFY(N,3)*ttp^2+COEFY(N,4)*ttp^3;
   XX=(XT-XC)*tq/2+(XT+XC)/2;
  WT=(XT-XC)*(3*COEFY(N,4)*ttp^2+2*COEFY(N,3)*ttp+COEFY(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+WT*fnxy(m,XX,YT);
  end%end for q
end%end for p
end%end for kk
iii(ccc,1)=0;
for mm=1:M
   iii(ccc,1)=iii(ccc,1)+II(mm,1);
end
%disp(II)
%disp('-----------------------------------')
%disp('X=')
%disp(X)
%disp('-----------------------------------')
%disp('Y=')
%disp(Y)
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%end for m
%disp(iii)
format long g
for i=1:7
   III(i,1)=div;
   III(i,2)=i;
   III(i,3)=iii(i,1);
end
disp('-------------------------------------------------------------------------------------------------------')
disp('-------------------------------------------------------------------------------------------------------')
disp('                    div                    i          II(i)          ')
disp('-------------------------------------------------------------------------------------------------------')
disp(III)
disp('-------------------------------------------------------------------------------------------------------')

function[]=greens_notaknot_cubicspline_integration(div,n)
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer(div);
%disp('A=')
%disp(A)
%disp('B=')
%disp(B)
```

```
%A=[0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;1/2-sqrt(1/8);0;sqrt(1/8);0.5]
%B=[0;1/2-sqrt(1/8);0.5;1/2+sqrt(1/8);1;1/2+sqrt(1/8);1/2;sqrt(1/8);0]
for m=16:24
   if(m~=17)&(m~=22)
switch m
 case 16
    %disp(' fn=(x+y)^19')
 case 18
 %disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
  case 19
  %  disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
   %    disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
 case 21

    % disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
     % disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
      %disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
       % disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
        %disp('fn=f1+f2+f3+f4;')
case 23
   %disp('fn=1')
   case 24
    %  disp('fn=cos(20*(x+y))')

otherwise
   disp('something wrong')
end
format long e
ccc=ccc+1;
   N=0;M=0;
   t=0:4*div;
   x(1:(4*div+1))=A(1:(4*div+1),1);y(1:(4*div+1))=B(1:(4*div+1),1);
 %[coefx]=cubicspline(t,x,t,3);
 %[coefy]=cubicspline(t,y,t,3);
 pp=spline(t,x);
 [breaks,coefs,npolys,ncoefs,dim]=unmkpp(pp);
 coefx=coefs;
 pp=spline(t,y);
 [breaks,coefs,npolys,ncoefs,dim]=unmkpp(pp);
 coefy=coefs;
 for kk=1:4*div
          N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=coefx(N,4)+coefx(N,3)*ttp+coefx(N,2)*ttp^2+coefx(N,1)*ttp^3;
  yt=coefy(N,4)+coefy(N,3)*ttp+coefy(N,2)*ttp^2+coefy(N,1)*ttp^3;
   xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*coefy(N,1)*ttp^2+2*coefy(N,2)*ttp+coefy(N,3))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,1)=0;
for mm=1:M
   iii(ccc,1)=iii(ccc,1)+II(mm,1);
end
%disp(II)
%disp('----------------------------------')
```

```
%disp('X=')
%disp(X)
%disp('---------------------------------')
%disp('Y=')
%disp(Y)
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%end for m
%disp(iii)
format long g
for i=1:7
   III(i,1)=div;
   III(i,2)=i;
   III(i,3)=iii(i,1);
end
disp('-------------------------------------------------------------------------------------------------------')
disp('-------------------------------------------------------------------------------------------------------')
disp('                    div                    i          II(i)          ')
disp('-------------------------------------------------------------------------------------------------------')
disp(III)
disp('-------------------------------------------------------------------------------------------------------')



function[]=new_cubic_subbotin_spline_integration(div,n)
for qq=1:4
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer_quadrant(qq,div);
for m=16:24
 if(m~=17)&(m~=22)
 if qq==1
switch m
 case 16
    disp(' fn=(x+y)^19')
 case 18
 disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
  case 19
    disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
case 20
      disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
 case 21

     disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
      disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
      disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
       disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
       disp('fn=f1+f2+f3+f4;')
case 23
   disp('fn=1')
   case 24
     disp('fn=cos(20*(x+y))')

otherwise
   disp('something wrong')
end
end%qq==1
format long e
ccc=ccc+1;
   N=0;M=0;
nn=(length(A)-1)/3;
t=0:nn;
i=1;j=1;
```

```
 x(1)=A(1,1);x(2*(nn+1))=A(div+1,1);
 y(1)=B(1,1);y(2*(nn+1))=B(div+1,1);
 for i=1:nn
    x(2*i)=A(j+1,1);
    x(2*i+1)=A(j+2,1);
    y(2*i)=B(j+1,1);
    y(2*i+1)=B(j+2,1);
    j=i+2*i+1;
 end
for i=1:nn
hi=(t(i+1)-t(i));
h(i)=hi;
end
[px]=new_cubic_subbotin(t,h,x);
[py]=new_cubic_subbotin(t,h,y);

for kk=1:nn
          N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=px(N,1)+px(N,2)*ttp+px(N,3)*ttp^2+px(N,4)*ttp^3;
  yt=py(N,1)+py(N,2)*ttp+py(N,3)*ttp^2+py(N,4)*ttp^3;
  xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*py(N,4)*ttp^2+2*py(N,3)*ttp+py(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,qq,1)=0;
for mm=1:M
   iii(ccc,qq,1)=iii(ccc,qq,1)+II(mm,1);
end
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%for m
%disp(iii)
end%forqq
for i=1:7
   III(i,1)=0;
for qq=1:4
   III(i,1)=III(i,1)+iii(i,qq,1);
end
end
disp(III)

function[]=new_cubic_subbotin_spline_integration_octant(div,n)
for qq=1:8
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer_octant(qq,div);
for m=16:24
  if(m~=17)&(m~=22)
  if qq==1
switch m
  case 16
    disp(' fn=(x+y)^19')
  case 18
  disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
```

```
   case 19
     disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
       disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
  case 21

      disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
       disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
       disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
         disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
         disp('fn=f1+f2+f3+f4;')
case 23
    disp('fn=1')
    case 24
       disp('fn=cos(20*(x+y))')

otherwise
    disp('something wrong')
end
end%qq==1
format long e
ccc=ccc+1;
    N=0;M=0;
 nn=(length(A)-1)/3;
 t=0:nn;
 i=1;j=1;
 x(1)=A(1,1);x(2*(nn+1))=A(div+1,1);
 y(1)=B(1,1);y(2*(nn+1))=B(div+1,1);
 for i=1:nn
     x(2*i)=A(j+1,1);
     x(2*i+1)=A(j+2,1);
     y(2*i)=B(j+1,1);
     y(2*i+1)=B(j+2,1);
     j=i+2*i+1;
 end
for i=1:nn
hi=(t(i+1)-t(i));
h(i)=hi;
end
[px]=new_cubic_subbotin(t,h,x);
[py]=new_cubic_subbotin(t,h,y);

for kk=1:nn
           N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=px(N,1)+px(N,2)*ttp+px(N,3)*ttp^2+px(N,4)*ttp^3;
  yt=py(N,1)+py(N,2)*ttp+py(N,3)*ttp^2+py(N,4)*ttp^3;
  xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*py(N,4)*ttp^2+2*py(N,3)*ttp+py(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,qq,1)=0;
for mm=1:M
    iii(ccc,qq,1)=iii(ccc,qq,1)+II(mm,1);
end
%disp(iii(ccc,1))
end%if m~=17&m~=22
```

```
end%for m
end%forqq
for i=1:7
   III(i,1)=0;
for qq=1:8
   III(i,1)=III(i,1)+iii(i,qq,1);
end
end
disp(III)

function[]=new_cubic_subbotin_spline_integration_12tant(div,n)
for qq=1:12
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer_12tant(qq,div);
for m=16:24
  if(m~=17)&(m~=22)
  if qq==1
switch m
  case 16
     disp(' fn=(x+y)^19')
  case 18
  disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
   case 19
     disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
       disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
  case 21

      disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
       disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
       disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
         disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
         disp('fn=f1+f2+f3+f4;')
case 23
    disp('fn=1')
    case 24
       disp('fn=cos(20*(x+y))')

otherwise
    disp('something wrong')
end
end%qq==1
format long e
ccc=ccc+1;
    N=0;M=0;
 nn=(length(A)-1)/3;
 t=0:nn;
 i=1;j=1;
 x(1)=A(1,1);x(2*(nn+1))=A(div+1,1);
 y(1)=B(1,1);y(2*(nn+1))=B(div+1,1);
 for i=1:nn
     x(2*i)=A(j+1,1);
     x(2*i+1)=A(j+2,1);
     y(2*i)=B(j+1,1);
     y(2*i+1)=B(j+2,1);
     j=i+2*i+1;
 end
for i=1:nn
hi=(t(i+1)-t(i));
h(i)=hi;
end
[px]=new_cubic_subbotin(t,h,x);
```

```
[py]=new_cubic_subbotin(t,h,y);

for kk=1:nn
          N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=px(N,1)+px(N,2)*ttp+px(N,3)*ttp^2+px(N,4)*ttp^3;
  yt=py(N,1)+py(N,2)*ttp+py(N,3)*ttp^2+py(N,4)*ttp^3;
  xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*py(N,4)*ttp^2+2*py(N,3)*ttp+py(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,qq,1)=0;
for mm=1:M
   iii(ccc,qq,1)=iii(ccc,qq,1)+II(mm,1);
end
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%for m
end%forqq
for i=1:7
   III(i,1)=0;
for qq=1:12
   III(i,1)=III(i,1)+iii(i,qq,1);
end
end
disp(III)

function[]=new_cubic_subbotin_spline_integration_16tant(div,n)
for qq=1:16
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
xc=1/2;XC=xc;
[A,B]=buffer_16tant(qq,div);
for m=16:24
  if(m~=17)&(m~=22)
  if qq==1
switch m
  case 16
     disp(' fn=(x+y)^19')
  case 18
  disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
   case 19
     disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
       disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
  case 21

     disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
      disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
      disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
        disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
        disp('fn=f1+f2+f3+f4;')
case 23
   disp('fn=1')
   case 24
     disp('fn=cos(20*(x+y))')
```

```
otherwise
   disp('something wrong')
end
end%qq==1
format long e
ccc=ccc+1;
   N=0;M=0;
 nn=(length(A)-1)/3;
 t=0:nn;
 i=1;j=1;
 x(1)=A(1,1);x(2*(nn+1))=A(div+1,1);
 y(1)=B(1,1);y(2*(nn+1))=B(div+1,1);
 for i=1:nn
    x(2*i)=A(j+1,1);
    x(2*i+1)=A(j+2,1);
    y(2*i)=B(j+1,1);
    y(2*i+1)=B(j+2,1);
    j=i+2*i+1;
 end
for i=1:nn
hi=(t(i+1)-t(i));
h(i)=hi;
end
[px]=new_cubic_subbotin(t,h,x);
[py]=new_cubic_subbotin(t,h,y);

for kk=1:nn
          N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=px(N,1)+px(N,2)*ttp+px(N,3)*ttp^2+px(N,4)*ttp^3;
  yt=py(N,1)+py(N,2)*ttp+py(N,3)*ttp^2+py(N,4)*ttp^3;
  xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*py(N,4)*ttp^2+2*py(N,3)*ttp+py(N,2))*wp*wq/4;
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,qq,1)=0;
for mm=1:M
   iii(ccc,qq,1)=iii(ccc,qq,1)+II(mm,1);
end
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%for m
end%forqq
for i=1:7
   III(i,1)=0;
for qq=1:16
   III(i,1)=III(i,1)+iii(i,qq,1);
end
end
disp(III)
%=========================================================
function[]=new_cubic_subbotin_spline_integration_20tant(div,n)
for qq=1:20
ccc=0;
[s3,w3]=glsampleptsweights(n+3);
[s0,w0]=glsampleptsweights(n);
```

```
xc=1/2;XC=xc;
[A,B]=buffer_20tant(qq,div);
for m=16:24
  if(m~=17)&(m~=22)
  if qq==1
switch m
  case 16
     disp(' fn=(x+y)^19')
  case 18
  disp('fn=sqrt((x-1/2)^2+(y-1/2)^2)')
   case 19
     disp(' fn=exp(-((x-1/2)^2+(y-1/2)^2))')
 case 20
       disp('fn=exp(-100*((x-1/2)^2+(y-1/2)^2));')
  case 21

      disp('f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);')
       disp('f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));')
       disp('f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);')
         disp('f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);')
         disp('fn=f1+f2+f3+f4;')
case 23
    disp('fn=1')
    case 24
       disp('fn=cos(20*(x+y))')

otherwise
    disp('something wrong')
end
end%qq==1
format long e
ccc=ccc+1;
    N=0;M=0;
 nn=(length(A)-1)/3;
 t=0:nn;
 i=1;j=1;
 x(1)=A(1,1);x(2*(nn+1))=A(div+1,1);
 y(1)=B(1,1);y(2*(nn+1))=B(div+1,1);
 for i=1:nn
     x(2*i)=A(j+1,1);
     x(2*i+1)=A(j+2,1);
     y(2*i)=B(j+1,1);
     y(2*i+1)=B(j+2,1);
     j=i+2*i+1;
 end
for i=1:nn
hi=(t(i+1)-t(i));
h(i)=hi;
end
[px]=new_cubic_subbotin(t,h,x);
[py]=new_cubic_subbotin(t,h,y);

for kk=1:nn
           N=N+1; M=M+1;II(M,1)=0;

 for p=1:(n+3)
 for q=1:n
  tp=s3(p);wp=w3(p);
  ttp=(tp+1)/2;
  tq=s0(q);wq=w0(q);
  ttq=(1+tq)/2;
  xt=px(N,1)+px(N,2)*ttp+px(N,3)*ttp^2+px(N,4)*ttp^3;
  yt=py(N,1)+py(N,2)*ttp+py(N,3)*ttp^2+py(N,4)*ttp^3;
  xx=(xt-xc)*tq/2+(xt+xc)/2;
  wt=(xt-xc)*(3*py(N,4)*ttp^2+2*py(N,3)*ttp+py(N,2))*wp*wq/4;
```

```
  II(M,1)=II(M,1)+wt*fnxy(m,xx,yt);
end%end for q
end%end for p
end%end for kk
iii(ccc,qq,1)=0;
for mm=1:M
   iii(ccc,qq,1)=iii(ccc,qq,1)+II(mm,1);
end
%disp(iii(ccc,1))
end%if m~=17&m~=22
end%for m
end%forqq
for i=1:7
   III(i,1)=0;
for qq=1:20
   III(i,1)=III(i,1)+iii(i,qq,1);
end
end
disp(III)

function[A,B]=buffer(div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
A(1,1)=1/2;A(div+1,1)=1;
B(1,1)=0;B(div+1,1)=1/2;
A(2*div+1,1)=1/2;A(3*div+1,1)=0;
B(2*div+1,1)=1;B(3*div+1,1)=1/2;
n=4*div;
A(n+1,1)=A(1);B(n+1,1)=B(1);
% FOURTH QUADRANT
for k=1:(div-1)
A(1+k,1)=xc+r*cos(1.5*pi+k*pi/(2*div));
B(1+k,1)=yc+r*sin(1.5*pi+k*pi/(2*div));
end
% FIRST QUADRANT
for k=1:(div-1)
A(div+1+k,1)=xc+r*cos(k*pi/(2*div));
B(div+1+k,1)=yc+r*sin(k*pi/(2*div));
end
% SECOND QUADRANT
for k=1:(div-1)
A(2*div+1+k,1)=xc+r*cos(pi/2+k*pi/(2*div));
B(2*div+1+k,1)=yc+r*sin(pi/2+k*pi/(2*div));
end
% THIRD QUADRANT
for k=1:(div-1)
A(n+1-k,1)=r*cos(k*pi/(2*div));
B(n+1-k,1)=r*sin(k*pi/(2*div));
end
%=========================================================
function[X,Y]=buffer_quadrant(q,div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
switch q
 case 1
% FOURTH QUADRANT
X(1,1)=1/2;Y(1,1)=0;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(1.5*pi+k*pi/(2*div));
Y(1+k,1)=yc+r*sin(1.5*pi+k*pi/(2*div));
end
X(div+1,1)=1;Y(div+1,1)=1/2;
case 2
```

```
X(1,1)=1;Y(1,1)=1/2;
% FIRST QUADRANT
for k=1:(div-1)
X(1+k,1)=xc+r*cos(k*pi/(2*div));
Y(1+k,1)=yc+r*sin(k*pi/(2*div));
end
X(div+1,1)=1/2;Y(div+1,1)=1;
case 3
   X(1,1)=1/2;Y(1,1)=1;
% SECOND QUADRANT
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/2+k*pi/(2*div));
Y(1+k,1)=yc+r*sin(pi/2+k*pi/(2*div));
end
X(div+1,1)=0;Y(div+1,1)=1/2;
case 4
   X(1,1)=0;Y(1,1)=1/2;
% THIRD QUADRANT
for k=1:(div-1)
%X(div+1-k,1)=r*cos(k*pi/(2*div));
%Y(div+1-k,1)=r*sin(k*pi/(2*div));
X(1+k,1)=r*cos(pi/2-k*pi/(2*div));
Y(1+k,1)=r*sin(pi/2-k*pi/(2*div));
end
X(div+1,1)=1/2;Y(div+1,1)=0;
end
function[X,Y]=buffer_octant(q,div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
switch q
 case 1
% FOURTH QUADRANT:1-OCTANT
X(1,1)=1/2;Y(1,1)=0;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(1.5*pi+k*pi/(4*div));
Y(1+k,1)=yc+r*sin(1.5*pi+k*pi/(4*div));
end
X(div+1,1)=xc+r*cos(1.75*pi);
Y(div+1,1)=yc+r*sin(1.75*pi);
case 2
% FOURTH QUADRANT:2-OCTANT
X(1,1)=xc+r*cos(1.75*pi);
Y(1,1)=yc+r*sin(1.75*pi);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(1.75*pi+k*pi/(4*div));
Y(1+k,1)=yc+r*sin(1.75*pi+k*pi/(4*div));
end
X(div+1,1)=1;Y(div+1,1)=1/2;
case 3
% FIRST QUADRANT:3-OCTANT
X(1,1)=1;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(k*pi/(4*div));
Y(1+k,1)=yc+r*sin(k*pi/(4*div));
end
X(div+1,1)=xc+r*cos(pi/4);
Y(div+1,1)=yc+r*sin(pi/4);
case 4
% FIRST QUADRANT:4-OCTANT
X(1,1)=xc+r*cos(pi/4);
Y(1,1)=yc+r*sin(pi/4);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/4+k*pi/(4*div));
Y(1+k,1)=yc+r*sin(pi/4+k*pi/(4*div));
```

```
end
X(div+1,1)=1/2;Y(div+1,1)=1;
case 5
% SECOND QUADRANT:5-OCTANT
X(1,1)=1/2;Y(1,1)=1;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/2+k*pi/(4*div));
Y(1+k,1)=yc+r*sin(pi/2+k*pi/(4*div));
end
X(div+1,1)=xc+r*cos(3*pi/4);
Y(div+1,1)=yc+r*sin(3*pi/4);
case 6
% SECOND QUADRANT:6-OCTANT
X(1,1)=xc+r*cos(3*pi/4);
Y(1,1)=yc+r*sin(3*pi/4);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(3*pi/4+k*pi/(4*div));
Y(1+k,1)=yc+r*sin(3*pi/4+k*pi/(4*div));
end
X(div+1,1)=0;Y(div+1,1)=1/2;
case 7
 % THIRD QUADRANT:7-OCTANT
  X(1,1)=0;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=r*cos(pi/2-k*pi/(4*div));
Y(k+1,1)=r*sin(pi/2-k*pi/(4*div));
end
X(div+1,1)=r*cos(pi/4);
Y(div+1,1)=r*sin(pi/4);
case 8
 % THIRD QUADRANT:8-OCTANT
X(1,1)=r*cos(pi/4);
Y(1,1)=r*sin(pi/4);
for k=1:(div-1)
X(k+1,1)=r*cos(pi/4-k*pi/(4*div));
Y(k+1,1)=r*sin(pi/4-k*pi/(4*div));
end
X(div+1,1)=1/2;Y(div+1,1)=0;
end
function[X,Y]=buffer_12tant(q,div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
switch q
 case 1
% FOURTH QUADRANT:1-12TANT
X(1,1)=1/2;Y(1,1)=0;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(1.5*pi+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(1.5*pi+k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(5*pi/3);
Y(div+1,1)=yc+r*sin(5*pi/3);
case 2
% FOURTH QUADRANT:2-12TANT
X(1,1)=xc+r*cos(5*pi/3);
Y(1,1)=yc+r*sin(5*pi/3);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(5*pi/3+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(5*pi/3+k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(11*pi/6);
Y(div+1,1)=yc+r*sin(11*pi/6);
case 3
% FOURTH QUADRANT:3-12TANT
```

```
X(1,1)=xc+r*cos(11*pi/6);
Y(1,1)=yc+r*sin(11*pi/6);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(11*pi/6+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(11*pi/6+k*pi/(6*div));
end
X(div+1,1)=1;Y(div+1,1)=1/2;
case 4
% FIRST QUADRANT:4-12TANT
X(1,1)=1;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(k*pi/(6*div));
Y(1+k,1)=yc+r*sin(k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(pi/6);
Y(div+1,1)=yc+r*sin(pi/6);
case 5
% FIRST QUADRANT:5-12TANT
X(1,1)=xc+r*cos(pi/6);
Y(1,1)=yc+r*sin(pi/6);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/6+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(pi/6+k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(pi/3);
Y(div+1,1)=yc+r*sin(pi/3);
case 6
% FIRST QUADRANT:6-12TANT
X(1,1)=xc+r*cos(pi/3);
Y(1,1)=yc+r*sin(pi/3);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/3+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(pi/3+k*pi/(6*div));
end
X(div+1,1)=1/2;Y(div+1,1)=1;
case 7
% SECOND QUADRANT:7-12TANT
X(1,1)=1/2;Y(1,1)=1;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/2+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(pi/2+k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(2*pi/3);
Y(div+1,1)=yc+r*sin(2*pi/3);
case 8
% SECOND QUADRANT:8-12TANT
X(1,1)=xc+r*cos(2*pi/3);
Y(1,1)=yc+r*sin(2*pi/3);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(2*pi/3+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(2*pi/3+k*pi/(6*div));
end
X(div+1,1)=xc+r*cos(5*pi/6);
Y(div+1,1)=yc+r*sin(5*pi/6);
case 9
% SECOND QUADRANT:9-12TANT
X(1,1)=xc+r*cos(5*pi/6);
Y(1,1)=yc+r*sin(5*pi/6);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(5*pi/6+k*pi/(6*div));
Y(1+k,1)=yc+r*sin(5*pi/6+k*pi/(6*div));
end
X(div+1,1)=0;Y(div+1,1)=1/2;
case 10
 % THIRD QUADRANT:10-12TANT
```

```
  X(1,1)=0;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=r*cos(pi/2-k*pi/(6*div));
Y(k+1,1)=r*sin(pi/2-k*pi/(6*div));
end
X(div+1,1)=r*cos(pi/3);
Y(div+1,1)=r*sin(pi/3);
case 11
 % THIRD QUADRANT:11-12TANT
X(1,1)=r*cos(pi/3);
Y(1,1)=r*sin(pi/3);
for k=1:(div-1)
X(k+1,1)=r*cos(pi/3-k*pi/(6*div));
Y(k+1,1)=r*sin(pi/3-k*pi/(6*div));
end
X(div+1,1)=r*cos(pi/6);
Y(div+1,1)=r*sin(pi/6);
case 12
 % THIRD QUADRANT:12-12TANT
X(1,1)=r*cos(pi/6);
Y(1,1)=r*sin(pi/6);
for k=1:(div-1)
X(k+1,1)=r*cos(pi/6-k*pi/(6*div));
Y(k+1,1)=r*sin(pi/6-k*pi/(6*div));
end
X(div+1,1)=1/2;Y(div+1,1)=0;
end
function[X,Y]=buffer_16tant(q,div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
switch q
 case 1
% FOURTH QUADRANT:1-16TANT
X(1,1)=1/2;Y(1,1)=0;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(1.5*pi+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(1.5*pi+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(13*pi/8);
Y(div+1,1)=yc+r*sin(13*pi/8);
case 2
% FOURTH QUADRANT:2-16TANT
X(1,1)=xc+r*cos(13*pi/8);
Y(1,1)=yc+r*sin(13*pi/8);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(13*pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(13*pi/8+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(7*pi/4);
Y(div+1,1)=yc+r*sin(7*pi/4);
case 3
% FOURTH QUADRANT:3-16TANT
X(1,1)=xc+r*cos(7*pi/4);
Y(1,1)=yc+r*sin(7*pi/4);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(7*pi/4+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(7*pi/4+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(15*pi/8);
Y(div+1,1)=yc+r*sin(15*pi/8);
case 4
% FOURTH QUADRANT:4-16TANT
X(1,1)=xc+r*cos(15*pi/8);
Y(1,1)=yc+r*sin(15*pi/8);
```

```
for k=1:(div-1)
X(1+k,1)=xc+r*cos(15*pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(15*pi/8+k*pi/(8*div));
end
X(div+1,1)=1;Y(div+1,1)=1/2;
case 5
% FIRST QUADRANT:5-16TANT
X(1,1)=1;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(k*pi/(8*div));
Y(1+k,1)=yc+r*sin(k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(pi/8);
Y(div+1,1)=yc+r*sin(pi/8);
case 6
% FIRST QUADRANT:6-16TANT
X(1,1)=xc+r*cos(pi/8);
Y(1,1)=yc+r*sin(pi/8);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(pi/8+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(pi/4);
Y(div+1,1)=yc+r*sin(pi/4);
case 7
% FIRST QUADRANT:7-16TANT
X(1,1)=xc+r*cos(pi/4);
Y(1,1)=yc+r*sin(pi/4);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/4+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(pi/4+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(3*pi/8);
Y(div+1,1)=yc+r*sin(3*pi/8);
case 8
% FIRST QUADRANT:8-16TANT
X(1,1)=xc+r*cos(3*pi/8);
Y(1,1)=yc+r*sin(3*pi/8);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(3*pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(3*pi/8+k*pi/(8*div));
end
X(div+1,1)=1/2;Y(div+1,1)=1;
case 9
% SECOND QUADRANT:9-16TANT
X(1,1)=1/2;Y(1,1)=1;
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/2+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(pi/2+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(5*pi/8);
Y(div+1,1)=yc+r*sin(5*pi/8);
case 10
% SECOND QUADRANT:10-16TANT
X(1,1)=xc+r*cos(5*pi/8);
Y(1,1)=yc+r*sin(5*pi/8);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(5*pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(5*pi/8+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(3*pi/4);
Y(div+1,1)=yc+r*sin(3*pi/4);
case 11
% SECOND QUADRANT:11-16TANT
X(1,1)=xc+r*cos(3*pi/4);
```

```
Y(1,1)=yc+r*sin(3*pi/4);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(3*pi/4+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(3*pi/4+k*pi/(8*div));
end
X(div+1,1)=xc+r*cos(7*pi/8);
Y(div+1,1)=yc+r*sin(7*pi/8);
case 12
% SECOND QUADRANT:12-16TANT
X(1,1)=xc+r*cos(7*pi/8);
Y(1,1)=yc+r*sin(7*pi/8);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(7*pi/8+k*pi/(8*div));
Y(1+k,1)=yc+r*sin(7*pi/8+k*pi/(8*div));
end
X(div+1,1)=0;Y(div+1,1)=1/2;
case 13
 % THIRD QUADRANT:13-16TANT
  X(1,1)=0;Y(1,1)=1/2;
for k=1:(div-1)
X(1+k,1)=r*cos(pi/2-k*pi/(8*div));
Y(k+1,1)=r*sin(pi/2-k*pi/(8*div));
end
X(div+1,1)=r*cos(3*pi/8);
Y(div+1,1)=r*sin(3*pi/8);
case 14
 % THIRD QUADRANT:14-16TANT
X(1,1)=r*cos(3*pi/8);
Y(1,1)=r*sin(3*pi/8);
for k=1:(div-1)
X(k+1,1)=r*cos(3*pi/8-k*pi/(8*div));
Y(k+1,1)=r*sin(3*pi/8-k*pi/(8*div));
end
X(div+1,1)=r*cos(pi/4);
Y(div+1,1)=r*sin(pi/4);
case 15
 % THIRD QUADRANT:15-16TANT
X(1,1)=r*cos(pi/4);
Y(1,1)=r*sin(pi/4);
for k=1:(div-1)
X(k+1,1)=r*cos(pi/4-k*pi/(8*div));
Y(k+1,1)=r*sin(pi/4-k*pi/(8*div));
end
X(div+1,1)=r*cos(pi/8);
Y(div+1,1)=r*sin(pi/8);
case 16
 % THIRD QUADRANT:16-16TANT
X(1,1)=r*cos(pi/8);
Y(1,1)=r*sin(pi/8);
for k=1:(div-1)
X(k+1,1)=r*cos(pi/8-k*pi/(8*div));
Y(k+1,1)=r*sin(pi/8-k*pi/(8*div));
end
X(div+1,1)=1/2;Y(div+1,1)=0;
end
function[X,Y]=buffer_20tant(q,div)
%div=number of partitions/divisions in a quadrant
%div=2,4,6,.....
xc=1/2;yc=1/2;r=1/2;
for k=1:(div+1)
   X(k,1)=0;Y(k,1)=0;
end
% FOURTH QUADRANT:1,2,3,4,5-20TANT
if ((q>=1)&(q<=5))
   X(1,1)=xc+r*cos((15*pi+(q-1)*pi)/10);
```

```
    Y(1,1)=yc+r*sin((15*pi+(q-1)*pi)/10);
for k=1:(div-1)
X(1+k,1)=xc+r*cos((15*pi+(q-1)*pi)/10+k*pi/(10*div));
Y(1+k,1)=yc+r*sin((15*pi+(q-1)*pi)/10+k*pi/(10*div));
end
X(div+1,1)=xc+r*cos(1.5*pi+q*pi/10);
Y(div+1,1)=yc+r*sin(1.5*pi+q*pi/10);
end
%FIRST QUADRANT:6,7,8,9,10-20TANTS
if ((q>=6)&(q<=10))
s=q-5;
    X(1,1)=xc+r*cos((s-1)*pi/10);
    Y(1,1)=yc+r*sin((s-1)*pi/10);
for k=1:(div-1)
X(1+k,1)=xc+r*cos((s-1)*pi/10+k*pi/(10*div));
Y(1+k,1)=yc+r*sin((s-1)*pi/10+k*pi/(10*div));
end
X(div+1,1)=xc+r*cos(s*pi/10);
Y(div+1,1)=yc+r*sin(s*pi/10);
end
% SECOND QUADRANT:11,12,13,14,15-20TANTS
if ((q>=11)&(q<=15))
s=q-10;
    X(1,1)=xc+r*cos(pi/2+(s-1)*pi/10);
    Y(1,1)=yc+r*sin(pi/2+(s-1)*pi/10);
for k=1:(div-1)
X(1+k,1)=xc+r*cos(pi/2+(s-1)*pi/10+k*pi/(10*div));
Y(1+k,1)=yc+r*sin(pi/2+(s-1)*pi/10+k*pi/(10*div));
end
X(div+1,1)=xc+r*cos(pi/2+s*pi/10);
Y(div+1,1)=yc+r*sin(pi/2+s*pi/10);
end
%THIRD QUADRANT:16,17,18,19,20-20TANT
if ((q>=16)&(q<=20))
    s=q-15;
    X(1,1)=r*cos(pi/2-(s-1)*pi/10);
    Y(1,1)=r*sin(pi/2-(s-1)*pi/10);
 for k=1:(div-1)
X(1+k,1)=r*cos(pi/2-(s-1)*pi/10-k*pi/(10*div));
Y(1+k,1)=r*sin(pi/2-(s-1)*pi/10-k*pi/(10*div));
 end
X(div+1,1)=r*cos(pi/2-s*pi/10);
Y(div+1,1)=r*sin(pi/2-s*pi/10);
End
%=========================================================
function[p]=new_cubic_subbotin(x,h,f)
n=(length(x)-1);
%h=diff(x);
%v=diff(f);
%for i=1:n
 %   v(i)=(f(2*i+1)-f(2*i))/h(i);
 %end
A=zeros(n+1,n+1);
r=zeros(n+1,1);
for i=1:n-1
    A(i+1,i:i+2)=[7*h(i)^2*(h(i)+h(i+1)) 2*h(i)*(h(i)+h(i+1))*(3*h(i+1)+ 7*h(i)) 0];
    r(i+1)=162*h(i+1)*(f(2*i)-f(2*i+1))+27*h(i)*(-14*f(2*i+1)+7*f(2*i)+8*f(2*i+2)-f(2*i+3));
end
    A(1,1:2)=[2*h(1) h(1)];r(1)=27*(f(3)-2*f(2)+f(1))/h(1);
    A(n+1,n:n+1)=[h(n) 2*h(n)];r(n+1,1)=27*(f(2*(n+1))-2*f(2*n+1)+f(2*n))/h(n);
ds=A\r;     %solve A*ds=r
    p=zeros(n,4);
    for i=1:n
    p(i,1)=(ds(i+1)+2*ds(i))*h(i)^2/27+(2*f(2*i)-f(2*i+1));
    p(i,2)=3*(f(2*i+1)-f(2*i))-7*h(i)^2*ds(i+1)/(54)-10*ds(i)*h(i)^2/27;
```

```
   p(i,3)=ds(i)*h(i)^2/2;
   p(i,4)=(ds(i+1)-ds(i))*h(i)^2/6;
end
%disp(' U=u+hi/3, SPLINE COEFFICIENTS-p ')
%disp('-----------------------------------------')
%disp('      U^1      U^2      U^3    ')
%disp(p)
%disp('-----------------------------------------')
%disp('-----------------------------------------')
%disp('u=             SPLINE COEFFICIENTS-q ')
%disp('-----------------------------------------')
%disp('      u^1      u^2      u^3    ')
%disp(q)
%disp('-----------------------------------------')
%=====================================================================
function[fn]=fnxy(n,x,y)
switch n
  case 16
     fn=(x+y)^19;

  case 17
     fn=cos(30*(x+y));
  case 18
     fn=sqrt((x-1/2)^2+(y-1/2)^2);
  case 19
     fn=exp(-((x-1/2)^2+(y-1/2)^2));
  case 20
     fn=exp(-100*((x-1/2)^2+(y-1/2)^2));
  case 21
     f1=0.75*exp(-0.25*(9*x-2)^2-0.25*(9*y-2)^2);
      f2=0.75*exp((-1/49)*(9*x+1)^2-0.1*(9*y+1));
      f3=0.5*exp(-0.25*(9*x-7)^2-0.25*(9*y-3)^2);
        f4=-0.2*exp(-(9*y-4)^2-(9*y-7)^2);
        fn=f1+f2+f3+f4;
    case 22
     fn=1/sqrt((x-1/2)^2+(y-1/2)^2);
  case 23
     fn=(x^0)*(y^0);
  case 24
     fn=cos(20*(x+y));
  case 25
     fn=1;

otherwise
   disp('something wrong')
end
%disp('htr')

%=====================================================================


function [s,wwww]=glsampleptsweights(n)
switch n
    case 32
     table=[ -.48307665687738316234812570440502e-1,  .96540088514727789826546531216736e-1
  .48307665687738316234812570440502e-1,  .96540088514727789826546531216736e-1
   -.14447196158279649348518637359881,  .95638720079274848779142187802379e-1
    .14447196158279649348518637359881,  .95638720079274848779142187802379e-1
   -.23928736225213707454460320916550,  .93844399080804555198861114715938e-1
    .23928736225213707454460320916550,  .93844399080804555198861114715938e-1
   -.33186860228212764977991680573019,  .91173878695763874569648538561320e-1
    .33186860228212764977991680573019,  .91173878695763874569648538561320e-1
   -.42135127613063534536411943617243,  .87652093004403801391354951494255e-1
    .42135127613063534536411943617243,  .87652093004403801391354951494255e-1
   -.50689990893222939002374747437780,  .83311924226946745953632338867803e-1
```

```
    .506899908932229390023747474377 80,  .833119242269467459536323388678 03e-1
   -.587715757240762329040745476401 80,  .781938957870702977725618929683 63e-1
    .587715757240762329040745476401 80,  .781938957870702977725618929683 63e-1
   -.663044266930215200975115168663 25,  .723457941088484981768303543320 38e-1
    .663044266930215200975115168663 25,  .723457941088484981768303543320 38e-1
   -.732182118740289680387426665091 25,  .658222227763618395148328691500 17e-1
    .732182118740289680387426665091 25,  .658222227763618395148328691500 17e-1
   -.794483795967942406963097298970 45,  .586840934785355406165980697009 69e-1
    .794483795967942406963097298970 45,  .586840934785355406165980697009 69e-1
   -.849367613732569970133693004967 75,  .509980592623761705225585865548 87e-1
    .849367613732569970133693004967 75,  .509980592623761705225585865548 87e-1
   -.896321155766052123965307243719 20,  .428358980222266758912349862855 94e-1
    .896321155766052123965307243719 20,  .428358980222266758912349862855 94e-1
   -.934906075937739689170919134835 40,  .342738629130214292891350500351 97e-1
    .934906075937739689170919134835 40,  .342738629130214292891350500351 97e-1
   -.964762255587506430773811928118 25,  .253920653092620566307572871988 49e-1
    .964762255587506430773811928118 25,  .253920653092620566307572871988 49e-1
   -.985611511545268335400175044630 90,  .162743947309056687946289923578 32e-1
    .985611511545268335400175044630 90,  .162743947309056687946289923578 32e-1
   -.997263861849481563544981128665 05,  .701861000947009581946030161837 49e-2
    .997263861849481563544981128665 05,  .701861000947009581946030161837 49e-2];

      s=table(:,1);wwww=table(:,2);
  case 33

table=[                          0,  .937684461602099837918982072849 53e-1
 -.936310658547333856707429241225 40e-1,  .933564260655961034417281375595 07e-1
  .936310658547333856707429241225 40e-1,  .933564260655961034417281375595 07e-1
   -.186439298827991572335798759218 82,  .921239866433168336618827042786 18e-1
    .186439298827991572335798759218 82,  .921239866433168336618827042786 18e-1
   -.277609097152497029402348067298 32,  .900819586606385649665998879161 75e-1
    .277609097152497029402348067298 32,  .900819586606385649665998879161 75e-1
   -.366339257748073341070220623253 87,  .872482876188443257202090794013 74e-1
    .366339257748073341070220623253 87,  .872482876188443257202090794013 74e-1
   -.451850017272450695725993277240 77,  .836478760670386962173905909336 81e-1
    .451850017272450695725993277240 77,  .836478760670386962173905909336 81e-1
   -.533389904786347643548894264995 50,  .793123647948867275580592128836 08e-1
    .533389904786347643548894264995 50,  .793123647948867275580592128836 08e-1
   -.610242345836379027307287513538 10,  .742798548439541392222731278506 56e-1
    .610242345836379027307287513538 10,  .742798548439541392222731278506 56e-1
   -.681731959969742786268215946919 35,  .685945728186567034603442924529 74e-1
    .681731959969742786268215946919 35,  .685945728186567034603442924529 74e-1
   -.747230496449562157859055124897 95,  .623064825303174715427247902288 20e-1
    .747230496449562157859055124897 95,  .623064825303174715427247902288 20e-1
   -.806162356274166589796200870782 80,  .554708466316635537274010344200 32e-1
    .806162356274166589796200870782 80,  .554708466316635537274010344200 32e-1
   -.858009652676504064643061480146 05,  .481477428187116891102980978672 40e-1
    .858009652676504064643061480146 05,  .481477428187116891102980978672 40e-1
   -.902316767743433583040531331518 75,  .404015413316695860590042810853 55e-1
    .902316767743433583040531331518 75,  .404015413316695860590042810853 55e-1
   -.938694372611168350355835124363 55,  .323003586323289488810124377466 27e-1
    .938694372611168350355835124363 55,  .323003586323289488810124377466 27e-1
   -.966822909689992768928377706678 60,  .239155481017494770922246411988 04e-1
    .966822909689992768928377706678 60,  .239155481017494770922246411988 04e-1
   -.986455726230642488110375698264 65,  .153217015129346740402336632059 00e-1
    .986455726230642488110375698264 65,  .153217015129346740402336632059 00e-1
   -.997424694246455217266168017580 40,  .660622784758737715822862227544 74e-2
    .997424694246455217266168017580 40,  .660622784758737715822862227544 74e-2];


   s=table(:,1);wwww=table(:,2);
case 34
 table=  [ -.455098219531025427490756708519 30e-1,  .909567403302598608281113190508 14e-1
  .455098219531025427490756708519 30e-1,  .909567403302598608281113190508 14e-1
   -.136152357259182975894428824331 12,  .902030443706407168926749080524 55e-1
```

```
    .13615235725918297589442882433112,  .902030443706407168926749080524 55e-1
   -.22566669161644948386864118093435,  .887018978356938568168490129047 30e-1
    .22566669161644948386864118093435,  .887018978356938568168490129047 30e-1
   -.31331108133946324745831676565098,  .864657397470357376283914900858 69e-1
    .31331108133946324745831676565098,  .864657397470357376283914900858 69e-1
   -.39835927775864594063149475293236,  .835130996998456434462641317693 62e-1
    .39835927775864594063149475293236,  .835130996998456434462641317693 62e-1
   -.48010654519032703419410268050739,  .798684443397718335104495375868 33e-1
    .48010654519032703419410268050739,  .798684443397718335104495375868 33e-1
   -.55787550066974664273645988621625,  .755619746600319206478931720875 68e-1
    .55787550066974664273645988621625,  .755619746600319206478931720875 68e-1
   -.63102172708052854531777575551900,  .706293758142557150695509880136 61e-1
    .63102172708052854531777575551900,  .706293758142557150695509880136 61e-1
   -.69893911321626290793300010657580,  .651115215540764022247769896854 81e-1
    .69893911321626290793300010657580,  .651115215540764022247769896854 81e-1
   -.76106487662987301418740896897875,  .590541358275244848918138479988 36e-1
    .76106487662987301418740896897875,  .590541358275244848918138479988 36e-1
   -.81688422790093366459157890658695,  .525074145726780987865300674881 74e-1
    .81688422790093366459157890658695,  .525074145726780987865300674881 74e-1
   -.86593463833456446926357209067130,  .455256115233532660528421050107 22e-1
    .86593463833456446926357209067130,  .455256115233532660528421050107 22e-1
   -.90780967771832446880089988901930,  .381665937963875109556499759835 99e-1
    .90780967771832446880089988901930,  .381665937963875109556499759835 99e-1
   -.94216239740510709163167602546055,  .304913806384461275263670140699 04e-1
    .94216239740510709163167602546055,  .304913806384461275263670140699 04e-1
   -.96870826253334428176464657305600,  .225637219854949669116649834043 31e-1
    .96870826253334428176464657305600,  .225637219854949669116649834043 31e-1
   -.98722781640630948504975043109970,  .144501627485950333774092258425 24e-1
    .98722781640630948504975043109970,  .144501627485950333774092258425 24e-1
   -.99757175379084191924337243745465,  .622914055590868384080005921551 52e-2
    .99757175379084191924337243745465,  .622914055590868384080005921551 52e-2];

  s=table(:,1);wwww=table(:,2);

case 35

table=[                              0,  .8848679490710427507090099406343 0e-1
 -.88371343275659263600929433497550e-1,  .8814053043027544746435003678198 0e-1
  .88371343275659263600929433497550e-1,  .8814053043027544746435003678198 0e-1
   -.17605106116598956997430365644506,  .8710444699718351891920936503506 1e-1
    .17605106116598956997430365644506,  .8710444699718351891920936503506 1e-1
   -.26235294120929605797089520045558,  .8538665339209911020403934404995 3e-1
    .26235294120929605797089520045558,  .8538665339209911020403934404995 3e-1
   -.34660155443081394587697983493024,  .8300059372885657377779663099849 3e-1
    .34660155443081394587697983493024,  .8300059372885657377779663099849 3e-1
   -.42813754151781425418762061300147,  .7996494224232424886458833150450 7e-1
    .42813754151781425418762061300147,  .7996494224232424886458833150450 7e-1
   -.50632277324148861502429755583735,  .7630345715544204011474258740233 8e-1
    .50632277324148861502429755583735,  .7630345715544204011474258740233 8e-1
   -.58054534474976450993450200818970,  .7204479477256005199076409254086 3e-1
    .58054534474976450993450200818970,  .7204479477256005199076409254086 3e-1
   -.65022436466589038867579280898455,  .6722228526908689213802171170974 6e-1
    .65022436466589038867579280898455,  .6722228526908689213802171170974 6e-1
   -.71481450155662878326440863122445,  .6187367196608017800170797743319 3e-1
    .71481450155662878326440863122445,  .6187367196608017800170797743319 3e-1
   -.77381025228691255526742300920990,  .5604081621237011871917596818937 2e-1
    .77381025228691255526742300920990,  .5604081621237011871917596818937 2e-1
   -.82674989909222540683405061274855,  .4976937040135352104963298939002 5e-1
    .82674989909222540683405061274855,  .4976937040135352104963298939002 5e-1
   -.87321912502522233152328234914140,  .4310842232617021119809010565597 0e-1
    .87321912502522233152328234914140,  .4310842232617021119809010565597 0e-1
   -.91285426135931761446493706355575,  .3611011586346337418109013689754 3e-1
    .91285426135931761446493706355575,  .3611011586346337418109013689754 3e-1
   -.94534514820782732953872598553000,  .2882926010889424897796817422373 5e-1
    .94534514820782732953872598553000,  .2882926010889424897796817422373 5e-1
```

```
-.97043761603922983321507048258475,  .21322979911483577132043584500603e-1
 .97043761603922983321507048258475,  .21322979911483577132043584500603e-1
-.98793576444385149803511708918550,  .13650828348361489865321570092698e-1
 .98793576444385149803511708918550,  .13650828348361489865321570092698e-1
-.99770656909960029726016313931210,  .58834334204430839474263690978547e-2
 .99770656909960029726016313931210,  .58834334204430839474263690978547e-2];
s=table(:,1);wwww=table(:,2);

end
%=========================================================================
```

References-*II***

1. Abramowitz, M and Stegun, I.A.(1972) Handbook of Mathematical Functions, Dover Publications, New York.
2. Ahlberg, J.H, E.N.Nilson, J.L.Walsh,(1967) The Theory of Splines and Their Applications, Academic Press, Inc., New York.
3. Andrews, L.C and R.L.Philips (2005), Mathematical Techniques, For Engineers and Scientists, Prentice Hall of India, Pvt Ltd, New Delhi.
4. Antia, H.M. (1991) Numerical Methods For Scientists and Engineers, Tata Mc Graw Hill Company Ltd, New Delhi.
5. Apostol, T.M.(1969) Calculus, Vol.II, 2^{nd} edition, Blaisdell.
6. Atkinson, K. (1993) An Introduction to Numerical Analysis, 2nd Edition, John Willey, New York.
7. Bradie,B. (2008) A Friendly Introduction to Numerical Analysis, Pearson Education Inc.
8. Bansal, R.K, A.K.Goel and M.K.Sharma.(2009) MATLAB® and its Applications in Engineering, Pearson Education.
9. Boor, C.de. On Uniform Approximation by Splines, J.Approx. Theory I (1968) 219-235.
10. Boor, C.de. The Method of Projections as Applied to Numerical Solution of Two Point Boundary Value problems using cubic splines, Ph.D thesis University of Michigan (1966).
11. Boor, C.de. (1978) A practical guide to Splines, Springer-Verlag, New York.
12. Boris, I.Kvasov. Methods of Shape-Preserving spline approximation, World Scientific Publishing Co.,Inc.,River Edge, NJ, 2000.
13. Burden,R.L and J.D.Faires, (2001) Numerical Analysis 7th Edn.Pacific Grove, California: Brooks/Cole.
14. Cheney,w and D.Kinicaid,(2004) Numerical Mathematics and Computing, 5^{th} Edn, Thomson Brooks/cole.
15. Dagnino, C and Fiorentine C (1984) "Computation of Nodes and Weights of Extended Gaussian Rules", Computing Vol 32, pp (271-278).
16. Davis,P.J and Rabinowitz. P, (1984) Methods of Numerical Integration, 2^{nd} Edition, Academic Press, Orlando.
17. Elden,L, LeWittmeyer-Koch, H.B.Nielsen, (2006) Introduction to Numerical Computation, analysis and MATLAB® illustrations, Overseas Press India Pvt Ltd.
18. Erwin,Kreyszig, Advanced Engineering Mathematics (1999), John Wiley and Sons, Inc.,New York.
19. Gautschi, W. (1970) On the Construction of Gaussian Quadrature Rules from Modified Moments, Math.Comp., Vol 24, pp 245 – 260.
20. Gautschi, W. (1982) An Algorithmic Implementation of the Generalized Christoffel Theorem, in Numerical Integration, Internat.Ser.Numer.Math.,ed.G.Hammerlin, Birkhauser, Basel,57, pp.89-106.
21. Gautschi, W. (1985) Orthogonal polynomials-constructive theory and applications, J.Comput. Appl. Math.,12&13, pp.61-76.
22. Gautschi, W (1997) Numerical Analysis: An Introduction, Birkhauser, Boston.
23. Gerald, C.F. and P.O. Wheatly. (1999) Applied Numerical Analysis, 6th Edn, Pearson Education Inc.
24. Gilat, A. (2004) MATLAB® An Introduction With Applications, John Wiley and Sons, Inc.
25. Greville, T.N.E. (1969) Introduction to Spline Functions, in: Theory and Application of Spline Functions, Academic Press, New York.
26. Hanselman,D and Bruce Littlefield, (2005) Mastering MATLAB®, Pearson Education, Inc.
27. Hilderbrand,F.B, Introduction To Numerical Analysis, (1988), Tata McGraw-Hill Publishing Co.Limited. New Delhi.
28. Hunt,B.R,. Lipsman.R.L, Rosenberg.J.M (2006), A Guide to MATLAB for Beginners and Experienced Users, Cambridge University Press.
29. Jain,M.K, S.R.K. Iyengar, R.K.Jain,(2003), Numerical Methods for Scientific and Engineering Computation, New Age International Publishers.
30. Kammerer, W.J., G.W. Reddien, and R.S. Varga (1974), Quadratic splines, Numerisohe Mathematik, 22, 241-251.
31. Kopal,Z. [1961]: "Numerical Analysis,"2^{nd} ed.,John Wiley & Sons,Inc., New York.
32. Krylov, V.I, (1984), Approximate Calculation of Integrals (Trans. Stroud A H) McMillan New York.
33. Marchuck, G.I (1982) Methods of Numerical Mathematics, second Edn, Springer Verlag, New York.
34. Micula,G Sanda Micula, Hand Book of Splines, Kluwer Academic Publishers (1999).
35. Moler,C.B. (2004) Numerical Computing with MATLAB, Society for Industrial and Applied Mathematics, Philadelphia:

36. Mukundan,R.(1998) and K.R.Ramakrishanan Moment Functions in Image Analysis, World Scientific Publishing Co.Pvt Ltd.
37. Otto, S.R and Denier.J P, (2005) An Introduction to Programming and Numerical Methods in MATLAB, Springer International Edition, Springer- Verlag London Ltd.
38. Palm, III.W.J, Introduction to MATLAB 7.4, (2008), Special Indian Edition, Tata Mc Graw Hill.
39. Parent, (2008) Computer Animation, Algorithms and Techniques, Elsevier Inc, second Edn.
40. Phillips, G.M. and P.J.Taylor (1996) Theory and applications of Numerical Analysis, second Edition, Academic Press, London.
41. Prenter, P.M, (1975) Splines and Variational Methods, Jhon Wiley & sons Inc.
42. Rathod.H.T, H.Y.Shrivalli and C.S.Nagabhushana, (2010) On a New Cubic Spline Interpolation, International e-Journal of Numerical Analysis and related topics, vol.4, pp [1-18].
43. Rogers, D.F and J.A.Adams (2002) Mathematical Elements of Computer Graphics, Tata Mc Graw-Hill Publishing Co.Ltd.
44. Sastry,S.S (2003) Introductory Methods Of Numerical Analysis, Prentice, Hall of India. Private Limited.
45. Schilling,R.J and S.L.Harris (2007) Applied Numerical Methods for Engineering, using MATLAB® and C, Books/Cole, (a part of Cengage Learning).
46. Schoenberg,I.J. (1967) On Spline Functions, in Inequalities, O.Shisha, ed 255-291, New York, Academic Press.
47. Schoenberg.I.J. (1947) Contribution to the Problem of Approximation of Equidistant Data by Analytic Functions, Quart. Appl. Math.,4:45-99.
48. Schoenberg,I.J. (1952) On Smoothing Operations and Their Generating Functions, Nat, Bur. Standards Rept. 1734.
49. Schoenberg,I.J. (1971) On Equidistant Cubic Spline Interpolation, Bull. Amer. Math. Soc.,77:1039-1044.
50. Simmons, G.F. (1985) Calculus with Analytic Geometry, Mc Graw Hill, New York.
51. Sommariva,A and M.Vianello, (2006) Gauss-Green Cubature over Spline Curvilinear Polygons, Applied Mathematics and Computation, vol 183, No.2, pp 1098-1107.
52. Stroud, A.H and D. Secrest,(1966) Gaussian Quadrature Formulas, Prentice Hall, Engle wood Cliffs, New Jersey.
53. Subbotin,Y.N.(1967) On Piecewise-Polynomial Approximation, Mat. Zametcki 1, 63-70 (Translation:1967. Math Notes 1, 41-46).
54. Todd,J.(ed.) 1962, A Survey of Numerical Analysis" McGraw-Hill Book Company, New York.
55. Web Resource: http://en.wikipedia.org/wiki/File:Rungesphenomenon.png.
56. Yang,W.Y, W.Cao, T.S.Chung, and J.Morris, (2005) Applied Numerical Methods using MATLAB®, John Wiley and Sons.Inc.