Tutoraggio del corso di Laboratorio di Matematica
Anno Accademico 2005/2006
Giovanni Bazzoni

Calcolo simbolico in Matlab

Il pacchetto Symbolic Math Toolbox definisce un tipo di variabile differente da quelle viste in precedenza, che sono double e char. In effetti si tratta di una variabile simbolica, che viene dichiarata con il comando `sym` oppure `syms`. Tale variabile non è di tipo numerico, e pertanto non è soggetta ad errori di approssimazioni da parte del calcolatore.
Possiamo creare una variabile simbolica `x` con il comando

```
>> sym('x')
```

In alternativa è possibile convertire, sempre con il comando `sym`, una variabile double, oppure char, in una simbolica, mediante l'assegnazione

```
>> z_sym=sym('z')
```

dove `z` è una variabile numerica o una stringa. Tale nuova variabile viene utilizzata in maniera formale da parte di Matlab.
Per dichiarare variabili simboliche in maniera più rapida si può utilizzare il comando

```
>> syms a b c
```

Vediamo che effettivamente tali variabili hanno solo un valore formale; definiamo ad esempio

```
>> d=a*b-c
```

Con il comando `subs` sostituiamo alle variabili simboliche dei valori specifici. La sintassi è

```
>> subs(d,{a,b,c},{2,3,4})
```

che produce l'output

```
ans = 2
```

Quello che Matlab ha fatto è sostituire `{2,3,4}` alle variabili `{a,b,c}` rispettivamente e sostituire a `d` il risultato dell'operazione.

Un comando molto importante per maneggiare variabili simboliche è `solve`. Supponiamo ad esempio di voler risolvere il sistema lineare

$$\begin{cases} ax + by = p \\ cx + dy = q \end{cases}$$

nelle variabili `x` e `y`, dove `a`, `b`, `c`, `d`, `p` e `q` sono parametri. Per far ciò dichiariamo le variabili simboliche

```
>> syms a b c d p q x y
```

e utilizziamo il comando `solve`, che ha la seguente sintassi:

```
>> [x,y]=solve('a*x+b*y=p','c*x+d*y=q')
```

Otteniamo in risposta:

```
x =

-(-d*p+q*b)/(a*d-b*c)


y =

(a*q-p*c)/(a*d-b*c)
```

Matlab ha risolto il sistema lineare rispetto alle variabili `x` e `y` ed ha messo il risultato in due variabili simboliche che abbiamo chiamato ancora `x` ed `y`. Come ha fatto a sapere che volevamo ricavare proprio `x` e `y`? Il fatto è che Matlab è stato impostato per fare esattamente questo, sapendo che le incognite sono indicate, nella maggior parte dei casi, proprio da `x` e `y`.
Se siamo interessati nello specifico alla soluzione di

$$\begin{cases} 2x + 3y = 1 \\ -2x + y = 7 \end{cases}$$

è sufficiente utilizzare il comando `subs` nella seguente maniera:

```
>> subs(x,{a,b,c,d,p,q},{2,3,-2,1,1,7})
ans =

   -2.5000

>> subs(y,{a,b,c,d,p,q},{2,3,-2,1,1,7})
ans =

     2
```

Queste sono effettivamente le soluzioni esatte del sistema lineare.
Possiamo anche verificare che `x` e `y` sono le soluzioni *formali* del sistema lineare, andando a fare la sostituzione:

```
>> a*x+b*y

ans =

-a*(-d*p+q*b)/(a*d-b*c)+b*(a*q-p*c)/(a*d-b*c)
```

Questo non ha certo l'aria, a prima vista, di essere uguale a `p`. Tuttavia possiamo semplificare tale espressione con il comando `simple` in questo modo:

```
>> [ris,modo]=simple(ans)

ris =
```

```
p


modo =

factor
```

`simple` semplifica un'espressione simbolica e restituisce due variabili, per esempio `ris` e `modo` che contengono rispettivamente il risultato della semplificazione e il metodo di semplificazione usato. In effetti `p` è il risultato che ci aspettavamo.
Per quanto riguarda il metodo usato per semplificare, sono possibili le seguenti scelte:

1. `combine`: questa opzione viene utilizzata per semplificare formule trigonometriche; ad esempio, se abbiamo dichiarato una variabile simbolica `x` e l'espressione da semplificare è $\sin(x) - 2\cos^2(x) + 1$ abbiamo

   ```
   >> [ris,modo]=simple(sin(x)-2*cos(x)^2+1)

   ris =

   sin(x)-cos(2*x)


   modo =

   combine
   ```

2. `simplify`: questa opzione permette di semplificare espressioni di qualsiasi genere; tale comando `simplify` funziona anche nel modo

   ```
   >> simplify(a*b-c*c+3*d*b-b*d+b*b)

   ans =

   a*b-c^2+2*d*b+b^2
   ```

3. `collect`: anche questo comando può essere utilizzato come `simplify` ed è specifico per effettuare raccoglimenti di simboli uguali:

   ```
   >> collect(x^2*y+y+x^3*y^2-y^2,y)

   ans =

   (x^3-1)*y^2+(x^2+1)*y
   ```

Notare che abbiamo anche specificato che volevamo effettuare il raccoglimento rispetto alla variabile `y`;

4. `factor`: fattorizza prodotti notevoli, come potenze di binomi;

```
>> factor(x^4+4*x^3+6*x^2+4*x+1)

ans =

(x+1)^4
```

5. `expand`: fa esattamente l'opposto, ovvero svolge espressioni compatte;

```
>> expand(cos(4*x))

ans =

8*cos(x)^4-8*cos(x)^2+1
```

Riassumendo, per semplificare espressioni simboliche abbiamo due strade:

- chiamare la funzione `[ris,modo]=simple(`*espressione*`)` e lasciar scegliere a Matlab quale tecnica utilizzare;
- oppure utilizzare una delle opzioni `simplify`, `collect`, `expand` o `factor`.

Un'utile applicazione delle variabili simboliche è nel calcolo differenziale. Infatti Matlab è capace di calcolare le derivate e gli integrali delle principali funzioni elementari. Anche le regole di calcolo (derivata di un prodotto, derivata di un quoziente) sono implementate. L'utilità sta nel fatto che alcune derivate possono essere molto lunghe da calcolare e può essere impossibile calcolare integrali, anche per funzioni elementari.
Per prima cosa, vediamo come differenziare una funzione. Dichiariamo una variabile simbolica `x`, e scegliamo quale funzione vogliamo derivare; ad esempio, ci interessa la derivata di $f(x) = x^2$. In tal caso scriviamo

```
>> f=x^2;
>> diff(f)

ans =

2*x
```

Ci sono funzioni per le quali il calcolo della derivata può essere complicato; ad esempio, se la nostra funzione è $f(x) = \tan(x)^x$, abbiamo

```
>> f=tan(x)^x;
>> diff(f)

ans =

tan(x)^x*(log(tan(x))+x*(1+tan(x)^2)/tan(x))
```

Per chi la sa calcolare, può verificare che è proprio quella giusta!
Se poi abbiamo una funzione di due variabili, ad esempio $f(x,y) = xy - \log(x)e^y$, dobbiamo specificare rispetto a quale variabile ci interessa calcolare la derivata:

```
>> f=x*y-log(x)*exp(y);
>> diff(f,x)

ans =

y-1/x*exp(y)

>> diff(f,y)

ans =

x-log(x)*exp(y)
```

Nel caso in cui bisogna visualizzare sul terminale funzioni complicate, può essere utile il comando `pretty`; ad esempio, con $f(x) = \arccos(\tan(\sqrt{x^2-5}))$, abbiamo

```
>> f=arccos(tan(sqrt(x^2-5)));
>> diff(f,x)

ans =

-(1+tan((x^2-5)^(1/2))^2)/(x^2-5)^(1/2)*x/(1-tan((x^2-5)^(1/2))^2)^(1/2)

>> pretty(ans)

                                  2      1/2 2
                        (1 + tan((x  - 5)   ) ) x
                - ----------------------------------------
                     2     1/2             2     1/2 2 1/2
                  (x  - 5)    (1 - tan((x  - 5)   ) )
```

Matlab ci permette di vedere un po' meglio quale sia l'espressione della derivata.
Possiamo calcolare anche derivate di ordine superiore, aggiungendo agli argomenti del comando `diff` un parametro intero che indica l'ordine di derivazione desiderato:

```
>> f=sin(x)^2;
>> diff(f,3)

ans =

-8*cos(x)*sin(x)
```

Come ultima possibilità del calcolo differenziale simbolico abbiamo lo sviluppo in serie di Taylor di una funzione analitica centrato nel punto x_0:

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

Il comando è `taylor(`*funzione,ordine di arresto,centro*`)`, dove *funzione* è il nome della funzione, passato come stringa oppure come variabile simbolica, *ordine di arresto* è l'esponente di grado massimo che vogliamo compaia nello sviluppo in serie e *centro* è il punto nel quale vogliamo centrare lo sviluppo.

```
>> f=exp(x);
>> taylor(f,10,1)

ans =

exp(1)+exp(1)*(x-1)+1/2*exp(1)*(x-1)^2+1/6*exp(1)*(x-1)^3+1/24*exp(1)*(x-1)^4
```

Vediamo infine come procedere al calcolo di primitive. Il comando è `int`. Niente di meglio che utilizzarlo su un esempio concreto:

```
>> f=1/cos(x);
>> int(f)

ans =

log(sec(x)+tan(x))
```

Volendo verificare:

```
>> diff(ans)

ans =

(sec(x)*tan(x)+1+tan(x)^2)/(sec(x)+tan(x))

>> [ris,modo]=simple(ans)

ris =
```

```
1/cos(x)


modo =

simplify
```

Possiamo calcolare anche integrali definiti, modificando leggermente il comando `int`. Ad esempio

```
>> f=cos(x);
>> int(f,-pi/2,pi/2)

ans =

2
```

La sintassi quindi è `int(`*funzione,primo estremo di integrazione,secondo estremo di integrazione*`)`.
Possiamo anche calcolare integrali impropri, quando siano convergenti. Supponiamo infatti di dover calcolare

$$\int_0^1 \frac{\arctan(x)}{x^{3/2}} dx$$

Osserviamo che questa funzione è integrabile in senso improprio in 0. Il comando è ancora

```
>> f=atan(x)/x^(3/2);
>> int(f,0,1)

ans =

-1/2*pi+1/2*2^(1/2)*log(2+2^(1/2))-1/2*2^(1/2)*log(2-2^(1/2))+1/2*2^(1/2)*pi
```

La risposta è esatta formalmente; in altre parole, Matlab calcola la primitiva di $f(x)$, e poi applica il teorema fondamentale del calcolo. L'espressione sostituita è *esatta*, cioè non è approssimata. La variabile `pi` contiene il valore esatto di π. Siccome però a noi interessa un valore numerico, utilizziamo il comando

```
>> double(ans)

ans =

    1.8971
```