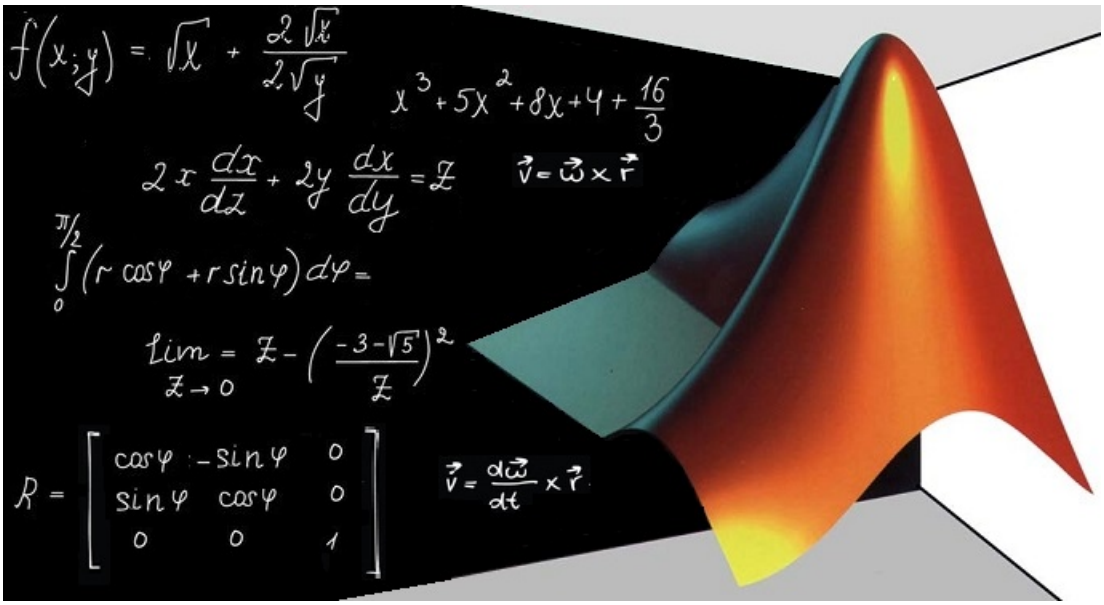
Fabio Nelli (meccanismocomplesso)

# CALCOLO SIMBOLICO CON MATLAB (PARTE 2)

22 March 2014




*matlab_symbolic_calculation.jpg*

Benvenuti alla seconda e ultima parte del calcolo simbolico con Matlab. Nella parte precedente abbiamo visto:

- risoluzione delle equazioni
- manipolazione e semplificazione delle formule matematiche
- derivate e integrali

In questa parte vedremo:

- limiti
- serie numeriche e sommatorie (Taylor,..)
- trasformate (Fourier, Laplace, ecc.)
- matrici (applicabile al calcolo vettoriale)

**Limiti**

Anche i limiti possono essere calcolati simbolicamente e i risultati a volte possono risultare particolarmente interessanti. Per il calcolo dei limiti si usa la funzione

**limit()**. Giusto per vedere la potenzialità di questa funzione, prendiamo come esempio uno dei limiti più popolari:

$$\lim_{x\to 0}\frac{\sin(x)}{x}=1$$

```
>> syms x
>> limit(sin(x)/x)
ans =        1
```

La funzione **limit()** specificata con un solo argomento, implicitamente considera il caso in cui la variabile (in questo caso x) tende a zero. Invece, se consideriamo il caso più generale in cui la variabile tende ad un altro valore, dovremo specificarlo all'interno della funzione **limit()** passandolo come secondo argomento.

```
>> syms x
>> limit(sin(x)/x, inf)
ans =        0
```

**Serie e sommatorie**

*Sommatorie*

Si può utilizzare Matlab anche per il calcolo delle sommatorie, se esistono, utilizzando la funzione symsum(). Per esempio, è possibile calcolare la sommatoria della seguente serie:

$$1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ \ldots+\frac{1}{N^2}\quad ,n=1,\ldots,\infty$$

$$\sum_{n=1}^{\infty}\frac{1}{n^2}=\frac{\pi^2}{6}$$

```
>> syms x k
>> s = symsum(1/k^2,1,inf)
s =      pi^2/6
```

Un altro esempio simile potrebbe essere la sommatoria di una serie geometrica:

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \ , \quad with\ |x \leq 1|$$

```
>> syms x k
>> s = symsum(x^k,k,0,inf)
s =piecewise([1 <= x, Inf], [abs(x) < 1, -1/(x - 1)])
```

*La serie di Taylor*

Anche la serie di Taylor è ottenibile mediante questa toolbox. Infatti è possibile calcolare la serie di Taylor di una generica funzione usando la funzione **taylor()**. Per esempio, consideriamo la seguente funzione:

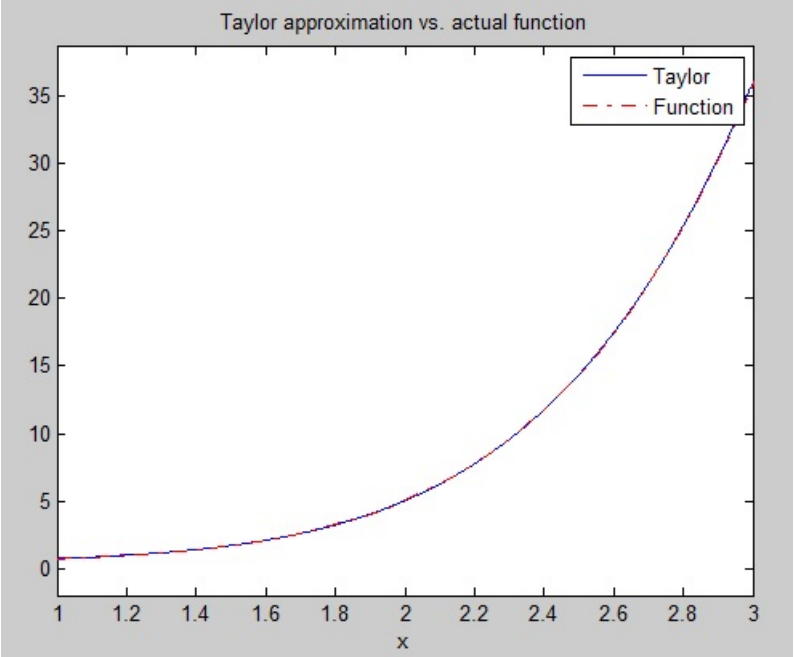
$$f(x) = \log(1+x)$$

e ora calcoliamo la serie di Taylor:

```
>> syms x
>> f = taylor(log(1+x))
f =x^5/5 - x^4/4 + x^3/3 - x^2/2 + x
>> pretty(f)
    5    4    3    2
   x    x    x    x
  -- + -- + -- + -- + x
   5    4    3    2
```

Da notare il risultato ottenuto mediante l'uso aggiuntivo della funzione **pretty()**. Questa funzione stampa un output simbolico in modo simile a come si presenta visualmente l'espressione della formula matematica. I seguenti comandi Matlab ci permettono di mostrare un grafico in cui vengono messi a paraganone l'andamento dell'approssimazione di Taylor con la funzione reale.

```
>> xd = 1:0.05:5;
>> yd = subs(f,x,xd);
>> ezplot(f, [1, 3]);
>> hold on;
>> plot(xd, yd, 'r-.')
>> title('Taylor approximation vs. actual function');
>> legend('Taylor','Function')
```




**Calcolo matriciale**

Non c'è alcun bisogno di aggiungere che il pane quotidiano di tutti gli ingegneri è il calcolo matriciale. Ancora di più quando consideriamo il grande potenziale che sta nel calcolare le matrici mantenendo i parametri indefiniti lungo tutti i vari calcoli matriciali. Consideriamo due matrici generiche 2×3 con tutti gli elementi al loro interno definiti da parametri. Ci riferiremo a queste due matrici come A e B.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Ora, passando a Matlab, per prima cosa dobbiamo definire le 8 variabili come simboli, ciascuna corrispondente ad un diverso elemento all'interno delle due matrici. Successivamente, definiremo le due matrici come abbiamo sempre fatto con Matlab.

```
>> syms a b c d e f g h
>> A = [a,b;c,d];
>> B = [e,f;g,h];
```

Ora che abbiamo definito tutto ciò di cui abbiamo bisogno, vediamo come addizionare queste due matrici. L'addizione di due matrici parametriche genererà un'ulteriore matrice parametrica 2×2:

```
>> C = A + BC = [a+e, b+f][c+g, d+h]
```

In maniera simile, anche il prodotto tra le matrici A e B produce una nuova matrice che mantiene al suo interno i parametri iniziali.

```
>> D = A*BD =[a*e+b*g, a*f+ b*h][c*e+d*g, c*f+d*h]
```

Certamente a questo punto ti starai chiedendo: OK, sto lavorando simbolicamente e possono vedere i parametri con cui sono partito, ma ora vorrei sostituirli con dei valori numerici... Ok. Nessun problema. Lavorando simbolicamente con le matrici alla fine otteniamo un insieme di matrici contenenti gli 8 parametri di partenza. Ora è possibile valutare tutte queste matrici numericamente. E' solamente necessario assegnare agli 8 parametri di partenza un set di valori appropriati e poi con il comando **eval** ottenere il risultato numerico delle matrici ottenute nei calcoli precedenti.

```
>> a=1;b=2;c=3;d=4;e=5;f=6;e=7;f=8;g=9;h=0;
>> eval(A)
ans = 1 2 3 4
```

Un'altra operazione molto comune che si fa sulle matrici è il calcolo dell'inversa. Si può ottenere l'inversa di A utilizzando lo stesso comando **inv()** che si utilizza per le matrici numeriche, ma il risultato in questo caso sarà un'altra matrice simbolica.

```
>> D = inv(A)D = [ d/(a*d-b*c), -b/(a*d-b*c)][-c/a*d-b*c),a/[a*d-b*c)]
```

Nuovamente, se si desidera valutare numericamente la matrice inversa di A, si può utilizzare anche qui il comando **eval()**.

```
>> Dn = eval(inv(A))Dn = -2.0000 1.00001.5000 -0.5000
```

Tutte le persone che si avvicinano al mondo della robotica, scopriranno presto (o hanno già scoperto) quanto sia importante l'approccio simbolico (parametrico) nel calcolo matriciale, specialmente durante il calcolo del jacobiano. Comunque anche per tutti gli altri che generalmente lavorano con Matlab avranno ormai compreso l'importanza di lavorare con le espressioni matematiche "simbolicamente".