Matematica e Statistica con R

Federico Comoglio e Maurizio Rinaldi

11 novembre 2015

Indice

Prefazione

Queste dispense devono la loro esistenza al corso di R che ormai da alcuni anni accompagna il corso di Matematica e Statistica presso il Dipartimento di Scienze del Farmaco e si rivolgono anche agli studenti di Biotecnologie presso il Dipartimento di Scienze della Salute di Novara. Esse seguono i principali temi del corso e fanno uso consistente di esempi. La maggior parte degli esempi sono stati pensati da noi, ma non rivendichiamo che essi siano esclusivi o abbiano una rilevanza particolare.
Il materiale è al momento provvisiorio e richiama anche alcune nozioni di matematica o di statistica laddove queste siano ritenute utili alla comprensione. Inoltre, numerosi punti sono ancora carenti di adeguate referenze, che verranno aggiunte in maniera esauriente nelle prossime versioni.
Queste dispense sono state realizzate in LaTeX, con praticamente tutte le figure generate *on the fly* in R e incluse nel testo utilizzando `Sweave`.
Nonostante abbiamo speso tempo nel pensare ai concetti da includere ed organizzarne il contenuto, ci assumiamo le nostre responsabilità per errori che sono ancora presenti nel testo, figure o codice. A questo proposito ma non solo, ogni vostro suggerimento è ben accetto e potenzialmente molto prezioso.

Grazie,
Novara, 11 novembre 2015
Maurizio e Federico

Capitolo 1

Preambolo

R è un programma *freeware* (di distribuzione gratuita) e *open-source* (il cui codice sorgente può essere modificato e redistribuito) distribuito dalla "R foundation", utilizzato per il calcolo statistico ed in grado di svolgere calcoli matematici ed algebrici (sicuramente di livello superiore ad una normale calcolatrice tascabile) ma che non rientrano nello scopo primario di R. La versione cui sono riferite queste dispense è la 3 2.1. Assumiamo che lo studioso abbia scaricato dal sito `http://www.r-project.org/` tale versione di R e la abbia installata sul suo computer.

1.1 Introduzione

Il programma si presenta all'utente con una interfaccia per l'inserimento dei dati, una finestra chiamata "R console". Per esempio digitando a console (dopo il *prompt* >) il comando

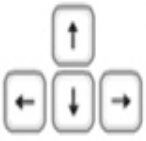
```
> 1+1
```

e premendo il tasto di invio appare come risposta

```
[1] 2
```

Oltre alla console R dispone di finestre di altra natura, per esempio finestre grafiche e finestre di *help*.
Iniziamo a descrivere alcune peculiarità di R.
L'insieme dei comandi eseguiti costituisce la storia o *cronologia* (*history*) di una sessione di lavoro. È possibile muoversi all'interno della cronologia, cioè richiamare i comandi precedenti e successivi utilizzando le frecce ↑ e ↓ della tastiera.

Possiamo scrivere diversi comandi prima di eseguirli (tasto di invio): basta separarli con un ";".
Si ricordi sempre che R è *c*ase sensitive ossia vi è differenza tra lettere minuscole e maiuscole. Questo è particolarmente importante in quanto ad esempio le variabili `a` ed `A` rappresentano entità distinte.



Editori esterni

E' molto comodo scrivere le porzioni di codice che si intendono eseguire in R su un file di testo esterno in modo da disporre a fine sessione di un listato dei comandi usati (con eventuali

commenti), scevro da errori e da operazioni superflue o ripetute. Gli editor esterni hanno in particolare la proprietà di

- Riconoscere la sintassi di R
- Interagire con R, consentendo l'esecuzione di comandi direttamente dall'editor.

Ne menzioniamo quattro di semplice utilizzo: Rstudio, Tinn-R, Komodo Edit e TextWrangler, i cui dettagli sono presenti in tabella 1.1.
A parte Tinn-R e RStudio , gli altri due editori necessitano di estensioni (*plug-in*) dedicate

Editor	Sistema Operativo	Website
RStudio	Mutlipiattaforma	`http://www.Rstudio.org`
Tinn-R	Windows	`http://www.sciviews.org/Tinn-R`
Komodo Edit	Multipiattaforma	`http://www.activestate.com/komodo-edit`
TextWrangler	Mac OSX	`http://www.barebones.com/products/textwrangler/`

Tabella 1.1: Alcune informazioni sugli editor citati nel testo (IDE)

per riconoscere la sintassi di R, segnalare errori, fornire suggerimenti ed eseguire comandi in R direttamente dall'editor. Il *plug-in* per Komodo Edit si chiama SciViews-K (`http://www.sciviews.org`), mentre il *plug-in* per TextWrangler è scaricabile dal sito `http://www.smalltime.com/gene/R.plist`. Quest'ultimo deve essere poi inserito nella directory (librerie globali): `~/Library/Application Support/TextWrangler/Language Modules/`.
A livello di sviluppo sono invece disponibili editori dedicati, che forniscono numerose funzioni di diagnostica e *debug* del codice. Senza qui addentrarci nei dettagli, Emacs (`http://www.gnu.org/software/emacs/`) con il *plug-in* ESS (Emacs *Speak Statistics*, `http://ess.r-project.org/`) ha trovato negli ultimi anni largo impiego anche come editor per R.
Un discorso più esteso spetta ad RStudio che presenta un ambiente di sviluppo integrato particolarmente adatto a fini didattici e come già detto è *free* e multipiattaforma.
Adotteremo in questo libro RStudio per la facilità d'uso e per il fatto che consente una visualizzazione completa contestuale di diversi aspetti di una sessione di lavoro. RStudio ha inoltre una versione server che è stata resa disponibile all'indirizzo `http://rstudio.med.unipmn.it:8787/` per gli studenti di questo corso[1]

1.2 Introduzione ad Rstudio

All'apertura di RStudio appaiono diversi pannelli mostrati in Figura 1.1

Il pannello fondamentale (a sinistra nella figura) è la classica console di R. Ci sono poi 2 altri 2 pannelli a sinistra. Quello superiore con 2 linguette `Environment` e `History` e quello inferiore con 5 linguette `Files, Plots, Packages,Help, Viewer` il cui significato sarà presto chiarito.

La barra dei menu fornisce diversi menu a tendina che scopriremo procedendo nello studio.

Script di codice

Dovendo scrivere porzioni di codice si può aprire un nuovo documento selezionando dal menu di R in corrispondenza a `File` il tab `New Document` come in figura 1.2

[1]Ringraziamo il Dottor Valter Rolando per essersi adoperato in tal senso

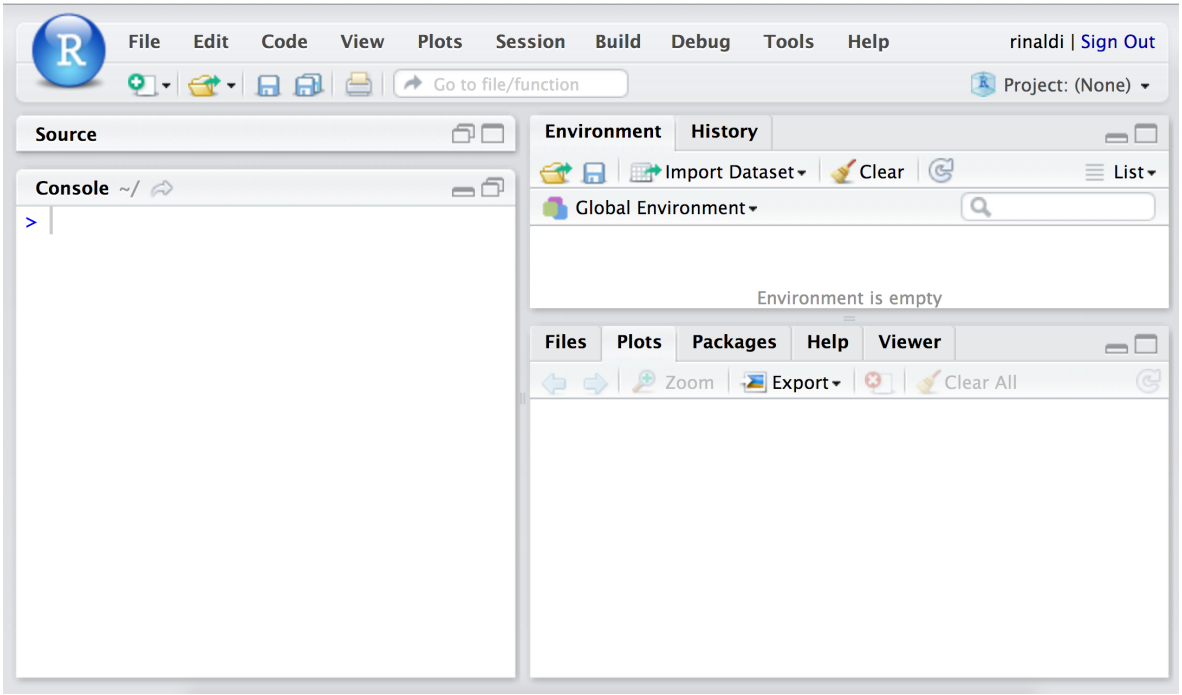


Figura 1.1: Finestra iniziale di RStudio

In RStudio invece nel menu a relativo a `File` si può selezionare `New file` e aprire file di diversa natura, in particolare porzioni di codice di RStudio (R Script) come in figura 1.3. Si provi per esempio ad aprire uno script di RStudio e ad inserire il comando 1+1.

Usando le frecce o le scorciatoie da tastiera nel menu a tendina in corrispondenza a `code` si possono inviare i comandi a R per la loro esecuzione.

Il file va poi salvato in una cartella opportuna con estensione `.R`.

Help e guida on-line

Una qualunque richiesta di informazioni può essere effettuata digitando il comando: ?*argomento* (o `help`(*argomento*)). Ad esempio ?`sin` consente di ottenere informazioni sulla funzione seno e altre funzioni trigonometriche in quanto la guida è completamente indicizzata. È anche possibile utilizzare il menu a tendina (*help*), con modalità diverse a seconda del sistema operativo.
In RStudio basta invece clicare sul menu Help selezionare la casellina Home e scrivere il comando al quale si è interessati.

Inserire i commenti

I commenti sono frasi, annotazioni o codice da non eseguire momentaneamente molto utilizzati in programmazione. L'introduzione di un commento in R è possibile mediante il simbolo `#` (cancelletto singolo o *hash*). Per evitare che un commento venga cancellato durante il normale *editing* del testo e che esso venga eseguito si consiglia di raddoppiare i simboli (`##`). Ad esempio:

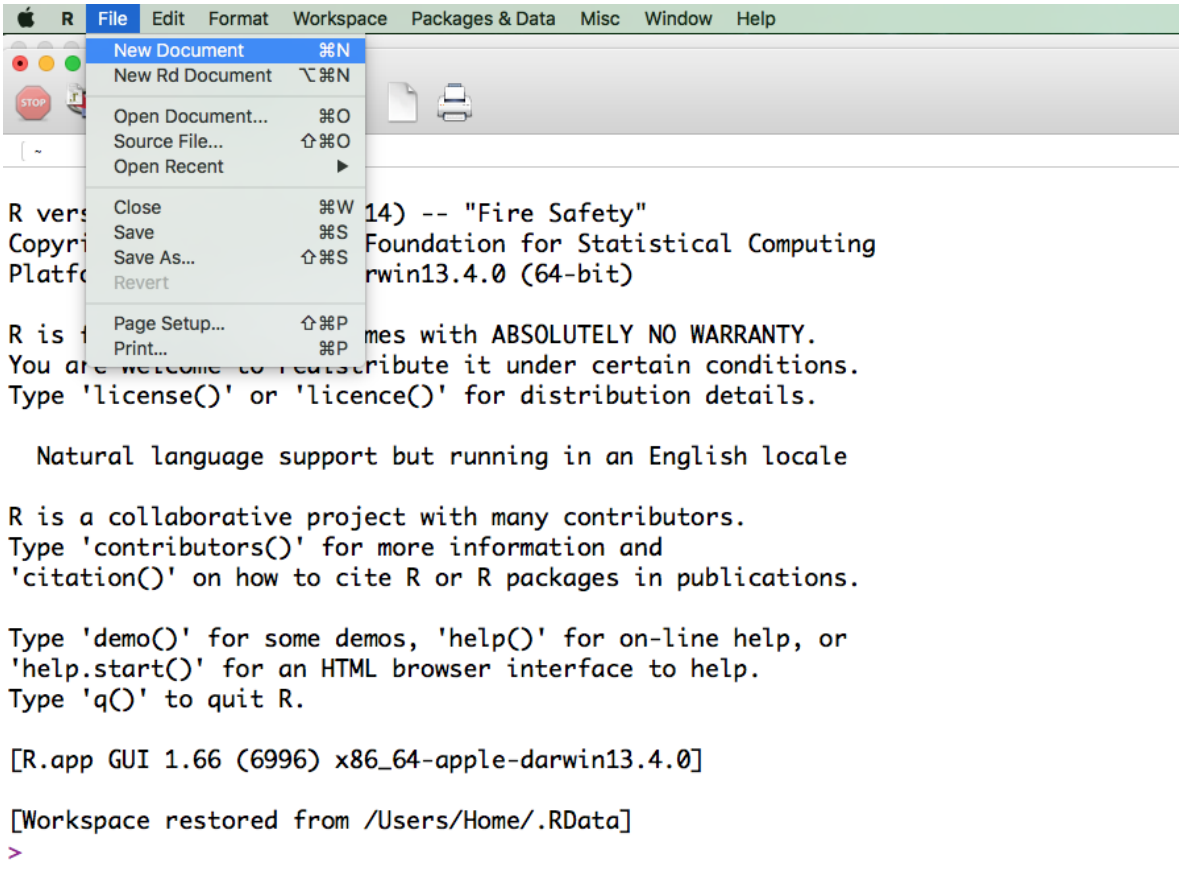


Figura 1.2: Apertura di uno script di R in R .

```
> 1+1 # dovrebbe fare 2
[1] 2
```

non produrrà errore perché riconoscerà la stringa scritta dopo `#` come commento.

Quando un commento è utile e quando è superfluo?

A prescindere dalle preferenze personali, un commento dovrebbe essere inserito quando la porzione di codice che si sta considerando risulta a noi non ovvia, quando le dimensioni, tipo e ruolo delle variabili coinvolte non sono ovvie o semplicemente quando si ha bisogno di introdurre una annotazione che renda più facile la lettura del codice non solo a noi stessi ma anche ad altri. Quest'ultimo punto è di fondamentale importanza quando si vuole condividere il codice con altri membri della comunità scientifica. Per capire se i commenti che inserite sono superflui o mancanti:

Ad un certo punto del corso salvate una porzione di codice (o una funzione) commentata in un file `.R`, lunga almeno 25 righe di codice. Aggiungete la data corrente al nome del file e

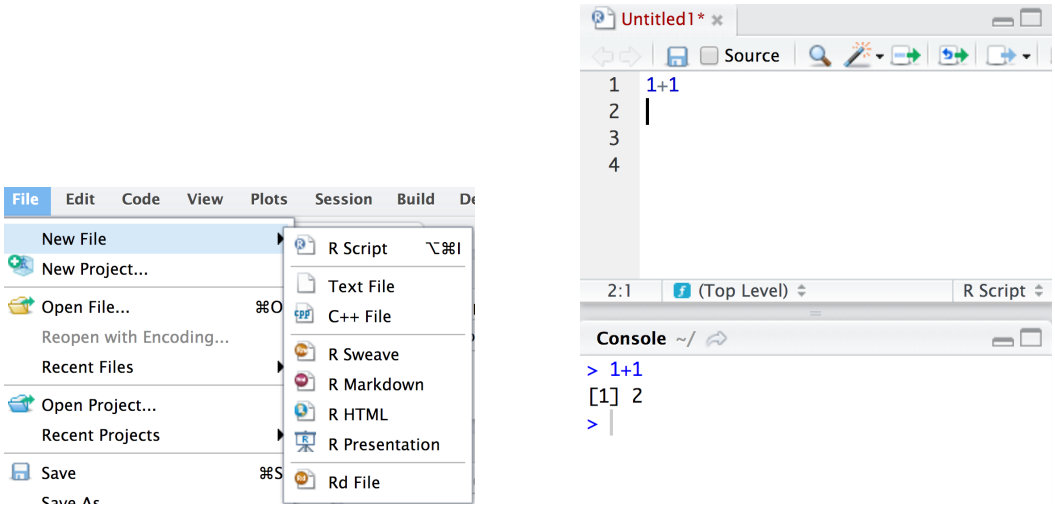


Figura 1.3: Apertura ed esecuzione di uno script di R in RStudio

in un periodo compreso tra 6 e 12 mesi semplicemente aprite il file e rileggetene il codice. Se siete in grado di capirne il contenuto, avete probabilmente fatto buon uso dei commenti.

Working directory

Il comando

```
> getwd()
```

indica la directory di lavoro mentre il comando

```
> setwd("indirizzo")
```

imposta la *working directory* nella posizione specificata da `"indirizzo"`. Lavorando con RStudio per impostare la directory si può usare il menu a tendina `Session`; è in particolare possibile posizionare la `working directory` in corrispondenza al file di codice sorgente (`To Source File location`).

Pacchetti

Oltre all'aggiornamento costante del programma, lo sviluppo di R consiste nello creazione e aggiornamento continui da parte della comunità mondiale di una serie di pacchetti addizionali. Un pacchetto contiene in particolare un insieme di comandi finalizzati alla realizzazione di un determinato compito: esistono per esempio pacchetti per la risoluzione di equazioni differenziali, pacchetti per l'analisi statistica di *microarray*, pacchetti per il *clustering* di dati. Per non appesantire R i pacchetti (con poche eccezioni) vengono caricati solo se richiesto dalle esigenze dell'utente. L'utente deve quindi inizialmente localizzare (per esempio via web) il pacchetto che lo interessa e scaricarlo localmente. Per fare questo deve selezionare il sito CRAN da cui scaricarlo e poi (direttamente dal menu a tendina di R) scaricarlo (con annesse eventuali *dependencies*). Una volta scaricato in locale il pacchetto può essere messo a disposizione dell'utente con il comando

```
> library("nome.pacchetto")
```

Se si lavora all'interno di RStudio il caricamento dei pacchetti è più semplice: in uno dei pannelli è presente il menu `Packages` dal quale l'utente può scaricare il pacchetto usando il sottomenu `Install` e caricarlo all'interno di R semplicemente barrandone la casellina relativa.

Capitolo 2

Introduzione ad R

2.1 Variabili

Una variabile è una porzione di memoria caratterizzata da un nome che contiene dati che possono essere modificati durante l'esecuzione del programma.
È possibile definire una qualunque variabile mediante l'assegnazione di un valore ad essa.
Sono valide entrambe le notazioni seguenti: *variabile* ← *valore* e *valore* → *variabile*; ad esempio:

```
> a<-2
```

associa il valore 2 alla variabile `a`. Per comporre le frecce si usano i segni "meno" "maggiore" `->`(freccia a destra) o "minore" "meno" `<-`(freccia a sinistra). Si abbia cura di evitare di inserire spazi tra il nome della variabile e la freccia, così come tra la freccia ed il valore assegnato. Il programma per assegnazioni semplici non invia alcun messaggio di errore, ma si possono verificare problemi quando vi siano più membri costituenti il valore assegnato. Rivolgere **sempre** la freccia dal valore alla variabile, infatti essa è una scatola in grado di contenere il valore voluto. Ad esempio non è valida la stringa: *variabile* → *valore* o *valore* ← *variabile*.
Non lasciare spazi vuoti nel definire il nome della variabile, è possibile invece utilizzare il sottotratto (*underscore*) _ ad esempio definendo: `variabile_uno`, **non** `variabile uno`. Si può anche usare il punto e scrivere `variabile.uno`. I nomi delle variabili sottostanno ad alcune regole. Per esempio `variabile1` è un nome lecito mentre non lo è `1variabile`.
L'assegnamento di uno stesso valore a più variabili (o costanti) in un unico comando è possibile con una sintassi del tipo:

```
> a<-b<-c<-5
```

L'assegnazione può avvenire anche usando il segno '=', ad esempio

```
> a=5
```

o il comando

```
> assign("a",5)
```

Ogni variabile è richiamabile scrivendo il suo nome. Ad esempio: `a` richiama la variabile `a` appena definita e mostra il valore di `a` in output, ovvero 5[1], Per rimuovere una variabile o un

[1]Spesso dopo aver scritto un comando ad esempio `a<-5` ci si interroga sull'assenza di output. In realtà il comando è stato eseguito. Per verificarlo basta scrivere `a` ed inviare. In modo alternativo il comando `(a<-5)` oltre a definire l'associazione di 5 ad `a` ne stampa il valore.

generico oggetto si utilizza il comando:

$$\texttt{rm}(\textit{name}),$$

ossia *remove memory*, *name* è il nome della variabile da cancellare. Ad esempio:

```
> rm(a)
> a=5;a
[1] 5
```

svuota il contenuto della memoria `a`. Per conoscere gli oggetti presenti nella sessione corrente basta il comando

```
> ls()
[1] "a" "b" "c"
```

o in modo equivalente

```
> objects()
[1] "a" "b" "c"
```

L'insieme degli oggetti presenti in una sessione costituisce il *worksapce* o *environment* (ambiente di lavoro) il cui contenuto è facilmente visualizzabile in RStudio cliccando su `Environment`. Scoprire come eliminare gli oggetti.

2.2 Costruzione di vettori

Un vettore (o *array* monodimensionale) può essere visto come un casellario, le cui caselle sono dette celle (*slot*) del vettore stesso. Tutte le celle sono variabili di uno stesso tipo, detto tipo base del vettore. Si possono così definire vettori di numeri, di caratteri e vettori logici. In R l'assegnamento della dimensione di un vettore non è obbligatorio. Un vettore può essere definito utilizzando il comando `c` di sintassi generale:

$$\texttt{c}(\textit{valore}_1, \textit{valore}_2, \ldots, \textit{valore}_n)$$

Ad esempio

```
>  c(1,5,9,11,17,34)
[1]  1  5  9 11 17 34
```

Si può associare il vettore ad una variabile con la notazione:

$$\texttt{c}(\textit{valore}_1, \textit{valore}_2, \textit{valore}_n) \rightarrow \textit{variabile}$$

Per esempio:

```
> c(1,5,9,11,17,34)->vettore
```

Due operazioni fondamentali sono l'estrazione di una cella

```
> i=4;vettore[i]
```

```
[1] 11
```

e la sostituzione

```
> vettore[i]<-5
```

Si noti anche che se scriviamo

```
> vettore[i]<-"testo"
> vettore
[1] "1"     "5"     "9"     "testo" "17"    "34"
```

abbiamo anche cambiato il tipo di base del vettore mentre, se scriviamo

```
> vettore[i]<-11
> vettore
[1] "1"  "5"  "9"  "11" "17" "34"
```

il vettore non viene restaurato. Per farlo occorre in questo caso scrivere

```
> as.numeric(vettore)->vettore
```

Possiamo anche definire celle con indice superiore alla lunghezza

```
> vettore[10]<-3
> vettore
 [1]  1  5  9 11 17 34 NA NA NA  3
```

il vettore viene prolungato fino a comprendere 10 celle. Le celle non assegnate vengono etichettate con `NA` (not assigned). Se invece si scrive

```
> vettore[-3]
[1]  1  5 11 17 34 NA NA NA  3
```

la cella di posizione 3 viene eliminata. Possiamo determinare la lunghezza di un vettore con la funzione `length`:

```
> length(vettore)
[1] 10
```

Possiamo poi ovviamente eseguire operazioni sui vettori

```
> 3*vettore
 [1]   3  15  27  33  51 102  NA  NA  NA   9
```

Il vettore di nome `vettore`, viene moltiplicato per 3, ovvero ciascuna cella viene moltiplicata per 3.

2.2.1 Generazione di sequenze: `rep` e `seq`

È possibile definire sequenze di numeri consecutivi come *n:m*. Ad esempio:

```
> 1:11
 [1]  1  2  3  4  5  6  7  8  9 10 11
```

consente di ottenere la sequenza dei numeri interi da 1 a 11. È anche possibile aggiungere un numero alla sequenza con il comando: `c`(*n:m*, *numero aggiunto*). Per esempio se vogliamo aggiungere il numero 100 alla lista precedente scriveremo:

```
> c(1:11,100)
 [1]   1   2   3   4   5   6   7   8   9  10  11 100
```

Nota: l'aggiunta è posizionale, cioè rispetta l'ordine con cui compaiono gli elementi.

È inoltre possibile generare liste costanti di valore k con il comando: `rep`(*k,n*) Se vogliamo una lista i cui elementi sono tutti 3 per un totale di 10 elementi scriveremo:

```
> rep(3,10)
 [1] 3 3 3 3 3 3 3 3 3 3
```

Infine il parametro `each=`*n* consente di ripetere ciascun elemento della sequenza *n* volte. Ad esempio:

```
> rep(c(3,2,1),each=3)
[1] 3 3 3 2 2 2 1 1 1
```

Ripete 3 volte il numero 3, 3 volte il numero 2 e 3 volte il numero 1. Diverso è il risultato con il comando

```
> rep(c(3,2,1),times=5)
 [1] 3 2 1 3 2 1 3 2 1 3 2 1 3 2 1
```

dove si ha la ripetizione `times` volte dell'intero vettore. Cosa succede se 2 vettori hanno lunghezza diversa? In matematica tradizionalmente questa somma non è definita. In R invece vale la regola del riciclaggio (*recycling rule*). Se dobbiamo sommare i vettori che seguono

$$\begin{array}{l} (1,2,3,4,5,6,7,8)+ \\ (3,4,5) \qquad\qquad\quad = \\ \hline \end{array}$$

si procede cosí: la somma si calcola riciclando il vettore più breve

$$(1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8)+ \tag{2.1}$$
$$(3,\ 4,\ 5,\ 3,\ 4,\ 5,\ 3,\ 4) = \tag{2.2}$$
$$\text{---------------------} \tag{2.3}$$
$$(4,\ 6,\ 8,\ 7,\ 9, 11, 10, 12) \tag{2.4}$$
$$\tag{2.5}$$

In modo simile se dobbiamo sommare i vettori che seguono

$$\begin{array}{l} (1,2,3,4,5,6,7,8)+ \\ (3) \qquad\qquad\qquad\quad = \\ \hline \end{array}$$

si esegue la somma

$$(1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8)+ \tag{2.6}$$
$$(3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3,\ 3) = \tag{2.7}$$
$$\text{---------------------} \tag{2.8}$$
$$(4,\ 5,\ 6,\ 7,\ 8,\ 9, 10, 11) \tag{2.9}$$
$$\tag{2.10}$$

Ad esempio

```
> 1:8+3:5
[1]  4  6  8  7  9 11 10 12
> 1:8+3
[1]  4  5  6  7  8  9 10 11
```

In pratica il vettore più corto viene riciclato tante volte quanto serve per raggiungere la lunghezza del vettore più lungo. Se la ripetizione non occorre un numero intero di volte allora viene emesso un messaggio di *Warning*.

Il comando `seq` consente di generare una sequenza (vettore) i cui elementi sono collegati da una proprietà caratteristica: sono tutti numeri pari, dispari, di passo n, ecc.
Si definisce una sequenza come: `seq`(*n,m*)
Ad esempio per generare la sequenza dei numeri da 0 a 100 a passo 1 (default) si scriverà:

```
> seq(0,100,1) ##analogo di seq(0,100)
  [1]   0   1   2   3   4   5   6   7   8   9  10  11  12
 [14]  13  14  15  16  17  18  19  20  21  22  23  24  25
 [27]  26  27  28  29  30  31  32  33  34  35  36  37  38
 [40]  39  40  41  42  43  44  45  46  47  48  49  50  51
 [53]  52  53  54  55  56  57  58  59  60  61  62  63  64
 [66]  65  66  67  68  69  70  71  72  73  74  75  76  77
 [79]  78  79  80  81  82  83  84  85  86  87  88  89  90
 [92]  91  92  93  94  95  96  97  98  99 100
```

Possiamo ottenere la sequenza di tutti i numeri dispari da n a m utilizzando il parametro `by`,che consente di impostare il passo nella sequenza. Il comando:

```
> seq(1,100,by=2)
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35
[19] 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71
[37] 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

restituisce tutti i numeri tra 1 e 100 a passo 2, ovvero tutti i numeri dispari compresi nell'intervallo. Per ottenere i numeri pari compresi nell'intervallo possiamo reiterare il comando precedente ma partendo da 0 (e volendo possiamo sottintendere la specifica `by=`)

```
> seq(0,100,2)
 [1]   0   2   4   6   8  10  12  14  16  18  20  22  24  26
[15]  28  30  32  34  36  38  40  42  44  46  48  50  52  54
[29]  56  58  60  62  64  66  68  70  72  74  76  78  80  82
[43]  84  86  88  90  92  94  96  98 100
```

il parametro `length.out` (brevemente `len` o `length` consente di definire il numero totale di elementi della sequenza, ovvero divide l'intervallo da m a n utilizzando un numero di punti pari alla lunghezza assegnata. Ad esempio se nella sequenza di numeri da 1 a 100 volessimo avere in totale 10 numeri scriveremo:

```
> seq(1,100,length=10)
 [1]   1  12  23  34  45  56  67  78  89 100
```

In effetti l'intevallo da 1 a 100 di lunghezza 99 viene diviso da 10 punti in 9 parti di ampiezza 11. Useremo tale parametro per il calcolo degli integrali con il metodo dei rettangoli.

1. Costruire un vettore contenente i numeri da 1 a 15. Identificare almeno due modi per invertire l'ordine degli elementi. Scambiare poi il 5^o e il 6^o ingresso del vettore e stampare a video il risultato.
2. Calcolare $23 + 37/10 - 2^3$
3. Scrivere il vettore $x = (1, 1, 2, 3, 5, 8)$. Stampare il 3^o elemento di x. Determinare la lunghezza di x. Costruire un vettore i cui elementi siano la radice quadrata degli elementi di x.
4. Si consideri la funzione $f(x) = x * \sin(x)$. Si determinino i valori di f negli estremi della suddivisione dell'intervallo $[2, 5]$ in 20 sotto-intervalli di ugual ampiezza.

Capitolo 3

Matematica di Base

In quanto segue immaginiamo di operare direttamente in R anche se avere a disposizione il listato dei comandi può essere utile.

3.1 Numeri, operazioni aritmetiche e variabili

3.1.1 I numeri

Se scriviamo in R qualunque numero la rappresentazione che ne abbiamo è quella di un numero reale dove il punto viene usato come separatore decimale. Inoltre per numeri molto grandi o molto piccoli R ricorre alla notazione scientifica. Un numero "grande" viene scritto come *ae* + *b* (senza spazi) mentre un numero "piccolo" viene scritto come *ae* - *b* (senza spazi) dove a varia tra 1 e 9 e b è l'esponente opportuno (in base 10). Ad esempio:

```
> 0.0001
[1] 1e-04
> 5.4e6
[1] 5400000
```

Si abbia cura di utilizzare il punto '.' invece della virgola ',' come separatore decimale. Il comando `class()` fornisce la natura di un oggetto.

Per quanto riguarda le operazioni aritmetiche fondamentali si usano i simboli: '+', '−','∗' e '/'. Per l'elevamento a potenza il simbolo è '^'.

Si noti che il numero di cifre di cui R dispone è superiore al numero di cifre visualizzate

```
> a=4/3
> a
[1] 1.333333
> a*3
[1] 4
> 1.333333*3
[1] 3.999999
```

3.1.2 Costanti fondamentali

Per costante ci si riferisce ad un valore non modificabile. Ogni linguaggio permette di utilizzare diversi tipi di costanti: numeri, caratteri e stringhe.
Tra le costanti definite in R di interesse generale abbiamo

- π definito come `pi`.
- `e`: numero di Nepero definito come il valore in $x = 1$ della funzione esponenziale `exp(1)`. La notazione è equivalente ad e^1. Non è possibile definire `e` solo con la stringa `exp` senza assegnare l'esponente, in quanto esso è definito come una funzione di x. Ovviamente le costanti sono utilizzabili in qualsiasi calcolo:

  ```
  > duepi<-2*pi
  > duepi
  [1] 6.283185
  ```

  Abbiamo assegnato alla variabile `duepi` valore $2 * \pi$ e l'abbiamo richiamata per visualizzare il risultato.
  Nota bene: se si intende visualizzare il valore e non ad utilizzarlo in calcoli successivi è più comodo ricorrere alla visualizzazione a *console*, come nell'esempio:

  ```
  > 2*pi
  [1] 6.283185
  ```

  in output si ottiene direttamente il risultato del calcolo.

3.1.3 Interi e frazioni

Se scriviamo

```
> class(5)
[1] "numeric"
```

notiamo che il numero intero 5 è stato interpretato come un numero reale. Se desideriamo espressamente ricordarci che 5 è un intero possiamo scrivere 5L

```
> class(5L)
[1] "integer"
```

Possiamo inoltre lavorare con le frazioni caricando il pacchetto `MASS` e usando la sintassi `fractions(m/n)`

```
> library(MASS)
> fractions(1/4)+fractions(2/3)
[1] 11/12
```

- **Esercizio**
  Si calcoli la frazione 17/5+20/7

  Soluzione
  Si determina il denominatore comune prendendo il minimo comune multiplo di 5 e 7.

- **Esercizio**
  Si calcoli la frazione 1/2+7/8

- **Esercizio**
  2/5+6/5

3.2 Funzioni e grafici

3.2.1 Definizione di funzioni

Per definire una funzione f di una variabile *variabile* si utilizza il comando:

$$f \leftarrow \texttt{function}(\textit{variabile})\ \textit{espressione}$$

Ad esempio per definire la funzione

$$f(x) = x^2$$

scriveremo:

```
> f<-function(x) x^2
```

Possiamo calcolare il valore della funzione in un qualunque valore di x con il comando: `f`(*valore*) ad esempio in $x = 6$ si avrà

```
> f(6)
[1] 36
```

Attenzione: nell'esempio che segue viene richiesto il valore di $f(x)$ in $x = 0$:

```
> f<-function(x)  1/x
> f(0)
[1] Inf
```

La dicitura `Inf` corrisponde ad ∞, infatti la funzione esplode in $x = 0$.

3.2.2 Funzioni predefinite

R consente l'utilizzo di funzioni matematiche già definite, come seno, coseno, tangente, le funzioni trigonometriche inverse, esponenziali, logaritmi, radici,ecc. Ecco di seguito le principali, la cui sintassi può essere visualizzata utilizzando l'*help* di R inserendo la parola chiave corretta (ad esempio `?sin`) per ottenere informazioni sul formalismo corretto:

- Funzioni trigonometriche

`cos`	coseno
`sin`	seno
`tan`	tangente
`acos`	arcocoseno
`asin`	arcoseno
`atan`	arcotangente

- Logaritmi.

  `log` è il logaritmo naturale. Ad esempio:

```
> log(3)
[1] 1.098612
```

  `log2` è il logaritmo di base 2

```
> log2(3)
[1] 1.584963
```

  Controlliamo:

```
> log2(3)->a
> 2^a
[1] 3
```

  Sullo stesso principio il logaritmo in base 10 sarà `log10`. Ad esempio:

```
> log10(100)
[1] 2
```

  Per basi diverse si usa il parametro `base` per specificare la base (si noti che il nome del parametro può anche essere omesso).

```
> log(64,base=4)
[1] 3
> log(64,4)
[1] 3
```

- Esponenziali.

  La funzione esponenziale (come già visto) è denotata da `exp`, si abbia cura di indicare `exp` tutto minuscolo. Ad esempio per ottenere il valore numerico dell'espressione e^4 si scrive:

```
> exp(4)
[1] 54.59815
```

- Radice quadrata.

  Si indica come `sqrt` (*square root*), ad esempio per ottenere la radice quadrata di 225 scriveremo:

```
> sqrt(225)
[1] 15
```

  Ovviamente la radice quadrata equivale ad elevare a potenza 1/2 per cui la radice precedente può essere espressa anche come[1]:

```
> 225^(1/2)
[1] 15
```

  Tale procedura può essere utilizzata per radici di indice qualunque, ad esempio:

```
> 16384^(1/7)
[1] 4
```

3.2.3 Grafici

Il comando `curve`, consente di tracciare il grafico di una funzione; `curve`(*funzione*), automaticamente assegna estremi sull'asse x da 0 ad 1 (a meno che non siano già stati definiti). Volendo possiamo specificare gli estremi (`from` e `to`) con il comando:

`curve`(*espressione/nome funzione*, *estremo inferiore* , *estremo superiore*)

Ad esempio volendo tracciare il grafico della funzione $1/x$ nell'intervallo $(0, 2)$ possiamo definire la funzione oppure scriverne direttamente l'espressione. La differenza tra le due scelte dipende dall'uso futuro della funzione, se essa sarà applicata in seguito conviene assegnarle un nome. I comandi:

```
> curve(1/x,0,2)
```

[1] Nel comando che segue non si dimentichino le parentesi.

oppure:

```
> f<-function(x) 1/x
> curve(f,0,2)
```

o anche

```
> curve(1/x,from=0,to=2)
```

generano lo stesso risultato, illustrato in Figura 3.1.

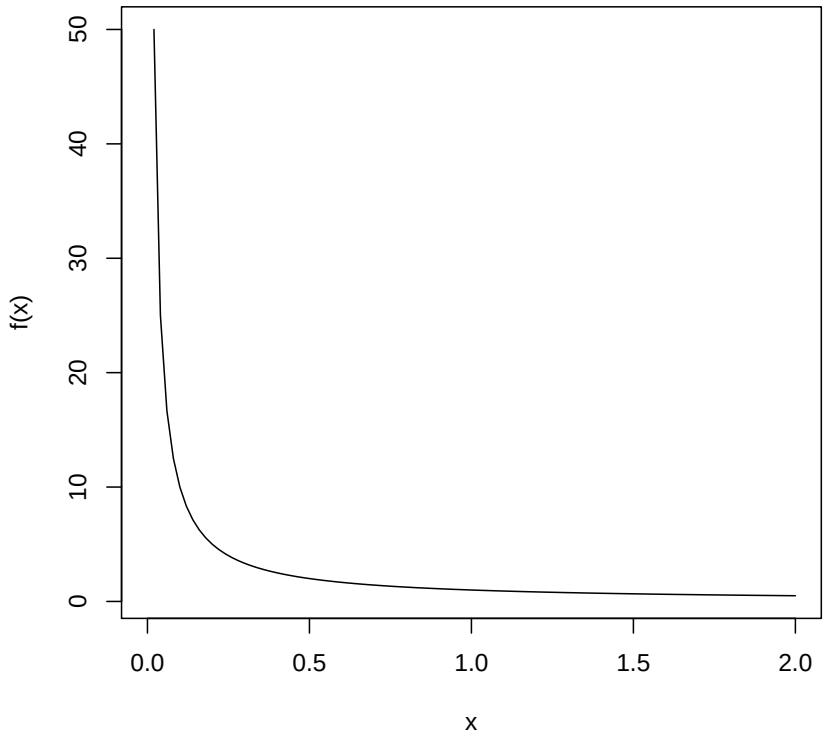


Figura 3.1: Grafico di $1/x$ nell'intervallo $(0, 2)$.

Può essere utile sovrapporre ad un grafico già tracciato un ulteriore grafico, tracciando ad esempio l'asse x, l'asse y, o una ulteriore funzione; la finestra grafica è quella preesistente e non deve essere ridefinita ogni volta (si suggerisce di realizzare per primo il grafico con finestra grafica più ampia). Il parametro da utilizzare in tale caso è `add` (variabile booleana, può assumere solo valore `True` o `False`, indicabili come `T` o `F` o per esteso come `TRUE` o `FALSE`, consigliato), ovviamente nel caso `F` l'assegnazione del parametro non è richiesta). Ad esempio per aggiungere la retta $y = 20\,x$ alla funzione precedente si scriverà:

```
> g<-function(x) 20*x
> curve (g,add=T)
```

ottenendo la Figura 3.2. Vediamo qui di seguito alcune opzioni grafiche:

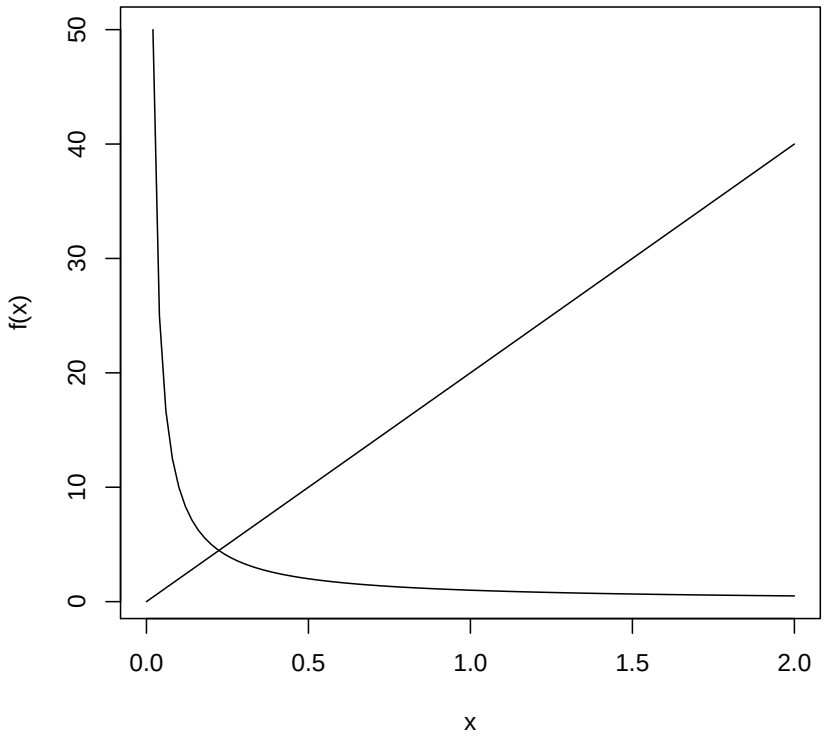


Figura 3.2: Sovrapposizione di 2 grafici

– Definizione della finestra grafica.
  Possiamo specificare completamente la finestra grafica. Per l'asse delle x si ricorre al parametro `xlim`:

  `xlim=c(`*estremo inferiore, estremo superiore*`)`

  e in modo simile per l'asse delle y il parametro è `ylim`:

  `ylim=c(`*estremo inferiore, estremo superiore*`)`

  Se ad esempio volessimo tracciare il grafico della funzione precedente tra -3 e 3 sull'asse x e tra -4 e 4 sull'asse y, scriveremo:

```
> curve(1/x,xlim=c( -3,3), ylim=c(-4,4))
```

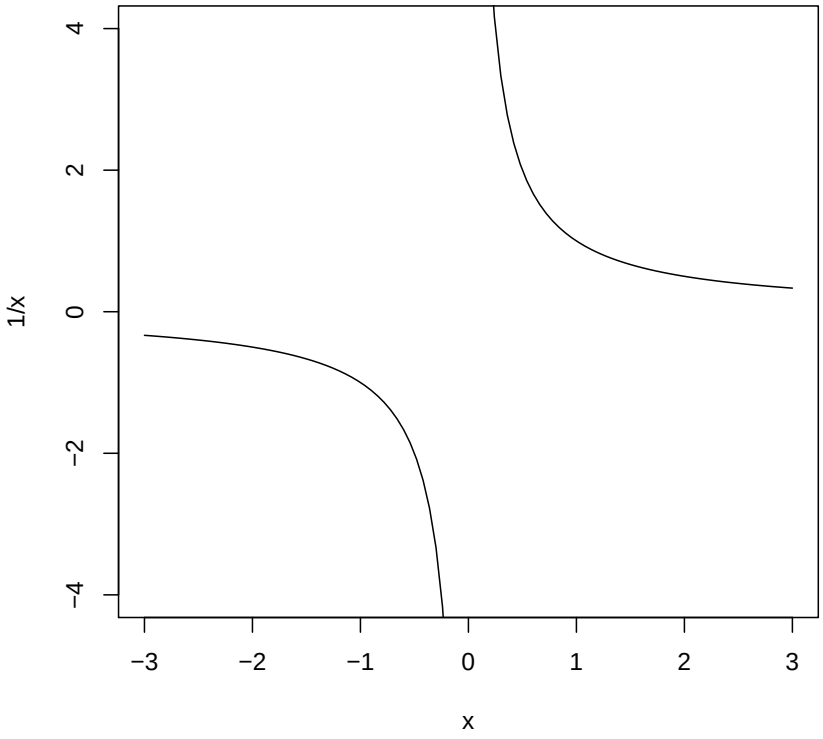


Figura 3.3: Scelta della finestra grafica

ed otterremo il grafico della figura 3.3. Se vogliamo restringere il grafico tra -2, e 2 della finiestra grafica precedente possiamo scrivere

```
> curve(1/x,from=-2,to=2,xlim=c( -3,3), ylim=c(-4,4))
```

ed otterremo il grafico della figura **??**.

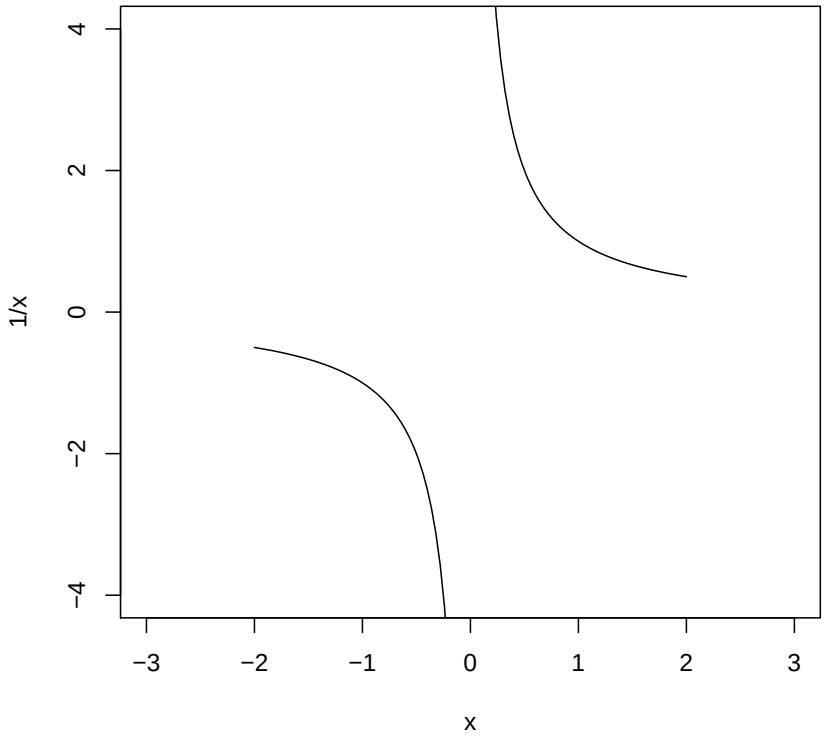


Figura 3.4: Scelta della finestra grafica: grafico con dominio ristretto

– Selezione del colore della curva.

Per cambiare il colore predefinito del grafico si deve utilizzare il parametro `col`, scrivendo `col("nome colore")`. Il nome del colore deve essere definito in lingua inglese. Per visualizzare l'elenco completo dei colori disponibili si usa il comando: `colors()`. Tra i principali colori troviamo

rosso	red
blu	blue
bianco	white
rosa	pink
verde	green
giallo	yellow
viola	purple
nero	black

Sono utilizzabili anche i parametri `light` e `dark`, esse devono precedere il nome del colore. Possiamo pensare di colorare la curva precedente di blu scuro e scrivere: Otteniamo la figura 3.5. Si noti che la dicitura `dark blue` e `darkblue` senza spazi

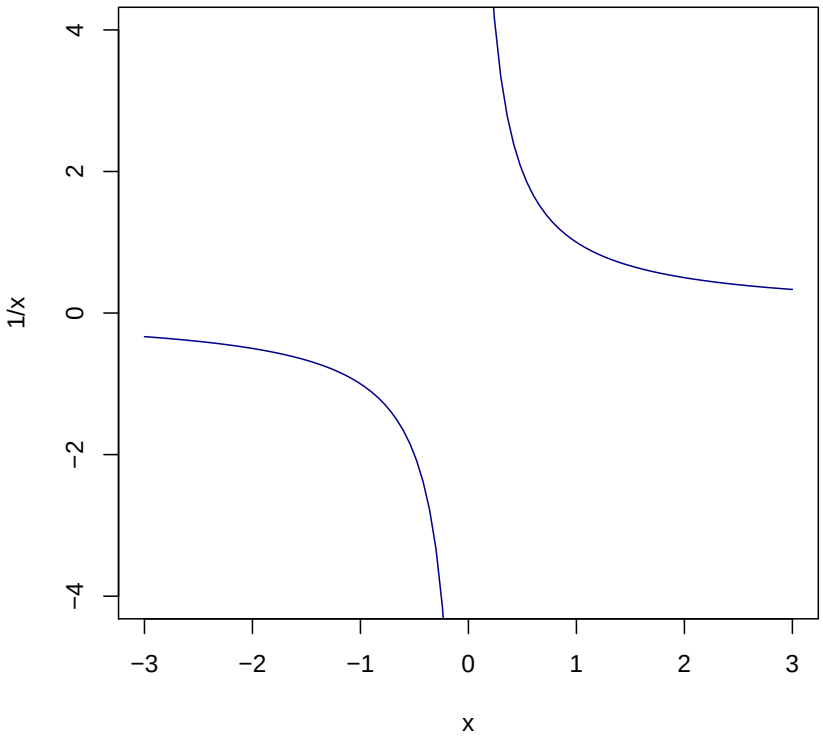


Figura 3.5: Scelta del colore

producono lo stesso risultato (sono implementati ambo i colori)

– Selezione dello spessore delle linee

Per cambiare lo spessore delle linee è possibile utilizzare il parametro `lwd`, esso può avere valori in una scala da 0 fino ad un numero qualunque (anche decimale) di *pixel*, se vogliamo visualizzare la curva precedente con spessore della linea pari a 5 pixel, scriveremo:

```
> curve(1/x,xlim=c( -3,3),ylim=c(-4,4),col="darkblue", lwd=5)
```

Otteniamo la figura 3.6

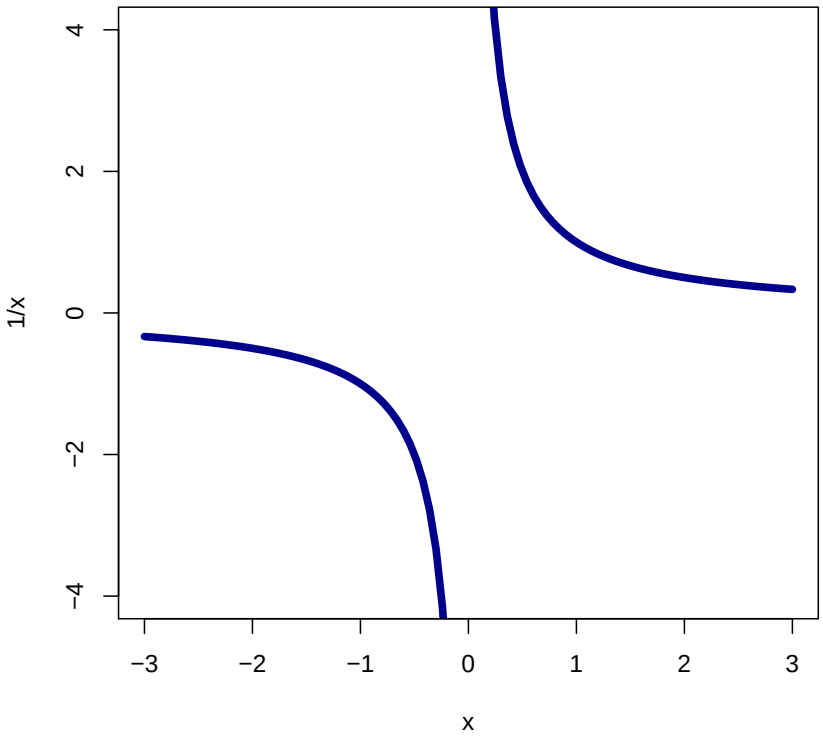


Figura 3.6: Spessore delle linee

– Scelta del tipo di linea.
  È possibile scegliere un numero di punti da visualizzare ed è anche possibile collegarli o meno. Per scegliere il numero di punti si usa il parametro `n` associandogli un numero intero. Per non collegare i punti si usa il parametro `type`, e si fa assumere valore `"p"`, si scrive quindi: `type= "p"`. Ad esempio se vogliamo un numero di punti pari a 100, non collegati, scriveremo:

```
> curve(1/x,xlim=c(-3,3),ylim=c(-4,4),
+ col="dark blue", lwd=2,n=100,type="p")
```

  il cui risultato è riportato in figura 3.7. Altre scelte possibili sono `type="n"` per il punto non vuoto ma anche `type="l"` per le linee (scelta di default) e `type="o"` per punti e linee.

– La funzione `abline`.
  La funzione `abline`, di sintassi: `abline(`*b*`,`*a*`)` consente di tracciare una retta di pendenza *a* e di intercetta *b*. Se scriviamo `abline(h=`*valore*`)` otteniamo una linea

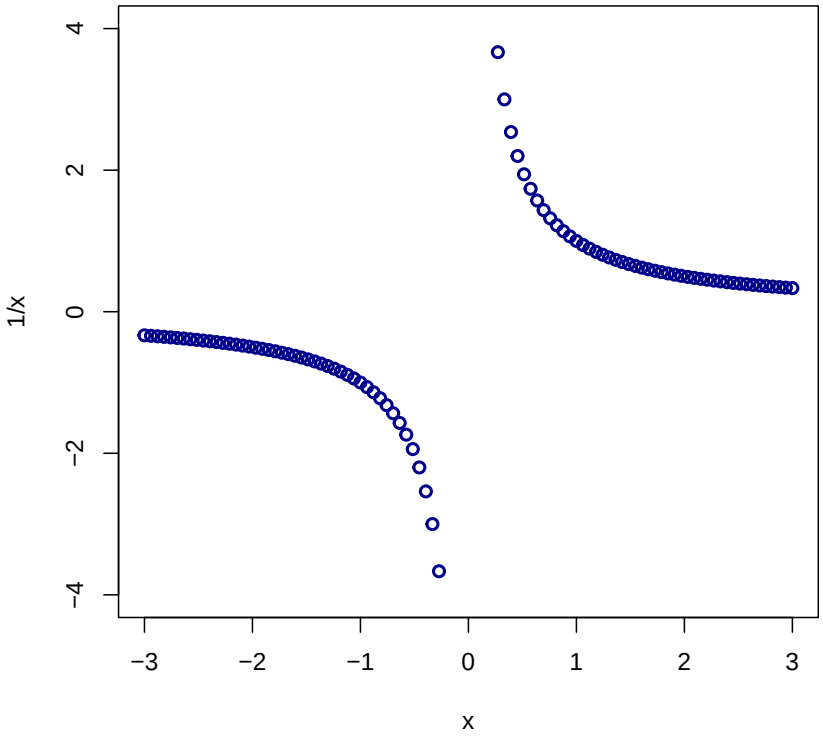


Figura 3.7: Grafico a punti

orizzontale a quota *valore* mentre con: `abline(v=`*valore*`)` otteniamo una retta verticale ad ascissa pari al *valore*. Dopo avere tracciato un grafico si possono quindi aggiungere con i comandi:`abline(h=0)` e `abline(v=0)` cioè gli assi di riferimento del sistema cartesiano (utili nel caso ad esempio in cui si debbano individuare le intersezioni con gli assi della funzione). Ad esempio se scriviamo:

```
>  curve(1/x,xlim=c( -3,3),
+  ylim=c(-4,4),col="darkblue",lwd=5);
>  abline(h=0);abline(v=0);
```

Otteniamo la figura 3.8. Notiamo che cambiando di poco la finestra grafica

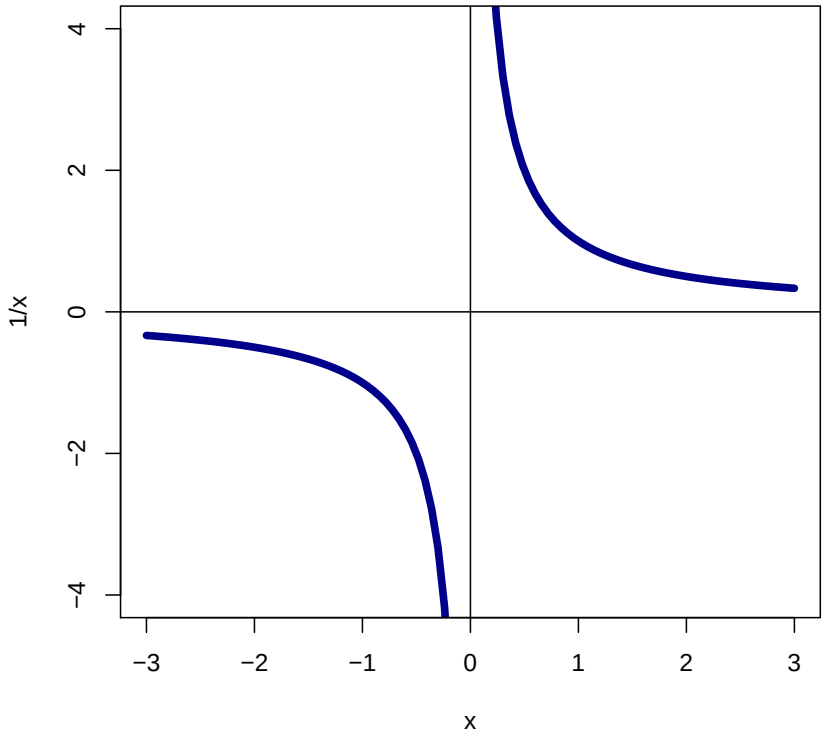


Figura 3.8: Iperbole con assi

```
>  curve(1/x,xlim=c( -3,3.1),
+  ylim=c(-4,4),col="darkblue",lwd=5);
>  abline(h=0);abline(v=0);
```

Otteniamo la figura 3.9.

In questa figura una porzione del grafico è stata sovraimpressa all'asse verticale. In effetti uno è fittizio determinato dal fatto che la funzione non è definita in $x = 0$ e che R nel tracciare il grafico cerca di congiungere l'ultimo punto a sinistra dello zero

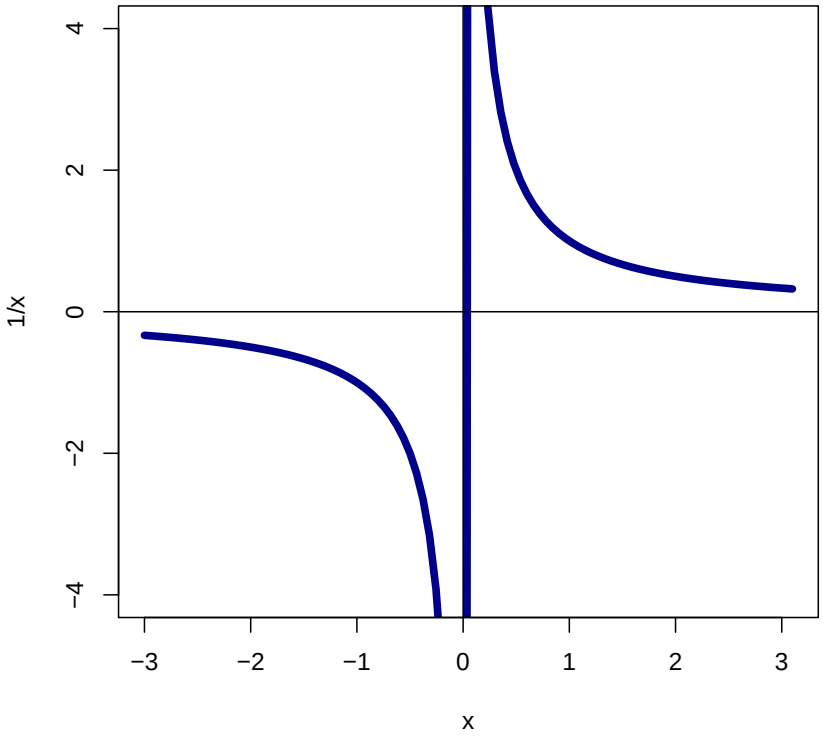
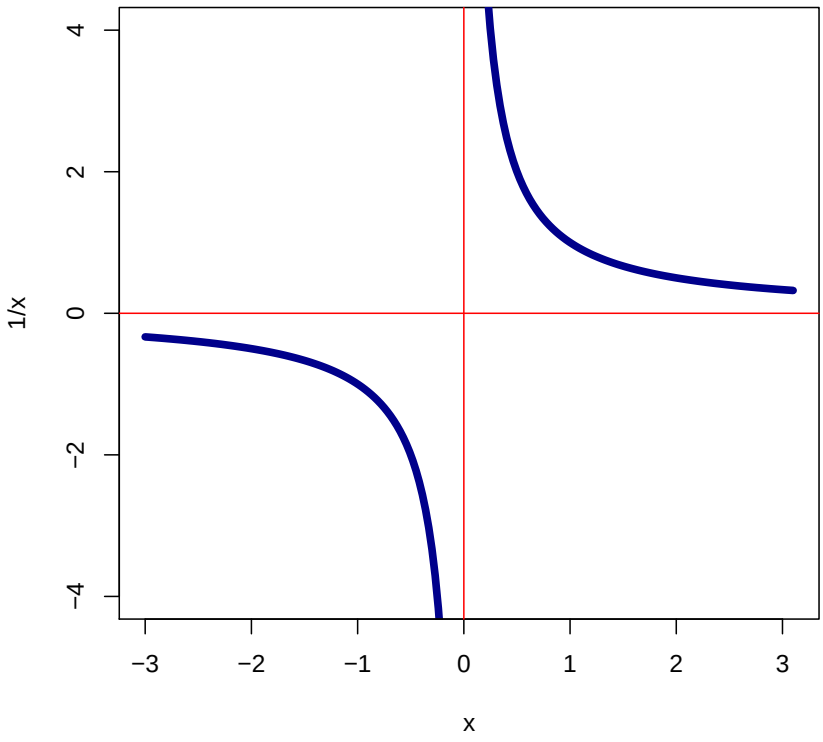


Figura 3.9: Problemi grafici

ed il primo a destra. Per eliminare questo problema si può ricorrere ai comandi che seguono, che generano la figura 3.10:.

```
> curve(1/x, xlim=c( -3,3.1),ylim=c(-4,4),type="n")
> curve(1/x,xlim=c(-3,0), ylim=c(-4,4), col="darkblue",lwd=5,add=T)
> curve(1/x,xlim=c(0,3.1),  ylim=c(-4,4),col="darkblue", lwd=5,add=T)
> abline(h=0,col="red"); abline(v=0,col="red")
```



3.3 Composizione di funzioni

Se abbiamo le funzioni

$$f\colon D_X \to D_Y$$

e

$$g\colon D_{Y'} \to D_Z$$

qualora i valori nell'immagine di f siano nel dominio di g (se Im(f) $\subseteq D_{Y'}$,cosa garantita se $D_Y = D_{Y'}$) possiamo usare l'uscita $y = f(x)$ di f come ingresso della funzione g e possiamo calcolare quindi $g(y) = g(f(x))$. La funzione cosí costruita si chiama funzione composta di f e g e viene indicata come $g \circ f$. Abbiamo quindi

$$g \circ f\colon D_X \to D_Z \tag{3.1}$$

$$x \mapsto y = g(f(x)) \tag{3.2}$$

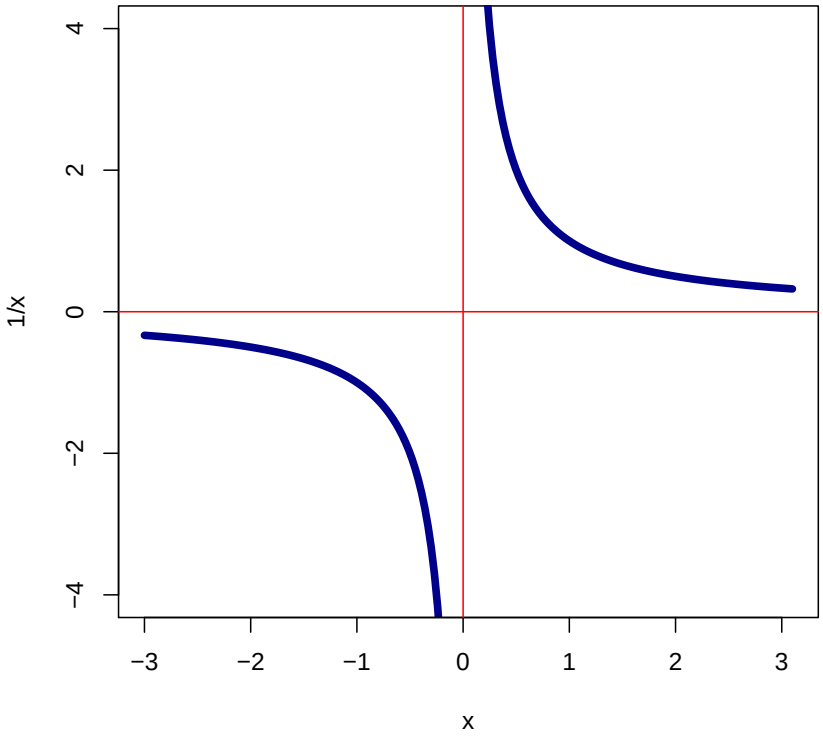


Figura 3.10: Grafico con assi e senza asintoto

3.3.1 Esempi

Si consideri la funzione composta $g \circ f$ delle funzioni $f(x) = x^3 + 3 \cdot x + 1$ e $g(x) = \tan(x) + 2$. Si calcoli $(g \circ f)(4)$. Con R chiamando gf la funzione $g \circ f$

```
> f<-function(x) x^3+3*x+1
> g<-function (x)  tan(x)+2
> gf<-function(x) g(f(x))
> gf(4)
[1] -30.26858
```

3.4 Funzioni inverse

Ricordiamo che una funzione $f\colon D_X \to D_Y$ è invertibile se comunque si scelga $y \in D_Y$ la soluzione x in D_X dell'equazione

$$f(x) = y$$

nella variabile x esiste ed è unica.
In tale situazione la funzione inversa esiste è denotata con f^{-1} ed è definita come

$$f^{-1}\colon D_Y \to D_X \tag{3.3}$$
$$y \mapsto x = f^{-1}(y) \mid x \text{ soluzione unica di } f(x) = y \tag{3.4}$$

Ne segue che ad un arbitrario $y \in D_Y$ possiamo associare in modo unico $x \in D_X$ tale che $f(x) = y$ e quindi $(f \circ f^{-1})(y) = f(f^{-1}(y)) = f(x) = y$ mentre se $x \in D_X$ $(f^{-1} \circ f)(x) = f^{-1}(f(x)) = f^{-1}(y) = x$. Quindi

$$f \circ f^{-1}\colon D_Y \to D_Y \tag{3.5}$$
$$x \mapsto f(f^{-1}(y)) = y \tag{3.6}$$

e

$$f^{-1} \circ f\colon D_X \to D_X \tag{3.7}$$
$$x \mapsto f^{-1}(f(x)) = x \tag{3.8}$$

sono le funzioni identiche di dominio e codominio rispettivamente.

- Si calcoli

$$5^{3/2 \log_5(4)}$$

3.4.1 Grafico delle funzioni inverse

Il grafico di una funzione invertible $f\colon D_X \to D_Y$ è l'insieme di punti nel piano

$$\Gamma_f = \{(x, f(x)) \mid x \in D_X\}$$

In modo simile il grafico di f^{-1} è l'insieme di punti

$$\Gamma_{f^{-1}} = \{(y, f(y)) \mid y \in D_Y\}$$

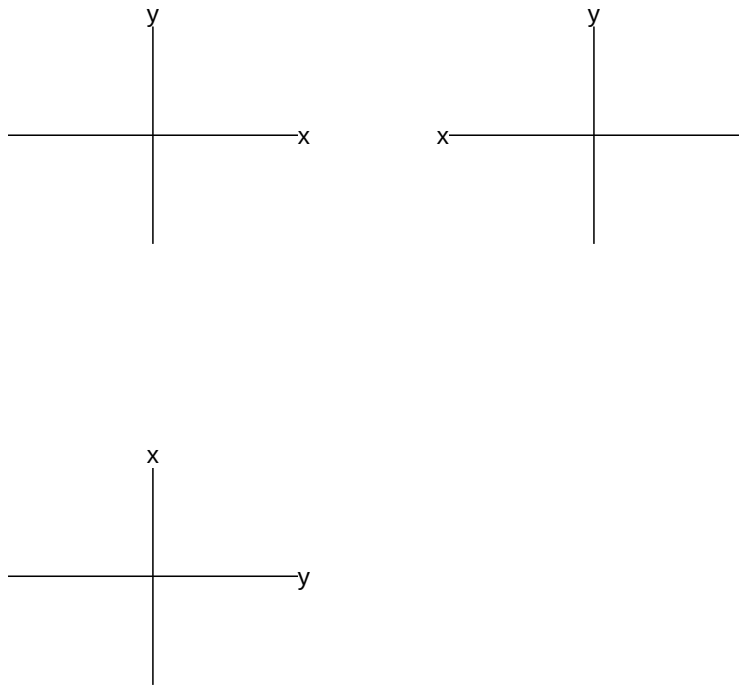


Figura 3.11: Trasformazione che consente di passare dal grafico di una funzione al grafico della sua funzione inversa

ma poiché l'equazione $f(x) = y$ ha una e una sola soluzione $x \in D_X$

$$\{(y, f^{-1}(y)) \mid y \in D_Y\} = \{(f(x), f^{-1}(f(x)) \mid x \in D_X\} = \{(f(x), x) \mid x \in D_X\}$$

Per ottenere il grafico della funzione inversa a partire dal grafico basta quindi trovare una trasformazione del piano che trasformi l'asse x nell'asse y e viceversa. Una possibile realizzazione di tutto questo è ripostato nella figura 3.11 Come si può realizzare concretamente questa trasformazione?
Partiamo per esempio dal grafico riportato in Figura 3.12 possiamo partire da una sequenza regolare di punti del dominio di f ed eseguire le trasformazioni

- Cambiare x in $-x$
- Cambiare $(x, f(x))$ in $(f(x), x)$

Nell'esempio abbiamo scelto come dominio l'intervallo $[-4, 4]$. Abbiamo quindi costruito una seguenza di punti in tale intervallo e poi eseguito i comandi

```
> par(mfrow=c(2,2))
> x=seq(-4,4,0.1)
```

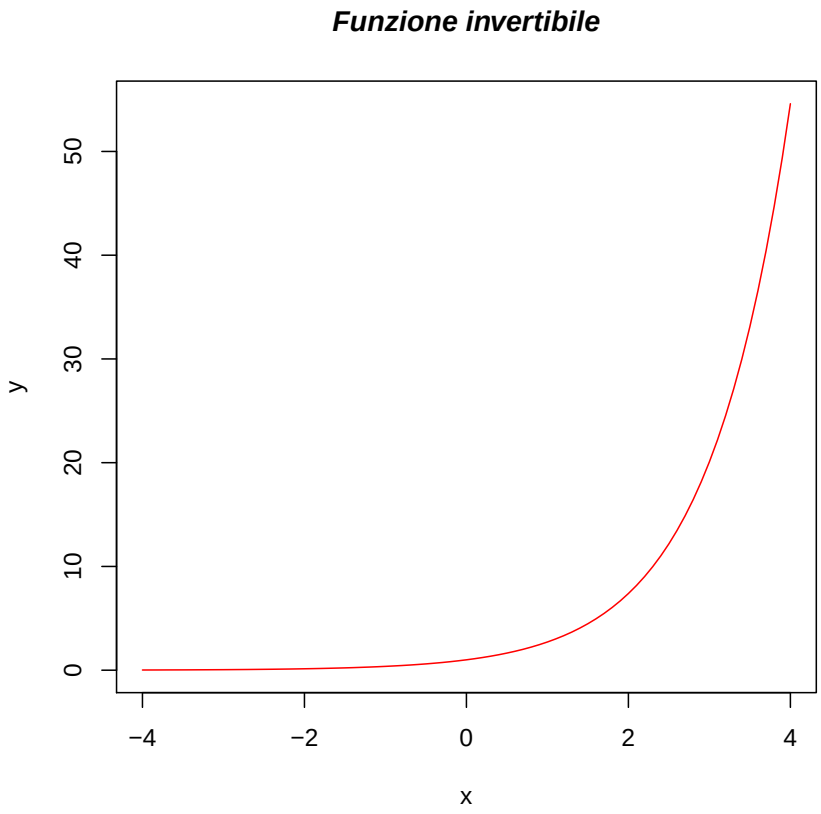
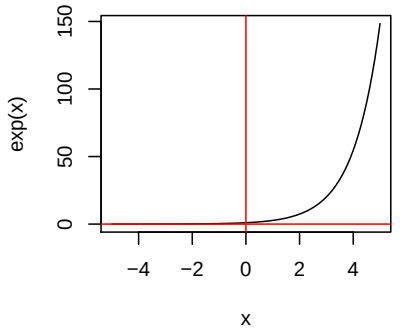
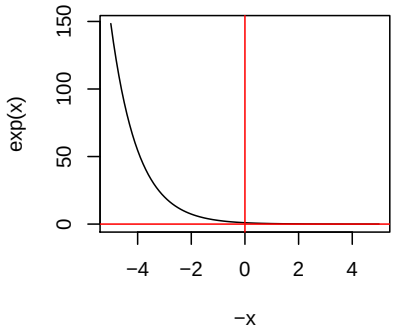
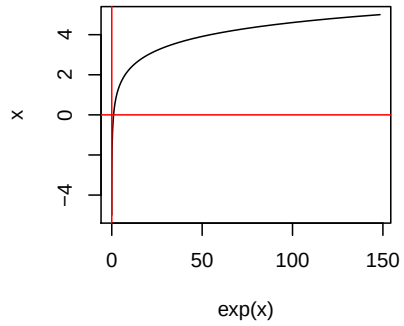


Figura 3.12: Grafico di una funzione invertibile (esponenziale)


exp(x)
0
50
100
150
−4
−2
0
2
4
x


exp(x)
0
50
100
150
−4
−2
0
2
4
−x


x
−4
0
2
4
0
50
100
150
exp(x)

```
> y=exp(x)
> plot(x,y,type="l");abline(h=0,col="red");abline(v=0,col="red")
> plot(-x,y,type="l");abline(h=0,col="red");abline(v=0,col="red")
> plot(y,x,type="l");abline(h=0,col="red");abline(v=0,col="red")
> par(mfrow=c(1,1))
```

Qui il comando `plot(x,y)` rappresenta i punti di coordinate (x, y) nel piano.

3.5 Funzioni inverse delle funzioni circolari

Come visto a lezione le funzioni seno, coseno e tangente non sono invertibili; se però consideriamo opportune restrizioni di tali funzioni, possiamo costruire le funzioni arcoseno, arcocoseno e arcotangente. Più precisamente la restrizione della funzione `sin`

$$\sin\colon [-\frac{\pi}{2}, \frac{\pi}{2}] \to [-1, 1]$$

è invertibile e la sua funzione inversa è la funzione arcoseno, che indicheremo con `asin`

$$\texttt{asin}\colon [-1, 1] \to [-\frac{\pi}{2}, \frac{\pi}{2}]$$

In modo simile la funzione

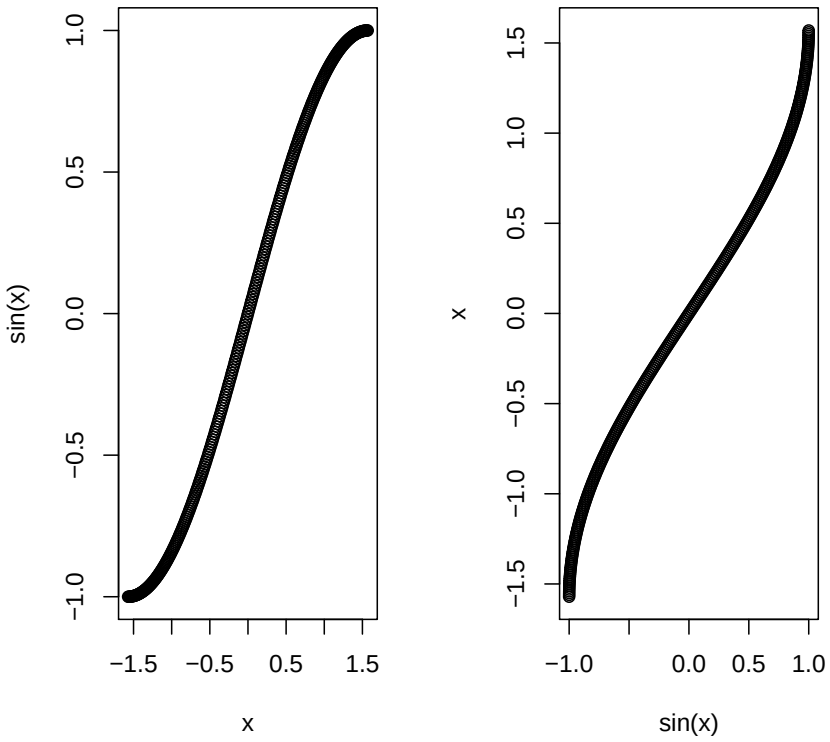
$$\cos\colon [0, \pi] - \to [-1, 1]$$

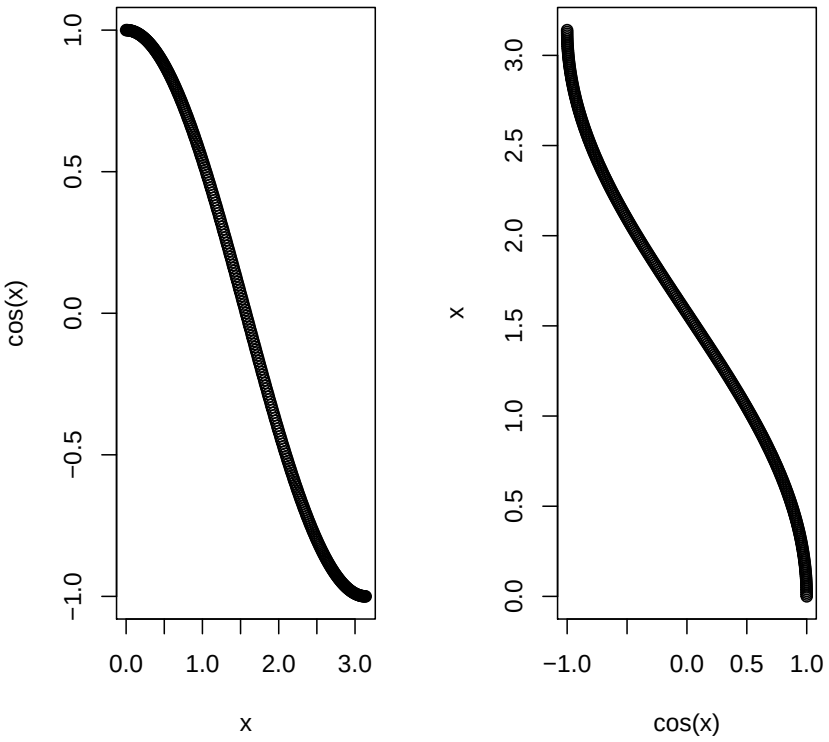
è invertibile e la sua funzione inversa è la funzione arcocoseno, che indicheremo con `acos`

$$\texttt{acos}\colon [-1, 1] \to [0, \pi]$$

```
> par(mfrow=c(1,2))
> x=seq(0,pi,0.01)
> plot(x,cos(x))
> plot(cos(x),x)
```

```
> par(mfrow=c(1,2))
> x=seq(-pi/2,pi/2,0.01)
> plot(x,sin(x))
> plot(sin(x),x)
```





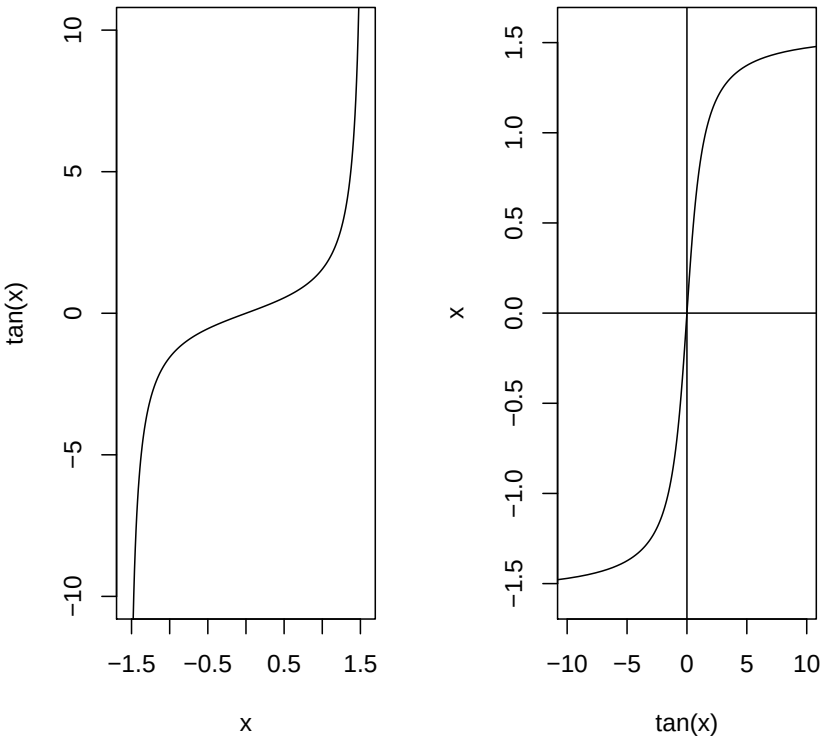
e lo è pure la funzione

$$\tan\colon (-\frac{\pi}{2}, \frac{\pi}{2}) \to \mathbb{R}.$$

La sua funzione inversa è la funzione arcotangente che indicheremo con `atan`

$$\texttt{atan}\colon \mathbb{R} \to (-\frac{\pi}{2}, \frac{\pi}{2})$$

```
> par(mfrow=c(1,2))
> x=seq(-pi/2,pi/2,0.01)
> plot(x,tan(x),ylim=c(-10,10),type="l")
> plot(tan(x),x,type="l",xlim=c(-10,10));abline(h=0);abline(v=0)
```



3.6 Derivate

Richiamiamo il significato geometrico della derivata: se una funzione f è derivabile in un punto x_0 allora il grafico si linearizza con un numero sufficiente di ingrandimenti attorno al punto $(x_0, f(x_0))$. Come illustrazione di questo processo consideriamo l'iperbole

```
> f<-function(x) 4*x/(2+x)
```

e disegniamo il suo grafico nella regione definita dalle condizioni $-d < x < d$ e $-d < y < d$, dove d è inizialmente pari a 8.

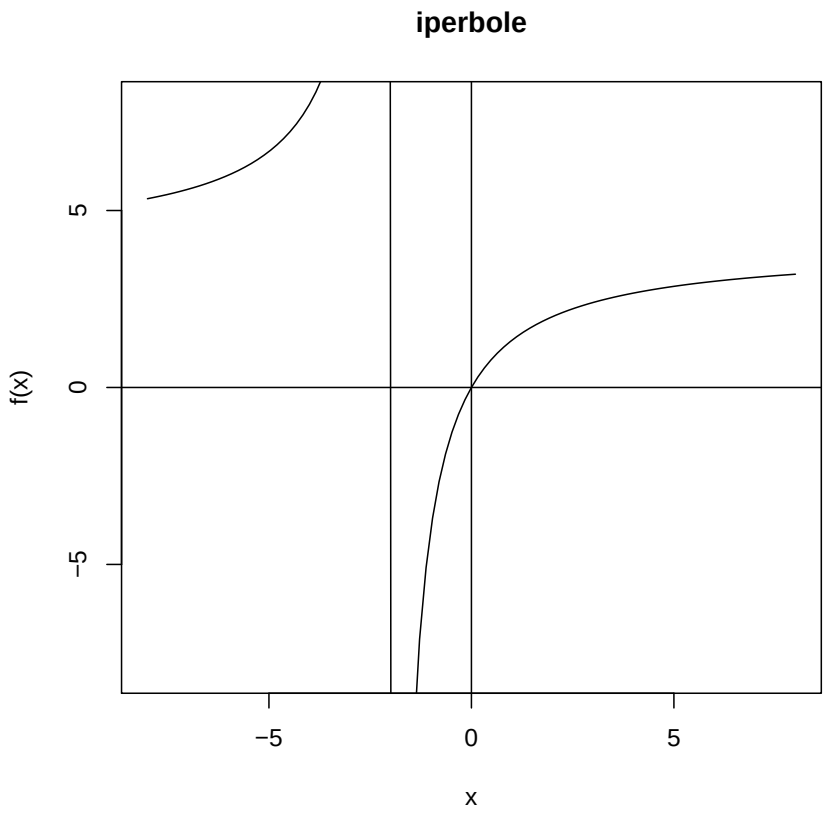
```
> d=8;curve(f,xlim=c(-d,d),ylim=c(-d,d), main="iperbole"); abline(h=0);abline(v=0)
```

Consideriamo ora l'ingrandimento di tale grafico con fattore di zoom `xf`=2 attorno al punto $(0, 0)$: in altre parole dimezziamo il valore di d:

```
> d=d/2;curve(f,xlim=c(-d,d),ylim=c(-d,d),
> main="zoom 2"); abline(h=0);abline(v=0)
```

Iterando tale comando un certo numero di volte si assiste alla linearizzazione progressiva del grafico come in figura 3.13.

```
> d=8;curve(f,xlim=c(-d,d),ylim=c(-d,d),
> main="iperbole"); abline(h=0);abline(v=0)
```



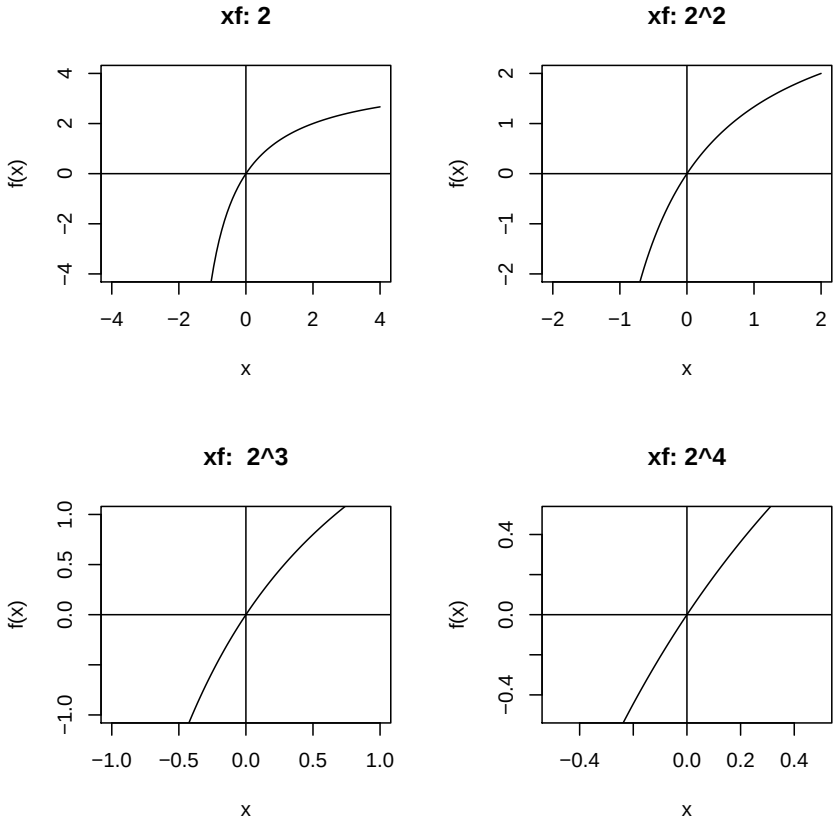


Figura 3.13: Linearizzazione di un grafico ottenuta attraverso una serie di ingrandimenti

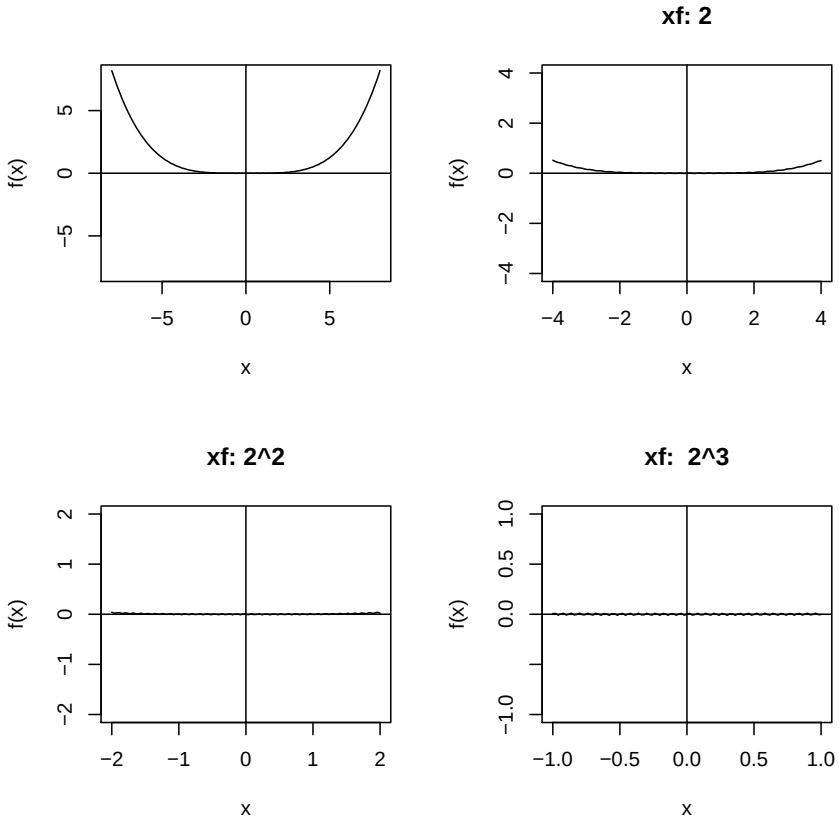


Figura 3.14: Linearizzazione progressiva di un grafico: prima fase

Consideriamo ora una funzione differente che combina un'espressione di quarto grado e una funzione circolare rapidamente oscillante

```
> f<-function(x) x^4/500 + sin(100*x)/100;
```

Il grafico per $-8 < x < 8$ come in precedenza suggerisce che la retta tangente in $x = 0$ sia orizzontale. Effettuando un certo numero di zoom con fattore di zoom 2 questa prima impressione sembra confermata, anche se possiamo iniziare a notare delle piccole oscillazioni nell'ultima figura del pannello 3.14.

Se procediamo con gli ingrandimenti questa sensazione viene confermata. Non ci saranno ulteriori sorprese da un certo punto in poi. Per quanto si prosegua la retta trovata in ultimo verrà confermata 3.15

3.6.1 Regola dei tre punti

Possiamo stimare la derivata di una funzione f in x con la regola dei tre punti nel punto x con incremento h

$$\texttt{trepunti}(f, x, h) = \frac{f(x+h) - f(x-h)}{2h}$$

Ora è sufficiente definire la funzione di cui vogliamo stimare la derivata, per poi richiamare la funzione `trepunti` fornendo il valore di x ed il valore dell'incremento h.

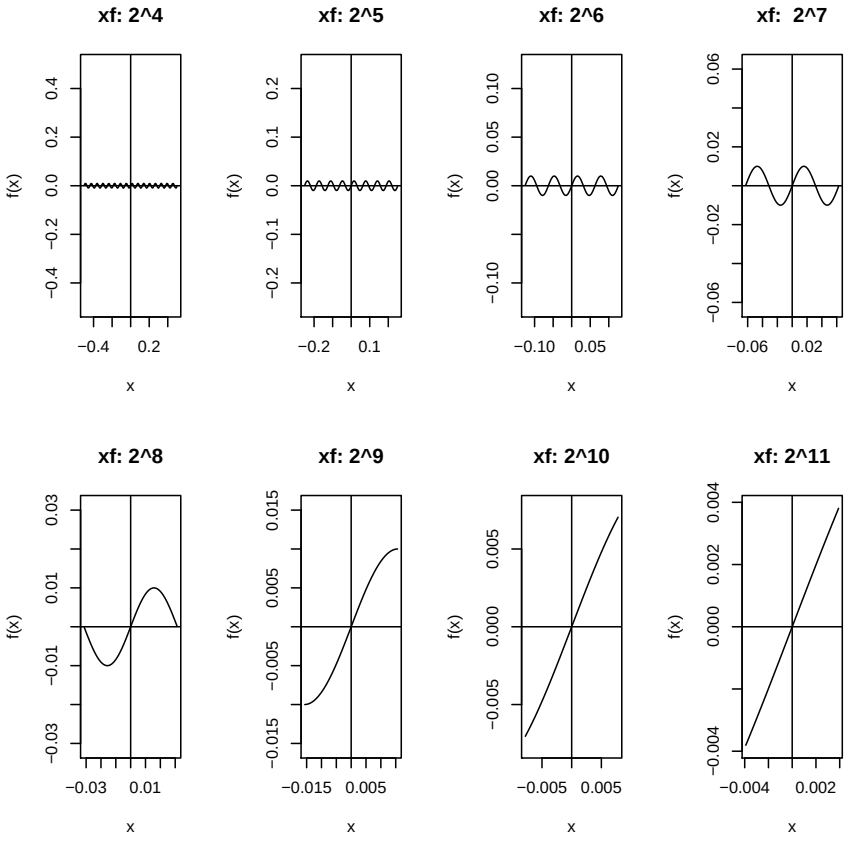


Figura 3.15: Linearizzazione progressiva di un grafico: seconda fase

Approssimiamo ad esempio la derivata di $x \to \sin(x)$ Definiamo la funzione e la regola della differenza centrale

```
> f<-function(x) sin(x)
> trepunti<-function(f,x,h) (f(x+h)-f(x-h))/(2*h)
```

Per esempio assegnando

```
> h=0.001
> x=1
```

Si ha

```
> trepunti(f,x,h)
[1] 0.5403022
```

Otteniamo così la stima della derivata nel punto 1 con incremento 0.001. Possiamo anche rappresentare graficamente la funzione derivata (insieme alla funzione di `sin`) scrivendo:

```
> curve(trepunti(f,x,0.001),0,2*pi,col="blue")
> curve(f,add=T,col="red")
```

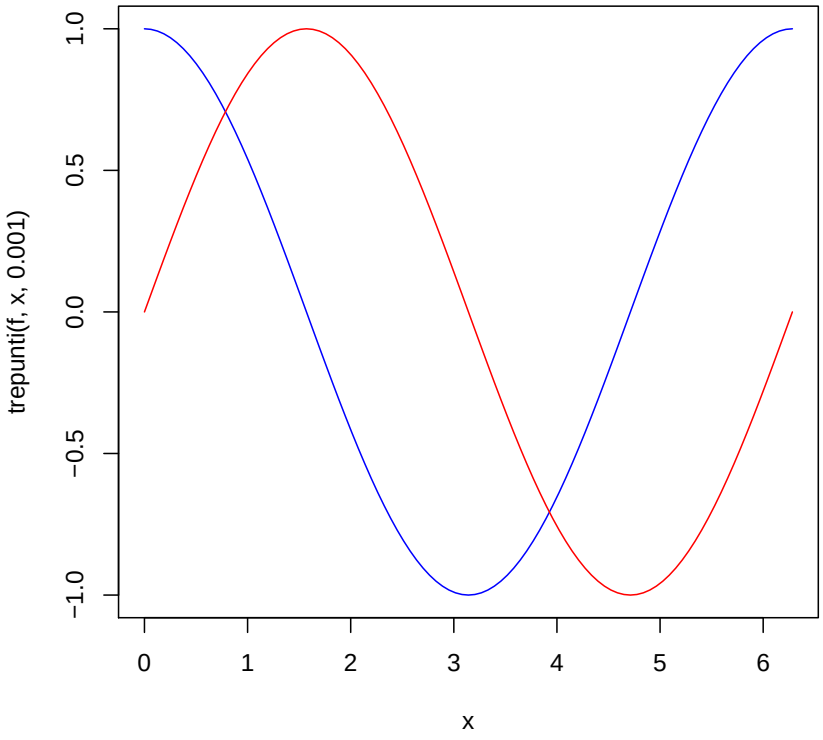


Figura 3.16: Grafico delle funzioni $sin(x)$ e della sua derivata calcolata numericamente

e ottenendo il grafico 3.16 da cui si evince che $D\sin(x) = \cos(x)$. In modo simile si può mostrare che $D\cos(x) = -\sin(x)$. Analizziamo ancora la derivata delle funzioni esponenziali. Consideriamo inizialmente la funzione

$$f(x) = 2^x$$

In questo caso

```
> f<-function(x) 2^x
> curve(trepunti(f,x,0.001),-2,2,
+ col="blue")
> curve(f,add=T,col="red")
> curve(trepunti(f,x,0.001)/f(x),add=T,
+ col="dark green")
```

sembra che la funzione (in rosso) e la sua derivata (in blu, approssimata con la regola dei 3 punti) siano proporzionali. Per verificarlo abbiamo tracciato anche il grafico di $f'(x)/f(x)$

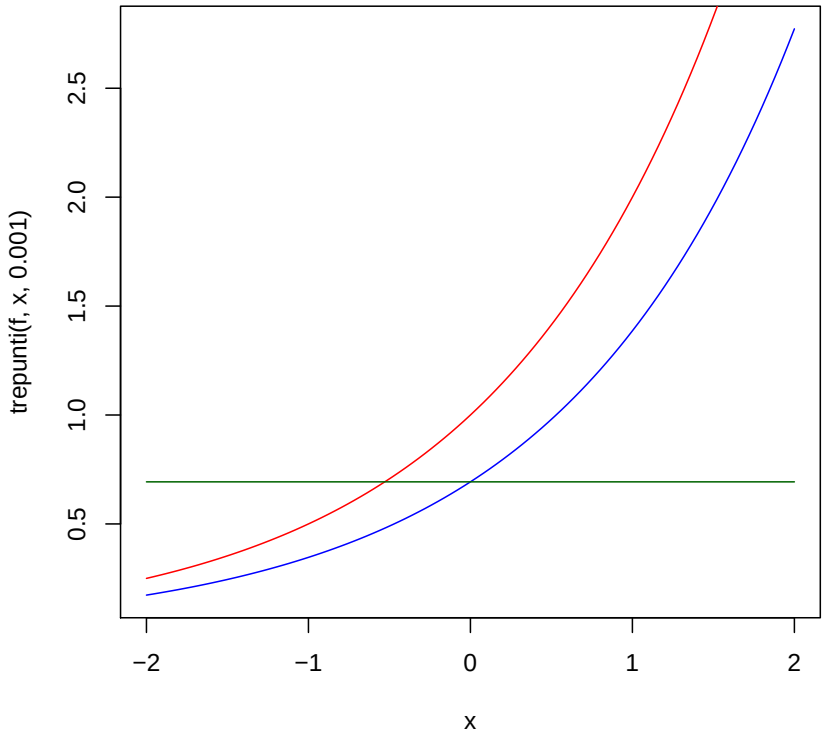


Figura 3.17: Grafico di 2^x e derivata

che è una retta orizzontale a quota

```
> trepunti(f,0,0.001)/f(0)
```

```
[1] 0.6931472
```

In modo simile possiamo considerare con la funzione 3^x:

```
> f<-function(x) 3^x
> curve(trepunti(f,x,0.001),-2,2,col="blue")
> curve(f,add=T,col="red")
> curve(trepunti(f,x,0.001)/f(x),add=T,col="dark green")
```

e anche qui le funzioni sembrano proporzionali. In questo caso però il rapporto derivata diviso funzione è maggiore di 1 e quindi la derivata è maggiore della funzione

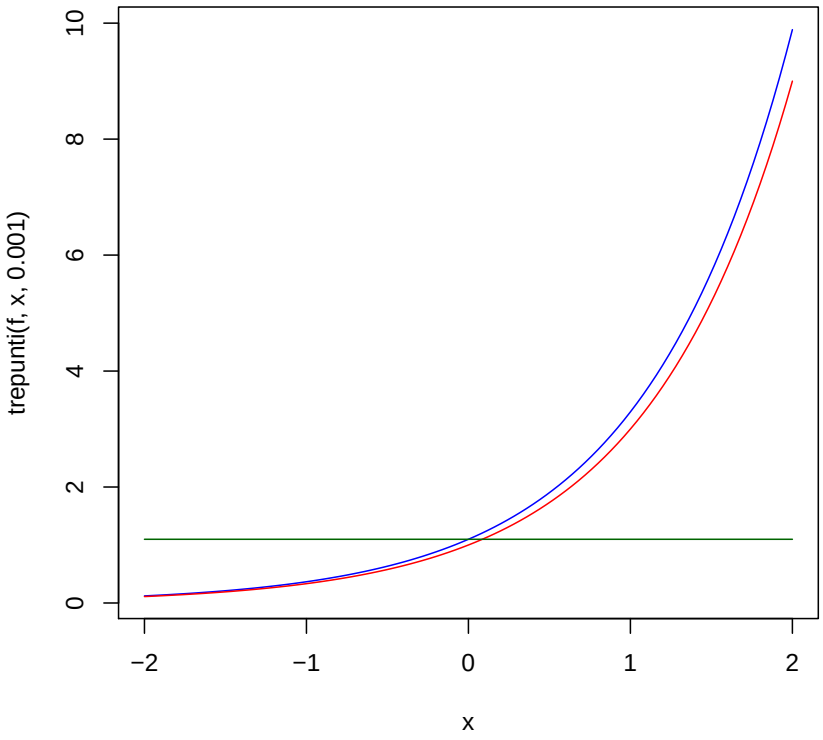
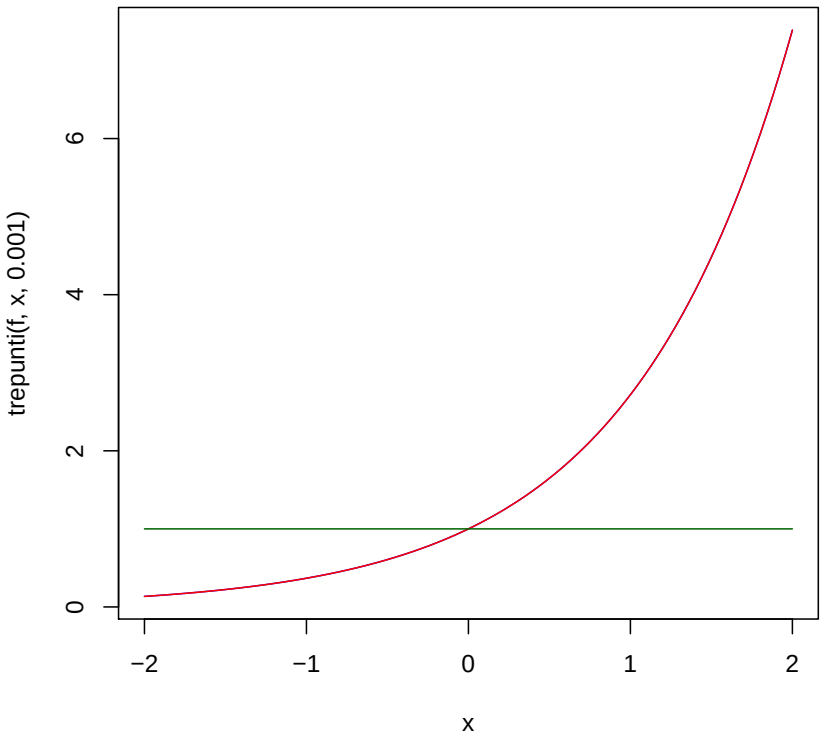


Figura 3.18: Grafico di 3^x e derivata

Per verificarlo abbiamo tracciato anche il grafico di $f'(x)/f(x)$ che è una retta orizzontale a quota

```
> trepunti(f,0,0.001)/f(0)
[1] 1.098613
```

Ci si può chiedere se esista un valore intermedio tra 2 e 3 per cui derivata e funzione coincidano esattamente. La risposta è precisamente il numero di Nepero `exp(1)`. Infatti abbiamo




```
> f<-function(x) exp(x)
> curve(trepunti(f,x,0.001),-2,2, col="blue")
> curve(f,add=T,col="red")
> curve(trepunti(f,x,0.001)/f(x),add=T, col="dark green")
> trepunti(f,0,0.001)/f(0)
```

Anche qui

```
> trepunti(f,0,0.001)/f(0)
[1] 1
```

mostra il valore del rapporto.

1. Ripeti le considerazioni precedenti usando il quoziente di Newton a destra.
2. Calcola il valore della approssimazione della derivata con la regola dei tre punti per la funzione $x^2 * sin(x)$ in $x = \pi$ con $h = \pi/6$.

3.6.2 Derivate "formali"

Per calcolare la derivata di una qualunque espressione possiamo scrivere:

`D(expression(`*funzione*`),`*variabile*`)`

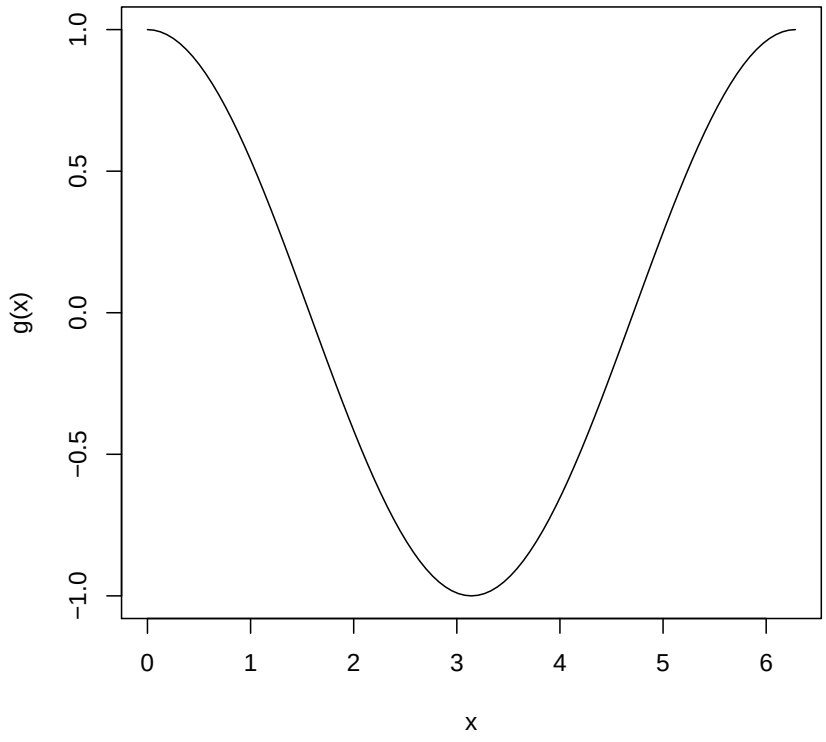
Ad esempio

```
> D(expression(x^4),"x")
4 * x^3
```

Un utile comando per valutare una derivata in un punto o per ricavare una funzione da una derivata è il comando `eval`, ovvero *evaluate*. Scriveremo `eval(`*nome derivata*`)`.

```
> x<-4
> eval(D(expression(x^4),"x"))
[1] 256
```

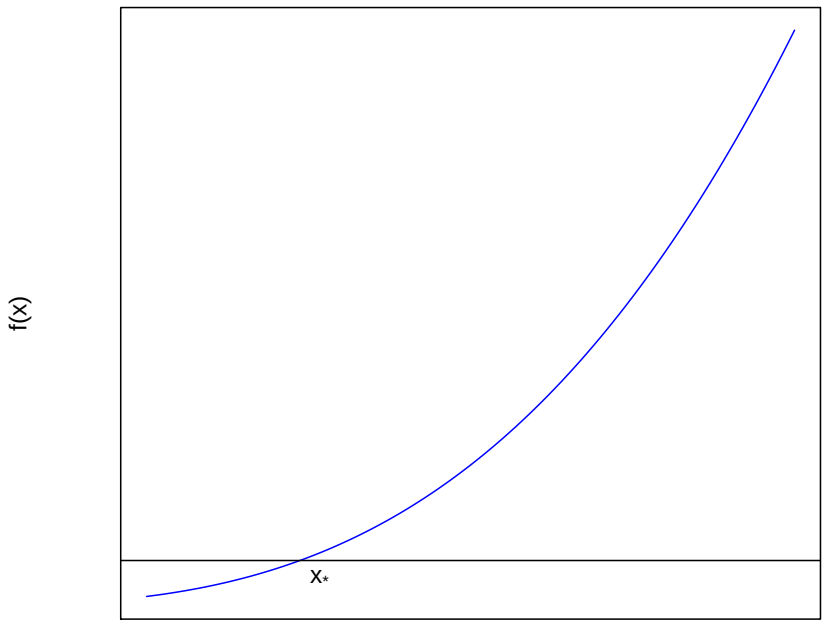
```
> g<-function(x) eval(D(expression(sin(x)),"x"))
> curve(g,0,2*pi)
```



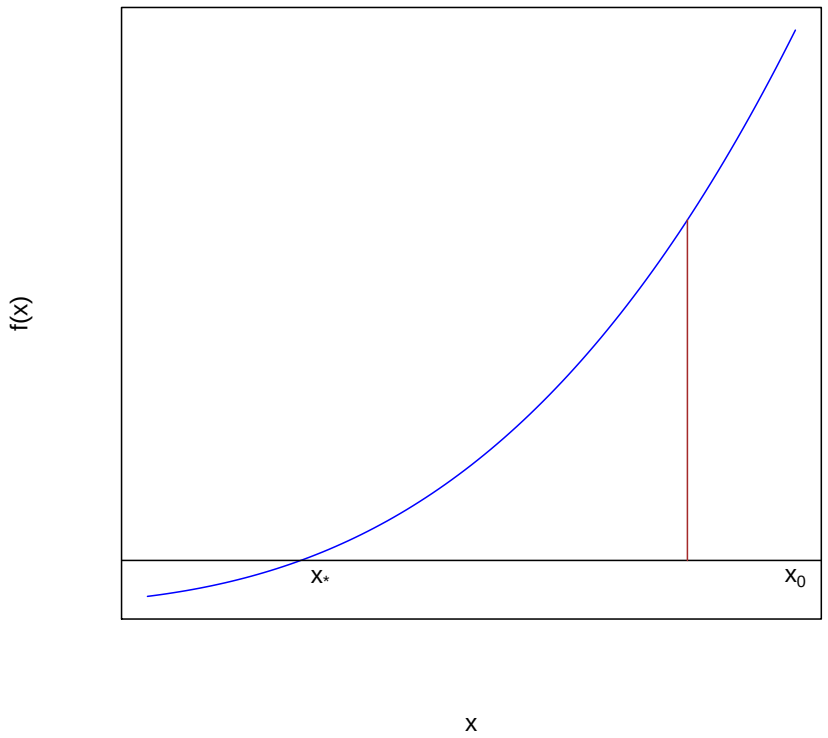
1. Calcola la derivata prima e seconda della funzione $x^2 * sin(x)$ in $x = \pi$.
2. Determina l'equazione della retta tangente al grafico della funzione precedente e traccia grafico e retta in una finestra grafica opportuna.

3.6.3 Metodo delle tangenti di Newton

Consideriamo ora una funzione generica f il cui grafico presenti delle intersezioni con l'as-



se delle x Nel metodo delle tangenti di Newton si parte da un'approssimazione iniziale x_0, che nel grafico illustrato si



situa alla destra di x_*.
Utilizziamo ora il metodo delle tangenti di Newton per determinare gli zeri di una funzione f. Prima di iniziare vediamo come possiamo procedere a livello grafico. Consideriamo una semplice funzione

$$f(x) = x^5 - 3x + 1$$

Iniziamo a selezionare una finestra grafica abbastanza ampia. Restringendola man mano vediamo meglio cosa succede.

```
> f<-function(x) x^5-3*x+1
> curve(f,-10,10)
> abline(h=0)
> curve(f,-2,2)
> abline(h=0)
```

Dal grafico 3.19 è evidente che ci sono 3 intersezioni con l'asse delle x che ci proponiamo di determinare usando il metodo delle tangenti di Newton. Ricordiamo che nel metodo delle tangenti di Newton si deve scegliere un punto iniziale x_0 ed il numero di iterazioni (o un criterio di arresto). Il passo iterativo è descritto dalla funzione:

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

In assenza della derivata esatta possiamo anche usare la regola dei 3 punti. In quanto segue ipotizzeremo di avere scelto un 10 iterazioni a partire dal valore $x_0 = 2$.

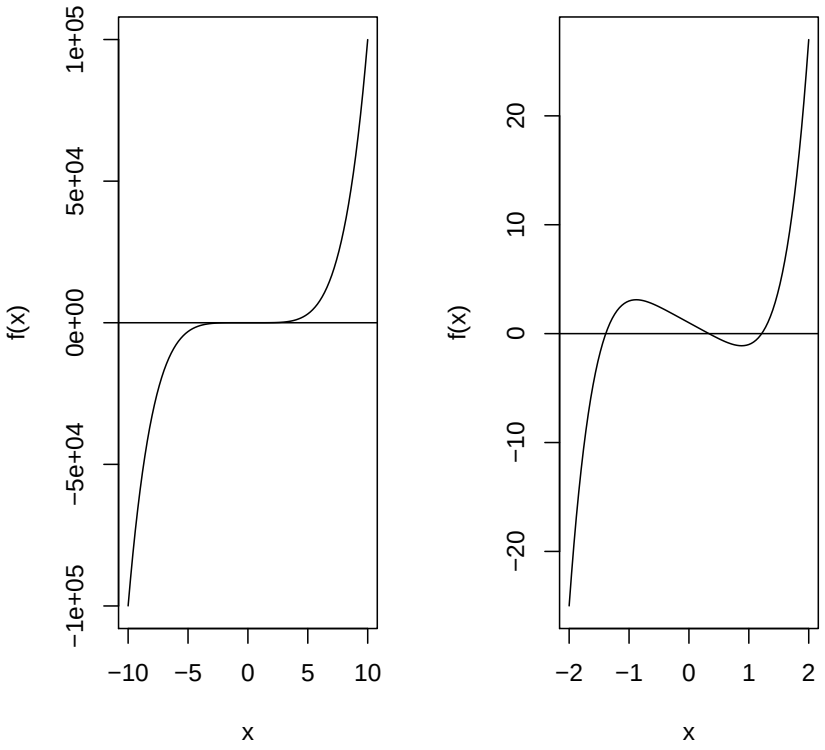


Figura 3.19: Intersezione con l'asse x

```
> x<-2
```

Scegliamo un incremento $h = 10^{-3}$ ed iteriamo il processo attraverso un ciclo `for`[2]:

```
> h<-0.001
> for(k in 1:9) x[k+1]<-x[k]-f(x[k])/trepunti(f,x[k],h)
> x
 [1] 2.000000 1.649351 1.406489 1.268587 1.220370
 [6] 1.214721 1.214648 1.214648 1.214648 1.214648
```

Si genera la lista di 10 passi iterativi, il fatto che la lista si stabilizzi è indice del fatto che la lista converge all'intersezione cercata. Ricerchiamo anche un altro valore della *radice*:

```
> x<- -2; iterazioni=10;
> for(k in seq(length=iterazioni)) x[k+1]<-x[k]-f(x[k])/
+ trepunti(f,x[k],h)
> x
 [1] -2.000000 -1.675325 -1.478238 -1.400446 -1.389020
 [6] -1.388792 -1.388792 -1.388792 -1.388792 -1.388792
[11] -1.388792
```

Per determinare la terza ed ultima intersezione partiamo da $x_0 = 0.5$

```
> x<-0.5
> for(k in 1:10) x[k+1]<-x[k]-
+ f(x[k])/trepunti(f,x[k],h);x
 [1] 0.5000000 0.3255812 0.3347240 0.3347341 0.3347341
 [6] 0.3347341 0.3347341 0.3347341 0.3347341 0.3347341
[11] 0.3347341
```

Le radici sono 0.3347341, -1.388792, 1.214648.

3.7 Integrali definiti

Ci occupiamo ora del calcolo di integrali definiti

$$\int_a^b f(x)dx$$

3.7.1 Metodi numerici di integrazione di R

Per il calcolo di integrali definiti si scrive il comando

`integrate`(*funzione*, *estremo inferiore*, *estremo superiore*)

Ad esempio per x tra -1 e 2

[2] Il comando `for(k in set) comando` esegue il `comando` (eventualmente dipendente da `k` per tutti i valori di `k` in `set`.

```
> integrate(function(x) x^4,-1,2)
6.6 with absolute error < 7.3e-14
```

Il risultato è un numero e di tale numero viene fornito l'errore assoluto, molto basso (la tecnica utilizzata è buona).

3.7.2 Metodi di integrazione: metodo dei rettangoli e metodo dei trapezi

Ci proponiamo di calcolare

$$\int_a^b f(x)dx$$

suddividendo $[a, b]$ in n sottointervalli di eguale ampiezza. Richiamiamo la procedura: si suddivide l'intervallo $[a, b]$ in n sottointervalli di egual ampiezza determinando il passo del metodo $h = \dfrac{b-a}{n}$ e gli estremi degli intervalli $x_k = a + k\,h$ con $k = 0, \ldots, n$. Si calcolano poi i corrispondenti valori $y_k = f(x_k)$ e si applicano poi le formule

$$L_n = h \sum_{i=0}^{n-1} y_k$$

$$R_n = h \sum_{i=1}^{n} y_k$$

$$T_n = \frac{L_n + R_n}{2}$$

che forniscono rispettivamente le stime a sinistra e a destra (metodo dei rettangoli) e la stima con il metodo dei trapezi. Integriamo per esempio una funzione $f(x) = x\,cos(x)$ con $n = 10$, tra $a = 1$ e $b = 4$. Per prima cosa occorre determinare il valore del passo.

```
> 10->n
> 1->a
> 4->b
> (b-a)/n->h
```

Con il comando

```
> a+(0:n)*h->x;x
 [1] 1.0 1.3 1.6 1.9 2.2 2.5 2.8 3.1 3.4 3.7 4.0
```

o anche con il comando

```
> x=seq(a,b,length=n+1)
```

si ottengono gli estremi degli intervalli. Si calcolano poi i valori corrispondenti della funzione e li si sostituiscono nelle formule di approssimazione

```
> f<-function(x) cos(x)*x
> f(x)
```

```
 [1]  0.54030231  0.34774848 -0.04671924 -0.61425018
 [5] -1.29470246 -2.00285904 -2.63822255 -3.09731897
 [9] -3.28711385 -3.13797012 -2.61457448
> y<-f(x)
> ln<-h*sum(y[1:n] )
> rn<-h*sum(y[2:(n+1)])
> tn<-(ln+rn)/2
> tn
[1] -5.042563
```

Il confronto con il valore dell'integrale

```
> integrate(f,1,4)
-5.062627 with absolute error < 6e-14
```

è abbastanza buono. Aumentando il numero di punti della suddivisione la stima migliora.

```
> 100->n
> (b-a)/n->h
> seq(a,b,length=(n+1))->x
> y<-f(x)
> ln<-h*sum(y[1:n])
> rn<-h*sum(y[2:(n+1)])
> tn<-(ln+rn)/2
> tn
[1] -5.062426
> rn
[1] -5.109749
> ln
[1] -5.015103
```

3.7.3 Metodo Montecarlo

Supponiamo di voler applicare il metodo Montecarlo alla determinazione dell'integrale

$$\int_1^4 (x^3 + 3x)dx$$

Occorre determinare la finestra grafica da utilizzare come bersaglio. Nell'esempio x varia tra 1 e 4. Per quanto riguarda la y tracciamo il grafico della funzione

```
> curve(x^3+3*x,1,4)
```

Questa è una funzione crescente (fatto che avrei potuto determinare anche dallo studio del segno della derivata) e nell'intervallo in esame ha un minimo in $x = 1$ e un massimo in $x = 4$. I corrispondenti valori sono $m = 4$ e $M = 76$.

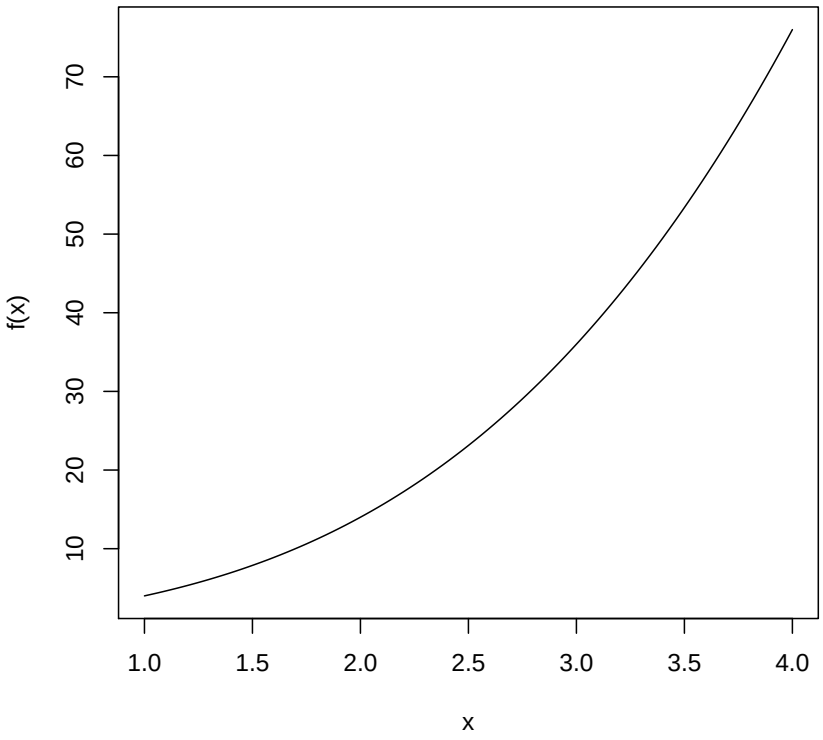


Figura 3.20: Determinazione del bersaglio

```
> f<-function(x) x^3+3*x
> f(4)
[1] 76
> f(1)
[1] 4
```

Possiamo quindi definire il bersaglio come

$$[1, 4] \times [0, 76]$$

Dobbiamo definire il numero di tiri. Il comando `runif`(n, a, b), (*r*andom-*unif*orm), permette di generare n numeri nell'intervallo (a, b) distribuiti uniformemente. Se l'intervallo non è specificato i numeri appartengono all'intervallo $(0, 1)$. Useremo il comando `runif` per effettuare i tiri.

```
> 1000->tiri;
> x<-runif(tiri,1,4)
> y<-runif(tiri,0,76)
> centri<-which(y<f(x))
> successi<-length(centri)
> successi/tiri*76*3
[1] 83.22
> integrate(f,1,4)
86.25 with absolute error < 9.6e-13
> plot(x[centri],y[centri],
+ col="red",cex=0.3)
> points(x[-centri],y[-centri],
+ col="blue",cex=0.3)
> curve(f,add=T)
```

```
[1] 80.712
86.25 with absolute error < 9.6e-13
```

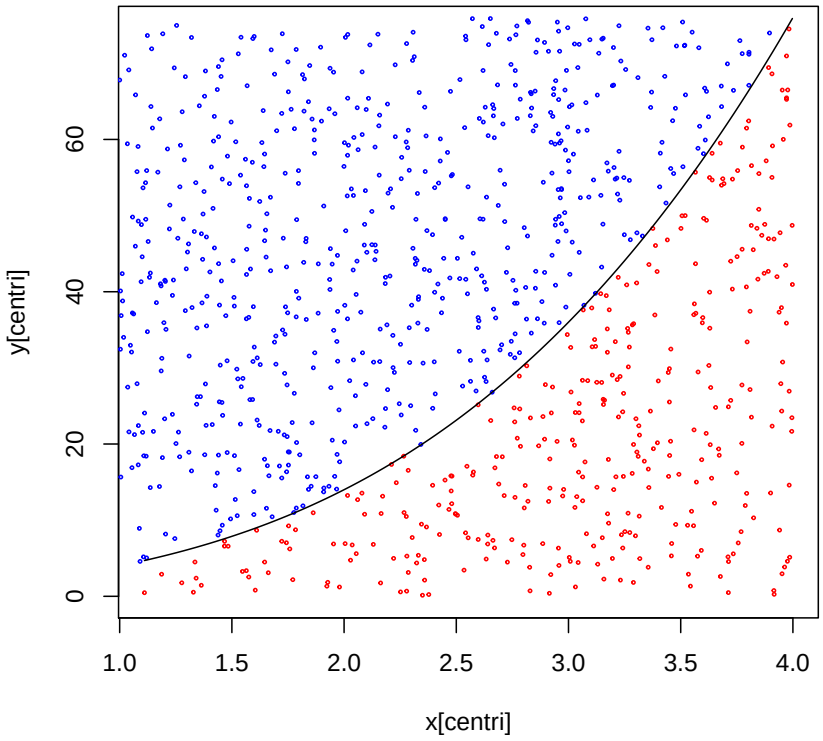


Figura 3.21: Metodo Montecarlo