Symbolics (autore: Vittorio Albertoni)

Premessa

Julia ha il grande vantaggio di poter utilizzare moduli Python nei modi illustrati nel manualetto allegato all'articolo «Un nuovo linguaggio per la data science» pubblicato sul mio blog all'indirizzo *www.vittal.it* nel febbraio 2021.

Per il calcolo simbolico possiamo pertanto ricorrere ai comandi del modulo Python SymPy attraverso il package Julia PyCall o addirittura utilizzare l'interfacciamento diretto attraverso il package Julia SymPy e il citato manualetto contiene esempi per entrambe le modalità.

Ritengo che questa sia ancora la soluzione ottimale per avere la semplicità di Python e l'accesso ad una gamma di funzioni vastissima, purché siamo disponibili a pagare questa comodità con una qualche lentezza di esecuzione.

Il più grande vantaggio nell'utilizzare Symbolics lo dovremmo trovare infatti nella velocità di esecuzione. Nè potrebbe essere diversamente, dal momento che, a differenza di SymPy, Symbolics è programmato in linguaggio Julia e fa parte direttamente dell'ecosistema Julia.

Propongo un confronto.

Sia da calcolare la derivata della funzione $y = \sin x^2 \exp x$.

Se utilizziamo il modulo Python SymPy procediamo in questo modo

```
julia> using SymPy
julia> x = symbols("x");
julia> derivata = diff(sin(x^2)*exp(x), x);
julia> print(simplify(derivata))
(2*x*cos(x^2) + sin(x^2))*exp(x)
```

Se utilizziamo il package Julia Symbolics procediamo in questo modo

```
julia> using Symbolics
julia> @variables x;
julia> derivata = Symbolics.derivative(sin(x^2)*exp(x), x);
julia> print(simplify(derivata))
sin(x^2)*exp(x) + 2x*exp(x)*cos(x^2)
```

Stessa quantità di istruzioni, ovviamente diverse tra loro, per arrivare allo stesso risultato, meglio espresso nel caso di SymPy. Forse per un errore di gioventù la funzione `simplify` di Symbolics non raccoglie il termine `exp(x)`.

Circa la maggiore velocità di elaborazione ottenibile con l'uso di Symbolics questo piccolo esempio non può ovviamente dimostrare nulla.

Da dilettanti, tutto sommato, direi che, una volta che abbiamo imparato ad usare SymPy, potremmo avvalercene sia programmando in Python sia programmando in Julia.

A chi voglia cimentarsi con Symbolics dedico questo manualetto che ne guida un utilizzo di base, esonerando così il dilettante, in mancanza di qualsiasi abbordabile indicazione in lingua italiana e facilmente comprensibile, dal dover affrontare la consultazione della documentazione di Julia, notorimente carente sul piano della chiarezza.

Il tutto essendo consapevoli del fatto che, nel momento in cui scrivo (settembre 2025) Symbolics non ha ancora la completezza di funzioni che si trovano in SymPy e ce ne accorgiamo leggendo questo manualetto che ho proposto nella stessa impostazione di quello a suo tempo proposto per SymPy.

In rete troviamo vari suggerimenti per ovviare alle ancora molte carenze di Symbolics con il ricorso a vari packages che ho riscontrato funzionano quando vogliono loro, non danno alcun affidamento e gonfiano inutilmente il software Julia mettendone anche a rischio il funzionamento. Altro serio motivo per tenerci comunque disponibile SymPy, con cui possiamo fare veramente tutto.

Per intanto qui mi limito a illustrare compiutamente ciò che funziona avendo semplicemente installato Symbolics.

Indice

1 Installazione **3**

2 Importazione **3**

3 Operazioni di base **3**
3.1 Dichiarazione dei simboli 3
3.2 Costruzione delle espressioni 3
3.3 Sostituzione di simboli 4
3.4 Dall'espressione numerica al numero 4
3.5 Visualizzazione 5

4 Elaborazione delle espressioni **5**
4.1 Semplificazioni 5
4.2 Espansioni polinomiali 6
4.3 Operazioni sui polinomi 6

5 Calcolo **6**

6 Risolutori **7**

7 Matrici **7**

8 Avvertenza conclusiva **8**

1 Installazione

Symbolics è un package Julia, cioè una delle tante librerie destinate ad arricchire le funzionalità di base del linguaggio Julia.

L'installazione avviene utilizzando la shell di Julia in modalità `Pkg`, modalità in cui si entra inserendo al prompt di Julia una parentesi quadra chiusa (`]`).
In questa modalità il prompt si trasforma da `julia>` a `pkg>` e installiamo con il comando
`add Symbolics`.

Dal momento che siamo in modalità installazione può essere utile installare un altro pacchetto di arricchimento, chiamato Latexify, che consente di tramutare qualsiasi espressione prodotta da Julia in linguaggio LaTeX, in modo da poter facilmente inserire in documenti prodotti con LaTeX, con copia e incolla, quando necessario per ottenere la stessa espressione scritta nell'insuperabile ben noto stile LaTeX.
Se interessa, installiamo con
`add Latexify`.

Per tornare al prompt di Julia utilizziamo il tasto BACKSPACE.

2 Importazione

Per utilizzare un package non basta averlo installato ma occorre anche dichiarare di volerlo usare importandolo.

Per utilizzare Symbolics dobbiamo scrivere il comando
`using Symbolics`
nella shell di Julia o all'inizio del file in cui inseriamo uno script Julia.

Se prevediamo di utilizzare anche Latexify lo dichiariamo aggiungendo il comando
`using Latexify`

Potremmo, in tal caso, fare prima con
`using Symbolics, Latexify`

3 Operazioni di base

3.1 Dichiarazione dei simboli

I simboli non numerici che intendiamo usare nelle nostre elaborazioni vanno preventivamente dichiarati.

La dichiarazione avviene usando la macro `variables` con la sintassi
`@variables` <simbolo>, <simbolo>, ...
dove <simbolo> è una lettera dell'alfabeto latino direttamente prodotta con la tastiera o una lettera dell'alfabeto greco prodotta con le dizioni `alpha, beta, gamma, delta, epsilon, zeta, eta, theta, iota, kappa, lamda, mu, nu, xi, omicron, pi, rho, sigma, tau, upsilon, phi, chi, psi, omega` precedute da \ e con pressione del tasto TAB dopo averle scritte.

Nello stesso modo dovremo scrivere questi simboli espressi con lettere dell'alfabeto greco quando li inseriremo nelle espressioni.

Il fatto che la macro si chiami `variables` non ha niente a che vedere con le variabili del linguaggio Julia.

3.2 Costruzione delle espressioni

Le espressioni matematiche si compongono utilizzando i simboli dichiarati, funzioni riconosciute da Symbolics e numeri, combinati con gli operatori aritmetici del linguaggio Julia (`^ // * / + -`).

L'espressione può essere assegnata a una variabile.

Esempio:
```
e = sin(x)^2 + cos(x)^2 - x + 2*x*y
```
assegna alla variabile `e` l'espressione matematica $sin^2(x) + cos^2(x) - x + 2xy$

Possiamo creare un'espressione da una stringa utilizzando la sintassi:
```
<nome_espressione> = eval(Meta.parse("<stringa>")
<nome_espressione> = eval(Meta.parse(<nome_stringa>)
```
La stringa può essere un nome o una frase qualsiasi ma, ovviamente, a noi servirà che sia un'espressione matematica.
Nel primo caso scriviamo la stringa tra le parentesi e tra apici.
Nel secondo caso importiamo una variabile stringa già esistente con un suo nome.
In questo modo possiamo acquisire come espressione matematica elaborabile in Symbolics un'espressione generata al di fuori di Symbolics, per esempio introdotta dall'utente da tastiera come risposta all'invito di input da uno script Julia.
Esempi:
```
e = eval(Meta.parse("4*x^2-5"))
```
assegna alla variabile `e` l'espressione matematica $4x^2 - 5$.
```
e = eval(Meta.parse(formula))
```
assegna alla variabile `e` l'espressione contenuta in una stringa denominata `formula`.

3.3 Sostituzione di simboli

Le espressioni matematiche di Symbolics sono immutabili.

E' tuttavia possibile generare altre espressioni matematiche sostituendo simboli di una espressione matematica con altri simboli o valori numerici.

A questo scopo dobbiamo costruire un dizionario, che possiamo chiamare `sostituzioni`, indicando in chiave il o i simboli da sostituire e in loro corrispondenza l'elemento o gli elementi da mettere in sostituzione (altri simboli o valori numerici).

Poi possiamo effettuare le sostituzioni con la sintassi
```
substitute(<vecchia_espressione>, sostituzioni)
```
Esempio:
Data l'espressione `e` dell'ultimo esempio del paragrafo precedente, dichiarate le variabili `x` e `y`, con
```
sostituzioni = Dict([x=>y])
```
e
```
substitute(e, sostituzioni)
```
generiamo l'espressione $4y^2 - 5$.
Con
```
sostituzioni = Dict([x=>2])
```
e
```
substitute(e, sostituzioni)
```
generiamo l'espressione matematica $4 * 2^2 - 5$, cioè il numero 11.

In ogni caso l'espressione `e` rimane $4x^2 - 5$.
Se vogliamo salvare l'espressione modificata dobbiamo assegnare il risultato della sostituzione ad una nuova variabile.
Ad esempio con
```
new_e = substitute(e, sostituzioni)
```
creeremmo una variabile, chiamata `new_e` che, nel primo caso, conterrebbe l'espressione $4y^2 - 5$ e, nel secondo caso, conterrebbe il valore 11.

3.4 Dall'espressione numerica al numero

Quando un sistema di calcolo simbolico compie calcoli su numeri fornisce il risultato in una espressione numerica.

Nei sistemi di calcolo simbolico puro, come quello del modulo SymPy di Python, se questo risultato è un numero che esiste, l'espressione numerica coincide con il numero stesso e non ci accorgiamo nemmeno che è un'espressione numerica.

Se il risultato è un numero che non esiste e che non si può calcolare, come avviene per i numeri irrazionali, l'espressione numerica è, in realtà, un simbolo.

La funzione `sqrt(8)` del Sympy di Python fornisce il risultato $2\sqrt{2}$.

L'oggetto che contiene questo risultato ha una funzione membro, che si chiama `evalf()`, che forza la traduzione dell'espressione numerica simbolica in numero approssimato.

Nel linguaggio Julia il sistema di calcolo simbolico non è puro ma ibrido e qualsiasi espressione numerica viene forzata a diventare numero, magari approssimato.

Pertanto se in Julia utilizziamo i comandi del modulo SymPy di Python utilizzando il package Pycall abbiamo un sistema di calcolo simbolico puro.
Se utilizziamo i package SymPy o Symbolics di Julia abbiamo un sistema di calcolo simbolico ibrido.
Nel primo caso, il comando `sqrt(8)` fornisce il risultato $2\sqrt{2}$.
Nel secondo caso, il comando `sqrt(8)` fornisce il risultato 2.8284271247461903.

3.5 Visualizzazione

Per visualizzare il contenuto di variabili e i risultati che via via otteniamo dalle nostre elaborazioni, dal momento che lavoriamo in Julia, abbiamo ovviamente a disposizione la funzione `print()`, con la quale, tuttavia, otteniamo la visualizzazione delle espressioni così come si inseriscono da tastiera e non vi è modo di migliorare l'output in tal senso.

Se ci serve evidenziare i risultati delle nostre elaborazioni in maniera tipograficamente eccellente possiamo ricorrere al package Latexify ed al suo comando `latexify`.

Per esempio, se abbiamo nella variabile `derivata` il risultato ottenuto nell'esempio fatto in premessa:
`sin(x^2)*exp(x) + 2x*exp(x)*cos(x^2)`
con i comandi
```
using Latexify
latexify(derivata)
```
otteniamo a terminale il codice Latex per inserire in un documento Latex la nostra derivata

```
julia> latexify(derivata)
L"\begin{equation}
\sin\left( x^{2} \right) e^{x} + 2 x e^{x} \cos\left( x^{2} \right)
\end{equation}
"
```

Per esempio, inserendo con copia e incolla, tra due simboli $ e $ la riga centrale tra `\begin(equation)` e `\end` nel documento che sto scrivendo con l'editor LyX, dopo aver scelto lo strumento INSERISCI MATEMATICA ottengo
$\sin\left(x^2\right) e^x + 2xe^x \cos\left(x^2\right)$
che è un po' meglio da vedere e da leggere.

4 Elaborazione delle espressioni

4.1 Semplificazioni

Quando creiamo un'espressione dalla tastiera o importando una stringa, Symbolics la acquisisce apportando già le più immediate semplificazioni.

Per esempio, se inseriamo nella variabile `e` l'espressione
$sin^2(x) + cos^2(x) - x + 2xy + 4x - \frac{xy}{2}$
digitando la seguente istruzione
`e = sin(x)^2 + cos(x)^2 - x + 2*x*y + 4*x - (x*y)/2`
l'espressione che viene creata da Symbolics è
3x + (3//2)*x*y + sin(x)^2 + cos(x)^2
meglio «latexificabile» così
$3x + \frac{3}{2}xy + sin^2(x) + cos^2(x)$

nella quale abbiamo già $3x$ al posto di $-x + 4x$ e $\frac{3}{2}xy$ al posto di $2xy - \frac{xy}{2}$

Ma la nostra espressione contiene un'altra possibile semplificazione, meno immediata: la somma dei quadrati di $sin(x)$ e $cos(x)$, infatti, è uguale a 1, qualunque sia il valore di x.

Per eseguire le semplificazioni meno immediate Symbolics ci offre la funzione `simplify()` il cui argomento può essere un'espressione creata al momento o il nome di un'espressione esistente.
Esempi:
`simplify(sin(x)^2 + cos(x)^2)`
fornisce il risultato 1;
`simplify(e)` con riferimento all'espressione `e` del precedente esempio
fornisce il risultato
1 + 3x + (3//2)*x*y
meglio «latexificabile» così
$1 + 3x + \frac{3}{2}xy$.

4.2 Espansioni polinomiali

Per espandere un'espressione polinomiale abbiamo la funzione `expand()`, il cui argomento è un'espressione inserita direttamente o richiamata.
`expand((a+b)^3)` produce l'espressione $a^3 + 3a^2b + 3ab^2 + b^3$
Se abbiamo l'espressione `e = (x - 3)*(x + 1)`,
`expand(e)` produce l'espressione $-3 - 2x + x^2$
`expand((a+b)*(a-b))` produce l'espressione $a^2 - b^2$

4.3 Operazioni sui polinomi

Per sommare, sottrarre, moltiplicare ed elevare a potenza possiamo usare i soliti operatori `+` `-` `*` e `^` di Julia.
Esempi:
Siano A e B i due polinomi
`A =` $x^4 - 2x^3 - 8x + 16$
`B =` $2x^4 + 8x$
`A + B` ritorna $3x^4 - 2x^3 + 16$
`A - B` ritorna $-x^4 - 2x^3 - 16x + 16$
`A * B`, assoggettando il risultato a expand(), ritorna $2x^8 - 4x^7 - 8x^5 + 16x^4 - 64x^2 + 128x$
`B ** 2`, assoggettando il risultato a expand(), ritorna $4x^8 + 32x^5 + 64x^2$

Per la divisione Symbolics non è attrezzato e occorrerebbe ricorrere ad altri packages dal funzionamento incerto.

5 Calcolo

Symbolics possiede tutte le funzioni matematiche preconfezionate che ritroviamo nel linguaggio Julia e le possiamo applicare anche a simboli semplicemente richiamandole o richiamandole con la sintassi `Symbolics.`<nome_funzione>.

Sempre nell'ambito del calcolo, con la funzione
`Symbolics.derivative(`<funzione_da_derivare>`, `<variabile_di_derivazione>`)`
calcoliamo la derivata.

Con

```
julia> using Symbolics
julia> @variables x;
julia> Symbolics.derivative(log(x), x)
```

otteniamo

```
1 / x
```

Per altri tipi di calcolo, come integrali e limiti, Symbolics non è direttamente attrezzato ed occorrerebbe installare packages aggiuntivi installabili a rischio e pericolo dell'utente.

6 Risolutori

Stesso discorso di pacchetti aggiuntivi installabili a rischio e pericolo dell'utente.

7 Matrici

Per costruire una matrice lavorando con Symbolics usiamo la stessa sintassi che usiamo lavorando in Julia.

In Julia con

```
[1 2; 4 2]
```

costruiamo la matrice

$$\begin{vmatrix} 1 & 2 \\ 4 & 2 \end{vmatrix}$$

In Symbolics con

```
@variables a,b,c,d
[a b; c d]
```

costruiamo la matrice

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

Lo stesso dicasi per i vettori.

```
[a b]
```

costruisce il vettore riga $\begin{vmatrix} a & b \end{vmatrix}$

```
[a; b]
```

costruisce il vettore colonna $\begin{vmatrix} a \\ b \end{vmatrix}$

Con gli operatori `+`, `-`, `*`, `/` e `^` possiamo fare la somma, la differenza, la moltiplicazione, la divisione e l'elevamento a potenza di matrici.

Se dichiariamo anche l'uso del package LinearAlgebra, che fa parte della dotazione standard di Julia, possiamo calcolare la matrice inversa con

`inv(<matrice>)`

e il determinante con

`det(<matrice>)`.

Esempi:

Previo

```
using Symbolics
@variables x
```

Siano A = $\begin{vmatrix} x & 2 \\ 1 & 3 \end{vmatrix}$ e B = $\begin{vmatrix} 3 & 2 \\ 4 & 1 \end{vmatrix}$

`A + B =` $\begin{vmatrix} 3+x & 4 \\ 5 & 4 \end{vmatrix}$

`A * 3 =` $\begin{vmatrix} 3x & 6 \\ 3 & 9 \end{vmatrix}$

`A * B =` $\begin{vmatrix} 8+3x & 2+2x \\ 15 & 5 \end{vmatrix}$

`A ^ 2 =` $\begin{vmatrix} 2+x^2 & 6+2x \\ 3+x & 11 \end{vmatrix}$

`A ^ -1 =` $\begin{vmatrix} \frac{3}{-2+3x} & \frac{-2}{-2+3x} \\ \frac{-1}{-2+3x} & \frac{x}{-2+3x} \end{vmatrix}$

`B ^ -1 =` $\begin{vmatrix} -0.2 & 0.4 \\ 0.8 & -0.6 \end{vmatrix}$

`A.det() =` $-2 + 3x$

`B.det() =` -5

8 Avvertenza conclusiva

Se confrontiamo questo manualetto con quello che avevo dedicato a SymPy vediamo come la differenza tra Symbolics e SymPy sia ancora abissale.

Una cosa, poi, che non si capisce è se gli sviluppatori di Symbolics abbiano intenzione di costruire uno strumento monoblocco come è SymPy o se pensano di circondare Symbolics di altri pacchetti via via da aggiungere per fare una cosa piuttosto che un'altra.

Se la scelta fosse quest'ultima è molto meglio che ci teniamo Sympy.