Calcolo Numerico - Corso B: Laboratorio Lezione 0

Luca Gemignani <luca.gemignani@unipi.it>

20 Febbraio 2019

1 Perchè il Calcolo Numerico

Il corso di Calcolo Numerico introduce alla risoluzione di *problemi matematici* mediante l'utilizzo di *metodi numerici* (che operano con numeri) generalmente implementati ed eseguiti su un *calcolatore*. La richiesta di metodi numerici adatti all'implementazione su un calcolatore discende dall'osservazione che gran parte dei problemi matematici non possono essere risolti con carta e penna perchè:

1. il problema non ammette una soluzione esprimibile in forma chiusa come combinazione finita di funzioni ed operazioni elementari note (esempi sono forniti dalla risoluzione di equazioni non lineari e il calcolo di integrali definiti). La ricerca della soluzione sarà quindi affrontata mediante la costruzione di una sequenza di approssimazioni con grado di precisione crescente;

2. la dimensione del problema lo rende trattabile solo con l'utilizzo di strumenti automatici per il calcolo (esempi sono forniti dai classici problemi dell'algebra lineare: risoluzione di sistemi lineari e calcolo di autovalori);

3. la sperimentazione (effettuata a supporto dell'analisi di proprietà qualitativamente e quantitativamente significative della soluzione) coinvolge grosse basi di dati.

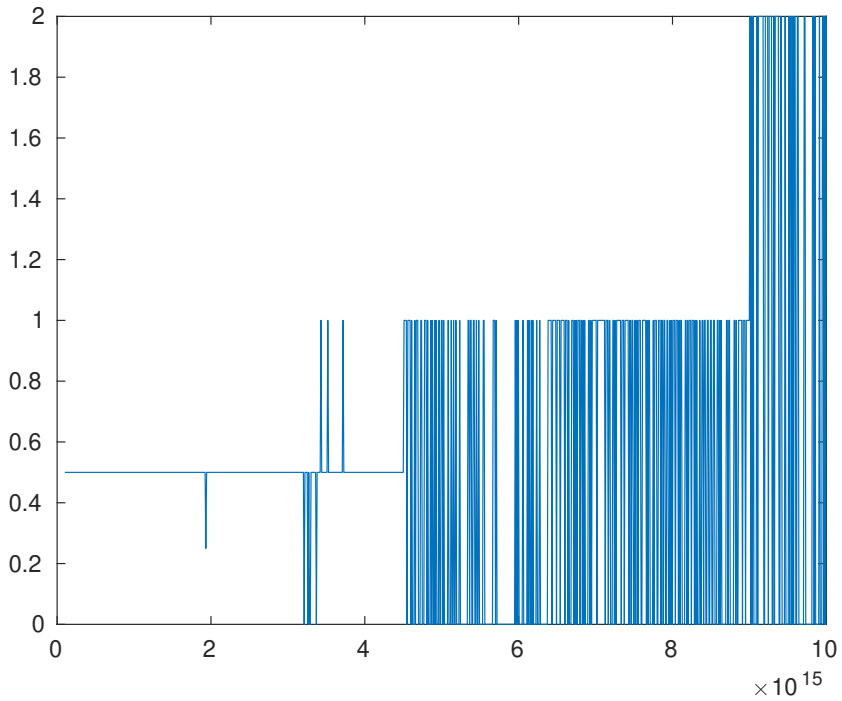
L'analisi e la sintesi dei metodi numerici non può pertanto prescindere dallo studio del loro comportamento quando implementati su un calcolatore che opera con un'aritmetica a *precisione finita*. Operare in questo ambito a sua volta genera nuovi problemi matematici. Per esemplificare le problematiche che emergono in questo rapporto sinergico introduciamo l'ambiente di calcolo MatLab. Supponiamo ad esempio di voler effettuare una sperimentazione numerica a supporto o verifica della nostra valutazione riguardo il valore del seguente limite

$$\lim_{x\to+\infty} \sqrt{x^2+x}-x, \quad f(x)=\sqrt{x^2+x}-x.$$

In ambiente Matlab scriveremo

```
>> f=@(x)sqrt(x.^2+x)-x

f =

  function_handle with value:

    @(x)sqrt(x.^2+x)-x

>> x=linspace(1.0e+14, 1.0e+16, 1000);
>> plot(x, f(x))
```

generando in uscita il seguente grafico



che non risulta particolarmente significativo. L'analisi della coda della sequenza di dati $y = f(x)$ è altrettanto imbarazzante evidenziando un comportamento erratico della sequenza.

```
>> y=f(x);
>> y(990:end)
```

```
ans =

0 2 0 0 0 2 2 0 2 0 2
```

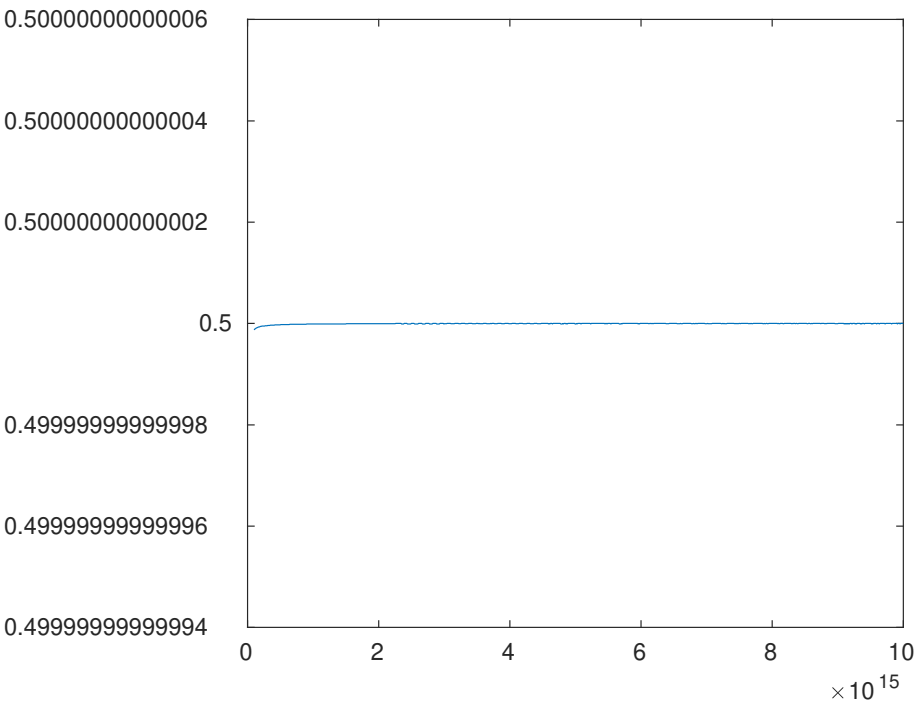
La funzione $f(x)$ può essere riscritta in forma equivalente come

$$f(x) = \sqrt{x^2+x} - x = \frac{(\sqrt{x^2+x} - x)(\sqrt{x^2+x} + x)}{\sqrt{x^2+x} + x} = \frac{x}{\sqrt{x^2+x} + x}.$$

Ponendo $g(x) = \frac{x}{\sqrt{x^2+x}+x}$ otteniamo in MatLab

```
>> g=@(x)x./(sqrt(x.^2+x)+x)

g =

  function_handle with value:

    @(x)x./(sqrt(x.^2+x)+x)

>> plot(x, g(x))
```

che restituisce il grafico



Il secondo grafico è assai più informativo del primo e rivela che il valore limite vale 0.5. Risulta comunque chiaro che algoritmi (funzioni) matematicamente equivalenti una volta implementati in macchina possono condurre a risultati completamente differenti.

Il calcolo delle funzioni $f(x)$ e $g(x)$ in macchina origina a sua volta un nuovo problema matematico. Più precisamente le funzioni effettivamente calcolabili su un calcolatore sono le funzioni razionali esprimibili come sequenza di operazioni aritmetiche. Pertanto per il calcolo delle radici quadrate coinvolte nell'espressione di f e g si dovranno indicare le corrispondenti funzioni razionali di approssimazione. Supponiano per fissare le idee di voler determinare un'approssimazione razionale di $\sqrt{a}$, $a > 0$. L'approccio usuale consiste nel considerare la risoluzione dell'equazione algebrica di secondo grado $x^2 = a$. Per l'approssimazione delle soluzioni dell'equazione si utilizza quindi un metodo iterativo che costruisce una successione di approssimazioni. Un metodo classico è il metodo delle tangenti o di Newton che a partire da un dato iniziale x_0 genera la sequenza di approssimazioni

$$x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k}, \quad k \geq 0.$$

Per costruire i primi 10 termini della successione con $a = 2$ e $x_0 = 2$ scriviamo

```
>> x0=2; a=2;
>> for k=1:10; x(k)=x0-(x0^2-a)/(2*x0); x0=x(k); end;
>> x

x =

  Columns 1 through 9

    1.5000 1.4167 1.4142 1.4142 1.4142 1.4142 1.4142 1.4142 1.4142

  Column 10

    1.4142
```

e con un differente formato di visualizzazione

```
  >> x

x =

  Columns 1 through 3

     1.500000000000000e+00 1.416666666666667e+00 1.414215686274510e+00

  Columns 4 through 6

     1.414213562374690e+00 1.414213562373095e+00 1.414213562373095e+00
```

```
  Columns 7 through 9

    1.414213562373095e+00 1.414213562373095e+00 1.414213562373095e+00

  Column 10

    1.414213562373095e+00

>>
```

Il valore restituito dalla funzione interna in MatLab vale

```
  >> sqrt(2)

ans =

    1.414213562373095e+00

>>
```

in perfetto accordo con il valore determinato sopra.
D'altra parte per $a = 2$ e $x_0 = -1$ otteniamo

```
x =

  Columns 1 through 3

   -1.500000000000000e+00 -1.416666666666667e+00 -1.414215686274510e+00

  Columns 4 through 6

   -1.414213562374690e+00 -1.414213562373095e+00 -1.414213562373095e+00

  Columns 7 through 9

   -1.414213562373095e+00 -1.414213562373095e+00 -1.414213562373095e+00

  Column 10

   -1.414213562373095e+00
```

mentre per $a = 2$ e $x_0 = 1000$ otteniamo

```
x =

  Columns 1 through 3
```

```
   5.000010000000000e+02 2.500024999960000e+02 1.250052499580004e+02

Columns 4 through 6

   6.251062464301702e+01 3.127130960206219e+01 1.566763299486836e+01

Columns 7 through 9

   7.897642347856357e+00 4.075441240519499e+00 2.283092824392554e+00

Column 10

1.579548752406015e+00
```

Quindi la convergenza della sequenza dipende criticamente dalla selezione del punto iniziale.