Alessandro Vicini
Dipartimento di Fisica, Università degli Studi di Milano

Introduzione alla programmazione in Mathematica

Alessandro.Vicini@mi.infn.it
http://wwwteor.mi.infn.it/users/vicini

Sommario

espressioni
funzioni
patterns
regole di sostituzione e definizioni
valutazione delle espressioni
package
MathLink

programma

presentazione teorica (3 lezioni)
scrittura di un package con esempi dettagliati dell'utilizzo di quanto esposto nelle prime lezioni +
utilizzo delle funzioni per la gestione dei dati (input/output) (7 lezioni)

commenti generali

- Mathematica è un programma di manipolazione algebrica
- Mathematica è un ambiente in cui far funzionare i propri packages
- stile di programmazione funzionale (vs. sequenziale)
- espressioni e loro valutazione (cosa è globale e cosa è locale)
- qualunque oggetto è manipolabile formalmente ($\rightarrow$ efficienza medio-bassa)
- per eventuali applicazioni specifiche, uso, dall'interno di Mathematica, di programmi esterni attraverso MathLink

esempio: <u>calcolo di sezioni d'urto in fisica delle particelle</u>
scrittura dei diagrammi di Feynman
algebra di Dirac ed altre semplificazioni
scrittura automatizzata di un programma Fortran

commento pratico

con un editor di testo (p.es. emacs) scrivere un file.m
digitando math iniziare la sessione interattiva con il kernel di Mathematica
con il comando <<file.m caricare il file che viene eseguito, riga per riga, come una sequenza di comandi interattivi
nel file, concludere ogni comando con ;

Espressioni 1

che cos'è un certo oggetto? *comando:* Head

es.1 Integer, Real, Complex,Symbol

es.2 Plus (ovvero somma), Times (ovvero prodotto), f

qualunque quantità in Mathematica è una espressione:
i numeri, i simboli, le funzioni, la Head di una funzione, una regola di sostituzione, una regola di assegnazione

qualunque espressione può essere sostituita con un'altra

qual è la struttura di una certa espressione?
comando: FullForm
tipica struttura di Mathematica è la List
es.1 prova=$\{a, b, c\}$=List[a,b,c]
è pratico organizzare gli oggetti in liste poichè:
i) possono essere trattati come gli argomenti di una funzione;
ii) su ciascun oggetto può essere iterata una funzione.
Per generare una lista si può usare il comando Table
es.2 Table[f[i],{i,1,3}] = {f[1],f[2],f[3]}
Qualunque espressione matematica è data da una sequenza di operazioni che assumono, in fin dei conti, la forma di una lista.
prova=FullForm[$ax^2 + b$] = Plus[b, Times[a, Power[x, 2]]]

Espressioni 2

prova=FullForm[$ax^2 + b$] = Plus[b, Times[a, Power[x, 2]]]

Elementi di una espressione *comando:*Part

es. Part[prova, 1]=prova[[1]]=b
Part[prova, {2,2,1}]=prova[[2,2,1]]=x
Part[prova, 0]=prova[[0]]=Plus

Lunghezza di una espressione *comando:*Length

es. Length[prova]=2

Aggiungere, togliere, inserire e sostituire elementi

es. prova2 = {a[1],a[2],a[3]}
Append[prova2,a[4]] = {a[1],a[2],a[3],a[4]}
Prepend[prova2,a[0]] = {a[0],a[1],a[2],a[3]}
Insert[prova2,new,3] = {a[1],a[2],new,a[3]}
Delete[prova2,2] = {a[1],a[3]}
ReplacePart[prova2,new,2] = {a[1],new,a[3]}

Posizione/livello degli elementi

es. Position[prova, b] = { 1 }
Position[prova, x] = { {2,2,1} }
Position[prova, c] = {}
Level[prova, {2}] = $\{a, x^2\}$

Funzioni 1

Una generica trasformazione associa ad un oggetto iniziale, secondo una certa regola, un oggetto finale.
Con Mathematica possiamo realizzare una trasformazione con definizioni e/o con regole di sostituzione.
Nei linguaggi tradizionali specifichiamo gli argomenti della funzione ed associamo al simbolo della funzione una combinazione "aritmetica" degli argomenti (p.es. f[$x\_$] := x^ 2).
Con Mathematica il nome della funzione e il nome degli argomenti sono entrambi, dal punto di vista della manipolazione, espressioni. Possiamo sostituire il nome della funzione con un altro, allo stesso modo in cui possiamo sostituire il nome di una variabile con un valore assegnato.
In Mathematica possiamo definire funzioni che hanno come effetto trasformazioni del secondo tipo. Per esempio definiamo
f := Function[x, x/. Plus→ Times]
f[Plus[a,b,c]] = Times[a,b,c]

Funzioni 2

applicare una funzione

Scrivere f[x] genera una nuova espressione che leggiamo come la applicazione di f ad x.

Possiamo considerare ciò che appare tra parentesi come un indice che distingue vari valori di f. Poiché x è un simbolo, cioè una quantità statica, Mathematica non è in grado di valutare f[y] sostituendo y ovunque compaia la x.

Generare una lista di oggetti indicizzati *comando:*Table

Table[f[i],{i,1,3}] = {f[1],f[2],f[3]} = List[f[1],f[2],f[3]]

Applicare una funzione ad una lista di oggetti *comando:*Apply

Apply[Plus, List[a,b,c]] = Plus[a,b,c] = a+b+c

con forza bruta avremmo scritto List[a,b,c] /. List→Plus = a+b+c

Applicare una funzione agli oggetti di una lista *comando:*Map

Map[f, {a,b,c}] = {f[a],f[b],f[c] }

Map[f, g[a,b,c]] = g[f[a],f[b],f[c]]

livelli dell'espressione ed applicazione di una funzione

Abbiamo visto che la struttura di un'espressione può avere vari livelli. E' possibile applicare una funzione solo al livello desiderato di una espressione.

Apply[f, {a,{b,c}}, 2] = {a, f[b,c] }

Map[f, {a,{b,c}}, {2}] = {a, {f[b],f[c]} }

Map[f, {a,{b,c}}, 2] = {f[a], f[{f[b],f[c]}] }

Funzioni 3

funzioni pure

L'azione sugli argomenti di una funzione può essere introdotta senza necessariamente dare una definizione ed un nome alla funzione.

es. Function[x,1-x] = (1-#1) &

Possiamo utilizzare questa notazione p.es. all'interno di Map, oppure, se ribattezziamo questa espressione con un simbolo:

f = (1-#1) &

si sottointende che la notazione f[x] valuta la funzione f con argomento x.

E' possibile avere n argomenti indicati con #1,...,#n .

Manipolare una lista utilizzando una funzione pura

Map[Part[#,2] & , {{a,b,c},{d,e,f},{g,h,i}}] = {b,e,h }

Select[x^ 2 + 2 y z + z^ 3, FreeQ[#,x] &] = 2 y z + z^ 3

E' possibile scrivere funzioni che hanno una Head "indicizzata" anziché un simbolo (p.es. una famiglia di funzioni).

es. f[2][x], Derivative[n][f][x]

Funzioni 4

operazioni strutturali

ordinare una lista *comando:* Sort[lista,criterio]

es. Sort[{5,1,8,2}, (#2 < #1)&] = {8,5,2,1}

ridurre le sottostrutture di una espressione *comando:* Flatten

propietà associativa es. Flatten[f[a,f[b,c]]] = f[a,b,c]

proprietà distributiva e linearità rispetto ad un'operazione *comando:* Distribute

es. Distribute[f[Plus[a,b]]] = Plus[f[a], f[b]]
Distribute[f[g[a,b]]] = g[f[a],f[b]]

associare elementi corrispondenti di liste *comando:* Thread

es. Thread[$f[\{a_1, a_2, a_3\}, \{b_1, b_2, b_3\}]$] =

$= \{ f[\{a_1, b_1\}], f[\{a_2, b_2\}], f[\{a_3, b_3\}] \}$

commento

Abbiamo visto come si applica una funzione.

Non abbiamo ancora detto nulla riguardo alla valutazione di una funzione.

Finora abbiamo stabilito delle regole lessicali e formali per associare degli oggetti.

Patterns 1

Nella manipolazione di oggetti è fondamentale riconoscere che un'espressione ha una certa struttura (pattern).
Mathematica identifica due espressioni solo se hanno la stessa struttura lessicale.

es.form = (1+x_)ˆ 2
  MatchQ[(1+a)ˆ 2,form] = True
  MatchQ[1+2 a+aˆ 2,form]=False

_ ovvero Blank[] indica una qualunque espressione di Mathematica
x_ ovvero Pattern[x,Blank[]] indica una qualunque espressione che viene associata al simbolo x.
x_h ovvero Pattern[x,Blank[h]] indica che l'espressione ha Head h .
__ indica una o più espressioni di Mathematica.
___ indica nessuna, una o più espressioni di Mathematica.
x:pattern ovvero un'espressione con un dato pattern che viene chiamata x.
x..indica un pattern ripetuto una o più volte
x... indica un pattern ripetuto zero o più volte

Patterns 2

verificare un pattern

MatchQ[expr,pattern] restituisce True se expr soddisfa pattern e False altrimenti.
Svariate domande del tipo "l'espressione data soddisfa una certa proprietà?", che hanno come risposta sì /no, sono realizzate in funzioni che terminano con Q.

cercare elementi che verificano un pattern

Cases[{a,b,c},pattern] genera una lista di elementi che soddisfano pattern
Count[{a,b,c},pattern] conta quanto elementi che soddisfano pattern
Position[expr, pattern] genera una lista con la posizione degli elementi che soddisfano pattern
Select[{a,b,c}, criterio] estrae gli elementi che soddisfano la funzione criterio

ulteriori vincoli su un pattern

pattern /; condizione
Un pattern esprime una struttura di tipo "lessicale"
è possibile introdurre ulteriori restrizioni di tipo matematico e/o contenutistico formulate come (pattern /; condizione)
p.es. fac[n_] := n! /; n>0
Nel caso di un insieme discreto di alternative che definiscono il pattern si può utilizzare pattern = a1 | a2 | a3 | ....

Patterns 3

strutture associative e commutative

Attribute: Flat, Orderless, OneIdentity

un pattern viene riconosciuto tenendo conto anche delle caratteristiche delle funzioni che fanno parte del pattern

il fatto che l'addizione sia commutativa permette che il pattern g[x_+y_,x_] sia soddisfatto sia da g[a+b,a] che da g[a+b,b]

Una funzione commutativa è detta Orderless

SetAttributes[f,Orderless] ha come effetto che

f[b,a,c] e che f[a,b,c] siano considerate equivalenti.

Gli argomenti vengono riordinati secondo una forma canonica, che facilita il pattern matching.

L'attributo Flat indica che la funzione è associativa.

Regole di sostituzione e definizioni

regole di sostituzione

Una regola di sostituzione rimpiazza un oggetto con un altro all'interno di una espressione.

expr /. *lhs* → *rhs*

Scriviamo esplicitamente

Replace[expr, Rule[*lhs*, *rhs*]]

Una regola costituisce anch'essa un'espressione di Mathematica che può essere manipolata in tutti i modi permessi (p.es. possiamo scrivere liste di regole di sostituzione).

lhs può essere un'espressione specifica, oppure un pattern.

Replace agisce una sola volta su tutte le parti dell'espressione.

ReplaceRepeated, abbreviato in //. ,riapplica la regola fino a quando l'espressione rimane inalterata.

definizioni

Una definizione può essere vista come una regola di sostituzione applicata da Mathematica ogni qual volta sia possibile, permanentemente (p.es. x=3).

La variabile x è globale ed il suo valore noto in tutta la sessione di Mathematica, a meno che non lo cancelliamo con il comando Clear[x].

Definizioni 2

assegnare valori ad oggetti indicizzati

es. a[1] = 3 ; a[2] = r ; a[5] = 0; a[x] = c

tutti i valori definiti possono essere richiamati con il comando ?a

in questo esempio la x in a[x] si comporta come un indice, fisso, e non come una variabile.

definire una funzione

f[x_] = x^ 2 dice che qualunque volta l'argomento di f sia una espressione di Mathematica (il pattern), allora viene valutata ed è x^ 2.

Si noti che non c'è alcuna verifica che x sia un numero: p.es. f["ciao"] = Power["ciao",2] = ciao^2

un esempio

f[1] = 1 ; f[n_] := n f[n-1]   funzione fattoriale

- le definizioni particolari vengono provate prima di quelle generali
- ogni definizione viene applicata fino a quando il risultato rimane inalterato

Mathematica cerca sempre di applicare immediatamente ogni definizione nota, fino a quando il risultato non rimane inalterato.

Regole di sostituzione e definizioni 3

definizioni immediate e ritardate

Non sempre si vuole applicare subito una definizione .

$lhs = rhs$

rhs viene subito valutato e associato a lhs

$lhs := rhs$

rhs non viene subito valutato; solo quando lhs viene richiamato, allora rhs viene valutato con i valori correnti delle variabili.

es1 ex[x_] := Expand[(1+x)^ 2]
    iex[x_] = Expand[(1+x)^ 2]

es2 x := Random[]
    ix = Random[]

Si definisce una regola di sostituzione ritardata
RuleDelayed[*lhs* , *rhs*] che viene abbreviata in :>
in cui *rhs* viene valutata solo al momento in cui la regola viene applicata all'espressione.

Regole di sostituzione e definizioni 4

down e upvalues

Le definizioni viste finora sono chiamate downvalues
poichè collegano direttamente la funzione che viene definita con l'espressione ad essa associata.
Si chiamano upvalues delle definizioni in cui si stabilisce una relazione tra una funzione della funzione che vogliamo definire è l'espressione ad essa associata.

f[args] :=	*rhs*	downvalue per f
g[f[args],...] ^:=	*rhs*	upvalue per f

esempio: quat[x_] + quat[y_] ^ := quat[x+y]
In questo caso la funzione g è data da Plus, e definiamo un upvalue per quat.
Come nella programmazione object-oriented, stiamo definendo come sommare due oggetti quat.
Anziché definire un operatore quatplus, si aggiunge alle regole del Plus standard questa riguardante la somma di due quat.

Intermezzo

qualunque oggetto è una espressione
di un'espressione possiamo manipolare qualunque parte
possiamo formalmente rappresentare applicazioni/funzioni e strutture
possiamo sostituire puntualmente o permanentemente oggetti

quando programmiamo vogliamo sapere:
in che ordine vengono effettuate le operazioni;
cosa è globale e cosa è locale.

Valutazione delle espressioni 1

oggetti di tipo diverso

→ sono tutti espressioni di Mathematica
→ subiscono una comune procedura di valutazione

Mathematica è un sistema di valutazione infinito

è importante sapere, in questo continuo processo di valutazione, quali operazioni hanno la precedenza sulle altre

per confrontare due espressioni è necessario ridurle ad una forma standard, sulla base della quale si decide se sono uguali.

riduzione delle espressioni ad una forma standard

esistono espressioni (p.es. i polinomi) per cui non esiste una sola forma standard (fattorizzati/espansi).

gli Attributes delle funzioni (Flat, Orderless, Listable,OneIdentity, Constant, Protected, Locked, HoldFirst, HoldAll, Temporary) permettono di velocizzare le procedure di pattern-matching.

Esistono una procedura standard ed una non-standard di valutazione, a seconda del tipo di espressione considerata.

Valutazione delle espressioni 2

procedura standard

valuta la Head

valuta ogni elemento dell'espressione

applica le trasformazioni legate agli Attributes

applica le definizioni date nella sessione

applica le funzioni intrinseche

valuta il risultato

le definizioni particolari hanno precedenza su quelle generali

le definizioni con upvalues hanno precedenza su quelle con downvalues

procedura non standard

non tutti gli elementi vengono valutati nell'ordine standard

x=y	la x non viene mai valutata
If[p,a,b]	valuta solo o a oppure b, a seconda che p sia True o False
Do[expr,{n}]	valuta expr n volte
Plot[f,{x,...}]	valuta f per un set di valori numerici di x
Function[{x},body]	viene valutata solo quando applicata

Mathematica non valuta gli elementi nel caso una funzione abbia Attributes della classe Hold

al contrario, per funzioni i cui elementi non sono di default valutati, è possibile forzare la valutazione con la funzione Evaluate

esempio: Plot[Evaluate[Table[Sin[n x],{n,1,3}]],{x,0,2 Pi}]

Valutazione delle espressioni 3

- nei patterns le singole parti vengono valutate, prima di essere confrontate con l'espressione in esame; per bloccare la valutazione delle parti del pattern, usare Literal[patt]
- nelle regole di sostituzione vengono valutate subito sia *lhs* che *rhs* nelle regole ritardate, viene valutata solo *lhs*
- nelle definizioni *symbol* = *value* viene valutato solo *value*; nelle definizioni ritardate non viene valutato nè l'uno nè l'altro.
- in Table, Sum, Do, Plot (iterazioni) l'espressione viene valutata solo dopo che l'indice ha assunto un valore definito
- nelle strutture condizionali (If oppure /;) viene valutata la condizione e, sulla base di questa, una delle espressioni da essa controllate

operatori logici

And	$expr_1$ && $expr_2$ && $expr_3$ && ...	valuta finchè trova False
Or	$expr_1$ || $expr_2$ || $expr_3$ || ...	valuta finchè trova True
SameQ	a === b	True solo se a e b identici
UnsameQ	a =!= b	True solo se a e b diversi

cicli e strutture di controllo

Do[expr,iterator]	Break[]
For[start, test, incr, body]	Continue[]
While[test,body]	Return[value]

Valutazione delle espressioni 4

- $RecursionLimit stabilisce il numero massimo di ricorsioni (massima profondità) accettate, senza che la procedura di valutazione raggiunga un risultato.
- $IterationLimit stabilisce la lunghezza massima di una procedura di valutazione che a causa della sua struttura (p.es. circolare) non raggiunge un risultato.
- Interrupt[] interrompe un calcolo e restituisce il prompt, permettendo di iniziare una sessione di Dialog[] o di attivare Trace
- Abort[] termina il calcolo in corso.

Modularità ovvero del nome delle cose

Moduli e variabili locali

Tutti i simboli introdotti nella sessione di Mathematica sono globali.

Spesso, per chiarezza e sicurezza della programmazione, ci servono delle variabili locali.

Il modulo permette di definire una serie di operazioni che vengono effettuate con variabili locali.

Se anche utilizziamo lo stesso nome per una variabile della sessione globale e per una variabile temporanea all'interno del modulo, Mathematica mantiene questi due oggetti distinti.

Module[var1,var2,..., body]

I valori delle variabili globali, i cui nomi siano diversi da quelli della lista locale, sono accessibili anche dall'interno di Module.

Module può essere eseguito interattivamente, ma più spesso serve nella programmazione per definire delle funzioni o delle vere e proprie subroutines.

Di un Module vengono valutati in primis i valori iniziali delle variabili locali, se assegnati. Il body del Module resta invece non valutato (attributo HoldAll), a meno che non si forzi il programma.

tmp = 1+x;

Module[x=4, tmp] = tmp

Module[x=4, Evaluate[tmp]] = 5

Costanti locali

Nella programmazione in Mathematica c'è il rischio concreto di utilizzare una variabile globale, già definita in precedenza con un certo valore.
Eṗossibile eseguire una porzione di programma introducendo una variabile locale il cui valore viene assegnato una volta per tutte come condizione iniziale.
With[{x=x_0}, body]
Si tratta di un modo veloce per mantenere la programmazione trasparente. With è in concreto una specializzazione di Module. Il suo utilizzo è analogo ad una regola di sostituzione.

Nomi delle variabili

Per mantenere tutte le variabili locali distinte da quelle globali, Mathematica introduce un nuovo simbolo per ogni variabile che compare nelle liste, ogni volta che un Module oppure un With vengono eseguiti.
La generazione di simboli può anche essere effettuata esplicitamente con il comando Unique[x] = x$nnn. E' buona norma lasciare che questi simboli vengano gestiti da Mathematica.

Mathematica ha degli “scoping constructs”, ovvero delle strutture con variabili locali:

Function	parametri
Rule	pattern
Set	pattern
With	costante
Module	variabile

Variabili mute

esempio di utilizzo di Module
p[n_] := Integrate[s^ n, {s,0,1}]
q[n_] := Module[s, Integrate[s^ n, {s,0,1}]

Calcolo correttamente q[s] ma non p[s].

Blocchi e valori locali

E‘ talvolta utile, nella programmazione interattiva, utilizzare un oggetto che appartiene alla sessione globale di Mathematica, assegnandogli temporaneamente un valore per prova. Block[{x=x_0}, body] utilizza la variabile x globale, e le assegna il valore temporaneo x_0, con cui viene valutato body.

Nel caso di Module viene attuato l’usuale “lexical scoping” che rende locale una variabile ovvero fisicamente accessibile ad una porzione del codice. Nel caso di Block viene realizzato il cosiddetto “dynamical scoping”, che rende locale il valore di una variabile in un intervallo della history della sessione.

Contesti

Module, With e Block possono essere utilizzati sia interattivamente che nella programmazione, per rendere parte delle operazioni locali.

Il problema di utilizzare variabili distinte e con nomi sufficientemente corti e maneggevoli è risolto introducendo i Context. Ogni variabile viene scritta come (contesto‘nome). Un contesto è cioè un prefisso che accomuna un gruppo di variabili legate tra loro. La struttura di contesto e nome breve della variabile è analoga alla relazione che c’è tra nome della directory e nome dei files. E‘ possibile in particolare creare strutture ramificate di contesti, del tipo c1‘c2‘c3‘nome.

Variabili con nome breve uguale, ma appartenenti a contesti diversi, sono fisicamente distinte.

$Context indica il contesto attuale

$ContextPath indica la lista dei contesti tra cui un nome breve verrà cercato.

Una variabile è identificata univocamente quando viene richiamata con tutte le sue estensioni. Nel caso contrario Mathematica provvede a cercarla nel $ContextPath. Se una definizione non viene trovata la variabile viene definita in Global‘ .

Packages

Stile di programmazione modulare
un package svolge una serie di operazioni complicate
comunica con l'esterno tramite una funzione
mantiene private, inaccessibili all'esterno, le operazioni interne.

In Mathematica il package viene definito in un suo Context c_package` che viene inserito nel $ContextPath.
Tutte le funzioni che devono restare private vengono definite in un Context c_package`Private
Quest'ultimo non viene aggiunto al $ContextPath e quindi le funzioni interne non possono essere richiamate dalla sessione di Mathematica.

MathLink

In certi casi risulta molto conveniente eseguire, durante la sessione di Mathematica, un programma esterno, scritto p.es. in C, il cui risultato viene quindi restituito a Mathematica per una successiva elaborazione.
MathLink permette di stabilire dei canali di comunicazione tra Mathematica e dei programmi esterni, oppure tra diverse sessioni di Mathematica.
Preparazione:
mcc -o f.exe f.tm f.c
f.tm è un template: contiene le dichiarazioni che spiegano la struttura delle funzioni scritte in C cui Mathematica ha accesso.
f.c è un programma in C, che interfaccia le funzioni in C vere e proprie con Mathematica. La sua struttura è del tipo

```
#include "mathlink.h"
type f( .......) {
qui richiama eventualmente altre funzioni che fanno il lavoro vero e proprio
MLPuttype(.....)
}

int main(int argc, char *argv[]) {
return MLMain(argc, argv);
}
```

Mathematica importa l'eseguibile f.exe e utilizza le nuove funzioni direttamente.
E' anche possibile fare partire un programma esterno (p.es. FORM), comunicargli dei dati, fargli eseguire delle operazioni, acquisire in Mathematica il risultato.

L'idea di base delle funzioni MLPuttype (MLPutInteger, MLPutReal, MLPutString, MLPutSymbol, MLPutFunction) è quella di prendere il risultato del programma in C, il cui tipo è in generale uno di quelli elementari, e restituirlo a Mathematica secondo le sue convenzioni. Poichè la struttura tipica dell'espressione di Mathematica è
Head[args]
MLPutFunction serve a restituire la Head, oltre al numero di argomenti che seguono
MLPuttype restituisce quindi i vari argomenti.