ALGORITMI E SIMULAZIONI

Michele Impedovo
michele.impedovo@unibocconi.it
IMQ Università Bocconi Milano

Una modesta proposta, in stile swiftiano: aboliamo i programmi e le materie. O lasciamo solo una traccia tenue di obiettivi generici, che costringa gli insegnanti a inventare come riempire il tempo che è dato loro, a far appassionare i ragazzi con vere sfide della mente, che facciano conoscere le imprese intellettuali della storia e rafforzino le loro capacità logiche.
Gabriele Lolli

Perché algoritmi e simulazioni?

Se l'insegnamento della matematica può essere alleggerito nelle parti di calcolo più faticose e noiose allora è possibile introdurre sistematicamente nell'insegnamento secondario (o addirittura primario) uno strumento concettuale forte, che è sempre stato un formidabile motore di sviluppo nella storia della matematica: l'algoritmo, inteso come una funzione che prende in ingresso *qualcosa* e fornisce in uscita *qualcosa*.
Se ci pensiamo bene, l'idea di algoritmo potrebbe addirittura essere presa a paradigma (uno dei tanti paradigmi, beninteso) dell'intera matematica: già la geometria di Euclide è una continua esposizione di algoritmi, dalla costruzione del triangolo equilatero (Libro I, prop. I), 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). E si prosegue nella storia della matematica attraverso algoritmi come pietre miliari: l'algoritmo di Newton per risolvere equazioni, l'algoritmo di Eulero per risolvere equazioni differenziali, …
L'insegnamento ha privilegiato, nell'ultimo secolo, quella matematica che, nei contenuti e nei metodi, potesse ricondursi a schemi preconfezionati, esaltando l'aspetto logico-formale della matematica ma mortificando la fantasia e lo spirito critico degli allievi. In un certo senso la *applicazione di formule* a cui purtroppo molta matematica viene ricondotta (rifugio sintattico per una gran parte di studenti) è ciò che resta, è la brutta copia della ricerca di algoritmi per la soluzione di un problema.
Un ben noto esempio è costituito dalla risoluzione di equazioni: già la *formula* per le equazioni di 2° grado introduce una distorsione, un equivoco, un'imprecisione.
La distorsione è quella per cui si tratta di un risultato estremamente avanzato, in un certo senso un *meta-risultato*: poter esprimere le soluzioni dell'equazione in funzione dei coefficienti svuota di significato la ricerca delle soluzioni stesse, costringe ad abusare della sintassi prima di aver compreso la semantica (e quindi a rifugiarsi nella formula anche quando, in casi semplici, l'equazione è facilmente risolubile). Inoltre induce a pensare, ahimè, che per ogni equazione esista una formula analoga.
L'equivoco è quello di lasciar credere agli allievi che una volta trovato un risultato simbolico come

$$\frac{3\pm\sqrt{7}}{2},$$

la nostra conoscenza sia aumentata. Non parlo qui degli studenti (e sono tanti) che non riconoscono in quella scrittura i due numeri distinti

$$\frac{3-\sqrt{7}}{2} \text{ e } \frac{3+\sqrt{7}}{2}$$

(sono gli stessi che risolvono la disequazione $x^2>1$ scrivendo $x>\pm 1$) bensì di coloro (e sono quasi tutti) che non si interrogano, poiché il contratto didattico non lo prevede, su quanto valgono circa quei numeri, dove si trovano sull'asse reale, come mai sono simmetrici rispetto a 1.5, e così via.
L'imprecisione è quella di chiamare *soluzione esatta* una scrittura che semplicemente rimanda ad un'altra equazione, $x^2-7=0$, perciò è auto-referenziale, tautologica, *simbolica*, ma non esatta.
Nello stesso modo se insegniamo a risolvere l'equazione $2^x=10$ fornendo la tautologica scrittura

$$x=\log_2(10)$$

e ci fermiamo qui, diamo davvero un'idea sbagliata di che cosa sia la matematica.
Occorrono algoritmi per approssimare numeri.
Il più semplice algoritmo è quello che chiamiamo (a volte con malcelato disprezzo) *per tentativi*. Calcolare per tentativi (cioè mediante le successive moltiplicazioni 2.1^2, 2.2^2, …) la prima cifra decimale di $\sqrt{7}$ è un esercizio che si può proporre in età scolare precoce ed è ricco di implicazioni e di conoscenze matematiche. Anche l'equazione $2^x = 10$ si risolve prima di tutto per tentativi.

Mi capita di dare, a studenti del I anno di un corso di laurea in economia, un test di ingresso; il quesito

Come si risolve un'equazione come $x^7+x=1$?

propone le seguenti risposte.

a) Applicando la formula risolutiva $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$
b) Con la formula di Ruffini
c) Non si può risolvere
d) Raccogliendo la *x*
e) Per tentativi

Nessuno sceglie la risposta *e*) che pure appare l'unica sensata e risulta essere l'unica praticabile. Tutte le altre risposte (soprattutto la *c*) ricevono consensi inspiegabili. Il fatto è che se ci limitiamo a relazioni simboliche, tutto si può "risolvere" in qualche modo; anche l'equazione polinomiale di 5° grado (per esempio ricorrendo alla funzione θ di Jacobi) può essere espressa in funzione dei coefficienti. Possiamo sempre inventare un simbolo nuovo come risposta ad ogni problema. Ma che cosa ne sappiamo di più del mondo?
Occorrono algoritmi, anche per approssimare numeri. Vedremo algoritmi per approssimare le radici di qualunque ordine e i logaritmi di qualunque base.

Se il concetto di algoritmo si perde nella notte dei tempi, l'idea di *simulazione* è più recente. In un certo senso, dai teoremi di incompletezza di Gödel (1931) ci siamo abituati all'idea, totalmente estranea a Hilbert, che certi problemi *non si possano risolvere*. Hilbert apre il secondo Congresso Internazionale di Matematica a Parigi, nell'agosto del 1900, con le seguenti parole:

> *Sentiamo in noi l'eterno richiamo: ecco il problema, cercane la soluzione. È possibile ottenerla facendo appello solo alla ragione, dal momento che in matematica non esiste "ignorabimus".*

E presenta un elenco di 23 problemi allora non risolti come sfida per la matematica negli anni successivi.
Nel 1970 il matematico russo Yuri Matijasevich (a ventidue anni!) risolve il decimo problema di Hilbert: determinare un algoritmo per stabilire se una equazione polinomiale in più incognite a coefficienti interi (*equazione diofantea*) ammetta soluzioni intere. La risposta di Matijasevich è 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. Hilbert era già morto: non lo avrebbe trovato carino.

Oggi siamo più ragionevolmente rassegnati (o propensi) ad aspettarci problemi la cui soluzione sia impossibile in termini simbolici elementari (intendo esprimibile mediante una funzione elementare), oppure semplicemente molto, molto complessa, anche quando la formulazione è relativamente semplice; si pensi ai frattali, agli innumerevoli campi di ricerca aperti dai sistemi dinamici caotici e dalla simulazione deterministica del caos. Uno dei più affascinanti problemi non risolti degli ultimi decenni è la cosiddetta *congettura di Collatz*: sia data la successione ricorsiva

$a_0 \in \mathbf{N}$

$$a_{n+1} := \begin{cases} n/2 & n \text{ pari} \\ 3n+1 & n \text{ dispari} \end{cases}$$

Per esempio, con $a_0 := 7$ si ottiene

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, …

Quando la successione assume il valore 1 perde di interesse, perché da quel punto assume ciclicamente i valori 4, 2, 1.

La congettura è la seguente:

Qualunque sia il numero naturale a_0 esiste sempre k tale che $a_k = 1$.

La proprietà è vera (è stata ovviamente testata su calcolatore) per tutti i numeri fino a 10^{12}, ma nessuno ne ha mai fornito una dimostrazione.

Paul Erdos ha addirittura dichiarato che "La matematica non è ancora pronta per questi problemi".

Mettiamo insieme tutti questi elementi:

- La nuova consapevolezza che molti problemi sono insolubili o molto difficili o la soluzione non è esprimibile in termini simbolici elementari; per esempio, la somma della serie $\sum_{k=1}^{\infty} \frac{1}{k^3}$, o la soluzione reale di $x+2^x = 0$.
- Lo sviluppo di strumenti automatici di calcolo e di programmazione sempre più potenti.
- La innata pigrizia dell'uomo (e degli adolescenti!) e la conseguente tendenza ad automatizzare operazioni di routine.
- La predisposizione cognitiva degli adolescenti di oggi, cresciuti tra telecomandi e telefonini, ai paradigmi dell'algoritmo.
- La richiesta che la scuola si occupi sempre più di statistica e di probabilità, più in generale di trattamento e analisi di dati.
- Il lento (ma inesorabile?) sovrapporsi del discreto al continuo, del finito all'infinito.

Tutti questi elementi ci inducono sempre più, anziché a risolvere, a *simulare* la soluzione di un problema.

Per esempio, un problema dalla formulazione semplice (ma dalla soluzione molto complicata) come il seguente: "Qual è la probabilità di fare k lanciando n dadi?" può prevedere agevolmente un approccio di simulazione: "lanciamo" n dadi un gran numero di volte e registriamo (facciamo registrare al calcolatore) quante volte la somma delle facce è k; la probabilità può essere approssimata dalla frequenza relativa.

Si tratta di una attività matematica nuova e interessante, ma richiede nuovi paradigmi e nuove abilità, alle quali spesso l'insegnante non è pronto (in generale lo è meno pronto dei propri studenti).

Poiché abbiamo a disposizione strumenti (addirittura tascabili) dotati di un'ampia libreria di funzioni predefinite (così come la riga e il compasso costituivano la *libreria* di Euclide), possiamo sistematicamente introdurre l'*algoritmo* e la *simulazione* come attività portanti del percorso didattico. Uno studente che compie tali attività acquisisce autonomia e fiducia in se stesso, perché il risultato viene cercato e non fornito a priori da un'autorità superiore; lo studente è maggiormente incuriosito dal problema che deve risolvere; la formulazione di congetture è comunque un lavoro di ottimo livello matematico (dico sempre che è molto più importante una congettura sensata che la soluzione "esatta" ma misteriosa dal punto di vista semantico); lo studente è naturalmente portato a

chiedersi *perché* una certa congettura funziona e quindi comincia a farsi strada l'esigenza di una dimostrazione.
Questa relazione vuole proporre qualche esempio e qualche traccia di percorso, attraverso la formulazione di problemi, che vada nella direzione descritta.

Numeri grandi

Problema. Qual è la 29-esima cifra di 29^{29}?

Innanzitutto: quante cifre ha 29^{29}? Ne ha almeno 29? Poiché un numero $N>1$ è compreso tra due potenze naturali consecutive di 10,

$$10^a \le N < 10^{a+1},$$

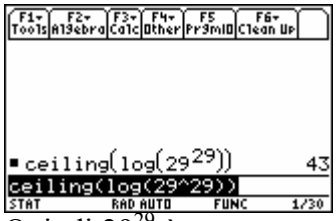
allora

$$a \le \log_{10}(N) < a+1$$

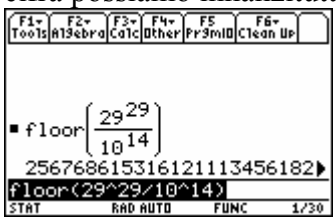
e quindi N possiede esattamente $a+1$ cifre intere.
In qualunque CAS esistono due importanti funzioni aritmetiche, `ceiling(x)` e `floor(x)`, che forniscono, rispettivamente, il "soffitto" e il "pavimento" di un numero x.

Allora il numero di cifre di N è

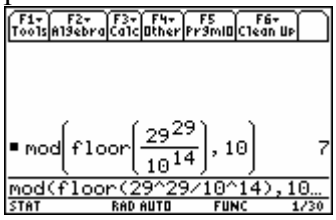
`ceiling(log10(N))`.



Quindi 29^{29} è un numero di 43 cifre, dato che è compreso tra 10^{42} e 10^{43}. Per conoscere la 29-esima cifra possiamo innanzitutto dividerlo per $10^{43-29} = 10^{14}$ e prenderne il "pavimento".

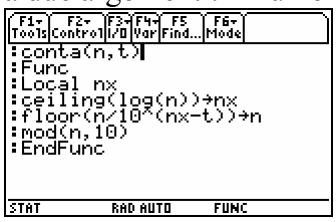


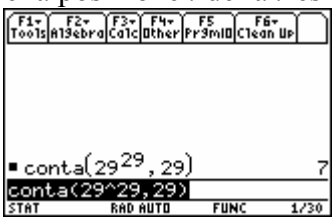
In questo modo la cifra che stiamo cercando è la cifra delle unità del numero appena trovato. Usando la funzione `mod(x,y)`, che fornisce il resto della divisione di x per y, possiamo risolvere il problema iniziale.



La TI-89 distingue tra *function* e *program*. La *function* è una vera e propria funzione: prende in ingresso un numero arbitrario di argomenti e restituisce il risultato (un numero, un'espressione, un vettore, …) in ambiente Home; può essere vista come una estensione del catalogo delle funzioni

predefinite della calcolatrice. Nel nostro caso creiamo una *function*, che chiamiamo `conta(n,t)` a due argomenti: il numero *n* e la posizione *t* della *t*-esima cifra di *n*.





La dichiarazione delle variabili locali è necessaria e non è consentito l'uso di variabili globali.

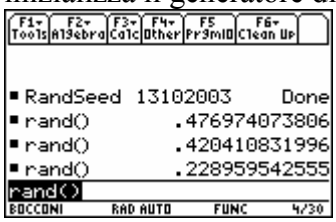
Tre dadi

Problema. Qual è la probabilità che lanciando tre dadi la somma delle facce sia 8?

Il punto di partenza, implementato in qualsiasi sistema di calcolo, è una funzione predefinita che genera un numero *casuale* compreso tra 0 e 1. Sulla TI-89 la funzione è `rand()`; il comando

```
RandSeed n
```

inizializza il generatore di numeri casuali.



Utilizzando solo la funzione `rand()` è possibile generare numeri casuali in qualunque intervallo [*a*, *b*] (è sufficiente una semplice trasformazione lineare)

```
a+(b-a)*rand()
```

e anche generare numeri interi casuali compresi tra *n* e *m*:

```
n+floor((b-a+1)*rand()).
```

La TI-89 possiede la funzione predefinita `rand(n)`, che prende in ingresso un numero naturale *n* fornisce in uscita un numero naturale casuale compreso tra 1 e *n*.

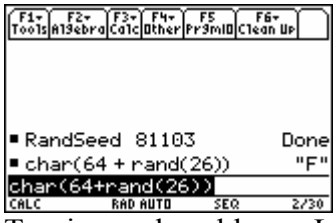
Da che lettera iniziare la prova orale degli esami? Possiamo inserire la data come numero che inizializza il generatore di numeri casuali

```
randseed 081103
```

e poi utilizzare il comando

```
char(64+rand(26))
```

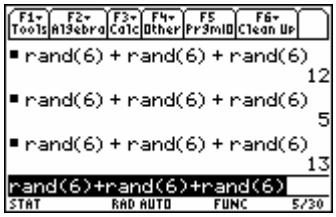
che fornisce una lettera casuale tra la "A" (char(65)) e la "Z" (char(90)).



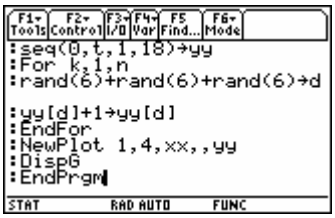
Torniamo al problema. La soluzione generale non è affatto banale. Vogliamo simulare il lancio di tre dadi un gran numero di volte e analizzare le uscite per farci un'idea della possibile soluzione. Il comando `rand(6)` è la simulazione del lancio di un dado equo e

```
rand(6)+rand(6)+rand(6)
```

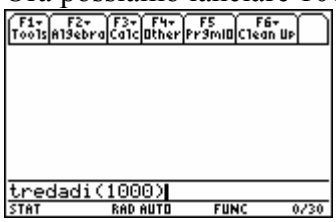
è la simulazione del lancio di tre dadi.

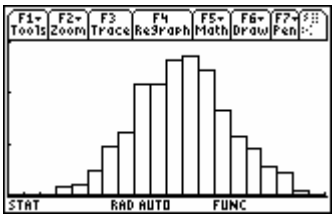


A differenza delle *function*, un *program* può sfruttare uno qualsiasi degli ambienti della calcolatrice; per esempio può fornire in uscita un grafico. Costruiamo dunque il *program tredadi*(*n*) che prende in ingresso il numero *n* di lanci di tre dadi e fornisce in uscita l'istogramma della distribuzione.

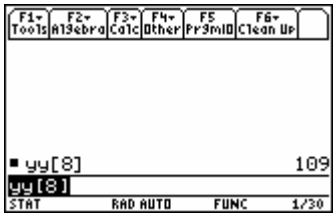


Ora possiamo lanciare 1000 volte tre dadi.





A differenza delle *function* un *program* consente l'utilizzo di variabili globali. Poiché le frequenze delle uscite sono elencate nella lista `yy`, possiamo cercare, in Home, la frequenza assoluta di 8.



Su 1000 lanci si è verificata 109 volte l'uscita 8, la frequenza relativa in questo esperimento è stata dunque 0.109. Vediamo ora una soluzione generale. Vogliamo mostrare che un dado equo a 6 facce è in un certo senso *equivalente* al polinomio

$$p(x) := x + x^2 + x^3 + x^4 + x^5 + x^6:$$

per ogni k=1, 2, …, 6 il coefficiente di x^k, diviso per 6, dà la probabilità che nel lancio di <u>un</u> dado esca k. Lanciamo ora due dadi, cioè calcoliamo $p(x)^2$. Risulta

$$p(x)^2 = x^2 + 2x^3 + 3x^4 + 4x^5 + 5x^6 + 6x^7 + 5x^8 + 4x^9 + 3x^{10} + 2x^{11} + x^{12};$$

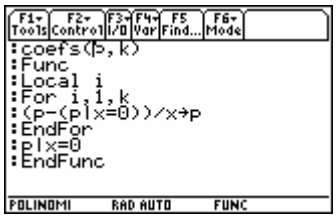
per ogni k=2, 3, …, 12 il coefficiente di x^k, diviso per 6^2, dà la probabilità che nel lancio di <u>due</u> dadi la somma delle facce sia k. Il trucco consiste nel sostituire alla somma delle facce dei dadi la somma degli esponenti di x: il coefficiente di x^k fornisce il numero di possibili combinazioni di due numeri compresi tra 1 e 6 la cui somma sia k.

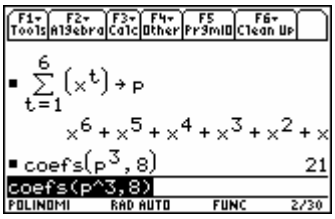
In generale, la probabilità che lanciano n dadi la somma delle facce sia k, con $n \le k \le 6n$, è dato dal rapporto tra il coefficiente di x^k nello sviluppo di $p(x)^n$ e 6^n. La soluzione del nostro problema è dunque la seguente: la probabilità che nel lancio di 3 dadi la somma delle facce sia 8 è uguale al rapporto tra il coefficiente di x^8 nello sviluppo di

$$(x + x^2 + x^3 + x^4 + x^5 + x^6)^3$$

e $6^3 = 216$. L'ultima difficoltà consiste nell'isolare il coefficiente di x^k in $p(x)^n$. Un algoritmo che estrae il coefficiente del termine di grado k da un polinomio in x potrebbe essere il seguente

```
coef(p, k)
for i, 1, k
      (p(x)−p(0))/x → p
p(0).
```



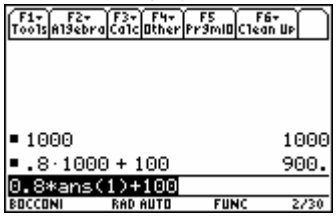


Possiamo finalmente rispondere al problema: la probabilità che nel lancio di tre dadi la somma sia 8 è uguale a 21/216 ≈ 0.097. L'errore relativo commesso con la simulazione è circa del 12%.
Ma siamo ancora molto lontani dalla generalizzazione a k e n qualsiasi. Esprimere in forma simbolica la soluzione in funzione dei dati, cioè n e k, è impresa assai ardua. La simulazione fornisce un risultato attendibile e la sua costruzione è, dal punto di vista cognitivo, interessante.

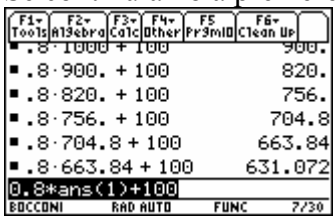
Sistemi dinamici discreti

Problema. Una foresta, inizialmente di 1000 alberi, è adibita a legname. Ogni anno viene tagliato il 20% degli alberi e vengono piantati 100 nuovi alberi. Qual è l'evoluzione della foresta? Si estingue, esplode, si stabilizza?

Si tratta della seguente successione ricorsiva:
$a_0 := 1000$
$a_{n+1} := 0.8a_n + 100$
Con la TI-89 possiamo agevolmente simulare l'evoluzione, mediante il magico comando `ans(1)` (si ottiene con la combinazione di tasti `2nd (−)`). Inseriamo il valore iniziale 1000, poi scriviamo il comando
`0.8*ans(1)+100`.



Se continuiamo a premere `Enter` otteniamo i successivi valori della successione.



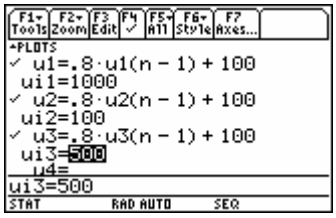
Proseguendo si osserva che la successione converge a 500. Se si chiede ora agli studenti che cosa sarebbe successo partendo da 2000 alberi anziché da 1000, molti congetturano che in questo caso la successione converge a 1000.
Si osserva invece, per via sperimentale, che qualunque sia il valore iniziale a_0 la successione converge sempre a 500, che è proprio il *punto di equilibrio* del sistema dinamico: è quel numero per il quale la diminuzione percentuale del 20% è esattamente uguale all'aumento assoluto di 100. In altri termini è quel numero la cui legge ricorsiva genera una successione costante:

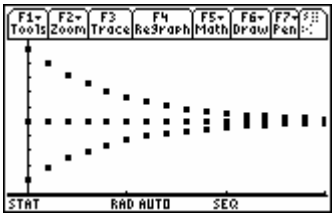
$$a_{n+1} = a_n.$$

In altri termini ancora è la soluzione dell'equazione

$$0.8x + 100 = x.$$

Questo primo approccio di tipo sperimentale è utile dal punto di vista didattico per costruire passo-passo la successione, per osservare come i successivi valori tendono a stabilizzarsi, per esplorare il problema con diversi valori iniziali e convincersi che le corrispondenti successioni convergono tutte allo stesso valore. Questo approccio solitamente induce la curiosità di capire *perché* tutte le successioni convergono a 500. I sistemi dinamici discreti possono costituire un nuovo, ricchissimo argomento per l'insegnamento secondario.
Con la TI-89 utilizziamo la modalità grafica `Sequence` (successione) per tracciare il grafico dei punti di coordinate (n,a_n). Nelle schermate seguenti si osserva la definizione e il grafico di tre successioni $a_{n+1} := 0.8a_n + 100$ con rispettivi valori iniziali 1000, 500, 100.





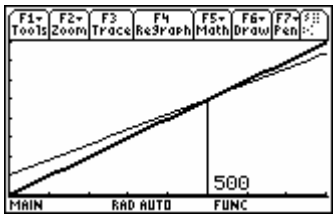
La TI-89, in ambiente `Sequence`, può anche tracciare automaticamente il cosiddetto *diagramma di fase* di una successione. Il punto di equilibrio del sistema si trova risolvendo l'equazione

$$x = 0.8x + 100,$$

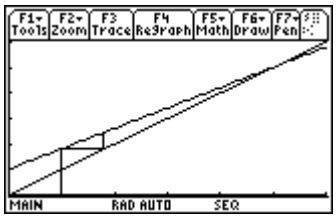
che è l'equazione risultante del sistema

$$\begin{cases} y = 0.8x + 100 \\ y = x \end{cases}$$

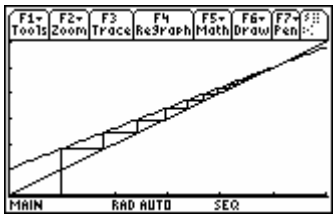
Risulta quindi naturale tracciare sullo stesso piano i grafici della funzione lineare $f: x \to 0.8x+100$ e della funzione identità $I: x \to x$.



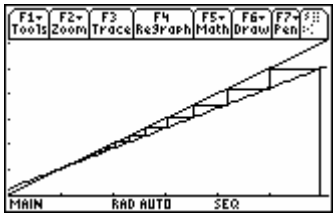
Il punto di equilibrio è l'ascissa x^*=500 del punto di intersezione tra le due rette. Poiché $f(x_t) = x_{t+1}$, se da un punto qualsiasi x_0 sull'asse delle ascisse, per esempio x_0:=100, saliamo in verticale fino al grafico della funzione f troviamo il punto di coordinate (x_0, ax_0+b) cioè il punto (x_0, x_1). Se ora dal punto (x_0, x_1) ci muoviamo prima in orizzontale fino alla retta $y=x$ e poi in verticale fino alla funzione f troviamo dapprima il punto (x_1, x_1) e poi il punto (x_1, x_2).



Di nuovo ci spostiamo in orizzontale fino a $y=x$ e in verticale fino a f trovando il punto (x_2, x_2) e poi il punto (x_2, x_3) e così via. Per ogni coppia di spostamenti (orizzontale e verticale) si passa dal punto (x_{t-1}, x_t) al punto (x_t, x_{t+1}): in definitiva le ordinate dei successivi punti su f sono i successivi valori della successione che parte da x_0. Se proseguiamo indefinitamente costruiamo una *ragnatela* e osserviamo che il sistema non può che convergere a 500.



Analogamente, se partiamo da un valore maggiore di 500, per esempio 1000, i successivi spostamenti ci riportano all'equilibrio. Nella figura seguente la finestra grafica è [400, 1000]×[400, 1000].



Il diagramma di fase ci fa capire da punto di vista *geometrico* perché tutte le successioni convergono all'equilibrio.

In modo analogo possiamo lavorare con un sistema di dimensione 2, come nel seguente, classico, esempio.

Problema. Nel Paese X, in cui votano costantemente 12 000 cittadini, si vota ogni anno per il rinnovo del Parlamento e ogni anno si presentano alle elezioni due partiti: il partito A e il partito B. Ad ogni elezione gli umori dell'elettorato in parte cambiano: non tutti quelli che hanno votato per un partito lo votano ancora l'anno successivo. Da stime e sondaggi si ricava la seguente tabella, che illustra come si spostano le preferenze dell'elettorato da un anno all'altro:

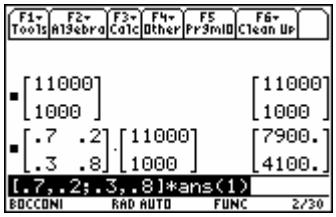
	da A	da B
a A	70%	20%
a B	30%	80%

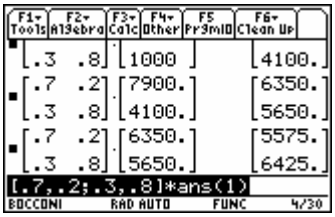
Se le ultime elezioni dell'anno 2003 hanno assegnato 11000 voti al partito A e 1000 voti al partito B, come andranno le elezioni del 2004? E nel 2010? E nel 2020? Qual è l'evoluzione del sistema?

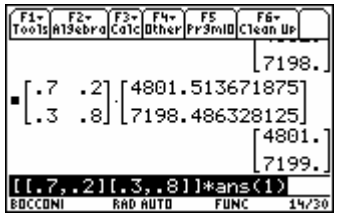
Il vettore iniziale è $\mathbf{a}_0 := [11000, 1000]$ e la legge ricorsiva è

$$\mathbf{a}_{n+1} := \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix} \mathbf{a}_n.$$

Ecco la simulazione sulla TI-89.







Si osserva che la successione converge al vettore [4800, 7200]. Si tratta, anche in questo caso, del *vettore di equilibrio* tale che

$$\mathbf{a}_{n+1} = \mathbf{a}_n,$$

cioè la soluzione dell'equazione vettoriale

$$\mathbf{x} = \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix} \mathbf{x}.$$

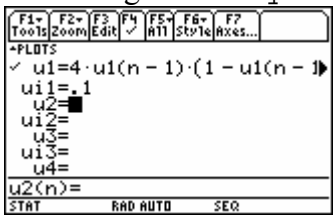
Chi tratta vettori e matrici può mettere parecchia carne al fuoco.

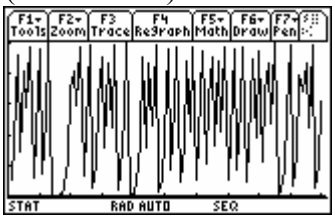
A proposito di sistemi dinamici, come non citare la *successione logistica*? Consideriamo la ben nota (e apparentemente innocua) successione

$a_0 := 0.2$

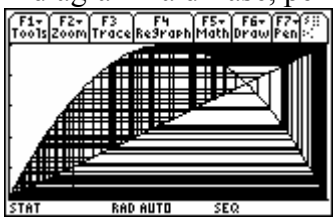
$a_{n+1} := 4a_n(1-a_n)$

e tracciamo il grafico della spezzata di vertici (n, a_n) per n da 1 a 100. Con la TI-89 utilizziamo la modalità grafica `Sequence` (successione).





Il sistema non sembra avere alcun comportamento prevedibile, oscilla in modo irregolare su tutto l'intervallo [0, 1] e pare che non abbia alcuna intenzione di stabilizzarsi, né per convergere né per divergere. Il *caos* è in agguato.
Il diagramma di fase, per le prime 100 iterazioni, ha il seguente inquietante aspetto.



Il metodo Montecarlo

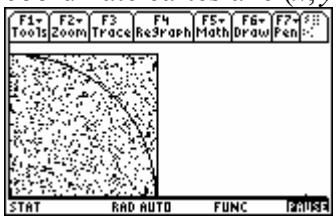
Con questo termine (Montecarlo è proprio il famoso casinò) si dovrebbe designare qualunque metodo di indagine basato sulla simulazione. Tuttavia esso è diventato, per antonomasia, il nome di un metodo di approssimazione delle aree (e degli integrali definiti). L'idea è semplice e la vediamo applicata sull'esempio del calcolo dell'area del cerchio.
Consideriamo sul piano cartesiano il quadrante del cerchio di centro l'origine e raggio 1 inscritto nel quadrato di vertici (0,0), (1,0), (1,1), (0,1). *Spariamo* a caso nel quadrato e cioè scegliamo un punto a caso di coordinate (x,y) con $0<x<1$, $0<y<1$: possiamo colpire un punto interno oppure esterno al cerchio, a seconda che la distanza del punto (x,y) sia minore o maggiore di 1. Ci aspettiamo che il rapporto tra il numero di "centri" c (punti interni al cerchio) e il numero di colpi sparati n si avvicini al rapporto tra l'area del quadrante di cerchio e l'area del quadrato, cioè $\pi/4$:

$$\frac{c}{n} \to \frac{\pi}{4}.$$

In questo modo possiamo approssimare π mediante $4c/n$.
Il programma è molto semplice:

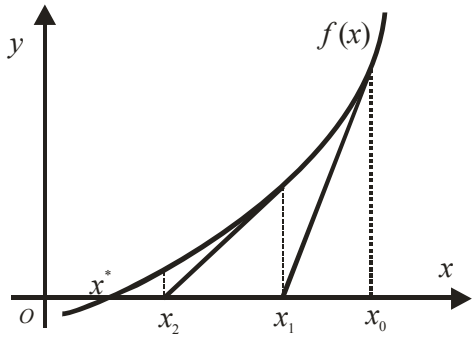
```
montecarlo(n)
0→c
For i,1,n
rand()→x:rand()→y
If x^2+y^2<1 Then
c+1→c
EndIf
4*c/n
```

Una versione più sofisticata con la TI-89 sfrutta il comando `PtOn x,y`, che accende il pixel di coordinate cartesiane (x,y) e fornisce i seguenti output.



Algoritmo di Newton

Un'attenzione particolare merita il classico algoritmo di Newton per l'approssimazione delle soluzioni di un'equazione $f(x) = 0$, dove $f(x)$ è una funzione derivabile.
Una soluzione di $f(x)=0$ è un punto x^* in cui il grafico di $y = f(x)$ interseca l'asse x. Se in un punto x_0 mandiamo la retta tangente a $f(x)$, questa interseca l'asse x in un punto x_1. Si manda la retta tangente a $f(x)$ in x_1 e così via: si costruisce così una successione $x_0, x_1, x_2, \ldots$ che, sotto opportune ipotesi, converge a x^*.



La retta tangente a $f(x)$ in x_0 ha equazione $y = f'(x_0)(x-x_0)+f(x_0)$ e interseca l'asse x in

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Se generalizziamo otteniamo la legge ricorsiva dell'algoritmo di Newton:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}$$

Consideriamo come esempio l'equazione $f(x) = 0$:

$$x^3 + x - 1 = 0,$$

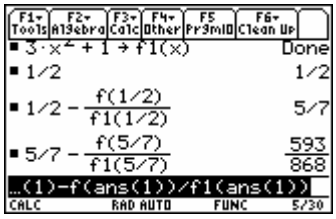
che ammette certamente uno zero x^* compreso tra 0 e 1, dato che $f(0) = -1$ e $f(1) = 1$. Risulta

$$f(x) = x^3 + x - 1, \qquad f'(x) = 3x^2 + 1.$$

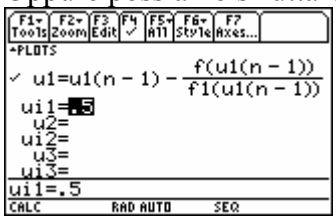
Se assumiamo come condizione iniziale il punto medio dell'intervallo [0, 1], $x_0 = 1/2$, otteniamo (in forma simbolica) la successione

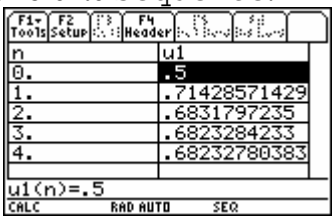
$$\frac{1}{2}, \frac{5}{7}, \frac{593}{868}, \ldots$$

Possiamo memorizzare le due funzioni $f(x)$ e $f'(x)$ direttamente nell'ambiente Home e generare la successione sfruttando il comando `ans(1)`.



Oppure possiamo sfruttare l'ambiente `Sequence`.



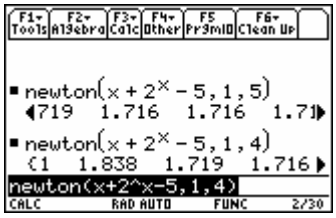


La successione generata dall'algoritmo di Newton ha una straordinaria rapidità di convergenza (in termini tecnici si dice che x^* è un *superattrattore*). Nel nostro esempio sono sufficienti 5 iterazioni per approssimare le prime 15 cifre decimali.

Il programma seguente prende in ingresso $f(x)$, il valore iniziale x_0, il numero di iterazioni n e fornisce in uscita la lista $\{x_0, x_1, \ldots, x_n\}$.

```
newton(f,a,n)
Func
Local  i,c,g
{a}→c
d(f,x)→g
For  i,1,n
```

```
approx(a-(f|x=a)/(g|x=a))→a
augment(c,{a})→c
EndFor
c
EndFunc
```



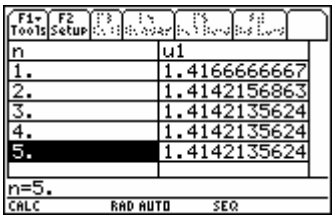
Naturalmente possiamo dare in uscita solo l'ultimo elemento della lista, oppure possiamo fornire in ingresso, anziché il numero di iterazioni, l'errore ε tale che risulti $|x_{n+1} - x_n| < \varepsilon$.

Radici

Se si applica l'algoritmo di Newton all'equazione $x^2 - N = 0$, con N positivo qualsiasi e con $x_0>0$, si costruisce una successione che converge a $\sqrt{N}$. Svolgendo i calcoli:

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

Si scopre così che l'algoritmo di Newton applicato all'equazione $x^2 - N = 0$ restituisce l'algoritmo di Erone per il calcolo della radice quadrata di N. Possiamo allora sfruttarlo per approssimare la radice quadrata di un numero positivo, per esempio $\sqrt{2}$, sapendo che è compresa tra 1 e 2, e quindi partendo da 1.5; la seconda iterazione, che in forma simbolica fornisce 577/408, approssima già le prime cinque cifre decimali.



In modo analogo possiamo approssimare rapidamente la radice n-esima di un numero positivo N: partiamo dall'equazione $x^n - N = 0$, e quindi:

$f(x) = x^n - N, \qquad f'(x) = nx^{n-1}$

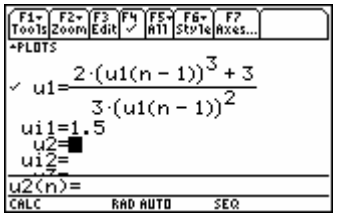
$$x_{t+1} = x_t - \frac{x_t^n - N}{nx_t^{n-1}} = \frac{(n-1)x_t^n + N}{nx_t^{n-1}}.$$

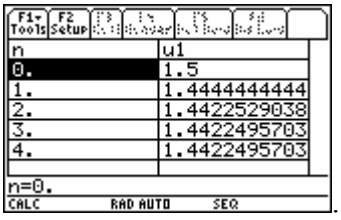
La successione ricorsiva per approssimare la radice cubica di N è dunque

$$x_{t+1} = \frac{2x_t^3 + N}{3x_t^2}.$$

Per esempio, per approssimare $\sqrt[3]{3}$, che è compreso tra 1 e 2, definiamo la successione

$$x_{t+1} = \frac{2x_t^3 + 3}{3x_t^2}, \text{ con } x_0 = 1.5.$$





La terza iterazione ha già approssimato le prime 10 cifre decimali.

Logaritmi

Qual è la soluzione di $2^x = 10$? Se vogliamo evitare l'equivoco di limitarci a fornire la scrittura simbolica e tautologica

$$x = \log_2(10)$$

dobbiamo dare uno strumento per approssimare i logaritmi. Sia ben chiaro: lo strumento più semplice è il comando `log` di qualsiasi calcolatrice: il tema nuovo che vogliamo introdurre è squisitamente matematico.

L'algoritmo di Newton non è *politically correct*: l'equazione $2^x - 10 = 0$ conduce alla successione ricorsiva

$$x_{t+1} = x_t - \frac{2^{x_t} - 10}{\ln(2)\,2^{x_t}}$$

in cui occorre necessariamente calcolare ln(2).

D'altra parte è sufficiente conoscere i logaritmi in una data base, per esempio in base 10. Tutti gli altri si ottengono da questi mediante il cambiamento di base:

$$\log_b(N) = \frac{\log(N)}{\log(b)}.$$

Un semplice algoritmo per l'approssimazione del logaritmo decimale di N (supponiamo per semplicità che N sia maggiore di 1) sfrutta le due banali osservazioni seguenti.

- Ogni numero positivo è compreso tra due potenze intere successive di 10; se
  $$10^k \le N < 10^{k+1}$$
  allora
  $$k \le \log(N) < k+1,$$
  cioè k è la parte intera di $\log(N)$.
- Per ogni numero intero k i due numeri $\log(N)$ e $\log(10^k N)$ hanno la stessa parte decimale. Infatti
  $$\log(10^k N) = k + \log(N).$$

Supponiamo per esempio di voler calcolare log(1789). Risulta

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

di conseguenza la parte intera di $\log(N)$ è 3:

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

cioè

$$\log(1789) = 3.\ \ldots$$

Ora dividiamo 1789 per 10^3, in modo da ottenere un numero compreso tra 1 e 10, e osserviamo che log(1789) e log(1.789) hanno la stessa parte decimale, quindi in particolare hanno la stessa prima cifra decimale; indichiamo con x tale cifra. Risulta

$$\log(1.789) = 0.\ x\ \ldots$$
$$10\log(1.789) = x.\ \ldots$$
$$\log(1.789^{10}) = x.\ \ldots$$

Siamo tornati al punto di partenza: poiché $1.789^{10} = 335.8\ldots$, allora

$$10^2 < \log(1.789^{10}) < 10^3$$
$$2 < \log(1.789^{10}) < 3$$

cioè

$$\log(1789) = 3.2\ \ldots$$

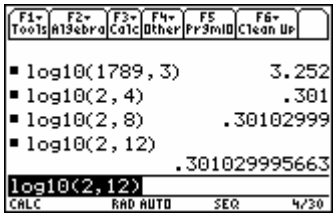
Possiamo proseguire approssimando ad ogni iterazione la successiva cifra decimale. Tutto quello che dobbiamo saper fare è:

- elevare un numero alla 10 (è un prodotto ripetuto);
- contare il numero di cifre intere del risultato precedente (è una divisione ripetuta per 10).

L'algoritmo potrebbe essere il seguente, in cui si approssimano le prime *c* cifre decimali di log(*n*).

```
log10(n,c)
0→x
for i,0,c
     0→k
while n>10
     n/10→n
     k+1→k
endwhile
x+k*10^(-i)→x
n^10→n
endfor
x
```

Ecco qualche applicazione con la TI-89. Si osservi che l'ultima cifra decimale è sempre troncata; non è difficile modificare il programma in modo da calcolare la cifra successiva e arrotondare.



La distanza media tra due punti

Problema. Consideriamo due punti scelti a caso nell'intervallo [0, 1]. Quanto vale in media la loro distanza? Qual è la probabilità che la loro distanza sia compresa tra 0.5 e 0.6?

L'approccio teorico al problema è certamente impegnativo. Proviamo una simulazione con Mathcad. Come in ogni sistema di calcolo automatico, anche in Mathcad esiste una funzione che fornisce un numero casuale compreso tra 0 e 1: rnd(1); si simula così una variabile aleatoria con distribuzione uniforme su [0, 1]. Prendiamo allora *n* coppie di numeri casuali compresi tra 0 e 1 e calcoliamone la distanza.

$n := 100000$

$k := 1 .. n \qquad a_k := |rnd(1) - rnd(1)|$

Possiamo valutare la media e la deviazione standard degli *n* dati.

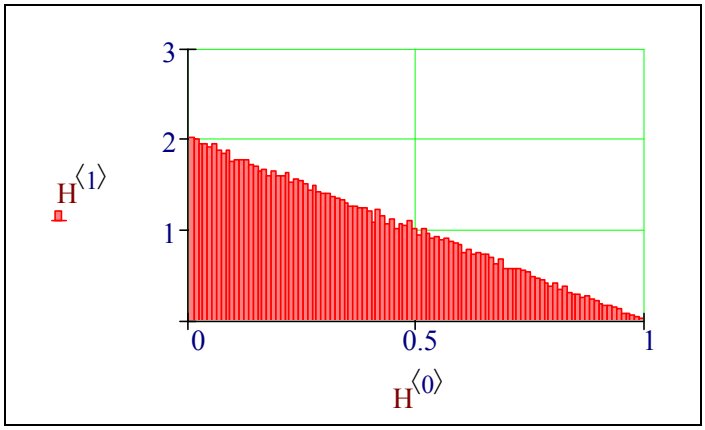
$mean(a) = 0.333$

$stdev(a) = 0.236$

Possiamo ora organizzare i dati in un istogramma, dividendo l'intervallo [0, 1] in un certo numero di classi di uguale ampiezza

$$A = \frac{b-a}{n},$$

calcolando per ciascuna la frequenza relativa e normalizzando dividendo per A.
Nella figura seguente il numero di classi è 100, l'ampiezza di ciascuna classe è 0.01, e il numero di esperimenti è 100000.



Quello che otteniamo è, in via sperimentale, il profilo della densità di probabilità, che appare, al di là di ogni ragionevole dubbio, lineare: è la funzione lineare $f(x) = 2–2x$. Il valore atteso è allora

$$\int_0^1 x(2-2x)\,dx = \frac{1}{3}$$

e la varianza è

$$\int_0^1 \left(x-\frac{1}{3}\right)^2 (2-2x)\,dx = \frac{1}{18} \approx 0.056,$$

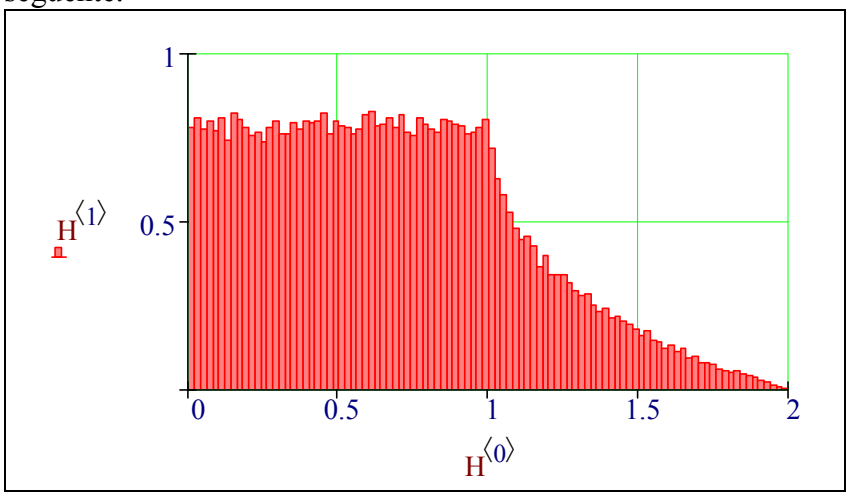
dando senso ai risultati sperimentali. La probabilità che la distanza sia compresa tra 0.5 e 0.6 è
Il fatto interessante è che dal punto di vista teorico i calcoli diventano subito proibitivi, mentre dal punto di vista della simulazione non c'è alcuna difficoltà: è sufficiente modificare la definizione della successione a_k.
Ecco per esempio il sorprendente risultato della distribuzione di probabilità di X^2+Y^2, dove X e Y sono due variabili aleatorie con distribuzione uniforme su [0, 1].

$n := 100000 \qquad nC := 100$

$k := 1 .. n \qquad a_k := rnd(1)^2 + rnd(1)^2$

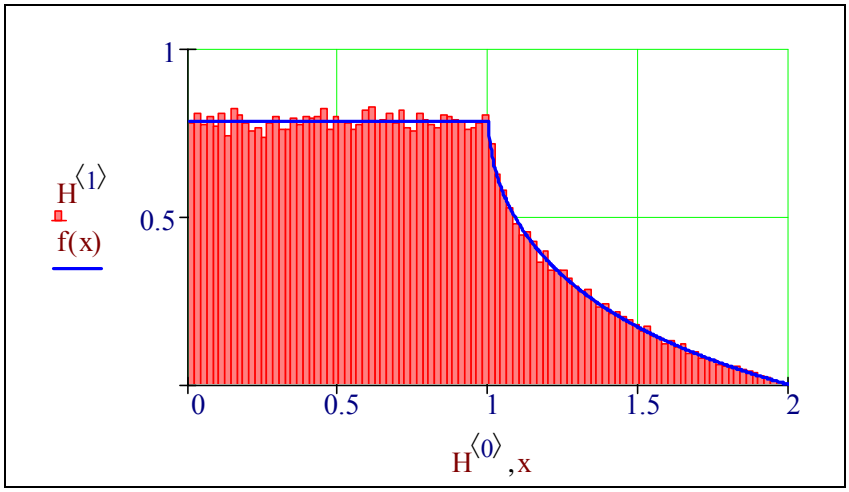
$mean(a) = 0.668$
$var(a) = 0.178$

L'istogramma normalizzato, sempre con 100000 esperimenti e 100 classi di ampiezza 0.02, è il seguente.



Si osserva che la densità è costante sull'intervallo [0, 1] (vale circa 0.8), e decresce rapidamente a 0 tra 1 e 2. Fatti i conti (molto complicati) risulta che la densità di probabilità è la funzione

$$f(x) := \begin{cases} \pi/4 & 0 \le x < 1 \\ \frac{1}{2}\arcsin\left(\frac{2-x}{x}\right) & x \ge 1 \end{cases}$$



La media è

$$\int_0^2 x\, f(x)\, dx = \frac{2}{3}$$

e la varianza è

$$\int_0^2 \left(x - \frac{2}{3}\right)^2 f(x)\, dx = \frac{8}{45} \approx 0.178.$$

In un certo senso la simulazione consente una sorta di "democratizzazione" della matematica: non conosco la teoria ma posso trovare un risultato ragionevole in modo sperimentale. Il computer che può usare uno studente è lo stesso che usa un ricercatore accademico.

Una conclusione provocatoria

Pochi giorni fa ho ricevuto la seguente mail, che riporto integralmente.

> Salve, mi chiamo Maurizio Castrini e sono un programmatore. Ho trovato casualmente il suo sito su internet. Volevo chiedere un aiuto se fosse possibile... Sono alle prese con un software che sto sviluppando in Visual Basic e non riesco a risolvere il seguente problema: lavoro in uno spazio 2D, ho disegnato poligono di 4 lati a caso nello spazio 2D. Passando con il mouse in questo spazio 2D, come faccio a sapere se il punto attuale del mouse è dentro al poligono che ho disegnato o meno? Se non la disturba, sa indicarmi un sito di riferimento dove trovare la risposta al mio problema? Grazie mille. Distinti Saluti.

Come è noto esiste un semplice algoritmo per stabilire se, percorrendo una retta AB di un piano, un punto P si trova nel semipiano "di destra" o "di sinistra" oppure sulla retta stessa: è sufficiente calcolare il prodotto vettoriale

$$\mathbf{v} = \underline{AB} \times \underline{AP}$$

tra i vettori $\underline{AB}$ e $\underline{AP}$. In definitiva, se i dati in ingresso sono le coordinate (x_1, y_1), (x_2, y_2), (x, y) di A, B, P si consideri la funzione

$$f(A, B, P) = (x_2 - x_1)(y - y_1) - (x - x_1)(y_2 - y_1)\ .$$

Se $f(A, B, P) > 0$ allora ABP è percorso in verso antiorario (P si trova "a sinistra " della retta orientata AB); se $f(A, B, P) < 0$ allora ABP è percorso in verso orario (P si trova "a destra" della retta orientata AB); se $f(A, B, P) = 0$ allora P appartiene alla retta AB.

Un punto P è allora interno ad un poligono $A_1A_2 \ldots A_n$ (convesso e non intrecciato), se percorrendone il perimetro il punto P si trova sempre "a destra" oppure sempre "a sinistra" (dipende dal verso di percorrenza del poligono), cioè se la funzione $f(A, B, P)$ applicata ad ogni terna A_i, A_{i+1}, P restituisce n numeri positivi oppure n numeri negativi.

Ma non è questo che mi interessa ora. La riflessione che vorrei fare è la seguente: uno degli aspetti che mi preoccupa di più del sapere scolastico è quella sorta di idealismo auto-referenziale di cui sono pervasi i libri di testo, dall'Italiano alla Matematica, che in molti casi insegna a dare risposte

convenzionali a domande convenzionali (risposte il cui senso si limita alla prova di valutazione in cui vengono richieste). Poiché viviamo in tempi in cui la *cultura* non ha più una definizione univoca e non può più auto-giustificarsi, la trasmissione del sapere è tanto più efficace quanto più è sottoposta, mediante i suoi stessi metodi, a continua verifica e auto critica.
Per questo temo molto (ho sempre temuto) qualunque invocazione al *rigore* (= rigor mortis) in matematica; ho sempre vissuto il rigore come un modo autoritario (e un po' terroristico, diciamolo) per non lasciare spazio alcuno a riflessioni *sulla* matematica. L'eccessivo richiamo al rigore rischia di trasformarsi in un malcelato desiderio di ridurre gli allievi a cervelli che galleggiano in un catino pieno d'acqua.
Bene, la costruzione di un algoritmo è probabilmente immune da questo rischio: perché in un algoritmo c'è una sintassi da rispettare, e perché c'è una semantica da soddisfare. Ecco una versione interessante di "rigore": l'algoritmo "gira"? produce quello che vogliamo?
Il problema che pone il messaggio che ho riportato potrebbe fornire spunti per pensare ad una scuola nuova.