JOURNAL OF THE
CHUNGCHEONG MATHEMATICAL SOCIETY
Volume **37**, No. 4, November 2024
http://dx.doi.org/10.14403/jcms.2024.37.4.201

NUMERICAL GEOMETRY OF A BÉZIER SURFACE

Jong Ryul Kim

Abstract. We calculate numerically the normal curvature, the principal curvatures and the principal directions of a Bézier surface. Also we find geodesics on a Bézier surface by using Runge-Kutta method with Python.

1. Introduction

A Bézier surface with the set P of control points is defined by

$$S(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}P_{ij}B_i^m(u)B_j^n(v),$$

where

$$B_i^m(u)=\frac{m!}{i!(m-i!)}(1-u)^{m-i}u^i,$$
$$B_j^n(v)=\frac{n!}{j!(n-j!)}(1-v)^{n-j}v^j$$

is the Bernstein polynomials of degree m,n, respectively [1].

A cubic Bézier curve with four control points Q_0,Q_1,Q_2 and Q_3 is defined by

$$b(t)=\sum_{i=0}^{3}Q_iB_i^3(t)=\sum_{i=0}^{3}Q_i\frac{3!}{i!(3-i!)}(1-t)^{3-i}t^i.$$

Plotting Bézier curves and surfaces with different control points could be one of the best way for understanding how they work. So simply consider the case of $m=n=3$ a so called cubic Bézier surface. The Bernstein polynomials of degree 3 are

$$B_0^3(u)=(1-u)^3,\quad B_1^3(u)=3(1-u)^2u,\quad B_2^3(u)=3(1-u)u^2,\quad B_3^3(u)=u^3,$$

Received January 10, 2024; Accepted November 29, 2024.
2020 Mathematics Subject Classification: Primary 53C22; Secondary 65L06.
Key words and phrases: Bézier curve, Bézier surface, Runge-Kutta method, geodesic equations.

$B_0^3(v) = (1-v)^3, \quad B_1^3(v) = 3(1-v)^2 v, \quad B_2^3(v) = 3(1-v)v^2, \quad B_3^3(v) = v^3.$

So the cubic Bézier surface is

$$\begin{aligned}
S(u,v) &= \sum_{i=0}^{3}\sum_{j=0}^{3} P_{ij}B_i^3(u)B_j^3(v)\\
&= P_{00}B_0^3(u)B_0^3(v) + P_{01}B_0^3(u)B_1^3(v) + P_{02}B_0^3(u)B_2^3(v) + P_{03}B_0^3(u)B_3^3(v)\\
&+ P_{10}B_1^3(u)B_0^3(v) + P_{11}B_1^3(u)B_1^3(v) + P_{12}B_1^3(u)B_2^3(v) + P_{13}B_1^3(u)B_3^3(v)\\
&+ P_{20}B_2^3(u)B_0^3(v) + P_{21}B_2^3(u)B_1^3(v) + P_{22}B_2^3(u)B_2^3(v) + P_{23}B_2^3(u)B_3^3(v)\\
&+ P_{30}B_3^3(u)B_0^3(v) + P_{31}B_3^3(u)B_1^3(v) + P_{32}B_3^3(u)B_2^3(v) + P_{33}B_3^3(u)B_3^3(v)
\end{aligned}$$

with the set $P = \{P_{ij}\}$ of control points for $i = 0,1,2,3$ and $j = 0,1,2,3$

$$\begin{aligned}
&P_{00} = (0,0,0), \quad P_{01} = (0,2,0), \quad P_{02} = (0,4,2), \quad P_{03} = (0,6,0),\\
&P_{10} = (2,0,4), \quad P_{11} = (2,2,6), \quad P_{12} = (2,4,6), \quad P_{13} = (2,6,4),\\
&P_{20} = (4,0,4), \quad P_{21} = (4,2,6), \quad P_{22} = (4,4,6), \quad P_{23} = (4,6,0),\\
&P_{30} = (6,0,0), \quad P_{31} = (6,2,2), \quad P_{32} = (6,4,2), \quad P_{33} = (6,6,0)\,.
\end{aligned}$$

We plot them and see how they work in section 2. We give Python code for the shape operator in section 3. Then we calculate the normal curvature, the principal curvatures and the principal directions of a surface and a Bézier surface as well. In the last section, we find numerical geodesics on a Bézier surface by using Runge-Kutta method given in [2], [3].

2. A Bézier surface plot with Python

Here we give a cubic Bézier surface plot with Python. We use it to calculate numerically the normal curvature, the principal curvatures and the principal directions in section 3. Also we use it to find geodesics numerically in section 4.

```
import math
from math import *
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# A cubic Bézier surface plot with the set P of control points
def Bezier_surface(p):
    #The Bernstein polynomials of degree 3
```

```
bu0=lambda u,v:(1-u)**3
bu1=lambda u,v:3*((1-u)**2)*u
bu2=lambda u,v:3*(1-u)*u**2
bu3=lambda u,v:u**3
bv0=lambda u,v:(1-v)**3
bv1=lambda u,v:3*((1-v)**2)*v
bv2=lambda u,v:3*(1-v)*v**2
bv3=lambda u,v:v**3
# A cubic Bézier surface S(u,v)
S=lambda u,v:np.array([p[0,0,0]*bu0(u,v)*bv0(u,v),\
p[0,0,1]*bu0(u,v)*bv0(u,v),p[0,0,2]*bu0(u,v)*bv0(u,v) ] )+\
np.array([p[0,1,0]*bu0(u,v)*bv1(u,v),p[0,1,1]*bu0(u,v)*bv1(u,v),\
p[0,1,2]*bu0(u,v)*bv1(u,v) ] )+\
np.array([p[0,2,0]*bu0(u,v)*bv2(u,v),p[0,2,1]*bu0(u,v)*bv2(u,v),\
p[0,2,2]*bu0(u,v)*bv2(u,v) ] )+\
np.array([p[0,3,0]*bu0(u,v)*bv3(u,v),p[0,3,1]*bu0(u,v)*bv3(u,v),\
p[0,3,2]*bu0(u,v)*bv3(u,v) ] )+\
np.array([p[1,0,0]*bu1(u,v)*bv0(u,v),p[1,0,1]*bu1(u,v)*bv0(u,v),\
p[1,0,2]*bu1(u,v)*bv0(u,v) ] )+\
np.array([p[1,1,0]*bu1(u,v)*bv1(u,v),p[1,1,1]*bu1(u,v)*bv1(u,v),\
p[1,1,2]*bu1(u,v)*bv1(u,v)] )+\
np.array( [p[1,2,0]*bu1(u,v)*bv2(u,v),p[1,2,1]*bu1(u,v)*bv2(u,v),\
p[1,2,2]*bu1(u,v)*bv2(u,v) ])+\
np.array([p[1,3,0]*bu1(u,v)*bv3(u,v),p[1,3,1]*bu1(u,v)*bv3(u,v),\
p[1,3,2]*bu1(u,v)*bv3(u,v) ] )+\
np.array([p[2,0,0]*bu2(u,v)*bv0(u,v),p[2,0,1]*bu2(u,v)*bv0(u,v),\
p[2,0,2]*bu2(u,v)*bv0(u,v)] )+\
np.array([p[2,1,0]*bu2(u,v)*bv1(u,v),p[2,1,1]*bu2(u,v)*bv1(u,v),\
p[2,1,2]*bu2(u,v)*bv1(u,v)] )+\
np.array([p[2,2,0]*bu2(u,v)*bv2(u,v),p[2,2,1]*bu2(u,v)*bv2(u,v),\
p[2,2,2]*bu2(u,v)*bv2(u,v) ])+\
np.array([p[2,3,0]*bu2(u,v)*bv3(u,v),p[2,3,1]*bu2(u,v)*bv3(u,v),\
p[2,3,2]*bu2(u,v)*bv3(u,v) ])+\
np.array([p[3,0,0]*bu3(u,v)*bv0(u,v),p[3,0,1]*bu3(u,v)*bv0(u,v),\
p[3,0,2]*bu3(u,v)*bv0(u,v) ])+\
np.array([p[3,1,0]*bu3(u,v)*bv1(u,v),p[3,1,1]*bu3(u,v)*bv1(u,v),\
p[3,1,2]*bu3(u,v)*bv1(u,v) ] )+\
np.array([p[3,2,0]*bu3(u,v)*bv2(u,v),p[3,2,1]*bu3(u,v)*bv2(u,v),\
p[3,2,2]*bu3(u,v)*bv2(u,v) ])+\
np.array([p[3,3,0]*bu3(u,v)*bv3(u,v),p[3,3,1]*bu3(u,v)*bv3(u,v),\
      p[3,3,2]*bu3(u,v)*bv3(u,v) ])
      return S
u=np.linspace(0,1,10)
v=np.linspace(0,1,10)
# Control points p
p=np.array([ [ [0,0,0],[0,2,0],[0,4,2],[0,6,0] ],
[ [2,0,4],[2,2,6],[2,4,6],[2,6,4] ],
[ [4,0,4],[4,2,6],[4,4,6],[4,6,0] ],
[ [6,0,0],[6,2,2],[6,4,2],[6,6,0] ] ])
u,v = np.meshgrid(u,v)
bs=Bezier_surface(p)(u,v)
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(55, 35)
ax.plot_wireframe(bs[0],bs[1],bs[2], color='blue')
plt.show()
```

Figure 1.

```
# A cubic Bézier curve plot with four control points
def Bezier_curve(p):
      b=lambda t:np.array([p[0,0]*(1-t)**3,p[0,1]*(1-t)**3, p[0,2]*(1-t)**3])+\
      np.array([p[1,0]*3*((1-t)**2)*t,p[1,1]*3*((1-t)**2)*t,p[1,2]*3*((1-t)**2)*t])+\
      np.array([p[2,0]*3*(1-t)*(t**2),p[2,1]*3*(1-t)*(t**2),p[2,2]*3*(1-t)*(t**2)])+\
      np.array([p[3,0]*(t)**3,p[3,1]*(t)**3,p[3,2]*(t)**3])
      return b
xh0=np.array([0,0,0,0])
yh0=np.array([0,2,4,6])
zh0=np.array([0,2,-2,0])
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(35, 10)
```

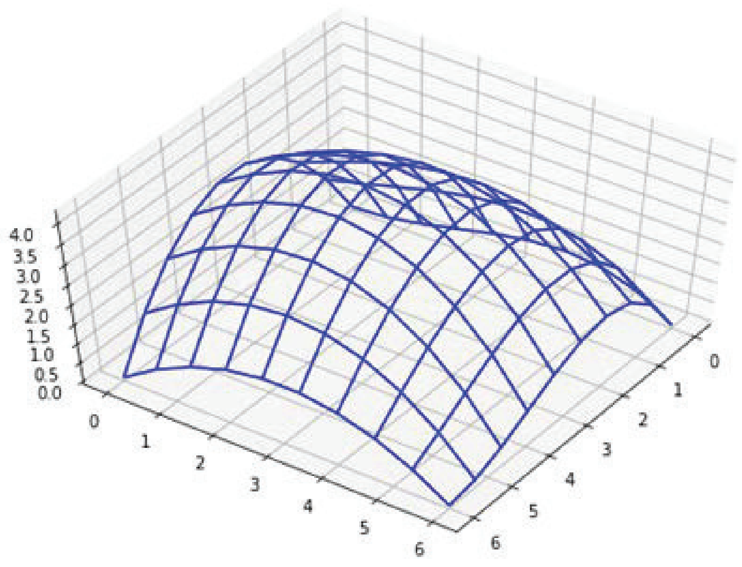

FIGURE 1. A Bézier surface with control points P

```
ax.set_xlim(0, 4)
ax.set_zlim(-2, 2)
ax.set_xlabel(”x”, size = 14)
ax.set_ylabel(”y”, size = 14)
ax.set_zlabel(”z”, size = 14)
p1=np.array([ [0,0,0],[0,2,2],[0,4,-2],[0,6,0] ])
b=lambda t:Bezier_curve(p1)(t)
t=np.linspace(0,1,15)
ax.plot(xh0,yh0,zh0,’g’,marker=’o’,markersize=10)
plt.plot(b(t)[0],b(t)[1],b(t)[2],’g’,marker=’>’,markersize=8)
plt.show()
```

Figure 2.

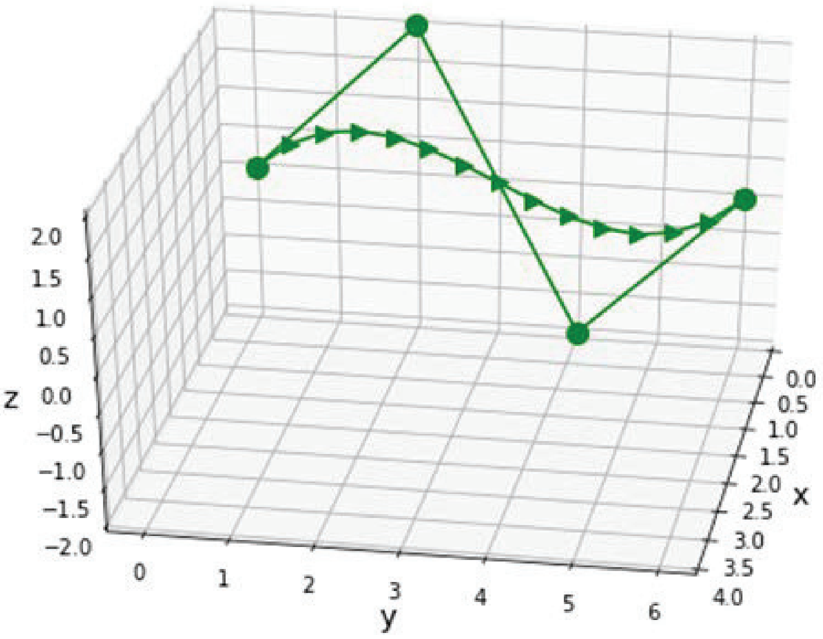

FIGURE 2. A Bézier curve of points p1

For the set P of control points in the introduction, we plot four cubic Bézier curves with the boundary four points

$$\begin{aligned}
&P_{00} = (0,0,0), \quad P_{01} = (0,2,0), \quad P_{02} = (0,4,2), \quad P_{03} = (0,6,0),\\
&P_{00} = (0,0,0), \quad P_{10} = (2,0,4), \quad P_{20} = (4,0,4), \quad P_{30} = (6,0,0),\\
&P_{30} = (6,0,0), \quad P_{31} = (6,2,2), \quad P_{32} = (6,4,2), \quad P_{33} = (6,6,0),\\
&P_{03} = (0,6,0), \quad P_{13} = (2,6,4), \quad P_{23} = (4,6,0), \quad P_{33} = (6,6,0)\,.
\end{aligned}$$

We plot the control points P, boundary Bézier curves and a Bézier surface of P.

Figure 3.

```
# plot two Bézier surfaces of control points p1 and p2
u=np.linspace(0,1,10)
v=np.linspace(0,1,10)
p1=np.array([ [ [0,0,0],[0,2,0],[0,4,2],[0,6,0] ],
```

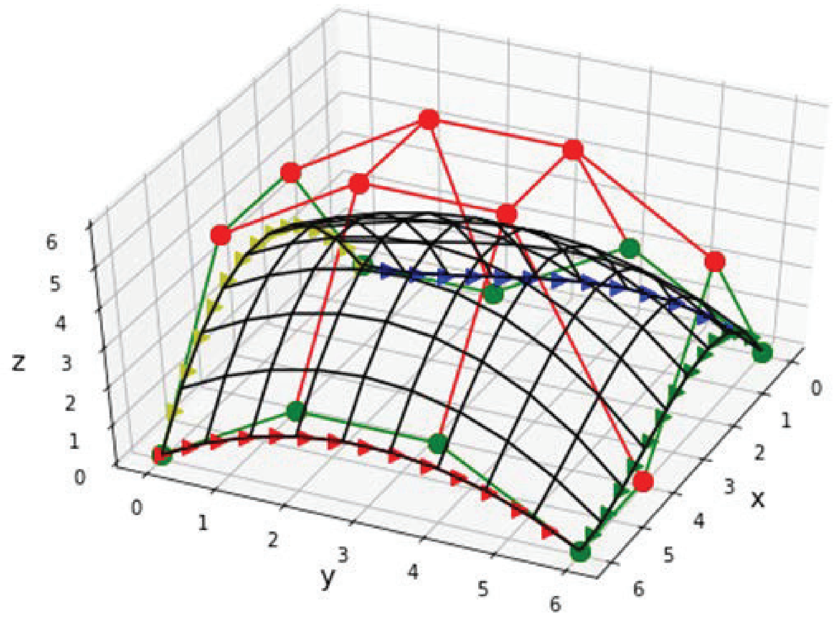

FIGURE 3. The control points P, boundary Bézier curves and a Bézier surface of P

```
[ [2,0,4],[2,2,6],[2,4,6],[2,6,4] ],
[ [4,0,4],[4,2,6],[4,4,6],[4,6,0] ],
[ [6,0,0],[6,2,2],[6,4,2],[6,6,0] ] ])
p2=np.array([ [ [0,0,0],[0,2,0],[0,4,2],[0,6,0] ],
[ [2,0+2,4],[2,2+2,6],[2,4+2,6],[2,6+2,4] ],
[ [4,0+2,4],[4,2+2,6],[4,4+2,6],[4,6+2,0] ],
[ [6,0,0],[6,2,2],[6,4,2],[6,6,0] ] ])
u,v = np.meshgrid(u,v)
bs1=Bezier_surface(p1)(u,v)
bs2=Bezier_surface(p2)(u,v)
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(55, 35)
ax.plot_wireframe(bs1[0],bs1[1],bs1[2], color='blue')
```

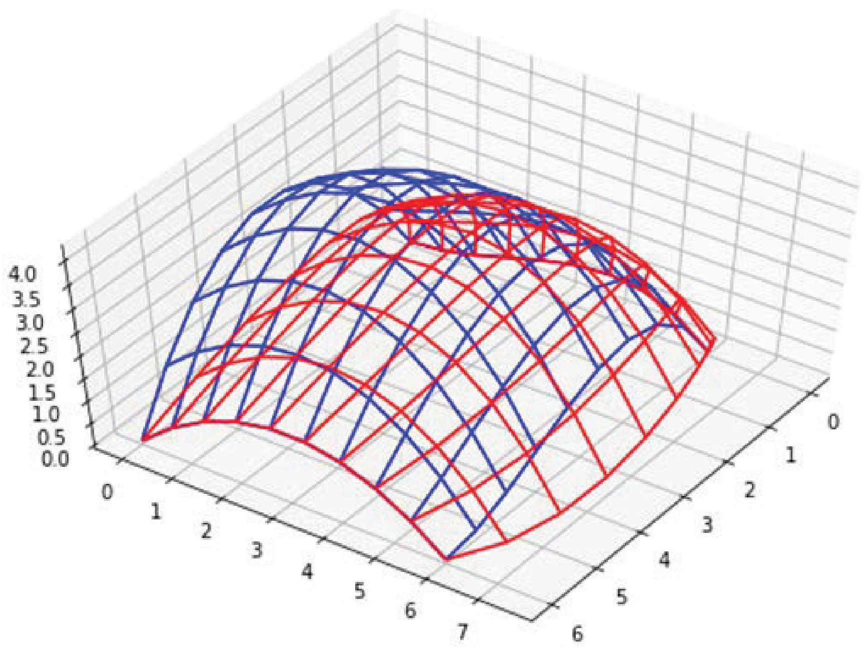

FIGURE 4. Two Bézier surfaces of control points p1 and p2

ax.plot_wireframe(bs2[0],bs2[1],bs2[2], color='red')

plt.show()

Figure 4.

3. Numerical shape operator of a Bézier surface

Let $\mathbf{X} : U \subseteq \mathbb{R}^2 \longrightarrow S \subset \mathbb{R}^3$ be a coordinate chart of a regular surface S

$$\mathbf{X}(u,v) = (x(u,v), y(u,v), z(u,v))\,.$$

Then we have a unit normal vector field $N = \frac{\mathbf{X}_u \times \mathbf{X}_v}{|\mathbf{X}_u \times \mathbf{X}_v|}$. The shape operator $S_N : T_pS \longrightarrow T_pS$ at a point $p \in S$ is defined by

$$S_N v = -D_v N, \quad v \in T_pS.$$

So the directional derivative of $N(u,v) = (N_1(u,v), N_2(u,v), N_3(u,v))$ in the direction of $\mathbf{X}_u, \mathbf{X}_u$ is given by

$$S_N\mathbf{X}_u = -D_{\mathbf{X}_u}N = -\begin{pmatrix} \frac{\partial N_1}{\partial u} & \frac{\partial N_1}{\partial v} \\ \frac{\partial N_2}{\partial u} & \frac{\partial N_2}{\partial v} \\ \frac{\partial N_3}{\partial u} & \frac{\partial N_3}{\partial v} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = -\begin{pmatrix} \frac{\partial N_1}{\partial u} \\ \frac{\partial N_2}{\partial u} \\ \frac{\partial N_3}{\partial u} \end{pmatrix},$$

$$S_N\mathbf{X}_v = -D_{\mathbf{X}_v}N = -\begin{pmatrix} \frac{\partial N_1}{\partial u} & \frac{\partial N_1}{\partial v} \\ \frac{\partial N_2}{\partial u} & \frac{\partial N_2}{\partial v} \\ \frac{\partial N_3}{\partial u} & \frac{\partial N_3}{\partial v} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -\begin{pmatrix} \frac{\partial N_1}{\partial v} \\ \frac{\partial N_2}{\partial v} \\ \frac{\partial N_3}{\partial v} \end{pmatrix},$$

respectively. For every unit tangent vector $u \in T_pS$, the normal curvature in the direction u is defined by $(S_Nu) \cdot u$. The maximum and minimum values of the normal curvature are called the principle curvatures and the corresponding directions are called the principle directions. For the different principle curvatures, the principle directions are orthogonal.

For Python code, from

$$S_N\mathbf{X}_u = a\mathbf{X}_u + c\mathbf{X}_v\,,$$

it follows that

$$(S_N\mathbf{X}_u) \cdot \mathbf{X}_u = a\mathbf{X}_u \cdot \mathbf{X}_u + c\mathbf{X}_v \cdot \mathbf{X}_u \tag{3.1}$$

and

$$(S_N\mathbf{X}_u) \cdot \mathbf{X}_v = a\mathbf{X}_u \cdot \mathbf{X}_v + c\mathbf{X}_v \cdot \mathbf{X}_v\,. \tag{3.2}$$

From (3.1), we get

$$\frac{(S_N\mathbf{X}_u) \cdot \mathbf{X}_u - c\mathbf{X}_v \cdot \mathbf{X}_u}{\mathbf{X}_u \cdot \mathbf{X}_u} = a\,.$$

Put it into (3.2), then we get

$$(S_N\mathbf{X}_u) \cdot \mathbf{X}_v = \frac{(S_N\mathbf{X}_u) \cdot \mathbf{X}_u - c\mathbf{X}_v \cdot \mathbf{X}_u}{\mathbf{X}_u \cdot \mathbf{X}_u}\mathbf{X}_u \cdot \mathbf{X}_v + c\mathbf{X}_v \cdot \mathbf{X}_v,$$

that is,

$$(S_N\mathbf{X}_u) \cdot \mathbf{X}_v = \frac{(S_N\mathbf{X}_u) \cdot \mathbf{X}_u}{\mathbf{X}_u \cdot \mathbf{X}_u}\mathbf{X}_u \cdot \mathbf{X}_v + c(\mathbf{X}_v \cdot \mathbf{X}_v - \frac{(\mathbf{X}_u \cdot \mathbf{X}_v)^2}{\mathbf{X}_u \cdot \mathbf{X}_u})\,.$$

So we get c and a. In the same way, we can calculate the following b and d

$$S_N\mathbf{X}_v = b\mathbf{X}_u + d\mathbf{X}_v\,.$$

Hence we get shape operator at each point $p \in S$

$$S_N = \begin{pmatrix} a & b \\ c & d \end{pmatrix}.$$

In this section, we calculate numerically the normal curvature of a Bézier surface by using Bézier surface plot with def Bezier_surface(p): in section 2.

```
u,v = np.meshgrid(u,v)
bs=lambda u,v:Bezier_surface(p)(u,v)
x=lambda u,v :bs(u,v)[0]
y=lambda u,v :bs(u,v)[1]
z=lambda u,v :bs(u,v)[2]
```

instead of a coordinate chart of a surface.

Let us plot two curves $S(0.5, v)$ and $S(u, 0.3)$ on a Bézier surface which meet at a point $S(0.5, 0.3)$ as in Figure 5. Then the tangent vectors $S_v(0.5, v)$ and $S_u(u, 0.3)$ are the basis of the tangent plane at a point $S(0.5, 0.3)$ instead of $\mathbf{X}_v(u_0, v)$ and $\mathbf{X}_u(u, v_0)$ at a point $\mathbf{X}(u_0, v_0)$ of a coordinate chart of a surface S.

```
p=np.array([ [ [0,0,0],[0,1,4],[0,2,4],[0,3,0] ],
[ [2,0,0],[2,1,4],[2,2,4],[2,3,0] ],
[ [4,0,0],[4,1,4],[4,2,4],[4,3,0] ],
[ [6,0,0],[6,1,4],[6,2,4],[6,3,0] ] ])
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(35, 10)
ax.set_zlim(-2, 4)
u=np.linspace(0,1,10)
v=np.linspace(0,1,10)
u,v = np.meshgrid(u,v)
bs=Bezier_surface(p)(u,v)
ax.plot_wireframe(bs[0],bs[1],bs[2], color='blue')
```

```
u1=np.array([0.5])
v1=np.array([0.3])
u1,v1 = np.meshgrid(u1,v1)
bs1=Bezier_surface(p)(u1,v1)
ax.plot(bs1[0,0],bs1[1,0],bs1[2,0],'r',marker='o',markersize=10,label='S(0.5,0.3)')
plt.legend(loc='upper right',fontsize='large')
u2=np.array([0.5])
v2=np.linspace(0,1,10)
u2,v2 = np.meshgrid(u2,v2)
bs2=Bezier_surface(p)(u2,v2)
ax.plot_wireframe(bs2[0],bs2[1],bs2[2], color='black',label='S(0.5,v)')
plt.legend(loc='upper right',fontsize='large')
u3=np.linspace(0,1,10)
v3=np.array([0.3])
u3,v3 = np.meshgrid(u3,v3)
bs3=Bezier_surface(p)(u3,v3)
ax.plot_wireframe(bs3[0],bs3[1],bs3[2], color='red',label='S(u,0.3)')
plt.legend(loc='upper right',fontsize='large')
plt.show()
```

Figure 5.

We calculate numerically the normal curvature, the principal curvatures and principal directions of a surface and a Bézier surface.

```
def diff_1(f,x,y):
    h=1e-4
    d=(f(x+h,y)-f(x-h,y))/(2*h)
    return d
def diff_2(f,x,y):
    h=1e-4
    d=(f(x,y+h)-f(x,y-h))/(2*h)
    return d
def dot_product(a,b):
```

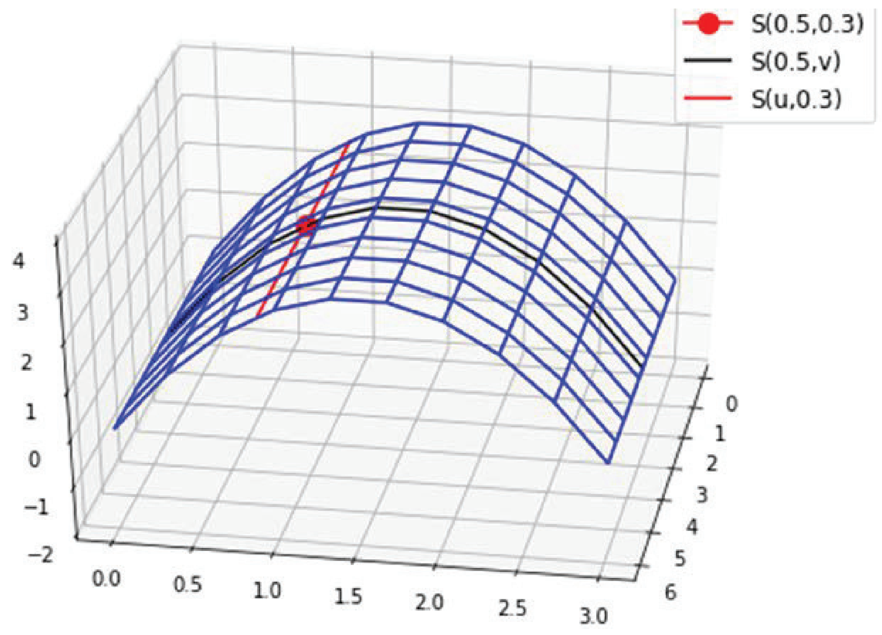

FIGURE 5. Two coordinate curves on a Bézier surface at a point S(0.5,0.3)

```
    n=len(a[0])
    sum=0
    for k in range(n):
        sum=sum+a[0,k]*b[0,k]
    return sum
def length_vector(a):
    n=len(a[0])
    sum=0
    for k in range(n):
        sum=sum+a[0,k]**2
    return sum**(1/2)
class normal_curvature:
    def __init__(self):
        self.p=None
    def derivative_normal(self,x,y,z,p):
        x_u=lambda u,v:diff_1(x,u,v)
        y_u=lambda u,v:diff_1(y,u,v)
        z_u=lambda u,v:diff_1(z,u,v)
        x_v=lambda u,v:diff_2(x,u,v)
        y_v=lambda u,v:diff_2(y,u,v)
        z_v=lambda u,v:diff_2(z,u,v)
        # X_u, X_v at a point p
        self.X_u=np.array([[x_u(p[0],p[1]),y_u(p[0],p[1]),z_u(p[0],p[1])]])
        self.X_v=np.array([[x_v(p[0],p[1]),y_v(p[0],p[1]),z_v(p[0],p[1])]])
        X_u=lambda u,v:[x_u(u,v),y_u(u,v),z_u(u,v)]
        X_v=lambda u,v:[x_v(u,v),y_v(u,v),z_v(u,v)]
        # cross product of X_u and X_v
        c_1=lambda u,v:X_u(u,v)[1]*X_v(u,v)[2]-X_u(u,v)[2]*X_v(u,v)[1]
        c_2=lambda u,v:-X_u(u,v)[0]*X_v(u,v)[2]+X_u(u,v)[2]*X_v(u,v)[0]
        c_3=lambda u,v:X_u(u,v)[0]*X_v(u,v)[1]-X_u(u,v)[1]*X_v(u,v)[0]
        cross_product=lambda u,v:np.array([[c_1(u,v),c_2(u,v),c_3(u,v)]])
        # unit normal vector field N
        N=lambda u,v:cross_product(u,v)/(length_vector(cross_product(u,v)))
        # DN Jacobian matrix of N
        N_1=lambda u,v:N(u,v)[0,0]
        N_2=lambda u,v:N(u,v)[0,1]
        N_3=lambda u,v:N(u,v)[0,2]
        N_1_u=lambda u,v:diff_1(N_1,u,v)
        N_1_v=lambda u,v:diff_2(N_1,u,v)
        N_2_u=lambda u,v:diff_1(N_2,u,v)
        N_2_v=lambda u,v:diff_2(N_2,u,v)
        N_3_u=lambda u,v:diff_1(N_3,u,v)
        N_3_v=lambda u,v:diff_2(N_3,u,v)
        N_1_u_v=lambda u,v:[N_1_u(u,v), N_1_v(u,v)]
        N_2_u_v=lambda u,v:[N_2_u(u,v), N_2_v(u,v)]
        N_3_u_v=lambda u,v:[N_3_u(u,v), N_3_v(u,v)]
        self.DN=lambda u,v:np.array([N_1_u_v(u,v),N_2_u_v(u,v),N_3_u_v(u,v)])
    def shape_operator(self,x,y,z,p):
        self.derivative_normal(x,y,z,p)
        x_u=np.array([[1,0]])
        x_v=np.array([[0,1]])
        E=dot_product(self.X_u,self.X_u)
        F=dot_product(self.X_u,self.X_v)
        G=dot_product(self.X_v,self.X_v)
        sn11=dot_product(np.dot(-self.DN(p[0],p[1]),x_u.T).T, self.X_u)
        sn12=dot_product(np.dot(-self.DN(p[0],p[1]),x_u.T).T, self.X_v)
        c=(sn12-(sn11*F)/E)/(G-(F**2/E))
        a=(sn11-c*F)/E
```

```
        sn21=dot_product(-np.dot(self.DN(p[0],p[1]),x_v.T).T, self.X_u)
        sn22=dot_product(-np.dot(self.DN(p[0],p[1]),x_v.T).T, self.X_v)
        d=(sn22-(sn21*F)/E)/(G-(F**2/E))
        b=(sn21-d*F)/E
        matrix=[]
        matrix.append(a)
        matrix.append(b)
        matrix.append(c)
        matrix.append(d)
        self.q=np.array(matrix).reshape(2,2)
        return self.q
    def normal_curvature_all_directions(self,x,y,z,p,n):
        self.shape_operator(x,y,z,p)
        list,ss,t=[ ],[ ],[ ]
        for k in range(n+1):
            u=np.array([[np.cos((2*pi)*(k/n)),np.sin((2*pi)*(k/n))]])
            v=np.cos((2*pi)*(k/n))*(self.X_u/length_vector(self.X_u))\
            +np.sin((2*pi)*(k/n))*(self.X_v/length_vector(self.X_v))
            # unit tangent vector
            unit=u/length_vector(v)
            ss.append((v/length_vector(v)).flatten().tolist())
            shape_operator=np.dot(self.q,unit.T)
            normal_curvature=dot_product(shape_operator.T,unit)
            t.append(normal_curvature)
            list.append([k,normal_curvature])
        tt=np.array(t)
        k_1,k_2=np.max(tt),np.min(tt)
        print("principal curvature max={ } min={ } ".format(k_1,k_2))
        self.list=np.array(list)
        self.ss=np.array(ss)
    def principal_directions(self,x,y,z,p,n):
        self.normal_curvature_all_directions(x,y,z,p,n)
        c=np.hstack((self.list,self.ss))
        self.c2=c1=c[:,1::]
        self.d=self.c2[np.argsort(self.c2[:,0])]
        v_1,v_2=self.d[0][1::],self.d[-1][1::]
        print("principal directions max={} min={} ".format(v_1,v_2))
```

We calculate the principal curvatures and principal directions of a torus.

```
n=normal_curvature()
```

```
p=np.array([0,0])

ff=lambda x,y:(6 + 2*np.cos(x))*np.cos(y)

g=lambda x,y:(6 + 2*np.cos(x))*np.sin(y)

h=lambda x,y:2*np.sin(x)

n.principal_directions(ff,g,h,p,20)
```

The output of the above code is

```
principal curvature max=0.49999999862793104 min=0.12499999965698276

principal directions max=[0.000000e+00 1.000000e+00 6.123234e-17] min=[
0.0000000e+00 -2.4492936e-16 1.0000000e+00] .
```

We calculate the principal curvatures and principal directions of a Bézier surface by using Bézier surface plot with def Bézier surface(p): in section 2.

```
q=np.array([ [ [0,0,0],[0,1,4],[0,2,4],[0,3,0] ],

[ [2,0,0],[2,1,4],[2,2,4],[2,3,0] ],

[ [4,0,0],[4,1,4],[4,2,4],[4,3,0] ],

[ [6,0,0],[6,1,4],[6,2,4],[6,3,0] ] ])

p1=np.array([0.5,0.3])

n=normal_curvature()

bs=lambda u,v:Bezier_surface(q)(u,v)

x=lambda u,v :bs(u,v)[0]

y=lambda u,v :bs(u,v)[1]

z=lambda u,v :bs(u,v)[2]

n.principal_directions(x,y,z,p1,100)
```

The output of the above code is

```
principal curvature max=-3.26898347899374e-11 min=-0.3970029923668647

principal directions max=[-3.92216785e-13 5.29998940e-01 8.47998304e-
01] min=[ 1.00000000e+00 -1.85037171e-13 0.00000000e+00] .
```

4. Numerical calculations of geodesics on a Bézier surface

Let $\mathbf{X} : U \subseteq \mathbb{R}^2 \longrightarrow S \subset \mathbb{R}^3$ be a coordinate chart of a surface S

$$\mathbf{X}(u,v) = (x(u,v), y(u,v), z(u,v)).$$

The Christoffel symbols of S in the parametrization $\mathbf{X}$ are defined by

$$\begin{aligned}
\mathbf{X}_{uu} &= \Gamma^1_{11}\mathbf{X}_u + \Gamma^2_{11}\mathbf{X}_v + (\mathbf{X}_{uu}\cdot N)N\,,\\
\mathbf{X}_{uv} &= \Gamma^1_{12}\mathbf{X}_u + \Gamma^2_{12}\mathbf{X}_v + (\mathbf{X}_{uv}\cdot N)N\,,\\
\mathbf{X}_{vv} &= \Gamma^1_{22}\mathbf{X}_u + \Gamma^2_{22}\mathbf{X}_v + (\mathbf{X}_{vv}\cdot N)N\,,
\end{aligned}$$

where $N = \frac{\mathbf{X}_u\times\mathbf{X}_v}{|\mathbf{X}_u\times\mathbf{X}_v|}$. So we get

$$\Gamma^1_{11} = \frac{GE_u - 2\,FF_u + FE_v}{2(EG-F^2)},\ \Gamma^1_{12} = \frac{GE_v - FG_u}{2(EG-F^2)},\ \Gamma^1_{22} = \frac{2\,GF_v - GG_u - FG_v}{2(EG-F^2)}\,,$$

$$\Gamma^2_{11} = \frac{-FE_u + 2\,EF_u - EE_v}{2(EG-F^2)},\ \Gamma^2_{12} = \frac{-FE_v + EG_u}{2(EG-F^2)},\ \Gamma^2_{22} = \frac{-2\,FF_v + FG_u + EG_v}{2(EG-F^2)}\,.$$

For an orthonormal basis $\{\alpha',\, N,\, \alpha'\times N\}$ along a unit speed curve $\alpha(t) \subset S$, the geodesic curvature of a curve on a surface is defined by $k_g = (\alpha''\cdot\alpha'\times N)$. If k_g is zero, then we have the following differential geodesic equations([4])

$$\begin{aligned}
u'' + \Gamma^1_{11}(u')^2 + 2\,\Gamma^1_{12}(u')(v') + \Gamma^1_{22}(v')^2 = 0\,,\\
v'' + \Gamma^2_{11}(u')^2 + 2\,\Gamma^2_{12}(u')(v') + \Gamma^2_{22}(v')^2 = 0\,.
\end{aligned}$$

Using Runge-Kutta method for solving the geodesic equations, we showed numerical geodesics on a surface in [2]. With the same method, we find numerical geodesics on a Bézier surface with

def Runge_Kutta_2_double_coordinate_chart(x,y,z,t0,t,h,u_0,u1_0,v_0,v1_0): in [2] and def Bezier_surface(p): in section 2.

```
u,v = np.meshgrid(u,v)

bs=lambda u,v:Bezier_surface(p)(u,v)

x=lambda u,v :bs(u,v)[0]

y=lambda u,v :bs(u,v)[1]

z=lambda u,v :bs(u,v)[2]
```

instead of a coordinate chart of a surface.

Exsample 1.

```
u=np.linspace(0,1,10)
v=np.linspace(0,1,10)
p=np.array([ [ [0,0,0],[0,2,0],[0,4,2],[0,6,0] ],
[ [2,0,4],[2,2,6],[2,4,6],[2,6,4] ],
[ [4,0,4],[4,2,6],[4,4,6],[4,6,0] ],
[ [6,0,0],[6,2,2],[6,4,2],[6,6,0] ] ])
u,v = np.meshgrid(u,v)
bs=lambda u,v:Bezier_surface(p)(u,v)
x=lambda u,v :bs(u,v)[0]
y=lambda u,v :bs(u,v)[1]
z=lambda u,v :bs(u,v)[2]
th=th=np.radians(0)
u1_0=np.cos(th)
v1_0=np.sin(th)
t1=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.7,0.03,0.2,u1_0,0.4,v1_0)
t1.append([rk[1],rk[2]])
t1=np.array(t1)
x1,y1,z1=bs(t1[0,0],t1[0,1])[0],bs(t1[0,0],t1[0,1])[1],bs(t1[0,0],t1[0,1])[2]
t2=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.7,0.03,0.2,u1_0,0.6,v1_0)
t2.append([rk[1],rk[2]])
t2=np.array(t2)
x2,y2,z2=bs(t2[0,0],t2[0,1])[0],bs(t2[0,0],t2[0,1])[1],bs(t2[0,0],t2[0,1])[2]
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(55, 35)
ax.plot_wireframe(x(u,v),y(u,v),z(u,v), color='blue')
ax.plot(x1, y1, z1, 'ro')
ax.plot(x2, y2, z2, 'go')
```

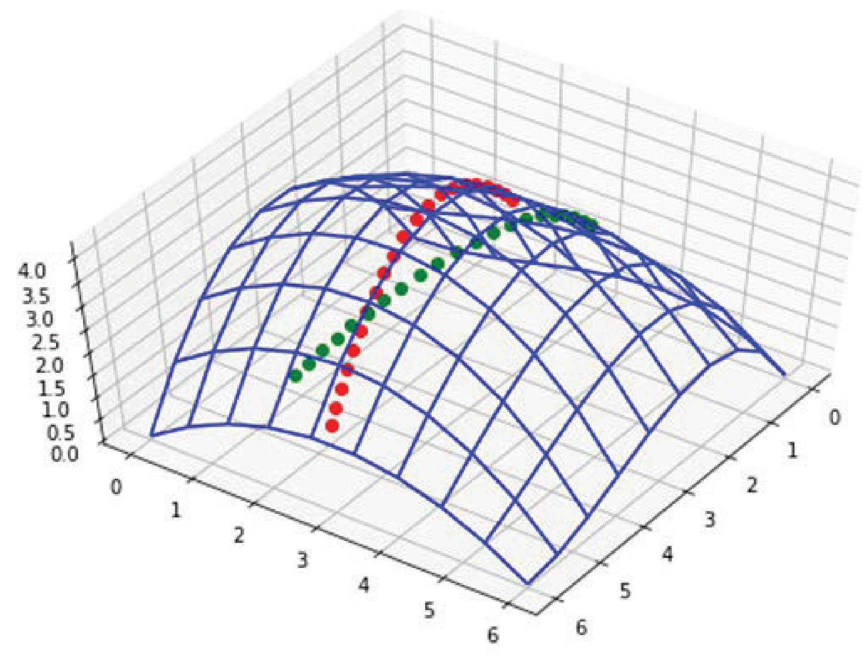

FIGURE 6. Geodesics on a Bézier surface (Example 1)

plt.show()

Figure 6.

Exsample 2.

u=np.linspace(0,1,10)

v=np.linspace(0,1,10)

p=np.array([[[0,0,0],[0,2,0],[0,4,2],[0,6,0]],

[[2,0+2,4],[2,2+2,6],[2,4+2,6],[2,6+2,4]],

[[4,0+2,4],[4,2+2,6],[4,4+2,6],[4,6+2,0]],

[[6,0,0],[6,2,2],[6,4,2],[6,6,0]]])

u,v = np.meshgrid(u,v)

bs=lambda u,v:Bezier_surface(p)(u,v)

x=lambda u,v :bs(u,v)[0]

```
y=lambda u,v :bs(u,v)[1]
z=lambda u,v :bs(u,v)[2]
th=th=np.radians(0)
u1_0=np.cos(th)
v1_0=np.sin(th)
t1=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.7,0.03,0.2,u1_0,0.4,v1_0)
t1.append([rk[1],rk[2]])
t1=np.array(t1)
x1,y1,z1=bs(t1[0,0],t1[0,1])[0],bs(t1[0,0],t1[0,1])[1],bs(t1[0,0],t1[0,1])[2]
t2=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.7,0.03,0.2,u1_0,0.6,v1_0)
t2.append([rk[1],rk[2]])
t2=np.array(t2)
x2,y2,z2=bs(t2[0,0],t2[0,1])[0],bs(t2[0,0],t2[0,1])[1],bs(t2[0,0],t2[0,1])[2]
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection=’3d’)
ax.view_init(55, 35)
ax.plot_wireframe(x(u,v),y(u,v),z(u,v), color=’blue’)
ax.plot(x1, y1, z1, ’ro’)
ax.plot(x2, y2, z2, ’go’)
plt.show()
```

Figure 7.

Exsample 3.

```
u=np.linspace(0,1,10)
v=np.linspace(0,1,10)
p=np.array([ [ [0,0,0],[0,2,4],[0,4,-4],[0,6,0] ],
[ [2,0,0],[2,2,4],[2,4,-4],[2,6,0] ],
```

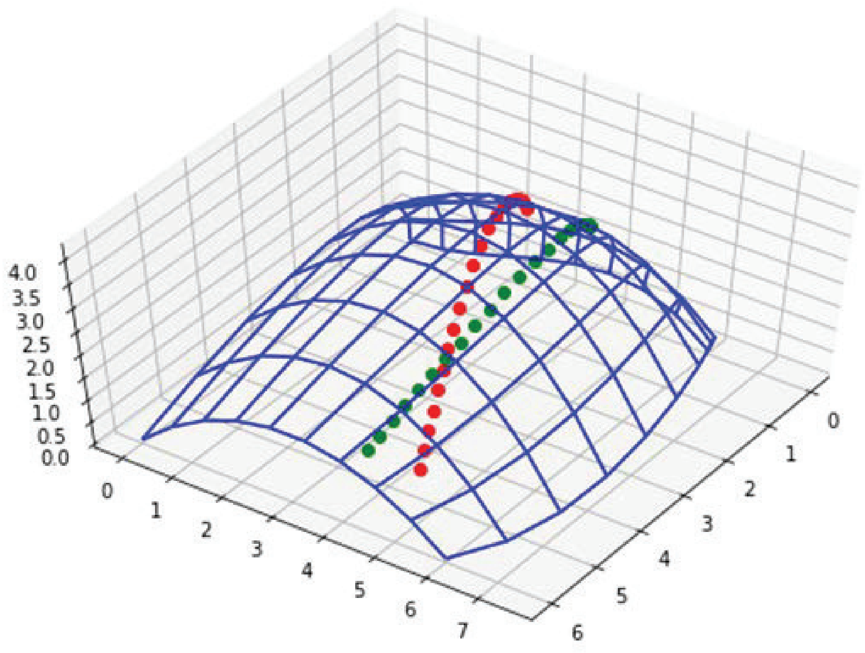

FIGURE 7. Geodesics on a Bézier surface (Example 2)

```
[ [4,0,0],[4,2,4],[4,4,-4],[4,6,0]],
[ [6,0,0],[6,2,4],[6,4,-4],[6,6,0]]])
u,v = np.meshgrid(u,v)
bs=lambda u,v:Bezier_surface(p)(u,v)
x=lambda u,v :bs(u,v)[0]
y=lambda u,v :bs(u,v)[1]
z=lambda u,v :bs(u,v)[2]
th=th=np.radians(90)
u1_0=np.cos(th)
v1_0=np.sin(th)
t1=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.8,0.05,0.3,u1_0,0.1,v1_0)
t1.append([rk[1],rk[2]])
t1=np.array(t1)
```

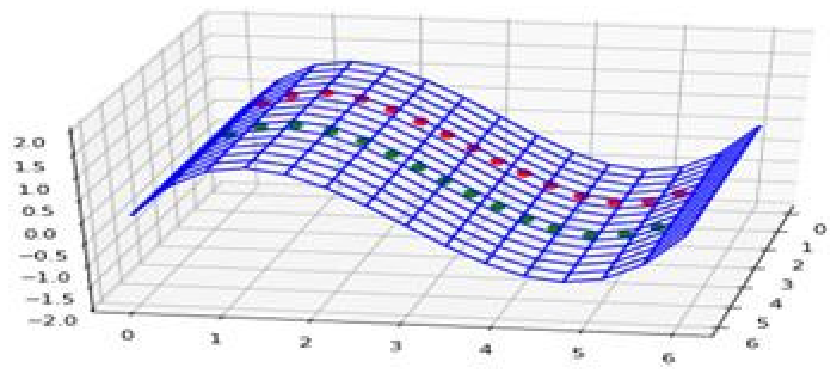

FIGURE 8. Geodesics on a Bézier surface (Example 3)

```
x1,y1,z1=bs(t1[0,0],t1[0,1])[0],bs(t1[0,0],t1[0,1])[1],bs(t1[0,0],t1[0,1])[2]
t2=[ ]
rk=Runge_Kutta_2_double_coordinate_chart(x,y,z,0.1,0.8,0.05,0.6,u1_0,0.1,v1_0)
t2.append([rk[1],rk[2]])
t2=np.array(t2)
x2,y2,z2=bs(t2[0,0],t2[0,1])[0],bs(t2[0,0],t2[0,1])[1],bs(t2[0,0],t2[0,1])[2]
fig = plt.figure(figsize=(9, 6))
ax = fig.add_subplot(111, projection='3d')
ax.view_init(55, 35)
ax.plot_wireframe(x(u,v),y(u,v),z(u,v), color='blue')
ax.plot(x1, y1, z1, 'ro')
ax.plot(x2, y2, z2, 'go')
plt.show()
```

Figure 8.

References

[1] G. Farin, *Curves and surfaces for CAGD a practical guide fifth edition*, San Diego, Academic Press, 2002.

[2] J. R. Kim, *Numerical Geodesics on a surface with Python*, J. Korean Soc. Math. Educ. Ser. B: Pure Appl. Math., **30** No. 1, (2023), 83–108

[3] J. R. Kim, *Python data analysis matrix mathematics*, Kyungmoon Press, 2020

[4] John McCleary, *Geometry from a Differentiable Viewpoint*, Cambridge university pres, 1994

Jong Ryul Kim
Department of Mathematics
Kunsan National University
Kunsan, 573-701, Republic of Korea
E-mail: `kimjr0@kunsan.ac.kr`