

Block Notes Matematico

Analisi numeriche e simulazioni

ing. Rosario Turco

Abstract

In questo articolo l'autore si sofferma su argomenti di analisi numerica, evidenziando aspetti algoritmici e di simulazione, mostrando come la matematica moderna sta evolvendo da pochissimi anni, grazie a questi due nuovi aspetti, a vantaggio del ragionamento logico delle nuove generazioni.

L'eterno ed amletico dilemma della didattica

La didattica ed i testi di matematica, in generale, nell'ultimo secolo hanno privilegiato molti schemi preconfezionati, esaltando taluni aspetti che potremmo definire del "non reinventare la ruota" ma sviluppando scarsamente altri. Scuramente è un argomento delicato, non semplice, il decidere cosa sia meglio fare in ambito didattico; ma certamente è importante fare considerazioni utili.

Ad esempio l'applicazione di formule circa le equazioni, che mettono in relazione la soluzione con i coefficienti, è tutto sommato per i giovani solo una forma di felice di "corner sintattico", un meta-risultato in cui si rifugiano ben volentieri, svuotando del tutto i percorsi semantici e le impalcature su cui le stesse formule sono nate e che potrebbero non solo appassionare alla materia i molti, ma anche offrire utili spunti a quei pochi che un giorno potrebbero riuscire a estrarre il talento sepolto, di "evangelistica memoria". L'altro errore indotto è che l'iniziato potrebbe erroneamente supporre che esista una formula, che sia soluzione per ogni tipo di equazione.

In realtà il matematico è innanzitutto un "ricercatore di schemi e di algoritmi", fa congetture che sperimenta, con strumenti automatici a supporto, fino alla formulazione del Teorema.

In realtà molte equazioni sono risolvibili con un "algoritmo per tentativi" o con un algoritmico non deterministico (intendendo senza una formula). Il valore di questo tipo di percorso è la realizzazione di una forma mentis capace di trovare soluzioni per qualsiasi tipo di equazione e problema.

Algoritmi, simulazione, logica

Il concetto di algoritmo risale agli Arabi ed ancor prima ai greci: già la geometria di Euclide è in realtà una esposizione indiretta di algoritmi, dalla costruzione del triangolo equilatero (Libro I, prop. 1), al celebre algoritmo per il calcolo del MCD (Libro VII, prop. 2), alla costruzione dell'icosaedro e del dodecaedro regolari (Libro XIII, prop. 16 e 17).

Nella storia della matematica ci sono poi state altre pietre miliari: l'algoritmo di Newton per risolvere le equazioni, l'algoritmo di Eulero per risolvere equazioni differenziali, l'algoritmo di Gauss etc.

L'idea della simulazione in matematica è più giovane e recente; coi teoremi di incompletezza di Gödel (1931) ci siamo un po' abituati all'idea, ancora estranea a Hilbert, che "certi problemi" non si possano risolvere. In altri termini la matematica, in certi settori, non è ancora del tutto esaustiva o tale da portarci a facili soluzioni. Nel 1970 il matematico russo Yuri Matijasevich risolse il decimo problema di Hilbert, formulato al Congresso di Parigi nel 1900: "determinare un algoritmo per stabilire se una equazione polinomiale in più incognite a coefficienti interi (equazione diofantea) ammetta soluzioni intere". La risposta di Matijasevich fu molto semplice: tale algoritmo non esiste. Non esiste un procedimento meccanico che, applicato ad una qualsiasi equazione diofantea, in un numero finito di passi possa decidere se questa ammette soluzioni intere.

Esaminiamo, nel seguito, come attraverso l'analisi numerica, gli algoritmi e la simulazione si può arrivare alla ricerca di soluzioni.

Numeri decimali con molte cifre

Poniamoci il seguente problema: Qual è la 29-esima cifra di 29^{29} ? Prima risposta: boh! Seconda: mi rimbocco le maniche e vediamo se con la logica ci arriviamo.

Essendo un numero decimale il suo valore deve essere compreso tra:

$$10^a \leq N \leq 10^{(a+1)}$$

Da cui il numero di cifre è l'intero più piccolo maggiore di $\log_{10} N$, ovvero mi darà $a+1$. Con PARI/GP abbiamo due funzioni **ceil** (*tetto*) e **floor** (*pavimento*). In questo caso dobbiamo arrivare al tetto!

In PARI/GP non c'è il log in base 10 ma quello in base naturale, per cui il numero di cifre nc ce lo ricaviamo come:

$$nc = \text{ceil}(\log(N)/\log(10))$$

Troviamo che $nc = 43$.

Come trovo la 29-esima cifra? Devo dividere 29^{29} per $10^{(43-29)}=10^{14}$ e prenderne il "pavimento" (floor). Perché? Nella numerazione decimale ogni cifra ha peso 10^i con $i=0, \dots, nc$.

```
? floor(29^29/10^14)
```

```
%6 = 25676861531612111345618282147
```

Il numero che cerchiamo è il 7. Siamo stati fortunati perché lo vediamo a video, ma avremmo potuto ottenerlo anche nel seguente modo:

```
? floor(29^29/10^14) % 10
```

```
%8 = 7
```

I sistemi dinamici discreti

Supponiamo il problema: Un bosco in montagna ha inizialmente 1000 alberi, è viene utilizzato per ricavare legname. Ogni anno viene tagliato il 20% degli alberi e contemporaneamente vengono piantati 100 nuovi alberi. Qual sarà l'evoluzione del bosco? Si estingue, cresce o si stabilizza?

Il problema è descrivibile con una sequenza:

$$a_0 = 1000 \quad (\text{la condizione iniziale})$$

$$a_{n+1} = 0,8 * a_n + 100 \quad (\text{evoluzione})$$

A questo punto dobbiamo ciclicamente calcolarci i valori. Basta creare un programma con PARI/GP del tipo:

```
SD(a0, b0, perc, c=10)= local(i=0); {  
  
an = a0;  
for(i=1,c,  
  an = an * perc + b0;  
);  
  
return(an);  
}
```

Ad esempio facciamo:

$$SD(1000,100,0.8,1)$$

Se al ciclo c (l'ultimo termine) diamo valori 1, 2, 10, 20, 40, 80, 100, 200, 500, 1000; si scopre che la sequenza prima decresce dal valore 900 poi converge a 500 alberi già dopo 30 anni. Per cui si stabilizza. Il punto 500 è un punto di equilibrio tra la situazione della diminuzione del 20% di alberi e l'aumento di 100.

Tutto ciò è anche soluzione delle semplici equazioni:

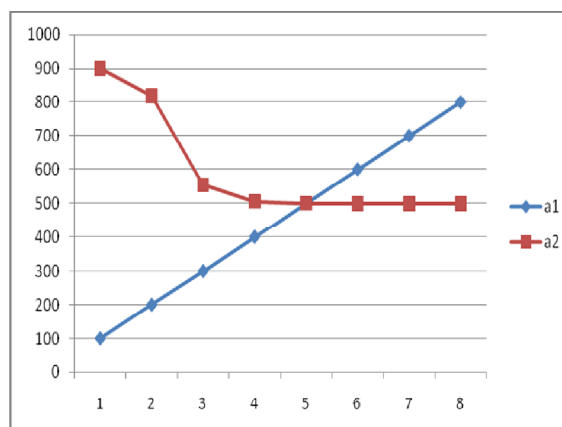
$$y = 0,8 x + 100$$

$$y = x$$

La seconda equazione è la funzione identità I: $x \rightarrow x$.

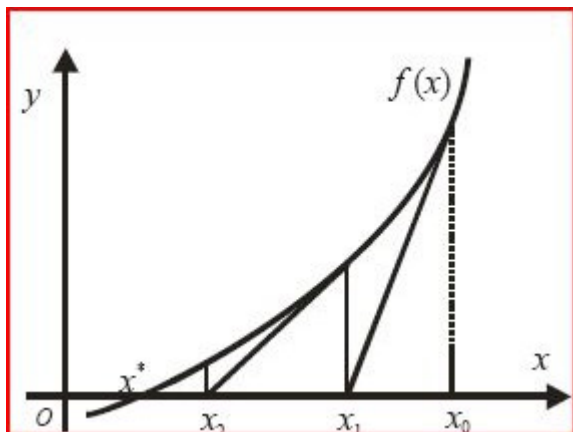
Se avessimo risolto banalmente le equazioni si sarebbe perso il concetto di convergenza della successione e della dinamicità del sistema.

C'è ovviamente la necessità di uno strumento a supporto per mostrare rapidamente i calcoli e la convergenza. D'altra parte anche un grafico è di necessaria utilità alla comprensione dove le due equazioni trovano il punto di equilibrio.



L'algoritmo di Newton

L'algoritmo di Newton permette di trovare le radici di una equazione di una funzione $f(x)$ derivabile. Una soluzione di $f(x)=0$ è un valore x^* , detto *radice o zero*, tale che $f(x^*)=0$, quindi dove $y = f(x)$ interseca l'asse delle x .



Se in un punto x_0 , tracciamo la tangente a $f(x)$, questa tangente intercetta l'asse delle x in x_1 . Poi tracciamo la verticale da x_1 a $f(x)$ otteniamo un punto sulla $f(x)$ dove tracciare una nuova tangente che intercetta l'asse delle x in x_2 etc. Proseguendo in questo modo abbiamo un procedimento di convergenza verso x^* , cioè quel punto dove $f(x)$ si annulla.

La retta tangente a $f(x)$ in x_0 ha equazione:

$$y = f'(x_0)(x - x_0) + f(x_0)$$

Ed intercetta l'asse delle x in: $x = x_0 - f(x_0)/f'(x_0)$. La derivata della funzione ricordiamo che coincide col concetto del coefficiente angolare o pendenza della retta tangente.

Allora possiamo generalizzare il tutto con una legge ricorsiva:

$$x_{t+1} = x_t - f(x_t)/f'(x_t) \quad (A)$$

Esempio

$$f(x) = x^3 + x - 1 = 0$$

Questa equazione ha sicuramente delle radici nell'intervallo $[0, 1]$ perché $f(0) = -1$ e $f(1) = 1$, quindi la funzione è crescente ed intercetta l'asse delle ascisse.

La derivata prima è $f'(x) = 3x^2 + 1$. Assumiamo come condizione iniziale $x_0 = 1/2$ in $[0, 1]$. In base alla (A) otterremo la successione di valori: $1/2, 5/7, 593/868, \dots$

Anche qui possiamo scrivere un piccolo programma in PARI/GP:

```
f(x)=local(); {
o=x^3+x-1;
return(o);
}
```

Lo *snippet* che presentiamo va ovviamente modificato in $f(x)$ e $f1(x)$ se la funzione considerata è diversa. Nel pezzo di codice $f1(x)$ rappresenta la derivata.

```
f1(x)=local(); {
o=3*x^2+1;
return(o);
}
```

L'algoritmo si può migliorare anche introducendo in input a Newt l'errore che si vuole ottenere: $|x_{t+1} - x_t| < \epsilon$.

```
Newt(x0, c=10) = local(i=0); {
x = x0;
if( c > 1,
for(i=2,c,
a = f(x)/f1(x);
x = x - a;
);
);
};
```

```
return(x);
}
```

Formula di Erone e la radice quadrata

Se l'equazione da risolvere è $x^2 - N = 0$, con $N > 0$, l'algoritmo di Newton trova per successive approssimazioni la radice quadrata di un numero N e, quindi, simula la formula di Erone. Difatti la formula di Newton se la sviluppiamo ci porta alla formula di Erone:

$$x_{t+1} = x_t - \frac{x_t^2 - N}{2x_t} = \frac{x_t^2 + N}{2x_t} = \frac{1}{2} \left(x_t + \frac{N}{x_t} \right) \quad (B)$$

Con l'algoritmo di Newton possiamo trovare ad esempio la $\sqrt{2}$ che è compresa tra 1 e 2. Ovviamente dobbiamo modificare $f(x)$ e $f1(x)$ come nello snippet successivo e partire da un valore iniziale come $3/2$ o 1.5 .

```
f(x,N)=local(); {
o=x^2-N;
return(o);
}
```

Con Newt(3/2,2,3) si ottiene subito 577/408 che già è una buona approssimazione a 5 cifre.

```
f1(x)=local(); {
o=2*x;
return(o);
}
```

```
Newt(x0, N, c=10) = local(i=0); {
```

```
x = x0;
```

```
if( c > 1,
for(i=2,c,
a = f(x,N)/f1(x);
x = x - a;
);
);
return(x);
```

```
}
```

Radice n-esima

In maniera analoga si può determinare la radice n-esima se l'equazione è $x^n - N = 0$. Anche qui se vogliamo simulare il risultato dobbiamo modificare $f(x,N)$ e $f1(x)$, tenendo conto della nuova equazione e della nuova derivata; $f(x) = x^n - N$ mentre $f1(x) = n * x^{n-1}$; per cui è:

$$x_{t+1} = x_t - \frac{(x_t^n - N)}{n x_t^{n-1}} = \frac{(n-1)x_t^n + N}{n x_t^{n-1}} \quad (C)$$

Per $n=3$ (radice cubica) la (C) diventa: $\frac{2x_t^3 + N}{3x_t^2}$

```
f(x, n, N)=local(); {
o=x^n-N;
return(o);
}
```

Calcoliamo ad esempio $\sqrt[3]{3}$.

```
f1(x,n)=local(); {
o=n*x^(n-1);
return(o);
}
```

Se si parte con $x_0 = 3/2$ o 1.5 dopo 3 iterazioni abbiamo già una buona approssimazione anche a dieci cifre decimali.

PARI/GP vi permette anche di lavorare in modo simbolico richiamando la funzione come:

```
Newt(x0, n, N, c=10) = local(i=0); {
```

Newt(3/2,3,3,3)

```
x = x0;
```

Ovviamente si potrebbe anche usare la formula (C).

```

if( c > 1,
  for(i=2,c,
    a = f(x, n, N)/f1(x,n);
    x = x - a;
  );
);
return(x);
}

```

Equazioni logaritmiche

L'equazione $2^x = 10$ sappiamo risolverla simbolicamente in $x = \log_2 10$. Perché l'algoritmo di Newton non va bene? In fondo l'equazione è: $2^x - 10 = 0$.

Se approfondiamo un attimo l'algoritmo di Newton conduce ad una successione del tipo:

$$x_{t+1} = x_t - 2^x - 10 / \log(2)2^{x_t} \quad (D)$$

La (D) costringe sempre a calcolare il $\log(2)$. Tanto vale risolvere $x = \log_2 10$. Il logaritmo in una data base non è un problema, lo abbiamo visto anche in qualche esempio precedente.

Se vogliamo calcolare ad esempio 1789, senza calcolatrice allora dobbiamo logicamente pensare che:

$$10^3 < 1789 < 10^4$$

Con i logaritmi in base 10 è:

$$3 < \log(1789) < 4$$

Allora $\log(1789)=3,....$

Come facciamo a stabilire dopo la virgola che viene? Dobbiamo fare delle considerazioni semplici. Sappiamo ad esempio che $\log(1789)=3.x$ dove x è la cifra decimale che non conosciamo ancora; ma anche $\log(1.789)=0.x$ avrebbe la stessa cifra decimale.

Ora è:

$\log(1.789)=0.x$ ma la posso riscrivere anche come $10 \cdot \log(1.789) = x$, il che equivale anche a $\log(1.789^{10}) = x$

Si osserva subito che $10^2 < 1.789^{10} < 10^3$ allora è: $2 < \log(1.789^{10}) < 3$, per cui $x=2$. Si può proseguire così anche con altre cifre decimali.

Se dobbiamo scrivere un algoritmo è preferibile però sfruttare gli shift di bit, nativi e performanti, di un linguaggio (è più veloce) che lavora sui numeri binari (anche PARI/GP).

Sappiamo che passare da una base ad un'altra si può fare in questo modo: $\log_b(x) = \log_2(x) / \log_2(b)$

Ad esempio $\log_e(10) = \log_2(10) / \log_2(e)$

Ora un algoritmo semplice per il logaritmo in base 2 che sfrutta gli shift di un intero (visto come binario) è:

```

/* Funzione per calcolare il logaritmo in base 2 di un intero */
int log2(int N){
int k=N, i=0;
while(k){

```

```

k>=1; // shift di bit
i++;
}
return i-1;
}

```

Ad esempio se abbiamo $n=1$ in binario è 001; in un binario ogni cifra ha peso potenza di 2. Ad esempio $001 = 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 = 1$ se facciamo lo shift a sinistra $001 \ll$ per 2 volte significa che il binario diventa 100 in realtà abbiamo moltiplicato per potenze di 2.

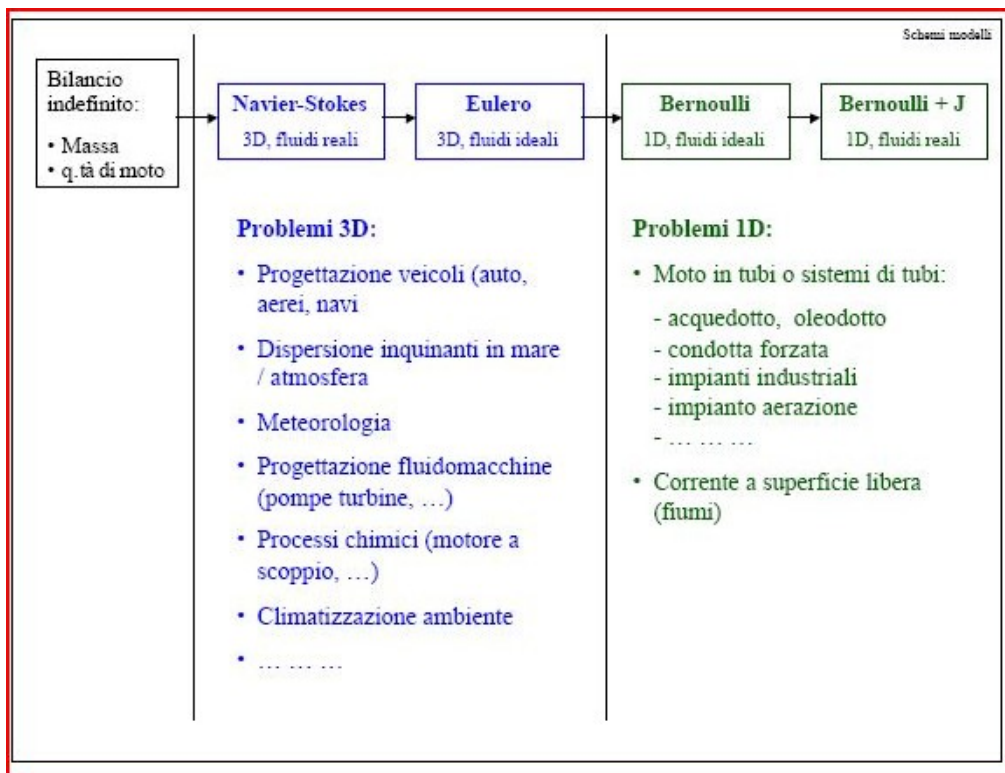
$$100 = 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 = 4$$

Se il 4 in binario lo shiftiamo a destra $100 \gg$ otteniamo che lo dividiamo per potenze di 2, ma in pratica stiamo cercando l'esponente a cui elevare la base 2 del potenza che mi ritorni 4 ovvero stiamo cercando il log in base 2 e il numero di shift che devo fare è il valore del log in base 2 ($\log_2(4)=2$).

Però dovendo cercare un criterio per arrestare l'algoritmo, si shifta a destra il numero (con PARI/GP è $\text{shift}(n,-1)$) finché non diventa 0. In tal caso il numero di shift è di una unità superiore al valore del log in base 2.

Problema del Millennio: Le equazioni di Navier - Stokes

L'uomo è ancora alla rincorsa di soluzioni e schemi preconfezionati, anche in questo problema del Millennio. Non si sa se esiste una soluzione, né se è possibile dimostrarlo. Siamo di fronte ad un analogo problema di Hilbert e aspettiamo un'altra dimostrazione negativa? Nel frattempo in fluidodinamica, l'analisi dei fluidi turbolenti e viscosi, vengono comunque fatte simulazioni per trovare delle soluzioni: gallerie del vento e modelli sono usati per la progettazione di profili aereo-dinamici. La pratica ha avuto il sopravvento sulla teoria o è possibile una soluzione?



This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.