Francesca Mazzia
Dipartimento Interuniversitario di Matematica
Università di Bari

MATLAB: analisi degli errori

Come esempio per l'analisi degli errori di arrotondamento scriviamo il seguente file di tipo script dal nome `scriptrad.m` . I commenti nel file vengono preceduti dal %

```
% Script file
% esempio errori di arrotodamento
% eseguiamo per n volte la radice quadrata di x e
% per n volte il quadrato del risultato. Prendiamo x=100

n = input(' n = ')
% l'istruzione input visualizza la stringa
% e legge il dato di input da tastiera.

xt = 100;
x  = 100;

for i=1:n
  x = sqrt(x);
end


for i=1:n
```

```
  x = x.^2;
end


%visualizziamo l'errore relativo
err = abs(x-xt)/abs(xt)
```

Adesso eseguiamo il file con diversi valori di n con la seguente istruzione

```
>> scriptrad
 n = 10

n =

    10


err =

  6.3665e-014


>> scriptrad
 n = 20

n =

    20


err =

  1.1555e-010

>> scriptrad
 n = 30
```

```
n =

    30


err =

  2.2990e-007

>> scriptrad
 n = 40

n =

    40


err =

  1.8975e-004

>> scriptrad
 n = 60

n =

    60


err =

  9.9000e-001

>> xt

xt =

   100
```

```
>> x

x =

     1
```

Un'altro esempio è il calcolo delle seguenti funzioni equivalenti

$$x * (\sqrt{x^2+1} - x)$$
$$x * \sqrt{x^2+1} - x^2$$
$$x/(\sqrt{x^2+1} + x)$$

con diversi valori di x.

Scriviamo il seguente file di tipo script

```
%
%risultati diversi calcolando funzioni equivalenti
%

x=logspace(0,9,500);

r1=x.*(sqrt(x.^2+1)-x);

r2=x.*sqrt(x.^2+1)-x.^2;

r3=x./(sqrt(x.^2+1)+x);

semilogx(x,r1,'r.',x,r2,'b.',x,r3,'g-');
```

La funzione `x=logspace(d1,d2,N)` genera un vettore x con N elementi con spaziatura logaritmica da 10^{d1} a 10^{d2}.
L'analoga funzione `x=linspace(d1,d2,N)` genera un vettore x con N elementi con spaziatura lineare da d1 a d2.
L'istruzione `semilogy` genera grafici in scala logaritmica. L'istruzione `plot` genera grafici in scala lineare.

Il punto prima di un operatore aritmetico permette di eseguire l'operazione sui vettori elemento per elemento.

Eseguiamo il file

```
>> scriptlimitefx
```

Calcolo del valore di un polinomio.

$$p(x) = a_0 x^N + a_1 x^{N-1} + \ldots + a_N$$

Come possiamo valutarlo al variare di x?

Un algoritmo standard è:

```
px = a(N)
for j=N-1:-1:0
   px = px + a(j) * x^(N-j)
end
```

Contiamo le operazioni aritmetiche:

addizioni : N

moltiplicazioni : $1 + 2 + 3 + \cdots N = \frac{N(N+1)}{2}$

Ogni termine $a_j x^{N-j}$ è stato calcolato indipendentemente dagli altri termini.

Possiamo modificare l'algoritmo calcolando ricorsivamente $x_j = x * x^{j-1}$.

L'algoritmo diventa:

```
px = a(N) + a(N-1)*x
potenza =  x
for j=N-2:-1:0
   potenza =  x * potenza
   px = px + a(j) * potenza
end
```

Le operazioni aritmetiche sono:

addizioni: N

moltiplicazioni : $N + N - 1 = 2N - 1$

Il costo è molto inferiore rispetto al primo algoritmo. Esempio: N=20

primo algoritmo: 210 moltiplicazioni

secondo algoritmo: 39 moltiplicazioni

Un algoritmo ancora più efficiente è la regola di Ruffini-Horner, che esegue le moltiplicazioni in modo innestato

ESEMPI:
$N = 2 : p(x) = a_2 + x(a_1 + a_0x)$
$N = 3 : p(x) = a_3 + x(a_2 + x(a_1 + xa_0))$
$N = 4 : p(x) = a_4 + x(a_3 + x(a_2 + x(a_1 + xa_0)))$

Il numero di operazioni è, rispettivamente, 2, 3 e 4 moltiplicazioni. Il secondo algoritmo ne richiedeva 3,5 e 7.

In generale:
$p(x) = a_N + x(a_{N-1} + \cdots + x(a_1 + a_0x)) \cdots)$

L'algoritmo:

```
px = a(0)
for j=1:N
  px = a(j) + px*x
end
```

Le operazioni aritmetiche sono:
addizioni: N
moltiplicazioni : N

L'analisi degli algoritmi dal punto di vista del costo computazionale va sotto il nome di *COMPLESSITÀ DI CALCOLO*.

L'algoritmo di Horner non solo è meno costoso ma anche più stabile.

Scriviamo tre funzioni Matlab per eseguire il calcolo del valore di un polinomio in un punto con i tre algoritmi. La prima funzione la chiamiamo `pol1,m`:

```
function py=pol1(p,x)
% Calcolo del valore del polinomio:
%
% p(1)x^(n-1) + p(2) x^(n-2) + .... + p(n-1) x + p(n)
%
```

```
% nel punti contenuti nel vettore x
%
n=length(p);
py = p(n);
for i=n-1:-1:1
   py = py + p(i)*x.^(n-i);
end
```

La seconda funzione la chiamiamo `pol2.m`:

```
function py=pol2(p,x)
% Calcolo del valore del polinomio:
%
% p(1)x^(n-1) + p(2) x^(n-2) + .... + p(n-1) x + p(n)
%
% nel punti contenuti nel vettore x
%
n=length(p);
xn = x;
py = p(n);
for i=n-1:-1:1
   py = py + p(i)*xn;
   xn = xn.*x;
end
```

La terza funzione la chiamiamo `horner.m`:

```
function y=horner(p,x)
% Calcolo del valore del polinomio:
%
% p(1)x^(n-1) + p(2) x^(n-2) + .... + p(n-1) x + p(n)
%
% nel punti contenuti nel vettore x
%
n=length(p);
y=p(1);
for i=2:n
   y=x.*y+p(i);
end
```

Scriviamo anche il seguente file di tipo script dal nome `scriptpol.m`

```
% script file per confrontare vari algoritmi per
% il calcolo dei valori  del polinomio (x-1)^6
%
p = [ 1 -6 15 -20 15 -6 1];
x = linspace(0.995,1.005,1000);

% Algoritmo 1
fp_pol1 = flops;
w1 = pol1(p,x);
fp_pol1 = flops - fp_pol1

% Algoritmo2
fp_pol2 = flops;
w2 = pol2(p,x);
fp_pol2 = flops - fp_pol2


% Algoritmo di horner
fp_horner =  flops;
y = horner(p,x);
fp_horner = flops-fp_horner

% espressione
fp_fun = flops;
z = (x-1).^6;
fp_fun = flops -fp_fun

plot(x,y,'.r',x,z,'-g',x,w1,'.b',x,w2,'.y')
```

Eseguiamolo con l'istruzione

```
>> scriptpol
```

Approssimazione della derivata prima.

Sappiamo che per definizione la derivata prima di una funzione $f(x)$ è data da

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{f(x+h) - f(x)}{h} = \frac{f(x) + hf'(x) + \frac{h^2}{2} f''(\xi) - f(x)}{h}$$

con $x \leq \xi \leq x + h$.

Da qui deriva che

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{h}{2} f''(\xi)$$

La quantità $\tau(h) = \frac{h}{2} f''(\xi)$ si chiama errore di troncamento e dipende da h. Possiamo dire che l'errore va a zero come $O(h)$ cioè

$$\left| \frac{\tau(h)}{h} \right|$$

è limitato per $h \to 0$.

Consideriamo il rapporto

$$\frac{f(x+h) - f(x-h)}{2h}$$

Utilizzando lo sviluppo in serie di Taylor abbiamo:

$$f(x+h) - f(x-h) =$$

$$= f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) + \frac{h^4}{24} f^{(4)}(\xi_1)$$

$$-f(x) + hf'(x) - \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(x) - \frac{h^4}{24} f^{(4)}(\xi_2) =$$

$$= 2hf'(x) + \frac{h^3}{3} f'''(x) + \frac{h^4}{24} (f^{(4)}(\xi_1) - f^{(4)}(\xi_2))$$

In definitiva abbiamo

$$\frac{f(x+h)-f(x-h)}{2h} =$$

$$= f'(x) + \frac{h^2}{6}f'''(x) + \frac{h^3}{24}(f^{(4)}(\xi_1) - f^{(4)}(\xi_2)) =$$

$$= f'(x) + O(h^2)$$

Scriviamo un file di tipo script per eseguire lo studio dell'errore nel calcolo della derivata prima. Chiamiamo tale file `deriv.m`:

```
%
% studio dell'errore nel calcolo della derivata prima di exp(1)
% con le differenze in avanti e le differenze centrali
%

h = logspace(-16,0,1000);

% approssimazione della derivata del primo ordine

df_1 = (exp(1+h)-exp(1))./h;

% approssimazione della derivata del secondo ordine

df_2 = (exp(1+h)-exp(1-h))./(2*h);

% grafico dell'errore

loglog(h,abs(exp(1)-df_1),'.b',h,abs(exp(1)-df_2),'.r');
```

L'istruzione `loglog` genera grafici in scala logaritmica su entrambi gli assi del sistema di riferimento.

Eseguiamo il file

```
>> deriv
```