UNIVERSITÀ
DEGLI STUDI
FIRENZE

Relazione di Metodi numerici per la grafica:

Interpolazione di superfici alla Hermite

Di

Filippo Mameli

Anno Accademico 2018-2019

1 Patch bicubico interpolante alla Hermite

L'interpolazione di Hermite a differenza dell'interpolazione di Lagrande si basa, non solo sull'utilizzo dei punti della funzione vettoriale, ma aggiunge come parametri in input anche le derivate parziali e miste. Il patch bicubico nella forma di Hermite è dato da:

$$X(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} h_{ij} H_3^3(u) H_3^3(v); \quad 0 \geq u, v \geq 1;$$

dove H_i^3 sono le funzioni cubiche di Hermite definite nel modo seguente:

$$H_0^3(t) = 1 - 3t^2 + 2t^3; H_1^3(t) = t - 2t^2 + t^3; H_2^3(t) = t^3 - t^2; H_3^3(t) = 3t^2 - 2t^3.$$

I valori di h_{ij} sono dati dalla matrice:

$$\mathbf{C} = \begin{pmatrix} \mathbf{X}(0,0) & \mathbf{X}_t(0,0) & \mathbf{X}_t(0,1) & \mathbf{X}(0,1) \\ \mathbf{X}_s(0,0) & \mathbf{X}_{st}(0,0) & \mathbf{X}_{st}(0,1) & \mathbf{X}_s(0,1) \\ \mathbf{X}_s(1,0) & \mathbf{X}_{st}(1,0) & \mathbf{X}_{st}(1,1) & \mathbf{X}_s(1,1) \\ \mathbf{X}(1,0) & \mathbf{X}_t(1,0) & \mathbf{X}_t(1,1) & \mathbf{X}(1,1) \end{pmatrix}$$

La matrice può essere vista come 4 gruppi di matrici 2x2 contenenti le informazioni sugli angoli del patch.

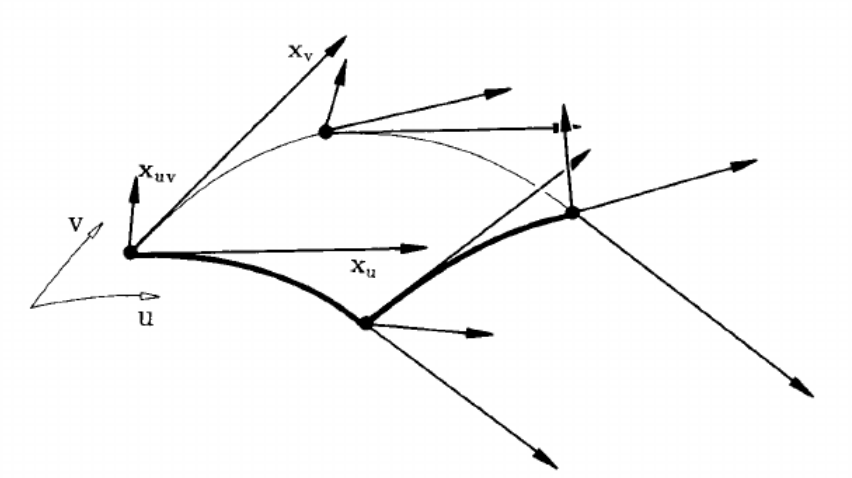

Figura 1: Rappresentazione dei punti e delle derivate

2 Implementazione in Matlab

Per testare il codice scegliamo come superficie il toroide in Figura 1. Da questo prendiamo una porzione più piccola per ricavare i dati da usare in input per l'interpolazione. La superficie del toro è stata disegnata eseguendo il Codice 1.

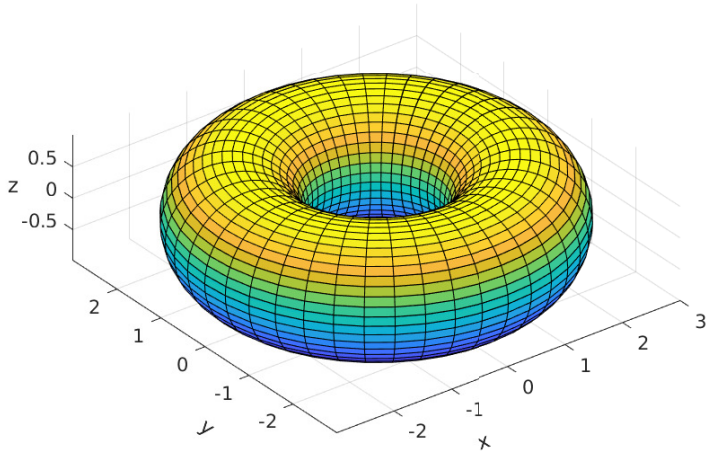

Figura 2: Superficie di test

```
s_a = 0; s_b = 7;
s = linspace(s_a, s_b, 50);
t_a = 0; t_b = 7;
t = linspace(t_a, t_b, 50);
[ss, tt] = meshgrid(s, t);
x = (2 + cos(tt)).*cos(ss);
y = (2 + cos(tt)).*sin(ss);
z = sin(tt);
surf(x, y, z);axis equal; xlabel('x', 'Rotation',20); ylabel('y',
    'Rotation',-20); zlabel('z', 'Rotation',0);
```

Codice 1: Plot della superficie

Prendiamo solo una porzione della superficie delimitata dai parametri s_a, s_b, t_a e t_b. Su questa con il Codice 2 visualizziamo in Figura 3 i quattro angoli che useremo per la costruzione della matrice di Hermite.

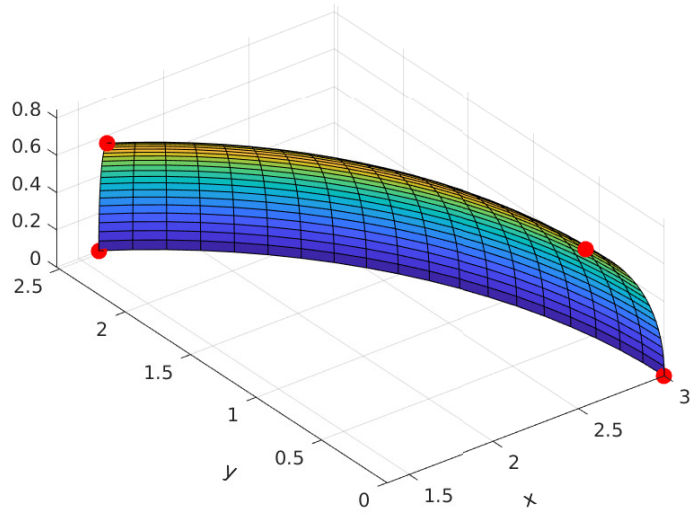

Figura 3: Angoli della porzione di toro

```
cx_1 = (2 + cos(t_a)).*cos(s_a);
cy_1 = (2 + cos(t_a)).*sin(s_a);
cz_1 = sin(t_a);

cx_2 = (2 + cos(t_b)).*cos(s_a);
cy_2 = (2 + cos(t_b)).*sin(s_a);
cz_2 = sin(t_b);

cx_3 = (2 + cos(t_b)).*cos(s_b);
cy_3 = (2 + cos(t_b)).*sin(s_b);
cz_3 = sin(t_b);

cx_4 = (2 + cos(t_a)).*cos(s_b);
cy_4 = (2 + cos(t_a)).*sin(s_b);
cz_4 = sin(t_a);
```

```
surf(x, y, z);axis equal; xlabel('x', 'Rotation',20); ylabel('y',
    'Rotation',-20); hold on;
plot3(cx_1, cy_1, cz_1, 'r.', 'MarkerSize', 30);
plot3(cx_2, cy_2, cz_2, 'r.', 'MarkerSize', 30);
plot3(cx_3, cy_3, cz_3, 'r.', 'MarkerSize', 30);
plot3(cx_4, cy_4, cz_4, 'r.', 'MarkerSize', 30);
```

Codice 2: Visualizzazione dei 4 angoli della porzione di toro

La matrice **C** è una matrice a tre piani, ciascuno 4x4. Per ricavare i valori delle derivate parziali e miste utilizziamo il *Symbolic Math Toolbox* di MATLAB . Definiamo le funzioni $f(x)$, $f(y)$ e $f(z)$ e i parametri s e t. Nel Codice 3 si può vedere come utilizzando la funzione *diff* si possa ricavare le derivate sui parametri s, t oppure s e t per ognuna delle funzioni definite.

```
C = zeros(4,4,3);
syms s t f_x(s,t) f_y(s,t) f_z(s,t)

f_x(s,t) = (2 + cos(t)).*cos(s);
f_y(s,t) = (2 + cos(t)).*sin(s);
f_z(s,t) = sin(t);

df_xt = diff(f_x, t);
df_xs = diff(f_x, s);
df_xst = diff(f_x, s, t);

df_yt = diff(f_y, t);
df_ys = diff(f_y, s);
df_yst = diff(f_y, s, t);

df_zt = diff(f_z, t);
df_zs = diff(f_z, s);
df_zst = diff(f_z, s, t);
```

Codice 3: Derivate delle funzioni della superficie

Per ogni piano della matrice la costruzione è uguale. Nel Codice 4 vediamo in dettaglio la definizione del piano X.

Analizzando i 4 elementi nell'angolo in alto a sinistra della matrice vediamo che:

- Nella posizione (1, 1) abbiamo il punto di interpolante;
- Nelle posizioni (1, 2) e (2, 1) abbiamo i valori delle funzioni derivate rispettivamente per t e per s in s_a e t_a;
- Nella posizione (2, 2) abbiamo il valore della funzione derivata mista in s_a e t_a;

```
C_x = zeros(4,4);
C_x(1,1) = f_x(s_a, t_a);
C_x(1,2) = df_xt(s_a, t_a);
C_x(2,1) = df_xs(s_a, t_a);
C_x(2,2) = df_xst(s_a, t_a);

C_x(1,4) = f_x(s_a, t_b);
C_x(1,3) = df_xt(s_a, t_b);
C_x(2,4) = df_xs(s_a, t_b);
C_x(2,3) = df_xst(s_a, t_b);

C_x(4,1) = f_x(s_b, t_a);
C_x(4,2) = df_xt(s_b, t_a);
C_x(3,1) = df_xs(s_b, t_a);
C_x(3,2) = df_xst(s_b, t_a);

C_x(4,4) = f_x(s_b, t_b);
C_x(4,3) = df_xt(s_b, t_b);
C_x(3,4) = df_xs(s_b, t_b);
C_x(3,3) = df_xst(s_b, t_b);
```

Codice 4: Definizione del piano X della matrice C

Iterando lo stesso procedimento per i piani Y e Z della matrice $\mathbf{C}$ possiamo passare al calcolo del patch in forma di Hermite. Utilizzando il calcolo matriciale abbiamo:

$$\mathbf{X}(s,t) = \mathbf{H}^T(s)\,\mathbf{C}\,\mathbf{H}(t)$$

dove $\mathbf{H}(t) = (H_0^3(t), H_1^3(t), H_2^3(t), H_3^3(t))$.
Nel Codice 5 vediamo prima la definizione delle funzioni della base di Hermite per poi passare calcolo di $\mathbf{X}(s,t)$. In Figura 4 viene mostrata la porzione di superficie disegnata utilizzando il patch bicubico ricavato.

```
syms H_0_t(t) H_1_t(t) H_2_t(t) H_3_t(t) H_0_s(s) H_1_s(s) H_2_s(s)
    H_3_s(s)

H_0_t(t) = 1 - 3*t.^2 + 2.*t.^3;
H_1_t(t) = t - 2*t.^2 + t.^3;
H_2_t(t) = t.^3 - t.^2;
H_3_t(t) = 3*t.^2 - 2*t.^3;

H_0_s(s) = 1 - 3*s.^2 + 2*s.^3;
H_1_s(s) = s - 2*s.^2 + s.^3;
H_2_s(s) = s.^3 - s.^2;
H_3_s(s) = 3*s.^2 - 2*s.^3;

H_t = [H_0_t(t); H_1_t(t); H_2_t(t); H_3_t(t)];
H_s = [H_0_s(s); H_1_s(s); H_2_s(s); H_3_s(s)];

X_st = symfun((H_s.') * C(:,:,1) * H_t, [s,t]);
Y_st = symfun((H_s.') * C(:,:,2) * H_t, [s,t]);
Z_st = symfun((H_s.') * C(:,:,3) * H_t, [s,t]);
```

Codice 5: Patch in forma di Hermite

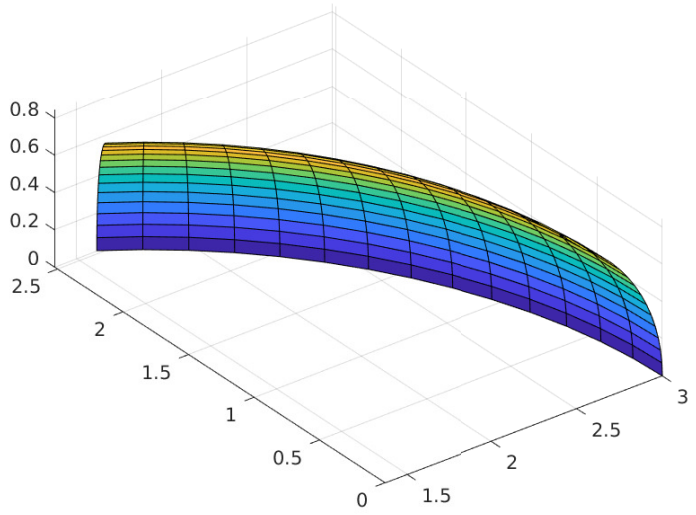

Figura 4: Visualizzazione del patch interpolante in forma di Hermite

3 Errore di interpolazione

In questo sezione analiziamo l'errore di interpolazione confrontando i valori esatti della superficie con i valori del patch interpolante. I parametri s_a, s_b, t_a e t_b presi per eseguire i test appartengono al dominio parametrico $[0,\ 1]^2$, utilizzando questi valori la norma 2, calcolata con il Codice 6, risulta:

- 0.0717 per le ascisse;
- 0.0396 per le ordinate;
- 0.0114 per la quota

data l'ampiezza del dominio parametrico-st i valori degli errori sono rilevanti come ci potevamo aspettare; passando da $[0, 1]^2$ a $[0, 0.5]^2$ abbiamo invece dei valori di norma 2 pari a:

- 0.0053 per le ascisse;
- 0.0014 per le ordinate;

- 3.7101e-04 per la quota.

L'errore in questo caso diminuisce notevolmente. Questo ci fa capire che questa tecnica di interpolazione è molto più efficace prendendo domini parametrici piccoli. Nella Sessione 6 vedremo come la divisione in sottointervalli sempre più piccoli porterà alla diminuzione e al quasi annullamento dell'errore.

```
norm(abs(X_hermite - X))
norm(abs(Y_hermite - Y))
norm(abs(Z_hermite - Z))
```

Codice 6: Calcolo della Norma 2

4 Dalla forma di Hermite alla forma di Bezier

Possiamo passare con dei semplici calcoli dalla forma di Hermite alla forma di Bezier in modo tale da utilizzare l'algoritmo di de Casteljau per la tabulazione. Il patch in forma di Bezier è così definito:

$$\mathbf{X}(s,t) = \mathbf{B}^T(s)\ \mathbf{M}\ \mathbf{B}(t)$$

con

$$B_0^3(t) = (1-t)^3, B_1^3(t) = 3t(1-t)^2 B_2^3(t) = 3t^2(1-t), B_3^3(t) = t^3$$

dove

$$\mathbf{B}(t) = (B_0^3(t), B_1^3(t), B_2^3(t), B_3^3(t))$$

e **M** è una matrice di dimensione 4x4 a tre piani.

Data la matrice di Hermite possiamo ad ogni elemento associare un corrispondente nella matrice nella forma di Bezier. Utilizziamo il Codice 7 per ricavare da un piano della matrice di Hermite il piano della matrice associato in forma di Bezier.

```
function M = bezier_matrix(C)
    M = zeros(4, 4);

    M(1, 1) = C(1, 1);
    M(1, 4) = C(1, 4);
```

```
    M(4, 1) = C(4, 1);
    M(4, 4) = C(4, 4);

    M(1, 2) = C(1, 1) + C(1, 2)/3;
    M(2, 1) = C(1, 1) + C(2, 1)/3;
    M(2, 2) = C(1, 1) + C(1, 2)/3 + C(2, 1)/3 + C(2, 2)/9;

    M(3, 1) = C(4, 1) - C(3, 1)/3;
    M(4, 2) = C(4, 1) + C(4, 2)/3;
    M(3, 2) = C(4, 1) + C(4, 2)/3 - C(3, 1)/3 + C(3, 2)/9;

    M(2, 4) = C(1, 4) + C(2, 4)/3;
    M(1, 3) = C(1, 4) - C(1, 3)/3;
    M(2, 3) = C(1, 4) + C(2, 4)/3 - C(1, 3)/3 - C(2, 3)/9;

    M(3, 4) = C(4, 4) - C(3, 4)/3;
    M(4, 3) = C(4, 4) - C(4, 3)/3;
    M(3, 3) = C(4, 4) - C(4, 3)/3 - C(3, 4)/3 + C(3, 3)/9;
end
```

Codice 7: Funzione per la creazione di un piano della matrice di Bezier

Riprodotto questo procedimento per tutti e tre i piani, come mostrato nel Codice 8, possiamo passare alla definizione del patch in forma di Bezier.

```
function M = full_bezier_matrix(C_x, C_y, C_z)
    M = zeros(4, 4, 3);

    M(:,:, 1) = bezier_matrix(C_x);
    M(:,:, 2) = bezier_matrix(C_y);
    M(:,:, 3) = bezier_matrix(C_z);
end
```

Codice 8: Creazione della matrice di Bezier completa

Nello stesso modo del patch di Hermite definiamo le funzioni di base per scrivere in forma matriciale il patch di Bezier. Visualizziamo nel Codice 9 come sono state create le funzioni del patch.

```
syms Ber_0_t(t) Ber_1_t(t) Ber_2_t(t) Ber_3_t(t) Ber_0_s(s)
    Ber_1_s(s) Ber_2_s(s) Ber_3_s(s)
Ber_0_t(t) = (1 - t).^3;
```

```
Ber_1_t(t) = 3*t*(1 - t).^2;
Ber_2_t(t) = 3*t.^2*(1 - t);
Ber_3_t(t) = t.^3;

Ber_0_s(s) = (1 - s).^3;
Ber_1_s(s) = 3*s*(1 - s).^2;
Ber_2_s(s) = 3*s.^2*(1 - s);
Ber_3_s(s) = s.^3;

Ber_t = [Ber_0_t(t); Ber_1_t(t); Ber_2_t(t); Ber_3_t(t)];
Ber_s = [Ber_0_s(s); Ber_1_s(s); Ber_2_s(s); Ber_3_s(s)];

XB_st = symfun((Ber_s.') * M(:,:,1) * Ber_t, [s,t]);
YB_st = symfun((Ber_s.') * M(:,:,2) * Ber_t, [s,t]);
ZB_st = symfun((Ber_s.') * M(:,:,3) * Ber_t, [s,t]);
```

Codice 9: Patch in forma di Bezier

Su questo nuovo patch possiamo fare un controllo di equivalenza con il patch in forma di Hermite. Eseguendo il Codice 10 questo ci da un risultato affermativo e quindi a meno di una tolleranza infinitesimale e assumendo che lo spazio parametrico sia compreso tra i valori presi in considerazione, i due patch bicubici sono uguali.

```
assume(s_a <= s <= s_b)
assume(t_a <= t <= t_b)
isAlways(abs(X_st - XB_st) < 1.0e-13 )
isAlways(abs(Y_st - YB_st) < 1.0e-13 )
isAlways(abs(Z_st - ZB_st) < 1.0e-13 )
```

Codice 10: Controllo di equivalenza dei due patch

Scriviamo l'algoritmo di de Casteljau in modo da sfruttare la forma di Bezier del patch. Utilizzando l'algoritmo i tempi per la tabulazione si riducono notevolmente. Passiamo dai 5 secondi della forma matriciale con il calcolo simbolico, a meno di un secondo utilizzando de Casteljau. In Figura 5 viene visualizzata la superficie calcolata e i nodi del patch. L'algoritmo di de Casteljau bivariato sul patch viene eseguito tramite il Codice 11 che richiama il Codice 12 di de Casteljau monovariato.

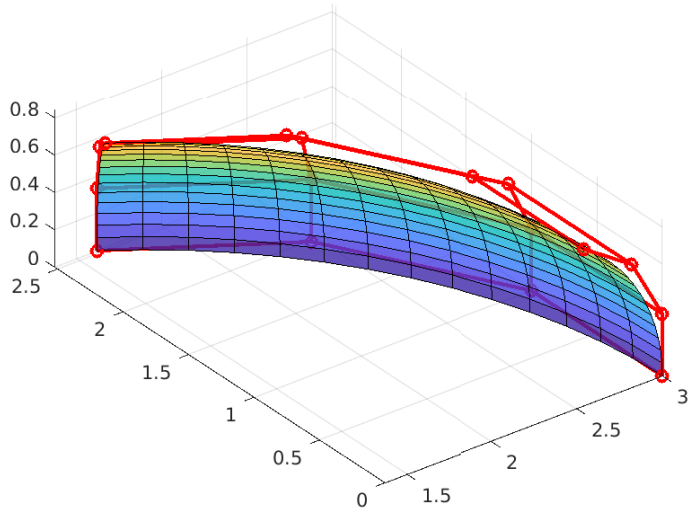

Figura 5: Plot dei nodi e della superficie calcolata con de Casteljau

```
function [s] = deCasteljau2(P, n, u0, v0)
    Q = zeros(1, n, 3);
    for i = 1:n
        Q(:,i,:) = deCasteljau(P(i,:,:), n, v0);
    end
    s = deCasteljau(Q, n, u0);
end
```

Codice 11: Algoritmo di de Casteljau per la superficie

```
function [c] = deCasteljau(P, n, u)
    Q = P;
    for k = 2:n + 1
        for i = 1:n - k + 1
            Q(:,i,:) = (1-u) * Q(:,i,:) + u*Q(:,i+1,:);
        end
    end
    c = [Q(1,1,1), Q(1,1,2), Q(1,1,3)]';
end
```

Codice 12: Algoritmo di de Casteljau a un solo parametro

5 Generalizzazione al caso spline

Utilizziamo l'unione di tanti patch per definire una superficie. Ogni patch è definito in un sottorettangolo del dominio parametrico. Determiniamo la superficie $\mathbf{X} : A \to E^3$ tale che, $\forall i = 0, \ldots, n$ e $\forall j = 0, \ldots, m$ risulti

$$\mathbf{X}(u_i, v_j) = \mathbf{P}_{i,j},\ \mathbf{X}_u(u_i, v_j) = \mathbf{d}_{i,j}^{(1,0)}\ \mathbf{X}_v(u_i, v_j) = \mathbf{d}_{i,j}^{(0,1)}\ \mathbf{X}_{uv}(u_i, v_j) = \mathbf{d}_{i,j}^{(1,1)}.$$

Poniamo inoltre la divisione in patch,

$$\mathbf{X}(u, v) = \mathbf{X}^{(i,j)}(u, v),\ \text{se } u \in [u_i, u_{i+1}],\ \text{e } v \in [v_j, v_{j+1}].$$

Posto

$$s = \frac{u - u_i}{\Delta u_i}, t = \frac{v - v_j}{\Delta v_j} \qquad \text{(parametri locali)}$$

con $\Delta u_i = u_{i+1} - u_i$ e $\Delta v_j = v_{j+1} - v_j$, utilizzando queste due variabili possiamo scrivere $\mathbf{X}^{(i,j)}(s, t) = \mathbf{H}^T(s)\ \mathbf{C}(i, j))\ \mathbf{H}(t)$. Per imporre le 16 condizioni di interpolazione sugli estremi del sotto-rettangolo in cui risulta definita $\mathbf{X}^{(i,j)}$, si pone allora

$$\mathbf{C}^{(i,j)} = \begin{pmatrix} \mathbf{P}_{i,j} & \Delta v_j \mathbf{d}_{i,j}^{(0,1)} & \Delta v_j \mathbf{d}_{i,j+1}^{(0,1)} & \mathbf{P}_{i,j+1} \\ \Delta u_i \mathbf{d}_{i,j}^{(1,0)} & \Delta u_i \Delta v_j \mathbf{d}_{i,j}^{(1,1)} & \Delta u_i \Delta v_j \mathbf{d}_{i,j+1}^{(1,1)} & \Delta u_i \mathbf{d}_{i,j+1}^{(0,1)} \\ \Delta u_i \mathbf{d}_{i+1,j}^{(1,0)} & \Delta u_i \Delta v_j \mathbf{d}_{i+1,j}^{(1,1)} & \Delta u_i \Delta v_j \mathbf{d}_{i+1,j+1}^{(1,1)} & \Delta u_i \mathbf{d}_{i+1,j+1}^{(0,1)} \\ \mathbf{P}_{i+1,j} & \Delta v_j \mathbf{d}_{i+1,j}^{(0,1)} & \Delta v_j \mathbf{d}_{i+1,j+1}^{(0,1)} & \mathbf{P}_{i+1,j+1} \end{pmatrix}$$

Nel Codice 13 viene descritta la funzione MATLAB per definire la matrice della forma di Hermite nel caso spline.

```
function [C_ij] = hermite_matrix_spine(f_x,s_a,s_b,t_a,t_b,du,dv)
    syms t s
    C_ij = zeros(4, 4);

    df_xt = diff(f_x, t);
    df_xs = diff(f_x, s);
    df_xst = diff(f_x, s, t);

    C_ij(1,1) = f_x(s_a, t_a);
```

```
    C_ij(1,2) = dv*df_xt(s_a, t_a);
    C_ij(2,1) = du*df_xs(s_a, t_a);
    C_ij(2,2) = du*dv*df_xst(s_a, t_a);

    C_ij(1,4) = f_x(s_a, t_b);
    C_ij(1,3) = dv*df_xt(s_a, t_b);
    C_ij(2,4) = du*df_xs(s_a, t_b);
    C_ij(2,3) = du*dv*df_xst(s_a, t_b);

    C_ij(4,1) = f_x(s_b, t_a);
    C_ij(4,2) = dv*df_xt(s_b, t_a);
    C_ij(3,1) = du*df_xs(s_b, t_a);
    C_ij(3,2) = du*dv*df_xst(s_b, t_a);

    C_ij(4,4) = f_x(s_b, t_b);
    C_ij(4,3) = dv*df_xt(s_b, t_b);
    C_ij(3,4) = du*df_xs(s_b, t_b);
    C_ij(3,3) = du*dv*df_xst(s_b, t_b);
end
```

Codice 13: Creazione matrice della forma di Hermite nel caso spline

Dividiamo il dominio parametrico di u in tre parti, $[0,\ 0.5,\ 0.9,\ 1]$. Nella Figura 6 vediamo come l'unione dei patch formi la superficie di partenza. Notiamo che il patch (in blu) definito nel dominio $u_1 = [0,\ 0.5]$ è quello più grossolano mentre il patch (in giallo) definito nel dominio $u_3 = [0.9,\ 1]$ è quello più fitto.

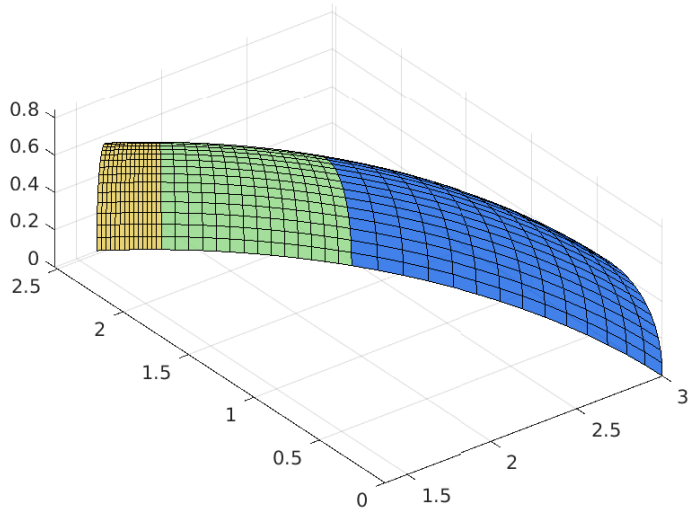

Figura 6: Patch spline

Usando tutto il dominio parametrico della superficie e dividendolo in 16 parti (dominio di u e di v diviso in 4 parti), vediamo in Figura 7 tutta la superficie definita dall'unione di 16 patch in forma di Hermite.

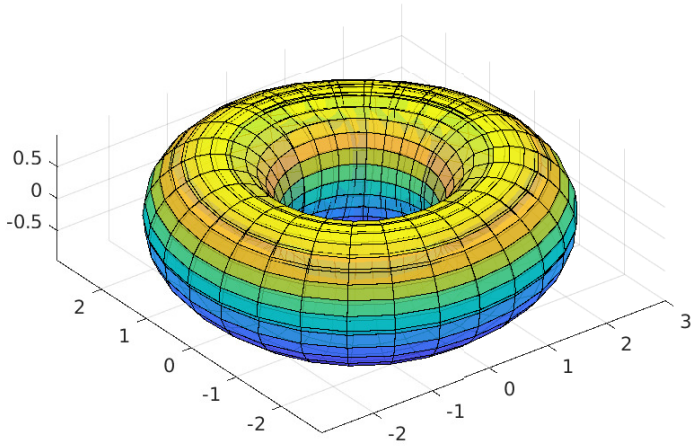

Figura 7: Superficie intera

6 Errore di interpolazione nel caso spline

Prendendo il patch nel dominio parametrico $[0,1]^2$ nel caso base ($n = 1$ e $m = 1$) abbiamo ovviamente gli stessi valori di errore che abbiamo trovato nella Sessione 3:

- 0.0717 per le ascisse;
- 0.0396 per le ordinate;
- 0.0114 per la quota.

Dividendo il dominio in due parti su u ($n = 2$ e $m = 1$) abbiamo i seguenti valori di errore:

- 0.0201 per le ascisse;
- 0.0114 per le ordinate;
- 0.0114 per la quota.

Dividendo il dominio anche su v ($n = 2$ e $m = 2$) abbiamo:

- 0.0113 per le ascisse;

- 0.0136 per le ordinate;

- 6.9577e-04 per la quota.

Dividendo ulteriormente il dominio sia su u che su v ($n = 3$ e $m = 3$) abbiamo i valori di errore:

- 0.0054 per le ascisse;

- 0.0075 per le ordinate;

- 1.3694e-04 per la quota.

Vediamo come l'errore sia diminuito di almeno un ordine di grandezza utilizzando al posto di un solo patch, 9 sottopatch definiti nei sottointervalli del dominio parametrico. Nel Codice 14 troviamo lo script MATLAB per il calcolo nell'errore.

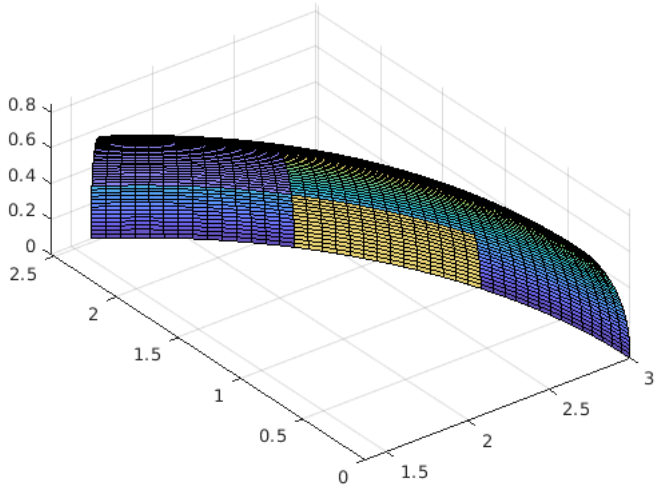

Figura 8: Superficie unione di 9 patch

```
n = 3; m = 3;
indexes_i = linspace(1, n, n);
indexes_j = linspace(1, m, m);
prec = 15;
u_i = linspace(0, 1, n + 1);v_j = linspace(0, 1, m + 1);
du_i = zeros(1, n);dv_j = zeros(1, m);
for k = indexes_i
   du_i(k) = u_i(k + 1) - u_i(k);
end
for k = indexes_j
   dv_j(k) = v_j(k + 1) - v_j(k);
end
X_norms = [];Y_norms = [];Z_norms = [];
colors = [[107, 98, 229];[141, 214, 130];[227, 201, 83]]/255;
for i = indexes_i
   for j = indexes_j
      C = full_hermite_matrix_spine(f_x, f_y, f_z, u_i(i), u_i(i
         + 1), v_j(j), v_j(j + 1), du_i(i), dv_j(j));
      P = full_bezier_matrix(C(:,:,1),C(:,:,2),C(:,:,3));
      [X_hermite, Y_hermite, Z_hermite] = plotDeCasteljau(P,
         prec);
      surf(X_hermite, Y_hermite, Z_hermite, 'FaceAlpha', .8,
         'FaceColor', colors(mod(i,3)+1,:)); axis equal;hold on;
      u = linspace(u_i(i), u_i(i + 1), prec);
      v = linspace(v_j(j), v_j(j + 1), prec);
      [uu, vv] = meshgrid(u, v);
      X = (2 + cos(vv)).*cos(uu);
      Y = (2 + cos(vv)).*sin(uu);
      Z = sin(vv);
      X_norms(end + 1) = norm(abs(X_hermite - X'));
      Y_norms(end + 1) = norm(abs(Y_hermite - Y'));
      Z_norms(end + 1) = norm(abs(Z_hermite - Z'));
   end
end
mean(X_norms)
mean(Y_norms)
mean(Z_norms)
```

Codice 14: Calcolo dell'errore interpolazione caso spline