

Laboratorio di Calcolo Scientifico  
a.a. 2002/03  
SOLUZIONE HOMEWORK #2

Udine, 23 giugno 2003

Prof.ssa R. Vermiglio, Dott. Ing. D. Breda  
Dipartimento di Matematica e Informatica,  
Università degli Studi di Udine  
vermiglio@dimi.uniud.it  
dbreda@dimi.uniud.it  
<http://www.dimi.uniud.it/~rossana>  
<http://www.dimi.uniud.it/~dbreda>

*NOTA: quando sono richieste implementazioni MATLAB che danno come output grafici, questi devono essere stampati e consegnati assieme all'homework (che può essere scritto a mano), includendo anche il testo dell'm-file e dei risultati numerici.*

1. Si consideri il problema dell'approssimazione di una quantità  $F$  mediante una formula  $F(h)$  tale che l'errore risulti

$$E(h) = F - F(h) = ch^2 + \mathcal{O}(h^4).$$

In particolare si vogliono approssimare le quantità

$$f'(0) = 1 \quad \text{e} \quad \int_0^{2\pi} f(x) dx = \frac{(3 - 10\pi)e^{-2\pi} - 3}{25}$$

dove

$$f(x) = xe^{-x} \cos 2x.$$

- i. Si scriva una *function* MATLAB con sintassi

`[Dh,EDh]=difcen(h)`

che implementi il metodo delle differenze centrate per l'approssimazione della derivata prima di  $f$  in 0. Sia  $h$  il passo utilizzato,  $Dh$  la derivata approssimata ed  $EDh$  l'errore assoluto commesso rispetto al valore esatto.

- ii. Si scriva una *function* MATLAB con sintassi

`[Th, ETh]=trapezi(h)`

che implementi la formula dei trapezi per approssimare l'integrale di  $f$  tra 0 e  $2\pi$ . Sia  $h$  il passo utilizzato,  $Th$  l'integrale approssimato ed  $ETh$  l'errore assoluto commesso rispetto al valore esatto.

- iii. Si analizzi l'andamento dell'errore assoluto in funzione del passo  $h$ , confermando con i risultati numerici che entrambe le *function* precedenti sono formule di approssimazione del tipo  $F(h)$  descritto sopra. In particolare si rappresenti su scala bilogaritmica l'errore assoluto in funzione del passo per un vettore di valori di  $h = (h_0, h_0/2, h_0/4, h_0/8, \dots)^T$  (con  $h_0$  opportunamente scelto) fino a raggiungere una tolleranza prefissata.

- iv. Si scriva una terza *function* con sintassi

`[Fh, EF]=richard(F, TOL, h0)`

che implementi l'estrapolazione di Richardson.  $h_0$  è un opportuno passo iniziale,  $TOL$  la precisione richiesta ed  $F$  la formula di approssimazione  $F(h)$ . L'algoritmo riportato in [1, §4.3] è il seguente:

$$\begin{aligned} F_{m,0} &= F(h_m), \quad m = 0, 1, \dots \\ F_{m,n+1} &= \frac{4^{n+1}F_{m,n} - F_{m-1,n}}{4^{n+1} - 1}, \quad n = 0, 1, \dots, m-1 \end{aligned}$$

con  $h_{m+1} = \frac{h_m}{2}$ ,  $m = 0, 1, \dots$  e  $h_0$  opportuno. *richard* costruisce la matrice delle approssimazioni

$$\begin{pmatrix} F_{0,0} & & & & \\ F_{1,0} & F_{1,1} & & & \\ F_{2,0} & F_{2,1} & F_{2,2} & & \\ F_{3,0} & F_{3,1} & F_{3,2} & F_{3,3} & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

a cui viene aggiunta una riga (cioè si dimezza il passo  $h$ ) ogni volta che l'errore assoluto stimato dall'estrapolazione

$$EF_m = |F_{m,m} - F_{m-1,m-1}|, \quad m = 0, 1, \dots$$

risulta superiore alla tolleranza  $TOL$  richiesta in input. Quando l'errore risulta inferiore a  $TOL$ , l'esecuzione termina ed il valore approssimato di  $F$  sarà  $Fh = F_{m,m}$ . *richard* deve restituire in output anche il vettore degli errori  $EF = (EF_1, EF_2, \dots, EF_m)^T$  e deve provvedere alla rappresentazione grafica (su scala bilogaritmica) dell'andamento dell'errore in funzione del passo.

- v. Si esegua *richard* per le formule implementate da *difcen* e da *trapezi*, scegliendo opportunamente il passo iniziale. Si confrontino i corrispondenti andamenti degli errori in funzione del passo ottenuti al punto *iii* con quelli ottenuti al punto *v*. Si commentino i risultati ottenuti.

- sol.1 i. L'approssimazione della derivata prima di una funzione  $f$  di classe almeno  $\mathcal{C}^2$  con il metodo delle differenze centrate rientra nel caso della formula  $F(h)$  riportata nel testo. L'errore analitico commesso risulta infatti

$$f'(x) - \frac{f(x+h) - f(x-h)}{2h} = \frac{f''(x)h^2}{6} + \mathcal{O}(h^4).$$

La seguente *function* MATLAB implementa l'approssimazione di  $f'(0)$  per un dato passo  $h$ :

```
function [Dh,EDh]=difcen(h)

f=inline('x.*exp(-x).*cos(2*x)'); % funzione f
x=0;                               % punto x=0
D=1;                               % derivata esatta
Dh=(f(x+h)-f(x-h))/(2*h);         % derivata appros.
EDh=abs(D-Dh);                    % errore assoluto
```

- ii. L'approssimazione dell'integrale di una funzione  $f$  di classe almeno  $\mathcal{C}^2$  con la formula dei trapezi composta rientra nel caso della formula  $F(h)$  riportata nel testo. L'errore analitico commesso risulta infatti

$$\begin{aligned} \int_a^b f(x)dx &= \frac{h}{2} \left( f(a) + 2 \sum_{i=1}^{n-1} f(a+ih) + f(b) \right) = \\ &= -\frac{f''(\xi)(b-a)h^2}{12} + \mathcal{O}(h^4) \end{aligned}$$

con  $\xi \in (a, b)$  e  $n = \frac{b-a}{h}$ . La seguente *function* MATLAB implementa l'approssimazione dell'integrale di  $f$  in  $[0, 2\pi]$  per un dato passo  $h$ :

```
function [Th,ETH]=trapezi(h)

f=inline('x.*exp(-x).*cos(2*x)'); % funzione f
a=0;b=2*pi;                       % estremi integrazione
T=((3-10*pi)*exp(-2*pi)-3)/25;    % integrale esatto
ff=f(a+(1:(b-a)/h-1)*h);          % valori intermedi f
Th=(h/2)*(f(a)+2*sum(ff)+f(b));    % integrale appros.
ETH=abs(T-Th);                    % errore assoluto
```

- iii. Per analizzare l'andamento dell'errore assoluto si eseguono le *function* precedenti per diversi valori del passo  $h = (h_0, h_0/2, h_0/4, h_0/8, \dots)^T$

con  $h_0 = 1$  per *difcen* e  $h_0 = 2\pi$  per *trapezi*. Il tutto può essere realizzato con le seguenti *function*

```
function EDh=test_difcen(h)

for i=1:length(h)
    [Dh(i),EDh(i)]=difcen(h(i));
end
loglog(h,EDh,'o')
xlabel('h')
ylabel('EDh')
grid on

function ETh=test_trapezi(h)

for i=1:length(h)
    [Th(i),ETh(i)]=trapezi(h(i));
end
loglog(h,ETh,'*')
xlabel('h')
ylabel('ETh')
grid on
```

Eseguendo

```
>>hD=1./2.^(0:15);
>>EDh=test_difcen(hD);
>>hT=2*pi./2.^(0:15);
>>ETh=test_trapezi(hT);
```

si ottengono i risultati

```
>>[hD',EDh']
ans=
    1.000000000000000e+000    1.642148124715520e+000
    5.000000000000000e-001    3.907410908422058e-001
    2.500000000000000e-001    9.484984944039332e-002
    1.250000000000000e-001    2.350808857298370e-002
    6.250000000000000e-002    5.863815797431560e-003
    3.125000000000000e-002    1.465121753680965e-003
    1.562500000000000e-002    3.662283198233585e-004
    7.812500000000000e-003    9.155382088099628e-005
    3.906250000000000e-003    2.288825150231766e-005
    1.953125000000000e-003    5.722050142820123e-006
    9.765625000000000e-004    1.430511739952678e-006
    4.882812500000000e-004    3.576278851946668e-007
    2.441406250000000e-004    8.940696827330896e-008
    1.220703125000000e-004    2.235174190179379e-008
    6.103515625000000e-005    5.587935447692871e-009
```

```

3.051757812500000e-005    1.396983861923218e-009
>>[hT',ETh']
ans=
6.283185307179586e+000    1.589844466262634e-001
3.141592653589793e+000    5.670578035070584e-001
1.570796326794897e+000    2.348282377978085e-001
7.853981633974483e-001    5.635281658942007e-002
3.926990816987241e-001    1.327424390518267e-002
1.963495408493621e-001    3.263203374110596e-003
9.817477042468104e-002    8.122894415675125e-004
4.908738521234052e-002    2.028521588293836e-004
2.454369260617026e-002    5.069926581814654e-005
1.227184630308513e-002    1.267395541093563e-005
6.135923151542565e-003    3.168435034853201e-006
3.067961575771282e-003    7.921053952614221e-007
1.533980787885641e-003    1.980261380984949e-007
7.669903939428206e-004    4.950652073010264e-008
3.834951969714103e-004    1.237662995701161e-008
1.917475984857052e-004    3.094156639238399e-009

```

rappresentati su scala bilogarithmica nelle figure 1 e 2. Si ha dunque conferma sperimentale che entrambi i metodi di approssimazione hanno ordine di convergenza 2 (=pendenza della retta).

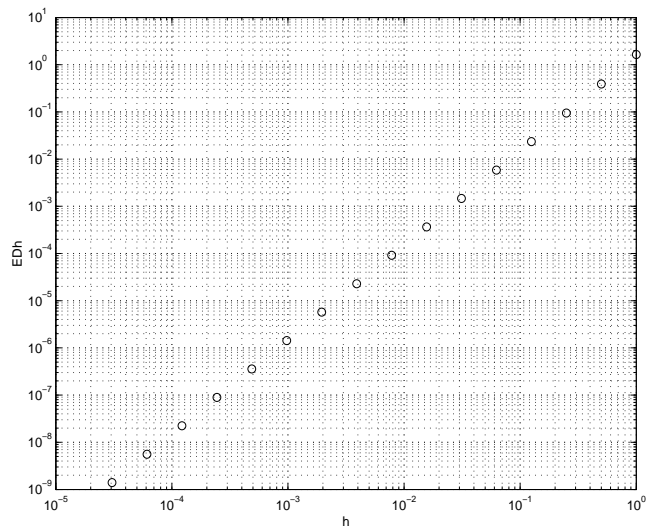


Figura 1: Errore nell'approssimazione di  $f'(0)$  con *difcen*.

Un'alternativa interessante (e generalmente valida quando si implementano funzioni di un parametro che vanno testate per diversi valori

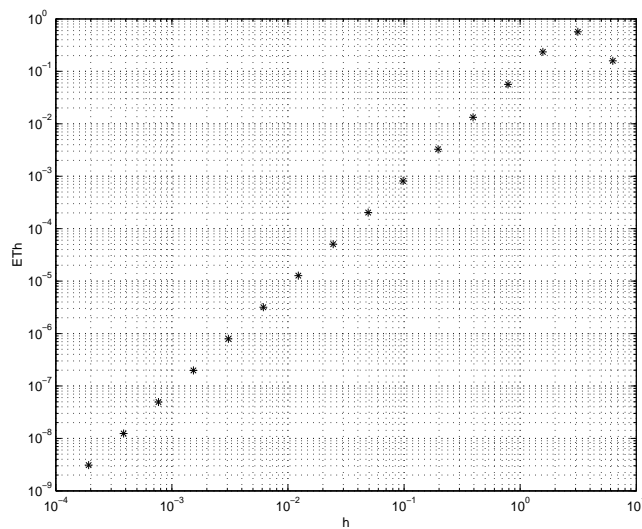


Figura 2: Errore nell'approssimazione di  $\int_0^{2\pi} f(x)dx$  con *trapezi*.

dello stesso parametro) è quella di modificare *difcen* e *trapezi* in modo da poter processare direttamente un intero vettore di passi  $h$  e di provvedere anche alla creazione del grafico:

```
function [Dh,EDh]=difcen(h,g)

% Questa versione funziona anche per h vettore

f=inline('x.*exp(-x).*cos(2*x)'); % funzione f
x=0;                               % punto x=0
D=1;                               % derivata esatta
Dh=(f(x+h)-f(x-h))./(2*h);        % derivata appross.
EDh=abs(D-Dh);                    % errore assoluto
% Il parametro g serve all'utente per richiedere (g=1)
% o meno (g diverso da 1) il grafico
if g==1
    loglog(h,EDh,'o')
    grid on
    xlabel('h')
    ylabel('EDh')
end

function [Th,ETH]=trapezi(h,g)

% Questa versione funziona anche per h vettore
```

```

f=inline('x.*exp(-x).*cos(2*x)'); % funzione f
a=0;b=2*pi; % estremi integ.
T=((3-10*pi)*exp(-2*pi)-3)/25; % integrale esatto
for i=1:length(h)
    ff(1:length(a+(1:(b-a)/h(i)-1)*h(i)),i)=...
    f(a+(1:(b-a)/h(i)-1)*h(i)); % valori inter. f
end
Th=(h/2).*(f(a)+2*sum(ff,1)+f(b)); % integrale appros.
ETh=abs(T-Th); % errore assoluto
% Il parametro g serve all'utente per richiedere (g=1)
% o meno (g diverso da 1) il grafico
if g==1
    loglog(h,ETh,'*')
    grid on
    xlabel('h')
    ylabel('ETh')
end

```

- iv. Una possibile implementazione dell'estrapolazione di Richardson per un generico metodo di approssimazione  $F(h)$  di ordine 2 è la seguente:

```

function [Fh,EF]=richard(F,TOL,h0,g)

% inizializzazione parametri
m=1;
n=1;
h(m)=h0;
EF(m)=1;
FF(m,n)=F(h(m));
% matrice delle approssimazioni
while EF(m)>TOL
    m=m+1;
    h(m)=h(m-1)/2;
    for k=1:n+1
        if k==1
            FF(m,k)=F(h(m));
        else
            FF(m,k)=(4^(k-1)*FF(m,k-1)-...
            FF(m-1,k-1))/(4^(k-1)-1);
        end
    end
    EF(m)=abs(FF(m,n+1)-FF(m-1,n));
    n=n+1;
end
Fh=FF(m,n);
[h',EF'] % per vedere i risultati
% plot dei risultati

```

```

if g==1
    loglog(h,EF,'o')
    grid on
    xlabel('h')
    ylabel('EF')
end

```

v. *richard* viene testata con le seguenti istruzioni:

```

>>F=inline('difcen(h)')
F=
    Inline function:
    F(h) = difcen(h)
>>[Fh,EF]=richard(F,1e-15,1,1);
ans=
    1.000000000000000e+000    1.000000000000000e+000
    5.000000000000000e-001    1.668542711831086e+000
    2.500000000000000e-001    2.412162427237852e-002
    1.250000000000000e-001    2.269801168789609e-003
    6.250000000000000e-002    3.161582119881601e-006
    3.125000000000000e-002    9.229950137523701e-011
    1.562500000000000e-002    2.198241588757810e-014
    7.812500000000000e-003    0
>>F=inline('trapezi(h)')
F=
    Inline function:
    F(h) = trapezi(h)
>>[Fh,EF]=richard(F,1e-15,2*pi,1);
ans=
    6.283185307179586e+000    1.000000000000000e+000
    3.141592653589793e+000    5.440978091743932e-001
    1.570796326794897e+000    1.285552896746760e+000
    7.853981633974483e-001    6.291238634850650e-001
    3.926990816987241e-001    4.645576526372360e-002
    1.963495408493621e-001    2.077531535587679e-004
    9.817477042468104e-002    1.031501676386493e-005
    4.908738521234052e-002    1.914182079509885e-008
    2.454369260617026e-002    3.377131907456032e-012
    1.227184630308513e-002    4.996003610813204e-016

```

I risultati sono rappresentati su scala bilogarithmica nelle figure 3 e 4. Con l'extrapolazione di Richardson l'efficienza è notevolmente migliorata per entrambi i metodi. Con un passo  $h \simeq 0.01$  si ottiene infatti un errore assoluto dell'ordine della precisione macchina, mentre senza extrapolazione si raggiunge una tolleranza di appena  $10^{-4}$  per *difcen* e di  $10^{-5}$  per *trapezi*.

Il confronto con e senza extrapolazione può essere direttamente imple-



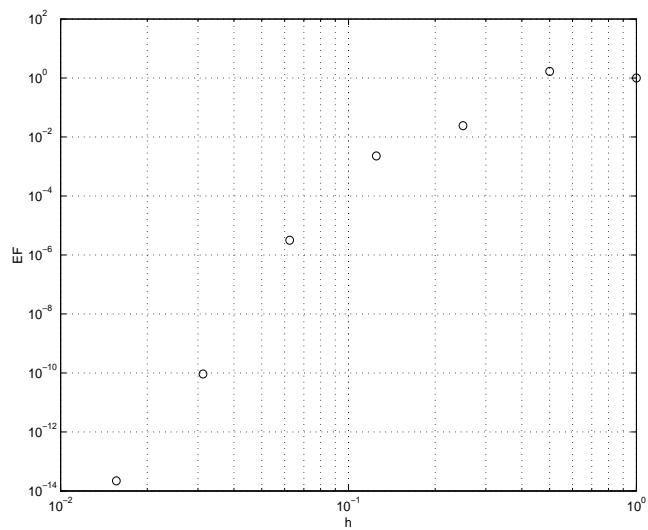


Figura 3: Errore nell'approssimazione di  $f'(0)$  con *richard* per *difcen*.

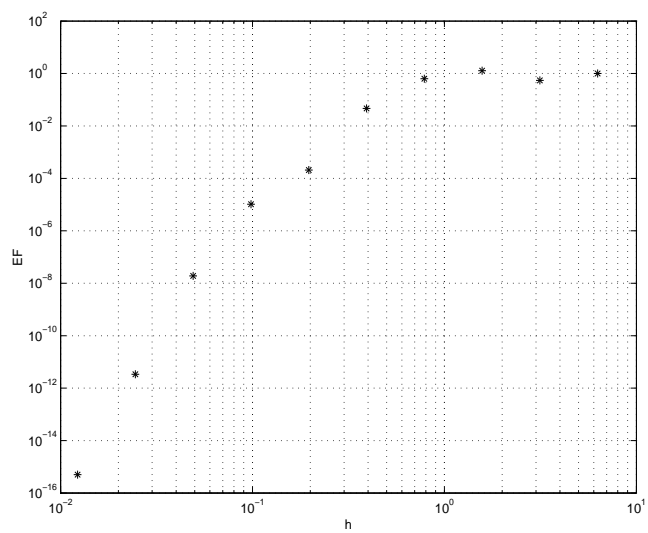


Figura 4: Errore nell'approssimazione di  $\int_0^{2\pi} f(x)dx$  con *richard* per *trapezi*.

mentato in *richard* aggiungendo il calcolo dell'errore assoluto lungo la prima colonna della matrice di approssimazione:

```
EF1(m)=abs(FF(m,1)-FF(m-1,1));
```

Questa corrisponde infatti alla formula di approssimazione senza l'estrapolazione. Modificando poi l'istruzione grafica con

```
loglog(h,EF,'o',h,EF1,'*')
```

si ottengono le figure 5 e 6 che mettono direttamente in risalto i vantaggi dell'estrapolazione.

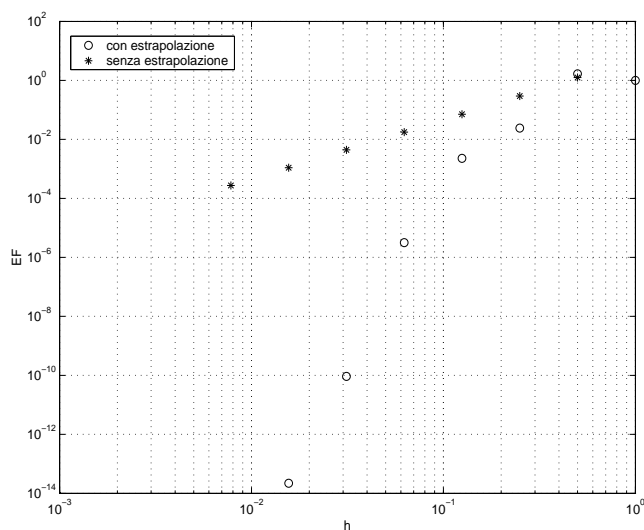


Figura 5: Errore nell'approssimazione di  $f'(0)$  con e senza *richard* per *difcen*.

2. Si scriva una *function* MATLAB per interpolare i punti del piano  $P_i = (x_i, y_i)$ ,  $i = 0, 1, \dots, n$ , con le spline parametriche, prendendo come parametro la lunghezza cumulativa  $t$  data da

$$\begin{aligned} t_0 &= 0 \\ t_i &= \sum_{k=1}^i l_k, \quad i = 1, \dots, n \end{aligned}$$

dove  $l_i$  è la lunghezza di ciascun segmento  $\overline{P_i P_{i-1}}$ ,  $i = 1, \dots, n$ . La sintassi della *function* dev'essere

```
[xx,yy]=spline_par(x,y)
```

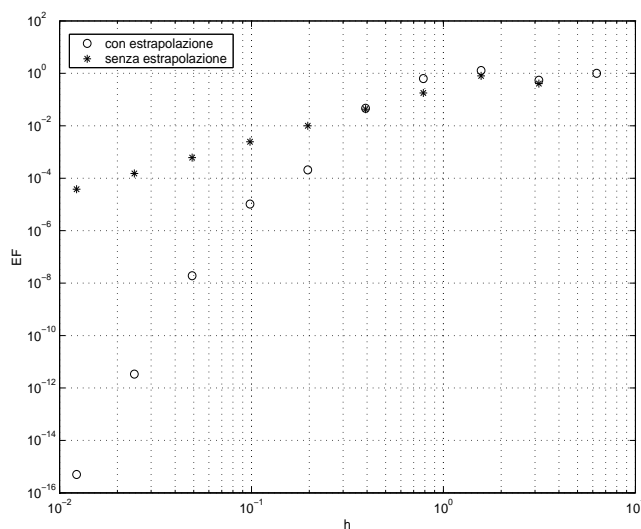


Figura 6: Errore nell'approssimazione di  $\int_0^{2\pi} f(x)dx$  con e senza *richard* per trapezi.

dove i vettori  $x$  e  $y$  rappresentano i punti da interpolare, mentre i vettori  $xx$  e  $yy$  rappresentano i valori assunti dalla spline parametrica in corrispondenza di un opportuno campionamento del parametro  $t$ .

Si esegua la *function* con i dati

$x_i$	8.125	8.4	9	9.445	9.6	9.959	10.166	10.2
$y_i$	0.0774	0.099	0.28	0.6	0.708	1.3	1.8	2.177

confrontando graficamente la spline parametrica con la spline ottenuta con il comando *spline* di MATLAB. Si verifichi inoltre la proprietà di invarianza geometrica delle spline parametriche (i.e. la curva interpolante ha la medesima rappresentazione indipendentemente dal sistema di coordinate usato), ruotando i punti di  $\theta = 36^\circ$  in senso antiorario. Ciò si può ottenere applicando ai vettori  $(x_i, y_i)^T$  la matrice di rotazione (di Givens, vedi [1, Appendice A.1.7])

$$R(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}$$

per cui i nuovi punti sono individuati dai vettori

$$\begin{pmatrix} x'_i \\ y'_i \end{pmatrix} = R(\theta) \begin{pmatrix} x_i \\ y_i \end{pmatrix}.$$

Si commentino i risultati ottenuti, in particolare si commenti il diverso comportamento tra spline (con opportuno codice MATLAB) e spline parametriche (con *spline-par*) quando cambia il sistema di riferimento.

(Oss.: ogni punto  $(x, y)$  sul piano cartesiano identifica un vettore  $(x, y)^T \in \mathbb{R}^2$ . La rotazione del vettore di un angolo  $\theta$  (in senso antiorario) comporta un cambio di coordinate  $(x, y) \mapsto (x', y')$ . Le nuove coordinate sono individuate dal vettore ruotato  $(x', y')^T$  come spiegato nel testo dell'esercizio e corrisponde alla rotazione del sistema di riferimento cartesiano di un angolo  $\theta$  in senso orario.

Sugg.: per verificare l'invarianza geometrica si applichi il comando

```
>>axis equal
```

al grafico corrente: tale istruzione permette di utilizzare la stessa scala (incrementi uguali) per entrambi gli assi.)

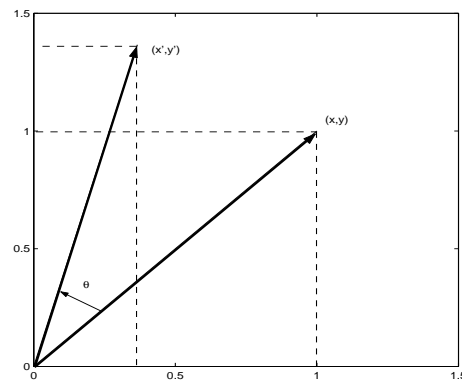


Figura 7: Rotazione antioraria del vettore  $(x, y)^T$  di un angolo  $\theta$ .

sol.2 Una possibile implementazione delle spline parametriche è la seguente:

```
function [xx,yy]=spline_par(x,y)

% crea il vettore di elementi x(i)-x(i-1)
dx=diff(x);
% crea il vettore di elementi y(i)-y(i-1)
dy=diff(y);
% crea la lunghezza cumulativa
t=[0;cumsum(sqrt(dx.^2+dy.^2))];
% campionamento della lunghezza cumulativa
tt=[0:t(length(t))/100:t(length(t))];
% calcola la spline paramtrica in tt
xx=spline(t,x,tt);
yy=spline(t,y,tt);
```

Aggiungendo le istruzioni

```

plot(x,y,'o',xx,yy)
hold on, grid on
xs=min(x):(max(x)-min(x))/100:max(x);
plot(xs,spline(x,y,xs),'--')

```

ed eseguendo il tutto con i dati assegnati si ottiene quanto rappresentato in figura 7. Si osservi come il comportamento oscillatorio della spline classica

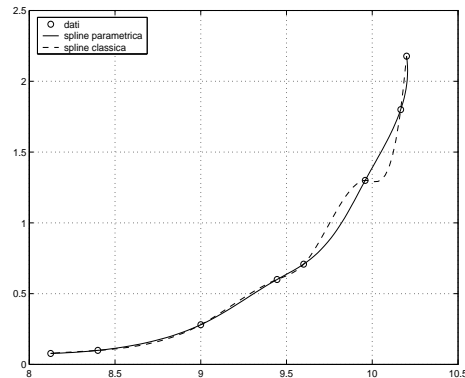


Figura 8: Spline parametrica (continua) e spline classica (tratteggiata).

sia smorzato con l'utilizzo della spline parametrica.

Per verificare l'invarianza geometrica delle spline parametriche si è innanzitutto costruita la seguente *function* per la rotazione dei punti assegnati di un angolo  $\theta$  in senso antiorario:

```

function [xr,yr]=rot(x,y,teta)

T=inline('[cos(teta),-sin(teta);sin(teta),cos(teta)]');
for i=1:length(x)
    r(:,i)=T(teta)*[x(i);y(i)];
end
xr=r(1,:);
yr=r(2,:);

```

*rot* si utilizza poi all'interno della seguente *function* che confronta direttamente i due tipi di spline quando i punti vengono ruotati:

```

function invspline(x,y,teta)

[xx,yy]=spline_par(x,y);           % spline parametrica
[xr,yr]=rot(x,y,teta);             % rotazione punti
[xxr,yxr]=spline_par(xr,yr);       % spline parametrica ruotata
plot(x,y,'o',xx,yy)

```

```

hold on, grid on
xs=min(x):(max(x)-min(x))/100:max(x);
plot(xs,spline(x,y,xs),'--')      % spline classica
plot(xr,yr,'o',xxr,yyr)
xrs=min(xr):(max(xr)-min(xr))/100:max(xr);
plot(xrs,spline(xr,yr,xrs),'--') % spline classica ruotata
axis equal

```

Eseguendo *invspline* con i dati assegnati e con  $\theta = 36^\circ$  si ottiene quanto rappresentato in figura 8. Si osservi come la spline parametrica conservi

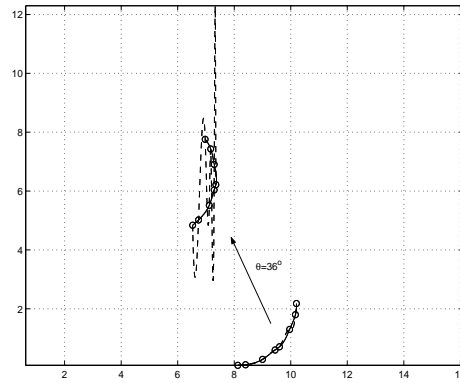


Figura 9: Spline parametrica (continua) e spline classica (tratteggiata) ruotate di  $36^\circ$  in senso antiorario.

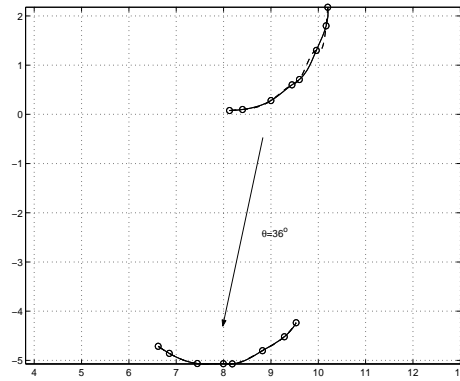


Figura 10: Spline parametrica (continua) e spline classica (tratteggiata) ruotate di  $36^\circ$  in senso orario.

la rappresentazione indipendentemente dal sistema di riferimento (invarianza geometrica), mentre la spline classica cambia rappresentazione variando il comportamento oscillatorio, in questo specifico caso peggiorando

notevolmente. Ripetendo il confronto con una rotazione oraria di  $36^\circ$  si ottiene quanto rappresentato in figura 9. In questo secondo caso la spline classica diminuisce il comportamento oscillatorio. La spline parametrica conserva, ovviamente, la sua rappresentazione.

## Riferimenti bibliografici

- [1] V. Comincioli, *Analisi Numerica: Metodi, Modelli e Applicazioni*, McGraw-Hill (1990), Milano.