Francesca Mazzia
Dipartimento Interuniversitario di Matematica
Università di Bari

SCILAB: 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

```
\\

function [x ] = radice(xt,n)
\\ function  [x ] = radice(xt,n)
\\ esempio errori di arrotodamento
\\ eseguiamo per n volte la radice quadrata di x e
\\ per n volte il quadrato del risultato.


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

for i=1:n
  x = x^2
end

endfunction
```

err = abs(x-xt)/abs(xt)

Adesso eseguiamo il file con diversi valori di n con la seguente istruzione ponendo `xt = 100`.

```
>> s-->getf("radice.sci")

-->[x]=radice(100,10)
 x  =

    1.000E+02

-->err = abs(x-100)/abs(100)
 err  =

    6.366E-14
 -->[x]=radice(100,20)
 x  =

    1.000E+02

-->err = abs(x-100)/abs(100)
 err  =

    1.155E-10

-->[x]=radice(100,30)
 x  =

    1.000E+02

-->err = abs(x-100)/abs(100)
 err  =

    2.299E-07

-->[x]=radice(100,40)
 x  =
```

```
    1.000E+02

-->err = abs(x-100)/abs(100)
 err  =

    5.437E-05


-->[x]=radice(100,60)
 x  =

    1.000E+00

-->err = abs(x-100)/abs(100)
 err  =

    9.900E-01

-->
```

Un'altro esempio è il calcolo delle seguenti funzioni equivalenti

$$
\begin{array}{l}
x * (\sqrt{x^2+1} - x) \\
x * \sqrt{x^2+1} - x^2 \\
x / (\sqrt{x^2+1} + x)
\end{array}
$$

con diversi valori di x.

Scriviamo la seguente function

```
function [x,r1,r2,r3]=limitefx()
// function  [x,r1,r2,r3]=limitefx()
// esempio:  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)

plot2d1('oln',x,[r1 r2 r3],[2 3 4])

endfunction
```

La funzione `x=logspace(d1,d2,N)` genera un vettore riga x con N elementi con spaziatura logaritmica da 10^{d1} a 10^{d2}.
L'istruzione `'` fa diventare il vettore `x` un vettore colonna, in pratica fa il trasposto di un vettore.
L'analoga funzione `x=linspace(d1,d2,N)` genera un vettore x con N elementi con spaziatura lineare da d1 a d2.
L'istruzione `plot2d1` genera grafici in scala logaritmica. In questo esempio la prima stringa 'oln', ha il seguente significato:

- o : significa che vi sono molte curve con un solo valore di x. Quindi x è un vettore colonna e [r1 r2 r3] costituiscono una matrice con tre colonne, ognuna della stessa lunghezza di x.
- l : significa che l'asse logaritmico è usato sull'asse delle x
- n : significa che l'asse normale è usato sull'asse delle y

Il vettore di tre elementi che segue determina lo stile che viene usato. In questo caso si usa una linea continua del colore definito dal numero i-esimo per ogni curva.

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

Eseguiamo il file

```
-->getf("limitefx.sci")

-->[x,r1,r2,r3]=limitefx();
```

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:
$N = 2 : p(x) = a_2 + x(a_1 + a_0 x)$

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

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

```
function py=pol1(p,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.sci`:

```
function py=pol2(p,x)
n=length(p);
```

```
xn = x;
py = p(n);
for i=n-1:-1:1
   py = py + p(i)*xn;
   xn = xn.*x;
end
endfunction
```

La terza funzione la chiamiamo `horn.sci`:

```
function [y]=horn(p,x)
y=[];
// 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

endfunction
```

Scriviamo anche il seguente file dal nome `polhorner.sci`

```
function []=polhorner()
// 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)';
```

```
// rappresentazione compatta
z = (x-1).^6;

// Algoritmo di horner

y = horn(p,x);


// Base delle potenze

w = pol1(p,x);

// Base delle potenze ottimizzato

s = pol2(p,x);

plot2d1('onn',x,[y z w s],[2 3 4 5])

endfunction
```

Eseguiamolo con l'istruzione

```
-->getf("pol1.sci")

-->getf("pol2.sci")

-->getf("horn.sci")

-->getf("polhorner.sci")

-->polhorner();
```

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 $\tau(h) = \dfrac{h}{2}f''(\xi)$ 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 una function per eseguire lo studio dell'errore nel calcolo della derivata prima. Chiamiamo tale file `derivata.sci`:

```
//
// studio dell'errore nel calcolo della derivata prima
// con le differenze in avanti e le differenze centrali
//
function [df_1,df_2]=derivata_exp(x)

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

// approssimazione della derivata del primo ordine

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

// approssimazione della derivata del secondo ordine

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

// grafico dell'errore
plot2d('ll',h,[,abs(exp(x)-df_1)/abs(exp(x)) abs(exp(x)-df_2)/abs(exp(x)) ],[2 3 4])
```

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

Eseguiamo il file

```
--> getf('derivata.sci')
--> derivata_exp(1);
```