Proposta esercizi 1

Calcolo Numerico (LT Informatica), a.a. 2019-2020

Dipartimento di Scienze Matematiche Fisiche e Informatiche
Università degli Studi di Parma

Docente:
Prof.ssa Chiara Guardasoni

Tutor:
Dott. Ariel S. Boiardi *

1 Numeri macchina e calcolo in virgola mobile

Rappresentazione dei numeri reali in diverse basi Quando scriviamo un numero intero positivo N con la usuale *notazione posizionale decimale*, ad esempio $N = 1234567890$, sottintendiamo la sviluppo in potenze di 10:

$$1234567890 = 1\cdot 10^9+2\cdot 10^8+3\cdot 10^7+4\cdot 10^6+5\cdot 10^5+6\cdot 10^4+7\cdot 10^3+8\cdot 10^2+9\cdot 10^1+0\cdot 10^0,$$

dove, naturalmente, l'ordine degli addendi è in'influente. In generale la scrittura posizionale decimale di un numero $N = d_n d_{n-1} \dots d_1 d_0$ sottintende lo sviluppo

$$N = d_n \cdot 10^n + d_{n-1} \cdot 10^{n-1} + \cdots + d_1 \cdot 10 + d_0,$$

dove $d_n, \dots, d_0$ sono cifre fra 0 e 9.

Naturalmente non vi è ragione intrinseca per la scelta della base 10 rispetto a qualunque altro numero: altre civiltà hanno usato numerazioni in basi diverse e, ancora oggi, contiamo ore e minuti in base 60, ricorrendo alla rappresentazione decimale solo per i sottomultipli inferiori all'unità del secondo (anche se in effetti non capita spesso di usare frazioni dell'ora diverse da $\frac{1}{2}$ o $\frac{1}{4}$...).

Nel calcolatore, come ben noto, le cifre vengono rappresentate mediante stati fisici, ed è quindi conveniente per ragioni tecniche utilizzare una numerazione in base 2. Tale sistema di numerazione è detto binario e le cifre si riducono dai due simboli 0 e 1.

La possibilità di rappresentare un numero qualunque in una base a piacere è garantita dal seguente fondamentale risultato

Teorema 1. *Dato un intero $\beta > 1$, un numero reale $x \neq 0$ può essere univocamente espresso nella seguente forma*

$$x = \underbrace{sign\,(x)}_{S} \underbrace{\left(d_1\beta^{-1} + d_2\beta^{-2} + \cdots\right)}_{M} \underbrace{\beta^p}_{E} \tag{1}$$

*Per dubbi e segnalazioni: `arielsurya.boiardi@studenti.unipr.it`

con $0 \leq d_i \leq \beta - 1$ *per ogni* i, $d_1 \neq 0$ *e* $d_i \neq \beta - 1$ *per qualche* i. *Nell'equazione precedente, S è il segno del numero, M è detta mantissa, P parte esponente, l'esponente* p *è chiamato caratteristica.*

Osserviamo che nel risultato precedente, per poter rappresentare un numero reale qualunque potremmo aver bisogno di infinite cifre, mentre nel calcolatore ogni variabile può, per ovvie ragioni, usare un numero finito di cifre. Se il numero di cifre significative è fissato a t, l'insieme dei numeri macchina, cioè quelli rappresentabili esattamente con t cifre in base β è

$$\mathbb{F}(\beta, t) = \{0\} \cup \left\{ x = \operatorname{sign}(x)\, \beta^p \sum_{i=1}^{t} d_i \beta^{-i} \right\}.$$

La maggiora parte dei calcolatori permettono di lavorare con variabili numeriche di lunghezze diverse, cui corrispondono diversi livelli di precisione e utilizzi di memoria.

Calcolo in virgola mobile Fino al 1985 i principali costruttori di calcolatori elettronici implementavano i numeri in virgola mobile ognuno con un proprio formato numeri e proprie convenzioni di calcolo. La necessità di scambiare dati fra diversi calcolatori portò nel 1985 all'istituzione dello standard IEEE 754[1] per i numeri in virgola mobile, oggi seguito da virtualmente tutti i produttori di sistemi di calcolo[1].

In MATLAB , ove non diversamente specificato, i numeri vengono rappresentati nel formato *double precision*[2], che utilizza 64 bits di memoria di cui 1 per il segno, 52 per la mantissa e 11 per la caratteristica. L'esponente è un intero da 11 bit, compreso quindi fra 0 e $2^{2^{11}-1} = 2^{2047}$; per riuscire ad esprimere anche esponenti negativi è stato fissato $2^{10} - 1 = 1023$ come zero per l'esponente, pertanto $p \in [-1022, 1023]$ e i due esponenti 1024 e -1023 sono utilizzati valori speciali che rappresentano 0, ∞ o risultati di operazioni impossibili.

Esercizio 1: Alla luce della spiegazione sopra sulla rappresentazione in doppia precisione, come possiamo esprimere esplicitamente le quantità `realmin, realmax` ed `eps` restituite dalle omonime functions in MATLAB . Si vede bene che `realmin` differisce (parecchio!) dall'`eps` macchina, quale è la differenza fra questi due valori, quale è il più rilevante per misurare l'accuratezza di un calcolo?

Ora che abbiamo chiaro il significato di `realmin, realmax, eps` possiamo usarli per osservare alcuni comuni problemi

Esercizio 2: Che risultato darà il seguente calcolo

```
>> realmax + 1 - realmax
```

e perché? Avremo qualche differenza invertendo l'ordine? Perché?

[1] Linguaggi, compilatori, interpreti, processori...
[2] Definito binary64 nelle ultime revisioni dello standard

```
>> realmax - realmax +1
```

Come noto l'*epsilon macchina* `eps` è il più piccolo numero macchina positivo tale che $1 + \texttt{eps} > 1$ e pertanto

```
>> 1 + eps - 1

ans =

        2.220446049250313e-16

>> ans == eps

ans =

        logical

        1
```

possiamo coerentemente vedere che $1 + \texttt{eps}$ differisce da 1 proprio di `eps`. Sarà anche vero che $2 + \texttt{eps} > 2$? Perché?

I fenomeni osservati nel precedente esercizio sono noti come *overflow*, quando qualcosa viene trascurato perché supera il massimo valore rappresentabile, e *undeflow*, quando qualcosa viene trascurato perché più piccolo della "sensibilità" del calcolatore.

Gli esempi appena mostrati mettono in evidenza come in aritmetica di macchina possano smettere di essere valide molte delle proprietà dell'aritmetica in campo reale. Va notato che alle volte sottovalutare questi problemi risulta catastrofico...[3]

Esercizio 3: Fissati $a = 0.233\,712\,58 \times 10^{-4}, b = 0.336\,784\,29 \times 10^{2}, c = -0.336\,778\,11 \times 10^{2}$, cosa ci aspettiamo dal confronto fra $(a + b) + c$ e $a + (b + c)$? E se il calcolo viene svolto in aritmetica di macchina? Come ti spiega il risultato?

Il fenomeno osservato è chiamato *eliminazione* e avviene quando vengono sottratti numeri molto simili fra loro. Naturalmente una eliminazione è presente in entrambi i calcoli dell'esercizio, ma data la differenza fra gli ordini di grandezza, nel primo calcolo è molto peggiore che nel secondo.

Esercizio 4: Calcolare in MATLAB la seguente somma

$$\sum_{k=1}^{10^4} \frac{1}{k},$$

confrontare il risultato ottenuto eseguendo la somma in verso naturale rispetto a quello ottenuto sommando da 10^4 fino a 1. Quale dei due risultati ci aspettiamo sia più affidabile? Usando la sintassi di MATLAB per le operazioni fra array il calcolo può essere svolto in modo molto naturale in un solo comando...come?

[3] http://www-users.math.umn.edu/ arnold/disasters/

Calcolo simbolico Avendo brevemente esplorato le difficoltà pratiche e teoriche del calcolo numerico, viene da chiedersi se non ci sia qualche modo per evitare i problemi visti, cioè riuscire a fare calcoli in modo automatico, ma senza dover fare troncamenti, rischiare overflow, underflow, eliminazioni... In effetti esistono diversi sistemi di calcolo automatico *simbolico*, cioè che implementano l'algebra del campo reale. Chi fosse curioso può sperimentare con uno di questi sistemi disponibile nella libreria `sympy` per Python

```
In [1]: from sympy import *

In [2]: sqrt(2)**2 - 2

Out[2]: 0
```

Questi sistemi però, per quanto interessanti, non sostituiscono il calcolo numerico.

2 Approssimazioni di π

Metodi classici Uno dei più antichi metodi per l'approssimazione di π è il metodo di esaustione formulato da Archimede di Siracusa. L'idea è di approssimare il perimetro di una circonferenza con il perimetro di un poligono ad essa inscritto, aumentando indefinitamente il numero di lati, la successione di poligoni esaurisce la circonferenza e permette di calcolare π.

Possiamo formulare l'algoritmo usando un linguaggio molto più moderno come segue: data un circonferenza unitaria un poligono di n lati ad essa inscritto ha i vertici a distanza $\frac{2\pi}{n}$ e lati di lunghezza $l = \sin\left(\frac{\pi}{n}\right)$. Il perimetro di tale poligono è

$$p_n = n \sin\left(\frac{\pi}{n}\right). \tag{2}$$

Ricordando che $\frac{\sin x}{x} \to 1$ per $x \to 0$ ricaviamo che

$$\lim_{n\to\infty} p_n = \pi.$$

Ad essere sinceri questa formulazione è fatta prendendo parecchie scorciatoie per fornire una formula semplice da implementare: l'uso delle funzioni trigonometriche (e del valore in doppia precisione di π) non sono chiaramente coerenti con il metodo di bisezione antico, per non parlare del delicato limite $\sin(x)/x$... Per questo motivo i risultati ottenuti con questa successione sono un po' più precisi rispetto a quelli ottenuti con il vero metodo di esaustione, chi fosse interessato ad una implementazione in MATLAB del metodo antico può fare riferimento a questa pagina[4]

Esercizio 5: Calcolare in MATLAB la successione dei perimetri utilizzando la formula in (2) e confrontare i risultati con il valore di `pi` restituito da MATLAB .

[4] https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/29504/versions/23/previews/html/

Il metodo di esaustione non era ovviamente nuovo al tempo di Archimede, ma fu Archimede il primo ad ottenere una buona approssimazione di π con questo metodo: calcolando il perimetro di un poligono a 96 lati Archimede ottenne la stima $3.1408 < \pi < 3.1429$[5]. Per i successivi circa 2000 anni idee simili furono applicate portando all'estremo delle sue possibilità il metodo di esaustione: nel Sedicesimo secolo François Viète calcolò il perimetro di un poligono di 393 216 lati, e pochi anni più tardi il matematico olandese Ludolph van Ceulen calclò il perimetro di un poligono a 2^{62} (fate il conto se volete avere un'idea...), che diede 35 cifre corrette dello sviluppo decimale di π.

A questo punto nello storia siamo agli albori del calcolo, e i suoi padri, Leibniz e Newton, non mancarono di usare il nuovo strumento per affrontare il problema del calcolo di π.

Attraverso un'espansione in serie di potenze si trova al formula di Leibniz[6]

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots \tag{3}$$

Esercizio 6: Implementare in MATLAB il calcolo di π mediante formula di Leibniz (3), confrontare i risultati arrestando le somme a diversi lavelli e con il valore di `pi`.

Osserviamo però che questa serie ha termine generale a segni alterni, fatto che potrebbe rallentare la convergenza. Alla luce inoltre delle problematiche già rilevate relativamente alla cancellazione numerica, anche nell'ottica del calcolo automatico, questa formula può essere migliorata. La correzione fu proposta da Newton nella seguente formula:

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{2^k (k!)^2}{(2k+1)!} \tag{4}$$

Esercizio 7: Implementare in MATLAB la formula di Newton (4) per il calcolo di π. Come risolviamo il problema della sommatoria infinita nell'ambito del calcolo numerico?

Hint. *Attenzione ai fattoriali e* `realmax`*...!*

Metodo Monte Carlo Pensando al significato geometrico di π possiamo osservare che il rapporto fra l'area di un cerchio di raggio unitario e l'area del quadrato ad esso circoscritto risulta $\frac{\pi}{4}$: se avessimo una stima dell'area del cerchio saremmo quindi in grado di approssimare π.

Se immaginiamo di lanciare casualmente una freccetta sul quadrato, la probabilità che essa cada all'interno del cerchio è proporzionale al rapporto fra l'area del cerchio

[5] Come già accennato provando a calcolare (2) con $n = 96$ otteniamo un risultato un po' più preciso.

[6] Lasciamo perdere la nomenclatura e le questioni storiche perché la vicenda è complicata... Ma anche Newton trovò la stessa formula.

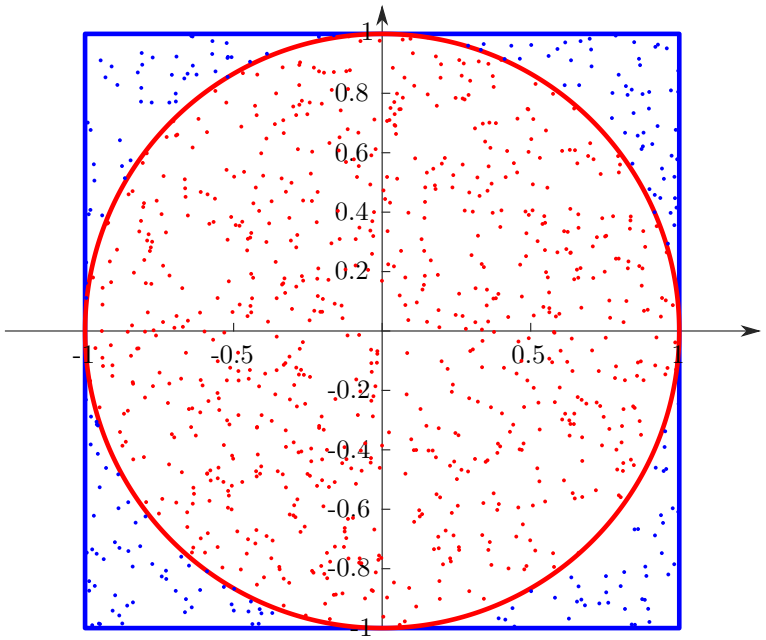


Figura 1: Cerchio di raggio unitario iscritto in un quadrato. L'area del cerchio è pari a π, quella del quadrato 4. I puntini rossi corrispondono alle "freccette" cadute all'interno del cerchio.

e quella del quadrato. Formalizzando questo procedimento si ottiene il *metodo Monte Carlo* per la stima degli integrali.

Il punto in cui cade la freccetta può essere generato come coordinate cartesiane casuali[7] fra -1 e 1:

$$x = -1 + 2 \cdot \texttt{rand}, \quad y = -1 + 2 \cdot \texttt{rand} \tag{5}$$

e il punto con queste coordinate cadrà all'interno del cerchio di raggio 1 se

$$x^2 + y^2 = \texttt{norm([x,y], 2)} \leq 1. \tag{6}$$

Naturalmente eseguendo un solo lancio avremo che la freccetta potrà o meno essere caduta ll'interno del cerchio, ma questo non dice molto... Effettuando tanti tentativi però (in fig. 1 1000 punti) potremo però considerare la frequenza con cui i punti casuali cadono nel cerchio al posto della probabilità, e quindi come stima per $\frac{\pi}{4}$; cioè lanciando N "freccette"

$$\frac{\pi}{4} \approx \frac{\#\{\texttt{[x,y]} : \texttt{norm([x,y],2)} <= 1\}}{N} \tag{7}$$

Esercizio 8: Implementare in MATLAB l'approssimazione di π mediante il lancio delle freccette descritta sopra.

[7] La function `rand` di MATLAB restituisce un numero reale estratto casualmente fra 0 e 1.

Soluzione: Una implementazione semplice è fornita nello script di seguito

```
clear; close all; clc

Nf = 1000; % Numero lanci
Nin = 0; % Niumero freccette nel cerchio
pi_est = zeros(Nf ,1);

for k=1:Nf
    % La posizione casuale della freccetta ha coordinate x e y
    x = -1+2*rand;
    y = -1+2*rand;

    % Contiamo quante freccette sono cadute all'interno del cerchio
    if norm([x,y], 2) <= 1
       Nin = Nin + 1;
    end

    % Per stimare pi
    pi_est(k) = Nin / k * 4;
end
fprintf("Ultima stima %f \n", pi_est(end))
plot(1:Nf, pi_est)
title("Successione delle approssimazioni")
xlabel("Tentativi")
```

Ora che abbiamo approssimato π in diversi modi potremmo chiederci se gli algoritmi sono tutti equivalenti, o se ce ne siano di migliori...

Esercizio 9: Una volta fissato un `Nmax`, calcolare le approssimazioni di π suggerite nei precedenti esercizi per `n = 1 : Nmax` salvando i rispettivi risultati per ogni `n`. Confrontare gli andamenti dei valori ottenuti in forma grafica. Cosa possiamo dire dei quattro algoritmi? Prestare particolare attenzione al confronto fra la formula di Leibnitz e il metodo di esaustione, potrebbe essere inaspettato.

Esercizio 10: A partire dai risultati dell'esercizio precedente, produrre un grafico di convergenza dei quattro algoritmi.

Hint. *Un grafico di convergenza in genere mostra qualche misura dell'errore, ad esempio il valore assoluto, in funzione del parametro di discertizzazione, in questo caso il numero di passi o di termini nelle sommatorie.*

Hint. *Per avere risultati utili dal metodo Monte Carlo potrebbe essere necessario aumentare di* 10 *o* 100 *punti ad ogni iterazione. Questo metodo infatti ha una convergenza piuttosto lenta (in compenso ogni iterazione non richiede altro che di estrarre due numeri casuali...).*

3 Radici di equazioni non lineari

Come noto la posizione verticale $y(t)$ di un corpo massiccio che cade da una altezza iniziale y_0 è

$$y(t) = y_0 - \frac{1}{2}gt^2, \tag{8}$$

dove g è l'accelerazione gravitazionale a $g \approx 9.80665\text{m/s}^2$. Il tempo di caduta di un corpo lasciato cadere da una altezza $y_0 = 1\text{m}$ può essere calcolato facilmente come radice dell'equazione non lineare

$$y(t) = 0 \tag{9}$$

e quindi

$$t_{\text{caduta }1,2} = \mp\sqrt{\frac{2}{g}} \tag{10}$$

dove la radice negativa rappresenta l'istante (nel passato) in cui il corpo dovrebbe essere stato lanciato per trovarsi in y_0 a tempo $t = 0$ con velocità nulla, e quella positiva l'istante (nel futuro) in cui il corpo cade nuovamente a terra ($y(t) = 0$). Ci concentriamo quindi sulla radice positiva: $t_{\text{caduta}} = t_{\text{caduta }2}$.

Esercizio 11: Dopo aver aperto un nuova finestra grafica in MATLAB , disegnare in blu il grafico della funzione (8) fra 0 e $t_{\text{caduta}} + 1$ e marcare con un asterisco rosso il punto in in cui il grafico interseca l'asse delle ascisse. Assegnare un titolo alla figura e dei label significativi agli assi.

Dal grafico si vede bene che, nell'intervallo considerato, la funzione cambia segno in particolare ha segni opposti agli estremi. Possiamo quindi trovare la radice dell'equazione non lineare mediante il metodo di bisezione.

Esercizio 12: Implementare in MATLAB il metodo di bisezione per il calcolo della radici di equazioni non lineari.

Esercizio 13: Testare l'algoritmo implementato nell'esercizio precedente sull'equazione (9). Quante iterazioni sono necessarie per raggiungere $y(t) \approx 0$ rispettivamente in semplice e doppia precisione? Confrontare i due risultati con il valore esatto in (10).

Consideriamo ora il polinomio

$$p(x) = x^3 - 8x^2 - x + 8 \tag{11}$$

Siccome il polinomio è di terzo grado, per il teorema fondamentale dell'algebra possiamo avere fino a tre radici reali (sempre tre radici complesse!). Per usare il metodo di bisezione dobbiamo trovare delle coppie di punti in cui la funzione abbia segni opposti...

Esercizio 14: Dopo aver prodotto un grafico del polinomio dovremmo avere un'idea di dove si trovino le radici. A partire da questa osservazione fissare tre coppie di punti

$(a_i, b_i)_{i=1,2,3}$ per cui $p(a_i)p(b_i) < 0$ e usare il metodo di bisezione per calcolare le radici del polinomio (11).

Cosa succede rispettivamente fissando

$$(a = -10, b = 10), \quad (a = -5, b = 10), \quad (a = -1, b = 5)$$

e perché?

Esercizio 15: Tenere tutto il script separati rende il lavoro sul codice e gli esperimenti molto macchinosi. Produrre una `function` MATLAB che mediante il metodo di bisezione calcoli le radici di una funzione data in input (come `handle`). Altri input che ragionevolmente vorremo consentire saranno gli estremi del dominio in cui cercare la radice, la tolleranza, il numero massimo di iterazioni (piuttosto importante per non rimanere intrappolati loop infiniti)...

Metodo di Newton Un altro metodo molto importante per il calcolo delle radici di equazioni non lineari è il metodo di Newton, in questo caso ogni iterazione prende la forma

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}, \quad n = 1, \dots \tag{12}$$

con $x^{(0)}$ fissato, possibilmente vicino alla radice...

L'idea del metodo di Newton è di approssimare la radice di f con la radice $x^{(n+1)}$ di una funzione lineare, in particolare la retta tangente al grafico di f nel punto $x^{(n)}$, come in fig. 2.

Esercizio 16: Implementare il metodo di Newton in una function MATLAB seguendo le indicazioni nell'esercizio 15, in questo caso come input servirà avere la derivata della funzione (che andrà calcolata a mano...) e un solo punto di partenza $x^{(0)}$ al posto degli estremi dell;intervallo.

Testare la `function` prodotta seguendo le indicazioni nell'esercizio 14. A che radice converge la successione fissando $x^{(0)} = 0$? E con $x^{(0)} = 0.1$?

Julia sets e altri mostri Abbiamo visto in conclusione all'esercizio 16 che spostando di un poco i punto iniziale nell'algoritmo di Newton andiamo a convergenza a radici diverse. È ragionevole pensare che se iniziamo molto vicino ad una radice, l'algoritmo ci porterà verso quella. In effetti è abbastanza vero, ma non del tutto, come possiamo vedere dal risultato del seguente

Esercizio 17: Passando in rassegna tutto l'intervallo $[-5, 10]$ con passo 0.01, in corrispondenza ogni punto iniziale $x^{(0)}$, plottare la radice raggiunta, a partire da quel punto, mediante il metodo di Newton. Cosa osserviamo nelle vicinanze di 0?

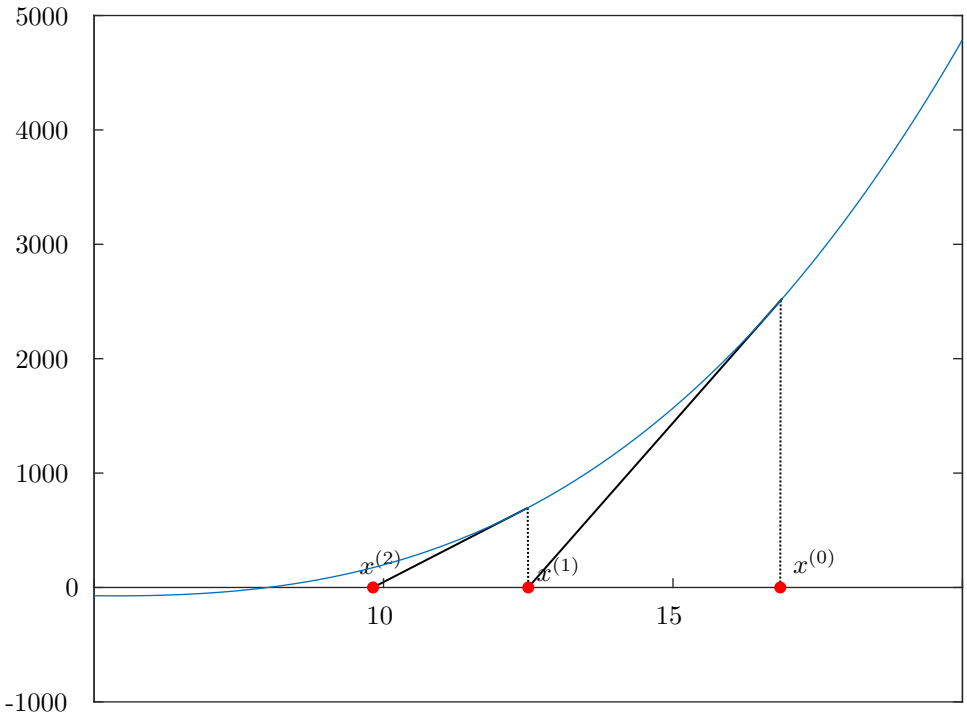


Figura 2: Sketch del significato geometrico del metodo di Newton.

Una procedura analoga al precedente esercizio 17 effettuata però su una regione del piano complesso permette di costruire figure di straordinaria complessità note come insieme di di Julia[8] di cui alcuni esempi sono mostrati sotto in

Convergenza I due metodi visti per la ricerca delle radici di equazioni non lineari sono piuttosto diversi, possiamo ora confrontarli e studiarne le caratteristiche:

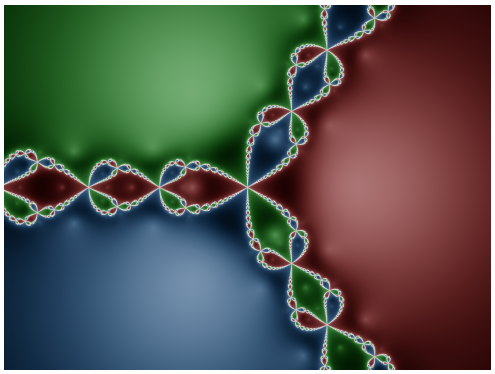
Esercizio 18: Usando un intervallo $[a, b]$ e un punto $x^{(0)}$ a piacere, confrontare le successive approssimazioni della radice $x = 1$ del polinomio (11) in un grafico che mostri le due approssimazioni in funzione del numero di iterazioni.

Produrre un grafico di convergenza che mostri l'errore in valore assoluto nelle approssimazioni in funzione del numero di iterazioni.

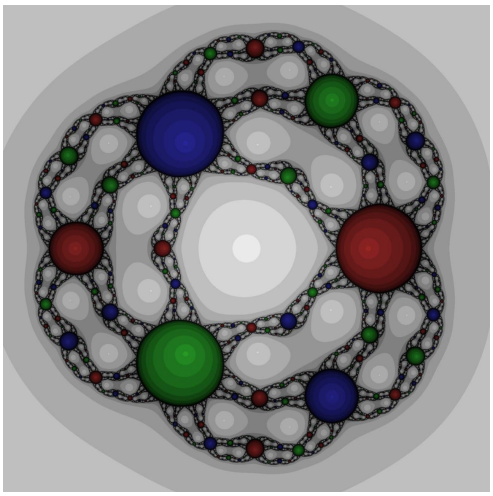
Riferimenti bibliografici

[1] «IEEE Standard for Binary Floating-Point Arithmetic». In: *ANSI/IEEE Std 754-1985* (1985), pp. 1–20. DOI: 10.1109/IEEESTD.1985.82928.

[8] Da Gaston Maurice Julia (3 February 1893 – 19 March 1978).



(a) $f(x) = x^3 - 1$ (da Wikipedia)



(b) $f(x) = x^2(x^3 - 1)$ (immagine propria)

Figura 3: Insiemi di Julia: ad ogni punto della regione del piano complesso inquadrata viene assegnato un colore in base alla radice a cui il metodo di Newton converge.