Laboratorio di Calcolo (GdA, 2018)

Raccolta di script e di comandi R

1 Script R

1.1 primo_script.R

```
# primo script di comandi R
# (tutte le linee che cominciano con cancelletto sono dei commenti)

cat( sprintf("Si va a cominciare\n") )

a = 5
b = 3
cat( sprintf("le variabili a e b sono definite\n") )

c = a*b

cat( sprintf("\ncomando 'a'\n") )
a  # così da solo non dà nessun risultato

cat( sprintf(" Non ha prodotto niente\n\n") )  # notare i due '\n'
cat( sprintf(" Proviamo con print(a)\n") )
print(a)

cat( sprintf("\n meglio ancora con 'cat(sprintf()'\n") )  # notare i due '\n'
cat( sprintf(" a = %.0f \n", a) )  # notare la differenza fra i diversi risultati
cat( sprintf(" a = %.3f \n", a) )
cat( sprintf(" a = %.5f \n", a) )
cat( sprintf(" a = %d \n", a) )

cat( sprintf("\nProvare a stampare anche le altre variabili e altri testi\n") )
```

1.2 somma_prodotto_numeri_complessi.R

```
# e ora un esempio di somma e prodotto di numeri complessi
z1 = 1 + 0.5i
z2 = 0.5 + 1i

# prepariamo un plot senza niente
plot(NULL, xlim=c(-2,2), ylim=c(-2,2), xlab='Re()', ylab='Im()', asp=1)
grid()

# ci mettiamo i due numeri
z=z1;
points(z, pch=19, col='blue'); lines( c(0, Re(z)), c( 0, Im(z)), col='blue')
z=z2
points(z, pch=19, col='blue'); lines( c(0, Re(z)), c( 0, Im(z)), col='blue')

# e la somma
z=z1+z2
points(z, pch=19, col='red'); lines( c(0, Re(z)), c( 0, Im(z)), col='red')

# e poi il prodotto
```

```
z=z1*z2
points(z, pch=19, col='green'); lines( c(0, Re(z)), c( 0, Im(z)), col='green')

# interessante proprietà del prodotto
#
Mod(z1)
Mod(z2)
Mod(z1*z2)
Mod(z1)*Mod(z2)

Arg(z1)
Arg(z2)
Arg(z1*z2)
Arg(z1)+Arg(z2)

# Provare con altre coppie z1 e z2
```

1.3 plot_im.R

```
z <- (0+1i)^(1/4) * c(1, 1.05, 0.95)

colori = c('blue', 'red', 'green')   # colori

plot(0+1i, xlim=c(-2,2), ylim=c(-2,2), xlab='Re()', ylab='Im()', asp=1, cex=2)

for (n in 1:3) {
  for( k in 1:16) {
     points(z[n]^k, pch=19, col=colori[n])
     lines(c(0, Re(z[n]^k)), c(0, Im(z[n]^k)), col=colori[n] )
     Sys.sleep(0.2)
  }
}
```

1.4 eq2_0.R

```
a=1; b=-1; c=-3

discr = b^2 - 4*a*c
print(discr)

sol1 = ( -b + sqrt(discr) ) / (2*a)
print(sol1)
sol2 = ( -b - sqrt(discr) ) / (2*a)
print(sol2)
```

1.5 eq2_2.R

```
a=1; b=-1; c=3

discr = b^2 - 4*a*c
print(discr)

if(discr <0) discr <- discr * (1+0i)
```

```
sol = ( -b + c(-1,1) * sqrt(discr) ) / (2*a)
print(sol)
```

1.6 equazione_2gr.R

```
# equazione di secondo grado e grafico della parabola

solve.eq2 <- function(a, b, c) {
   # att: non ci sono controlli su a, b e c !!
   #
   discr <- b^2-4*a*c                          # discriminante
   if (discr < 0) discr <- as.complex(discr)  # se negativo -> complesso
   ( -b + c(-1,1)*sqrt(discr) ) / (2*a)       # soluzioni
}

salva.su.file = !TRUE  # per stampare su file togliere '!'

if (salva.su.file) png("equazione_secondo_grado.png")

a=2; b=-1; c=-3
sol <- solve.eq2(a, b, c)
print(sol)

# plot della funzione, con le soluzioni, se reali

titolo <- sprintf("a=%.1f, b=%.1f, b=%.1f", a, b, c)

plot(function(x) a*x^2 + b*x + c, -2, 2, ylab='y', col='blue', main=titolo)
grid()
abline(h=0)
if (!is.complex(sol[1])) {
  abline(v=sol[1], lty=2, col='magenta')
  abline(v=sol[2], lty=2, col='magenta')
}

if (salva.su.file) dev.off()
```

1.7 potenza_numeri_complessi_1.R

```
#script per mostrare interessanti proprietà
# delle potenze dei numeri complessi
# rappresentiamo 'i' nel piano complesso

salva.su.file = TRUE
if (salva.su.file) png("potenze_nr_complessi.png")

# plot vuoto, con griglia
plot(NULL, xlim=c(-2,2), ylim=c(-2,2), xlab='Re()', ylab='Im()', asp=1, cex=2)
grid()

# numero complesso di partenza: radice quarta di 'i'
z = (0+1i)^(1/4)

for (k in 1:16) {
   points(z^k, pch=19, col='blue')
   lines( c(0, Re(z^k)), c( 0, Im(z^k)), col='blue', lwd=2)
```

```
}
# nota: lwd determina lo spessore delle linee e lo cambiamo
#       per riuscire a vederle anche quando si sovrappongono

# cambiamo il nr di partenza e ripetiamo, con diverso colore
z = 1.05 * (0+1i)^(1/4)

for (k in 1:16) {
   points(z^k, pch=19, col='red')
   lines( c(0, Re(z^k)), c( 0, Im(z^k)), col='red')
}

# ancora, con diverso fattore
z = 0.95 * (0+1i)^(1/4)

for (k in 1:16) {
   points(z^k, pch=19, col='green')
   lines( c(0, Re(z^k)), c( 0, Im(z^k)), col='green', lwd=3)
}

if (salva.su.file) dev.off() # importante chiudere il 'device'
                             # solo allora il file sarà salvato
                             # nella directory di lavoro
```

1.8 potenza_numeri_complessi_2.R

```
# Variante di numeri_complessi_1.R, nella quale definiamo
# una funzione per disegnare puntino e segmento
# e una per creare il plot iniziale

# definizione delle funzioni ----------------------------------------------
start.plot.im <- function(Z) {
   # Z rappresenta il max di |Re(z)| e |Im(z)| rappresentabile
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='Re()', ylab='Im()', asp=1)
   grid()
}

plot.im <- function(z, pch=19, col='blue', lwd=1) {
   points(z, pch=pch, col=col)
   lines( c(0, Re(z)), c(0, Im(z)), col=col, lwd=lwd)
}

# Plot --------------------------------------------------------------------
salva.su.file = !TRUE
if (salva.su.file) png("potenze_nr_complessi.png")

start.plot.im(2)

z = (0+1i)^(1/4); for (k in 1:16) plot.im(z^k, lwd=2, col='blue')
z = 1.05*(0+1i)^(1/4); for (k in 1:16) plot.im(z^k, lwd=2, col='red' )
z = 0.95*(0+1i)^(1/4); for (k in 1:16) plot.im(z^k, lwd=3, col='green' )

if (salva.su.file) dev.off()
```

1.9 potenza_numeri_complessi_3.R

```
# Variante di numeri_complessi_2.R, nella quale
```

```
# - 'looppiamo' sulle tre condizioni iniziali
# - mettiamo una breve pausa fra una potenza e l'altra

# definizione delle funzioni ----------------------------------------
start.plot.im <- function(Z) {
   # Z rappresenta il max di |Re(z)| e |Im(z)| rappresentabile
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='Re( )', ylab='Im( )', asp=1)
   grid()
}

plot.im <- function(z, pch=19, col='blue', lwd=1) {
   points(z, pch=pch, col=col)
   lines( c(0, Re(z)), c(0, Im(z)), col=col, lwd=lwd)
}

# Plot --------------------------------------------------------------
salva.su.file = !TRUE
if (salva.su.file) png("potenze_nr_complessi.png")

start.plot.im(2)

rho  <- c(1, 1.05, 0.95)
cols <- c('blue', 'red', 'green')
lwds <- c(2, 2, 3)
for (i in 1:3) {               # notare il diverso significato di 'i'
    z = rho[i] * (0+1i)^(1/4)
    for (k in 1:16) {
       plot.im(z^k, lwd=lwds[i], col=cols[i])
       if(!salva.su.file) Sys.sleep(0.1)
    }
}

if (salva.su.file) dev.off()
```

1.10 sol_zn_eq_i_comandi.R

```
#####################################################
#
#  soluzione delle equazioni
#                               z^2 = i    Eq.1)
#                               z^4 = i    Eq.2)
#
#  NOTA: quest non è uno script(*), bensì una lista di comandi commentata
#        - eseguire un comando alla volta mediante copia/incolla
#          (o ancora meglio riscrivendo i comandi e ripescare quelli
#           vecchi con la freccia in alto)
#        - eventualmente trasformare in uno script che esegua
#          tutti i comandi, stampi i risultati mediante print()
#          o cat(sprintf()), ed eventualmente aggiungendoci anche
#          i comandi per salvare il grafico finale su un png (o anche pdf,
#          dipende dall'uso che si vuol fare del risultato).
#        - ogni tanto controllare le variabili in gioco con ls()
#          (e, ancora meglio, lavorare in una nuova directory in modo
#           da mettersi nella situazione co cominciare da capo)
#
#  (*) in realtà si può eseguire e si vede al volo il plot finale,
#      rischiando di non capire cosa facciano i diversi comandi,
#      e inoltre non si vedono i risultati numerici in quanto è
```

```
#      stato evitato l'uso di print() o cat(sprintf()), lasciati
#      come esercizio (vedi sopra)
# ------------------------------------------------------------------------
#
# definizione delle funzioni ---------------------------------------------
# plot nuovo
start.plot.im <- function(Z) {
   # Z rappresenta il max di |Re(z)| e |Im(z)| rappresentabile
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='Re( )', ylab='Im( )', asp=1)
   grid()
}

# mpstra un numero immaginario nel plot
show.im <- function(z, pch=19, col='blue', lwd=1.5, ...) {
   points(z, pch=pch, col=col, ...)
   lines( c(0, Re(z)), c(0, Im(z)), col=col, lwd=lwd, ...)
}

# funzione che disegna un cerchio  (sull'argomento "..." torneremo a lezione)
cerchio <- function(C=c(0,0), r=1, ...) {
  th=seq(0, 2*pi, by=0.02)
  points(C[1] + r*cos(th), C[2] + r*sin(th), ty='l', ...)
}

#-------------------------------------------------------------------------

# definiamo i   # notare i diversi significati dei due 'i'
i <- 0 + 1i

# lo visualizziamo in un plot
start.plot.im(1.2)
cerchio(col='grey')
show.im(i)

# soluzioni di Eq.1  [si noti l'ordine c(1,-1),
#                     per seguire la convenzione +- ]
sol.Eq1 <- c(1, -1) * sqrt(i)
sol.Eq1
Mod(sol.Eq1)
Arg(sol.Eq1)

# rappresentazione grafica
show.im(sol.Eq1[1], col='green')
show.im(sol.Eq1[2], col='green', lty=2)

# soluzioni di Eq.2
# basta iterare quanto fatto precedentemente
sol.Eq2 <- c(1, -1) * sqrt(sol.Eq1[1])
sol.Eq2[3:4] <- c(1, -1) * sqrt(sol.Eq1[2])
sol.Eq2
Mod(sol.Eq2)
Arg(sol.Eq2)

# controlliamo che siano veramente soluzioni
sol.Eq2^4

# rappresentazione grafica
# (tratto e colore dovrebbere aiutare a identificare le soluzioni)
show.im(sol.Eq2[1], col='red')
```

```
show.im(sol.Eq2[2], col='red', lty=3)
show.im(sol.Eq2[3], col='red', lty=2)
show.im(sol.Eq2[4], col='red', lty=3)
```

1.11 media.R

```
media <- function(x) {
   s <- 0
   for (i in 1:length(x) ) {
       s <- s + x[i]
   }
   m = s / length(x)
  return(m)
}
```

1.12 int_rette.R

```
m1 = 1; q1= -5
m2 = -2; q2 = 3

M= 10
plot(NULL, xlim=c(-M, M), ylim=c(-M,M), xlab='x', ylab='y', asp=1)
grid(lwd=2)
abline(v=0)
abline(h=0)

abline(q1, m1, col='blue')
abline(q2, m2, col='red')

xi = (q2 - q1) / ( m1 - m2)
yi = m1 * xi + q1

print(xi)
print(yi)

points(xi, yi)
```

1.13 ossatura_rette_2p_interattivo.R

```
# ------ funzione per disegnare un piano cartesiano vuoto
piano.cartesiano <- function(Z=10) {
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='x', ylab='y', asp=1)
   grid()
   abline(h=0)
   abline(v=0)
}

piano.cartesiano(10)

while(1) {       #  "while(1)"  -> ciclo infinito, interrotto da 'break'

  cat(sprintf("\n Clicca su due punti (coincidenti per uscire)\n"))

  p <- locator(2)
```

```
# METTERE DENTRO l'if la condizione tale che
# "se i due punti sono  vicini il ciclo while viene interrotto !
# sostituire "....." con la condizione opportuna

  if ( .....  ) break

# mettiamo sul plot i due punti
  points(p)

# CALCOLARE m e q a partire da p$x[1], p$y[1], p$x[2] e p$y[2]
  m = .....
  q = .....

  abline(q, m, col=sample(rainbow(7))[1] )

# sample(rainbow(7))[1] sceglie un colore a caso fra 7 dell'arcobaleno
# (ma invece di 7 si può anche mettere un altro numero 'ragionevole - provare)

}
```

1.14 ossatura_cerchi_2p.interattivo.R

```
# ------ funzione per disegnare un piano cartesiano vuoto
piano.cartesiano <- function(Z=10) {
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='x', ylab='y', asp=1)
   grid()
   abline(h=0)
   abline(v=0)
}

piano.cartesiano(10)

# valori di phi per tracciare il cerchio mediante
# le equazioni parametriche

n=101
phi <- seq(0, 2*pi, len=n)

while(1) {   # loop infinito: attenzione!

  cat(sprintf("\n Clicca su due punti (coincidenti per uscire)\n"))

  p <- locator(2)

  # CALCOLARE r come distanza fra due punti nel piano
  # (per le coordinate dei punti vedi ossatura_cerchi_2p.interattivo.R)

  r <- .....

  if ( r < 0.2) break    # se è 'piccolo' vuol dire che vogliamo uscire

  # CALCOLARE i vettori x e y che contengono i punti sulla circonferenza
  x <- .....
  y <- .....

  colore <- sample(rainbow(7))[1]
```

```
  # centro
  points(p$x[1], p$y[1], pch=3, col=colore)
  # circonferenza
  points(x, y, ty='l', lwd=2, col= colore)
}
```

1.15 fatt_rec.R

```
fattoriale <- function(n) {
    if(n < 2) return(1)
    return( n * fattoriale(n-1) )
}
```

1.16 molla_xva.R

```
#--------------------------------------
#  Oscillazione della molla: valutazione numerica
#
#  (Per le note a fine linea si veda in fondo)

k  = 50      # (N/m)   costante della molla
m  = 0.5     # kg      massa sospesa
xM = 0.05    # m       allungamento massimo (a t=0)

dt   = 0.01  # s       intervallo di discretizzazione
tMax = 1.5   # s       tempo massimo

t = 0                                                    # 1)
x = xM
v = 0
a = - k * x / m

i = 2  # indice delle iterazioni

while( t[i-1] < tMax ) {
   t[i] <- t[i-1] + dt
   v.m0 <- v[i-1] + a[i-1]*dt/2                          # 2)
   x.m0 <- x[i-1] + v.m0*dt/2                            # 3)
   a.m  <- ( - k * x.m0 ) / m                            # 4)
   v[i] <- v[i-1] + a.m*dt                               # 5)
   x[i] <- x[i-1] + (v[i-1] + v[i])/2 * dt               # 6)
   a[i] <- ( - k * x[i] ) / m                            # 7)
   i <- i + 1
}

plot(t, x/max(x), pch=19, col='blue', cex=0.5,           # 8)
     xlab='t (s)', ylab='x/xMax;  v/vMax;  a/aMax',
     ylim=c(-1,1.2) )
abline(h=0, col='gray')
points(t, v/max(v),  pch=19, col='cyan', cex=0.5)
points(t, a/max(a),  pch=19, col='red', cex=0.5)
text(0.15*max(t), 1.15, sprintf("xMax = %.2f m", max(x)), col='blue', cex=1.7)
text(0.5*max(t), 1.15, sprintf("vMax = %.2f m/s", max(v)), col='cyan', cex=1.7)
text(0.85*max(t), 1.15, sprintf("aMax = %.2f m/s^2", max(a)), col='red', cex=1.7)

#-- Note ------------------
```

```
# 1) Si ricorda che in R le variabili sono per default dei vettori,
#   ovvero "t = 0" crea il vettore t e assegna 0 al primo elemento.
#   Quindi prima di entrare nel loop while t e t[1] sono la stessa cosa
#
# 2) Si calcola la velocità a metà del nuovo intervallino dt
#    usando come accelerazione quella calcolata per alla posizione
#    del tempo precedente
#
# 3) Posizione a metà del nuovo intervallino, basata sulla veocità
#    'intermedia' v.m0
#
# 4) Si calcola l'accelerazione dalla forza calcolata in x.m0
#    (si noti come dal punto di vista computazionale sarebbe stato
#     più efficiente calcolare una tantum k/m prima del loop,
#     ma per uno script di questo tipo tale accortezza non è cruciale e
#     rende lo script più intellegibile)
#
# 5-7) Calcoliamo finalmente velocità, posizione e accelerazione
#      in corrispondenza di t[i]
#
#
# 8) Vecchio plot apparso sul sito il 31 ottobre:
# plot(t, x*100, pch=19, col='blue', cex=0.8,
#     xlab='t (s)', ylab='x (cm);  v (dm/s)')
# abline(h=0, col='gray')
# points(t, v*10,  pch=19, col='red', cex=0.8)
# text(0.15, 3, "x", col='blue', cex=2)
# text(0.02, -3.2, "v", col='red', cex=2)
```

1.17 quarto_cerchio_sampling.R

```
#  Quarto di cerchio:
#   valutazioni area e pi greco mediante campionamento
                                          #
N = 10000

x <- runif(N)
y <- runif(N)
plot(x, y, pch=19, cex=0.2, col='blue', asp=1)

IN <- (x^2 + y^2) <= 1
points(x[IN], y[IN], pch=19, cex=0.25, col='red')

n.IN <- sum(IN)

area.stimata =  n.IN/N
cat( sprintf("Area stimata:  %f\n", area.stimata) )

pi.stimato = n.IN/N * 4
cat( sprintf("Pi greco stimato:  %f\n", pi.stimato) )
```

1.18 quarto_cerchio_numeric.R

```
#  Quarto di cerchio: valutazione numerica dell'area (e di pi)
#
```

```
# funzione del quarto di cerchio
# (generico nome fun per riutilizzarlo con altre funzioni)
fun <- function(x) sqrt(1-x^2)  # att!! senza controlli!!

# per un esempio di altra funzione, ben più complicata, vedi in fondo

N = 30   # nr di intervalli
stima.pi = TRUE

# disegniamo il quarto di cerchio di raggio unitario
x <- seq(0, 1, len=201)         # x usato per il disegno
plot(x, fun(x), ylim=c(0, max(fun(x))), ty='l', asp=1, col='blue')
abline(h=0)
abline(v=0)

x <- seq(0, 1, len=(N+1))       # x usato per discretizzazione
dx = 1 / N

l = 0
A = 0
for (i in 1:length(x)) {
    xi <- x[i]
    points(rep(xi,2) , c(0, fun(xi) ) , ty='l', lty=2)
    if (xi > 0) {
        xc <- xi - dx/2
        points(xc, fun(xc), pch=19)
        points(c(x[i-1], x[i]), c(fun(x[i-1]), fun(x[i])), ty='l', col='red')

        A = A + dx * fun(xc)
        dy = fun(x[i]) - fun(x[i-1])
        l = l + sqrt(dx^2 + dy^2)
    }
}

cat(sprintf("N = %d\n", N))

if(stima.pi) {
    cat(sprintf("Area = %f  (esatta: %f)\n", A, pi/4))
    cat(sprintf("Arco = %f  (esatto: %f)\n", l, pi/2))
    cat(sprintf("\nStime di pi greco\n"))
    cat(sprintf("Dall'area: %f\n", 4 * A))
    cat(sprintf("Dall'arco: %f\n", 2 * l))
} else {
    cat(sprintf("Area = %f\n", A))
    cat(sprintf("Arco = %f\n", l))
}


# esempio di funzione complicata
# fun <- function(x)  1 / log( 1 + sin(1+x)^4 ) - 1
# comando da aggiungere dopo plot per avere il titolo
# title(" y = 1 / log( 1 + sin(1+x)^4 ) - 1 ")
# (Att: in questo caso è preferibile togliere 'asp=1')
```

1.19 minimax.R

```
# ci creiamo un vettore di numeri aleatori (pseudo-)casuali
```

```
N = 10
v <- runif(10)

cat(sprintf("Valori generati\n"))
print(v)
cat(sprintf("Valori ordinati\n"))
print(sort(v))

min  = v[1]
imin = 1
max = v[1]
imax = 1

for (i in 2:N) {
  if (v[i] < min) {
     min <- v[i]
     imin = i
  }
  if (v[i] > max) {
     max <- v[i]
     imax = i
  }
}

cat(sprintf("\nminimo:  v[%d] = %f\n", imin, min))
cat(sprintf("massimo:  v[%d] = %f\n", imax, max))
```

1.20 intro_matrici.R

```
# come costruire/riempire una matrice

# 1) per righe
A = rbind( c(1, 3), c(-3, 5) )

# esempi di accesso
A[1,1]
A[1,2]

A[1,]
A[,2]

A[1,] = -1 * A[1,]
A[,2] = 2 * A[,2]
# (utile per il metodo di eliminazione di Gauss)

diag(A)
t(A)
diag(A) = c(7, -7)

# 2) per colonna
B = cbind( c(1, 3), c(-3, 5) )

# ordinamento interno in memoria (è sempre per colonne)
as.vector(A)
as.vector(B)

# dimensioni
dim(A)
```

```
# 3) elementi + dimensioni
C = matrix(1:4, c(2,2))

# 3a) ma è sufficiente dare anche solo in nr di righe o di colonne
D = matrix(1:6, c(2,3))
D = matrix(1:6, nrow=2)
D = matrix(1:6, ncol=3)

# le dimensioni possono essere cambiate
dim(D) = c(3,2)

# 4) scrivere la matrice tale e quale (se corta...) con opzione byrow=TRUE
A <- matrix(c(4, 2, 1,
              5, 6, 7,
              1, 0, 3), nrow=3, byrow=TRUE)


# operazioni fra matrici
# (i vettori sono visti come matrici di una sola colonna)
A %*% B

# A %*% D   # dà errore
D %*% A   # ok

1:3 %*% 1:3     # curiosamente funziona
t(1:3) %*% 1:3  # più rigorosamente

# per trasformare in semplice 'numero' (in realtà un vettore)
as.vector( t(1:3) %*% 1:3 )

1:3 %*% t(1:3)

outer(1:3, 1:3) # ricordate?

# matrice inversa
iA = solve(A)
iA %*% A
A %*% iA
```

1.21 trasposta_2.R

```
trasp <- function(A) {
  A = as.matrix(A)       # trasforma un eventuale vettore
                         # in una matrice di una sola colonna
  nr = dim(A)[2]
  nc = dim(A)[1]
  tA = matrix( rep(NA, nr*nc), c(nr, nc) )
  for(i in 1:nr) {
    for(j in 1:nc) {
      tA[i,j] = A[j,i]
    }
  }
  return(tA)
}
```

1.22 prodotto_matrici.R

```
mat.prod <- function(A, B) {
   # convertiamo eventuali vettori in matrici 1x1
   A = as.matrix(A)
   B = as.matrix(B)
   # controlliamo che le dimensioni siano 'consistenti'
   if(dim(A)[2] != dim(B)[1]) {
      cat(sprintf("Il nr di righe di B deve essere uguale al nr di col di A\n"))
      return(NULL)
   }
   nr = dim(A)[1]
   nc = dim(B)[2]
   nj = dim(A)[2]
   P = matrix( rep(0, nr*nc), c(nr, nc) )
   for(i in 1:nr) {
     for(k in 1:nc){
       for(j in 1:nj) {
         P[i, k] = P[i,k] + A[i,j]*B[j,k]
       }
     }
   }
   return(P)
}
```

1.23 Gauss_elim_steps.R

```
# metodo di eliminazione di Gauss

rid.Gauss.nz <- function(A) {
# caso di elementi tutti diversi da zero
    if(dim(A)[1] != dim(A)[2]) {
    cat(sprintf("La matrice deve essere quadrata\n"))
      return(NULL)
    }
    if (any(A ==0))  {
        cat(sprintf("Questo metodo funziona solo con tutti elementi !=0"))
        return(NULL)
    }

    nr = nc = dim(A)[1]
    if(nr == 1) {
        return(A)
    } else if (nr > 4) {
        cat(sprintf("Max matrici 4x4"))
        return(NULL)
    }
    print(A)

    # primo passo
    for (i in 2:nr) {
       A[i,] = A[i,] -  A[1,] *  A[i,1]/A[1,1]
    }
    print(A, dig=3)
    if(nr == 2) {
        return(A)
    }
```

```
    # secondo passo
    for (i in 3:nr) {
       A[i,] = A[i,] -  A[2,] *  A[i,2]/A[2,2]
    }
    print(A, dig=3)
    if(nr == 3) {
        return(A)
    }

    # terzo passo
    for (i in 4:nr) {
       A[i,] = A[i,] -  A[3,] *  A[i,3]/A[3,3]
    }
    print(A, dig=3)
    if(nr == 4) {
        return(A)
    }

}
```

1.24 Gauss_elim.R

```
# metodo di eliminazione di Gauss

elim.Gauss <- function(A, dbg=FALSE) {
# (con gestione di trattamento di pivot nulli)
    if(dim(A)[1] != dim(A)[2]) {
    cat(sprintf("La matrice deve essere quadrata\n"))
      return(NULL)
    }
    nr = nc = dim(A)[1]
    if(nr == 1) {
        return(A)
    }

    if(dbg) print(A)
    for (k in 2:nr) {
        if(dbg) print(k)
        pivot = A[k-1,k-1]
        if(dbg) cat(sprintf("Pivot riga %d: %f\n", k-1, pivot))
        if (pivot == 0) {   # cerca la prima riga con candidato pivot != 0
          if(dbg) cat(sprintf("Pivot nullo in riga = %d\n", k-1))
          new.row = 0
          for (i in k:nr) {
              if (A[i,k-1] != 0) {
                  new.row = i
                  if(dbg)  cat(sprintf("new.row %d \n", new.row ))
                  tmp = A[k-1,]
                  if(dbg) print(tmp)
                  A[k-1,] = A[i,]
                  A[i,] = tmp
                  if(dbg) print(A)
              }
              if(dbg) cat(sprintf("dopo scambio fra riga %d e riga %d\n", k-1, i))
              if(dbg) print(A)
              if(new.row != 0) break
          }
          if (new.row == 0) {
```

```
            cat(sprintf("Non è stato possibile scambiare due righe\n"))
            return(NULL)
          }
        }
        pivot = A[k-1,k-1]
        if(dbg) cat(sprintf("Pivot riga %d: %f\n", k-1, pivot))
        for (i in k:nr) {
            A[i,] = A[i,] -  A[k-1,] *  A[i,k-1] / pivot
        }
        if(dbg) print(A)
    }

    return(A)
}
```

1.25 sinusoidi_1.R

```
omega = 1 # s^-1
X     = 1 # cm
phi   = 0

T = 2*pi/omega

t <- seq(0, 2*T, len=201)

old.mar = par("mar")
par(mar=c(4.2,4,2.6, 0.3))
plot(t, X*cos(omega*t+phi), ty='l', col='blue', lwd=2,
     ylim=c(-1,1.2), xlab='t  (s)', ylab='x  (cm)',
main = "x(t) = X*cos(omega*t+phi);  X=1cm, omega=1s^-1")
points(t, omega*X*cos(omega*t+phi+pi/2), ty='l', col='cyan', lwd=2)
points(t, omega^2*X*cos(omega*t+phi+pi), ty='l', col='red', lwd=2)
text(1.,1.12, "phi=0", col='blue', cex=2)
text(4.7,1.12, "phi=pi/2", col='cyan', cex=2)
text(9.5,1.12, "phi=pi", col='red', cex=2)
par(mar=old.mar)
```

1.26 sinusoidi_2.R

```
omega = 1 # s^-1
X     = 1 # cm
phi   = 0

T = 2*pi/omega

t <- seq(0, 2*T, len=201)

old.mar = par("mar")
par(mar=c(4.2,4,2.5, 0.3))
plot(t, X*cos(omega*t+phi), ty='l', col='blue', lwd=2,
     ylim=c(-1,1.2), xlab='t  (s)',
     ylab="x (cm);  x' (cm/s);  x'' (cm/s^2)",
     main = "x(t) = X*cos(omega*t+phi),  x'(t),  x''(t);   X=1cm, omega=1s^-1")
points(t, omega*X*cos(omega*t+phi+pi/2), ty='l', col='cyan', lwd=2)
points(t, omega^2*X*cos(omega*t+phi+pi), ty='l', col='red', lwd=2)
text(0.8,1.12, "x(t)", col='blue', cex=2)
```

```
text(4.8,1.12, "x'(t)", col='cyan', cex=2)
text(9.5,1.12, "x''(t)", col='red', cex=2)
par(mar=old.mar)
```

1.27 carica_file.R

```
# carica un file di lanci

nomeFile <- readline(prompt="dai il nome del file (return per il precedente) ")

if (nomeFile != '') {
  nomeFileOld <- nomeFile
} else {
  nomeFile <- nomeFileOld
}

cat(sprintf("\nvado a caricare  %s\n", nomeFile))
lancio <- read.table(nomeFile, header=TRUE)

cat(sprintf("%s contiene %d istanze di %d variabili: ",
            nomeFile, dim(lancio)[1], dim(lancio)[2] ))
cat(sprintf("%s\n", paste(colnames(lancio), collapse = ', ') ))

# plot riassuntivo (matrice di plot!)
plot(lancio)
```

1.28 analizza_lancio.R

```
# analizza un file di lanci

source("carica_file.R")

readline(prompt="dai enter per continuare")
# equazioni orarie e velocità verticale
par(mfrow=c(3,1))
plot(lancio$t, lancio$x, ty='b', col='blue', xlab='t (s)', ylab='x (m)')
plot(lancio$t, lancio$y, ty='b', col='blue', xlab='t (s)', ylab='y (m)')
plot(lancio$t, lancio$vy, ty='b', col='orange', xlab='t (s)', ylab='vy (m/s)')
par(mfrow=c(1,1))

readline(prompt="dai enter per continuare")
# traiettoria
plot(lancio$x, lancio$y, asp=1, ty='b', col='brown',
     xlab='x (m)', ylab='y (m)')
abline(h=0, col='gray')
readline(prompt="dai enter per continuare")

# ricostruiamo la velocità lungo la y dai punti
dt = lancio$t[2] - lancio$t[1]
np = length(lancio$t)
vy.ric <- (lancio$y[2:np] - lancio$y[1:(np-1)] ) / dt

# e la confrontiamo con quella calcolata nel programma C
plot(lancio$t, lancio$vy, col='orange', xlab='t (s)', ylab='vy (m/s)')
readline(prompt="dai enter per continuare")
points(lancio$t[1:(np-1)] + dt/2, vy.ric, ty='b', col='red')
```

```
readline(prompt="dai enter per continuare")

# calcolamoci la lunghezza della traiettoria in funzione del tempo
# a partire dai tratti in ogni dt
ds = sqrt((lancio$y[2:np] - lancio$y[1:(np-1)] )^2 +
          (lancio$x[2:np] - lancio$x[1:(np-1)] )^2 )
s = cumsum(ds)
plot(lancio$t[1:(np-1)] + dt/2, s, ty='l',
     col='blue', xlab='t (s)', ylab='s (m)')
  # tracciamo un segmento fra primo e ultimo punto
  # per apprezzare la curvatura di s in funzione del tempo
points(c(lancio$t[1], lancio$t[np-1]), c(s[1],s[np-1]), ty='l', lty=2)
readline(prompt="dai enter per continuare")

# e, visto che ci siamo, dai ds ci calcoliamo anche il modulo
# della velocità in funzione del tempo (in t + dt/2 !!)
v = ds / dt

# che possiamo confrontare con quella da vx e vy
# (calcolata però per 0, dt, 2*dt, etc. )
v.da.componenti <- sqrt(lancio$vx^2 + lancio$vy^2)

plot(lancio$t, v.da.componenti, col='blue', ylim=c(0, max(v)),
     xlab='t (s)', ylab='v (m/s)', main="modulo della velocità")
readline(prompt="dai enter per continuare")
points(lancio$t[1:(np-1)] + dt/2, v, ty='b', col='red')
```

1.29 parametri_lancio.R

```
# ricava i parametri del lancio dal nome del file
parametri.lancio <- function(nome) {
   v        <- as.numeric( strsplit(nomeFile, '_')[[1]][2] )
   theta    <- as.numeric( strsplit(nomeFile, '_')[[1]][3] )
   altro    <- strsplit(nomeFile, '_')[[1]][4]
   s.dec    <- strsplit(altro, '.', TRUE)[[1]][1:2]
   dt <- as.numeric ( paste(s.dec, collapse='.') )

# in una sola riga, ma criptico
#  dt <- as.numeric ( paste( strsplit( strsplit(nomeFile, '_')[[1]][4], '.', 1)[[1]][1:2], collapse='.') )

   return(list(v=v, theta=theta, dt=dt))
}

par <- parametri.lancio(nomeFile)

print(par)

cat("Ora che abbiamo i parametri del lancio, possiamo ricalcolarci tutto in R\n")
```

1.30 carica_csv.R

```
# carica un file di lanci

nomeFile <- readline(prompt="dai il nome del file (return per il precedente) ")

if (nomeFile != '') {
```

```
  nomeFileOld <- nomeFile
} else {
  nomeFile <- nomeFileOld
}

cat(sprintf("\nvado a caricare  %s\n", nomeFile))
lancio <- read.csv(nomeFile, header=TRUE)

cat(sprintf("%s contiene %d istanze di %d variabili: ",
            nomeFile, dim(lancio)[1], dim(lancio)[2] ))
cat(sprintf("%s\n", paste(colnames(lancio), collapse = ', ') ))

# plot riassuntivo (matrice di plot!)
plot(lancio)
```

1.31 analizza_lancio_1.R

```
# analizza un file di lanci

source("carica_file.R")

readline(prompt="dai enter per continuare")

# ridefiniamo i margini
old.mar = par("mar")
par(mar=c(4,4,0.3, 0.3))

par(mfrow=c(2,1))
# equazioni orarie e velocità in funzione del tempo
plot(lancio$t, lancio$x, ty='b', col='blue', xlab='t (s)', ylab='x (m)')
plot(lancio$t, lancio$y, ty='b', col='blue', xlab='t (s)', ylab='y (m)')

plot(lancio$t, lancio$vx, ty='b', col='orange', ylim=c(0, max(lancio$vx)),
     xlab='t (s)', ylab='vx (m/s)')
plot(lancio$t, lancio$vy, ty='b', col='orange', xlab='t (s)', ylab='vy (m/s)')
abline(h=0, lty=2)
par(mfrow=c(1,1))

readline(prompt="dai enter per continuare")
# traiettoria
plot(lancio$x, lancio$y, asp=1, ty='b', col='brown',
     xlab='x (m)', ylab='y (m)')
abline(h=0, col='gray')
readline(prompt="dai enter per continuare")

# ricostruiamo la velocità lungo la y dai punti
dt = lancio$t[2] - lancio$t[1]
np = length(lancio$t)
vy.ric <- (lancio$y[2:np] - lancio$y[1:(np-1)] ) / dt

# e la confrontiamo con quella calcolata nel programma C
plot(lancio$t, lancio$vy, col='orange', xlab='t (s)', ylab='vy (m/s)')
readline(prompt="dai enter per continuare")
points(lancio$t[1:(np-1)] + dt/2, vy.ric, ty='b', col='red')
readline(prompt="dai enter per continuare")

# calcolamoci la lunghezza della traiettoria in funzione del tempo
# a partire dai tratti in ogni dt
```

```
ds = sqrt((lancio$y[2:np] - lancio$y[1:(np-1)] )^2 +
          (lancio$x[2:np] - lancio$x[1:(np-1)] )^2 )
s = cumsum(ds)
plot(lancio$t[1:(np-1)] + dt/2, s, ty='l',
     col='blue', xlab='t (s)', ylab='s (m)')

#  # tracciamo un segmento fra primo e ultimo punto
#  # per apprezzare la curvatura di s in funzione del tempo
# points(c(lancio$t[1], lancio$t[np-1]), c(s[1],s[np-1]), ty='l', lty=2)
readline(prompt="dai enter per continuare")

# e, visto che ci siamo, dai ds ci calcoliamo anche il modulo
# della velocità in funzione del tempo (in t + dt/2 !!)
v = ds / dt

# che possiamo confrontare con quella da vx e vy
# (calcolata però per 0, dt, 2*dt, etc. )
v.da.componenti <- sqrt(lancio$vx^2 + lancio$vy^2)

par(mar=c(4,4,3,0.3))
plot(lancio$t, v.da.componenti, col='blue', ylim=c(0, max(v)),
     xlab='t (s)', ylab='v (m/s)', main="modulo della velocità")
readline(prompt="dai enter per continuare")
points(lancio$t[1:(np-1)] + dt/2, v, ty='b', col='red')

par(mar=old.mar)
```

1.32 data_frame.R

```
# Non è uno script: da eseguire riga per riga

source("carica_file.R")  # ricarichiamo un file se non lo abbiamo fatto

str(lancio)

summary(lancio)

L <- as.matrix(lancio)    # anche 'data.matrix()'

# abbiamo quindi diversi modi di accedere ai vettori delle istanze
# di ciascuna variabile, ad esempio per t
lancio$t
lancio[[1]
L[,1]
L[,'t']      # vantaggio: non bisogna ricordare il numero d'ordine

# mentre con le righe abbiamo accesso alle variabili
# nelle stesse 'condizioni' (allo stesso tempo, nel nostro caso),
# ad esempio
L[1,]

# da notare come le colonne hanno dei nome, come abbiamo visto sopra
dimnames(L)

# se vogliamo, possiamo rimuovere i nomi
dimnames(L) = NULL
L[1,]
```

```
# o anche metterne altri (fuori programma...)
#-----------------------------------------------------
#
# etc. etc. (fuori programma...)

# per quello che ci interessa, spesso ci conviene
# mettere le colonne di interesse in vettori
# aventi i nomi delle colonne stesse e fare le nostre analist,
# ad esempio
t <- as.numeric(lancio$t)

#-------------------------------------------------------------
#-------------------------------------------------------------
# Sempre, FUORI PROGRAMMA, ecco come creare delle variabili al volo,
# in questo caso con il contenuto che ci interessa

rm(t,x,y,vx,vy)   # per essere sicuri di crearle veramente!

for(i in 1:dim(L)[2]) assign(dimnames(L)[[2]][i] , L[,i])

# controlliamo
t
x
y
vx
vy

# Quindi, potremmo fare una cosa del genere all'inizio,
# ad esempio
l <- read.table("lancio_10_45_0.05.txt", header=TRUE)
dimnames(l)[[2]]
for(i in 1:dim(l)[2]) assign(dimnames(l)[[2]][i] , l[[i]])
```

1.33 esempi_list.R

```
# esempi di 'liste'
# Non è uno script: eseguire da riga di comando con copia/incolla

a = list(c(2,3,4), "ciao a tutti", 23, matrix(1:4, nrow=2))
a
a[[4]]
a[[4]][1,2]

b = list(lista=a, vettore=1:4, matrice=rbind(c(3,-1), c(2,3)) )
b
b[[1]][[2]]
b$lista[[4]]
b$lista[[4]][2,2]
b$matrice
b$matrice[2,1]
b[[3]][2,1]
```

1.34 TerraGiove_vis.R

```
# prodotto scalare:
sc.prod <- function(v1, v2)  as.numeric( t(v1) %*% v2 )
```

```
# angolo fra vettori
angolo.fra.vettori <- function(v1, v2) {
  costh <- sc.prod(v1, v2) / (sqrt(sc.prod(v1, v1)) * sqrt(sc.prod(v2, v2)))
  return( acos(costh) * 180/pi )
}

#--------------------------------

R.T = 1 # U.A    (unità astronomica)
T.T = 1 # anno

R.G = 5.2 #
T.G = T.T * R.G^(3/2)

omega.T = 2*pi/T.T
omega.G = 2*pi/T.G

cat(sprintf("Distanza Giove-Sole: %.2fU.A; periodo: %.2f y\n", R.G, T.G))

cat(sprintf("Vel. angolari: Terra: %.3f rad/y;  Giove: %.2f rad/y\n",
            omega.T, omega.G))

t <- seq(0, 1, by=1/104)

xT = R.T * cos(omega.T*t)
yT = R.T * sin(omega.T*t)
plot(xT, yT, asp=1, col='blue')
readline(prompt = "Premi Invio per continuare")

t <- seq(0, 12, by=1/104)
xG = R.G * cos(omega.G*t)
yG = R.G * sin(omega.G*t)

xT = R.T * cos(omega.T*t)
yT = R.T * sin(omega.T*t)

plot(xG, yG, asp=1, col='orange')
points(xT, yT, cex=0.5, col='blue')

readline(prompt = "Premi Invio per continuare")
plot(xG-xT, yG-yT, asp=1, col='orange')

readline(prompt = "Premi Invio per continuare")
plot(xG[1]-xT[1], yG[1]-yT[1], asp=1, col='orange',
    xlab='xG - xT', ylab='yG - yT',
    xlim=range(xG-xT), ylim=range(yG-yT))
for(i in 2:length(t)) {
  points(xG[i]-xT[i], yG[i]-yT[i], col='orange')
  Sys.sleep(0.002)
}

# Complemento: plot solo se Giove è 'ben visibile'
readline(prompt = "Premi Invio per continuare")
plot(xG[1]-xT[1], yG[1]-yT[1], asp=1, col='orange',
    xlab='xG - xT', ylab='yG - yT',
    xlim=range(xG-xT), ylim=range(yG-yT))
for(i in 2:length(t)) {
  if (angolo.fra.vettori(c(xT[i],yT[i]), c(xG[i],yG[i])) < 45) {
```

```
    colore = 'orange'
  } else {
    colore = 'gray97'   # se non visibile
  }
  points(xG[i]-xT[i], yG[i]-yT[i], col=colore)
  Sys.sleep(0.002)
}
```

1.35 histo_etc.R

```
# Non è uno script: eseguire una riga alla volta
# ripartiamo dai numeri random uniformi continui
runif(1)

runif(1, 0, 10)

runif(10, -1, 1)
runif(10, -1, 1)

( x = runif(10) )
( x = runif(10) )

?runif

x = runif(100000)
hist(x, col='cyan')
hist(x, col='cyan', nc=100)
mean(x)

# numeri interi
sample(1:10)     # li mischia!

# mischia qualsiasi cosa  purché siano elementi di un vettore
sample( c('a', 'b', 'c', 'd') )

sample(letters)
sample(LETTERS)
sample(c(LETTERS, letters))
sample(c(LETTERS, letters), 10)

# generatore di password da 10 lettere non ripetute
paste(sample(c(LETTERS, letters), 10), collapse='' )

# ottimo per fare un generatore di tombola
n.estr <- sample(90)
length(n.estr)
n.estr

for(n in n.estr) {cat(sprintf("Uscito %d\n", n)); Sys.sleep(1)}

# ma può essere usato anche per fare un vero generatore
# di numeri (o altri 'oggetti') casuali, mediante "rep=TRUE"
# e il secondo argomento che indica il nr di valori

# 100 lanci di dadi
sample(6, 100, rep=TRUE)

# 100 numeri della roulette
```

```
sample(0:36, 100, rep=TRUE)

# o una password con lettere e cifre anche ripetute
# step 1, per capire cosa succede
sample(c(letters, LETTERS, 0:9))
# o, per aumentare la probabilità che ci siano delle cifre,
sample(c(letters, LETTERS, rep(0:9,3)))

# quindi
paste( sample(c(letters, LETTERS, rep(0:9,3)), 12), collapse='' )

#-----------------------------------------------------------------
# analisi di numeri random, per avere un'idea ...
# (niente di rigoroso dal punto di vista probabilistico)
n <- sample(6, 10000, rep=TRUE)
hist(n, col='cyan')
# -> brutto: hist non è fatto per queste cose...

table(n)
( tn <- table(n) )

barplot(tn, col='cyan')

barplot(tn, col='cyan', names=LETTERS[1:6] )

pie(tn)
# anche se è banale

citta = c("Roma", "Milano", "Napoli", "Torino", "Palermo")
mil.abi = c(2.9, 1.4, 0.97, 0.88, 0.67)
pie(mil.abi, citta)
pie(mil.abi, citta, col=rainbow(5))

barplot(mil.abi, names=citta, col=rainbow(5))
barplot(mil.abi, names=citta, col=rainbow(20)[8:12])

#-----------------------------------------------------------------
# Ripartiamo dai dadi
n1 <- sample(6, 1000, rep=TRUE)
n2 <- sample(6, 1000, rep=TRUE)

par(mfrow=c(2,1))
barplot(table(n1), col='cyan')
barplot(table(n2), col='cyan')

s = n1 + n2
d = n1 - n2

barplot(table(s), col='cyan')
barplot(table(d), col='cyan')

#-----------------------------------------------------------------
# Una cosa simile vale anche per variabili continue:
x1 <- runif(10000)
x2 <- runif(10000)


hist(x1, nc=50)
hist(x2, nc=50)
```

```
sx = x1 + x2
dx = x1 - x2
hist(sx, nc=50)
hist(dx, nc=50)

x3 <- runif(10000)
x4 <- runif(10000)
x5 <- runif(10000)

par(mfrow=c(3,1))
hist(sx, nc=50)
hist(sx + x3, nc=50)
hist(sx + x3 + x4 + x5, nc=50)

x6 <- runif(10000)
x7 <- runif(10000)
x8 <- runif(10000)
x9 <- runif(10000)

par(mfrow=c(1,1))
hist(sx + x3 + x4 + x5 + x6+x7+x8+x9, nc=50)
```

1.36 newton.R

```
# ricerca degli zeri mediante il metodo di Newton

pausa <- function() readline(prompt="\nDai Enter per continuare")

fun <- function(x) x^2 - N

N <- as.numeric(readline(
     prompt="Dai il numero del quale si vuol trovare la radice ") )

x    <- seq(-sqrt(N)*1.5, sqrt(N)*3, len=101)
xtxt = sqrt(N)*0.1
ytxt = max(fun(x))
dy   = ytxt/15
ylab <- sprintf("x^2 - %f", N)
main <- sprintf("Algoritmo di Newton per trovare Sqrt(%f)", N)
plot(x, fun(x), ty='l', col='blue', ylab=ylab, main=main,
     ylim=c(min(fun(x))*1.5, max(fun(x))))
abline(h=0, col='gray')
abline(v=0, col='gray')

cat(sprintf("Clicca su un punto delle ascisse con x positiva"))
p0 = locator(1)
x  = p0$x

x0 = Inf
while( abs(x-x0) > 0.000001 ) {
  testo = sprintf("x = %.6f", x)
  text(xtxt, ytxt, testo, cex=2, pos=4, col='blue')
  points(x, 0, pch=19)

  pausa()
  points(x, fun(x), pch=19, col='blue')
  lines(c(x,x), c(0,fun(x)), lty=2, col='orange')
```

```
  pausa()
  m = 2*x
  c = fun(x) - m*x
  abline(c, m, col='red', lty=2)
  x0 = x
  x=-c/m
  points(x, 0, pch=19)
  ytxt = ytxt - dy
}
```

1.37 minimizzazione.R

```
# Algoritmo di minimizzazione a 'scivolamento'

pausa <- function() readline(prompt="\nDai Enter per continuare")

fun  <- function(x) 0.3*x^3 + 3*x^2 + 3*x -10
dfun <- function(x) 3*0.3*x^2 + 2*3*x + 3

alpha = 0.2
x    <- seq(-8, 3, len=101)
main <- sprintf("Algoritmo di minimizzazione (alpha = %.2f)", alpha)
plot(x, fun(x), ty='l', col='blue', ylab='f(x)', main=main)
abline(h=0, col='gray')
abline(v=0, col='gray')

cat(sprintf("Clicca su un punto delle ascisse in 'prossimità' del minimo"))
p0 = locator(1)
x  = p0$x

i=0
x0 = Inf
while( abs(x-x0) > 0.000001  ) {
  cat(sprintf(" x = %f\n", x))
  points(x, 0, pch=19)
  points(x, fun(x), pch=19, col='blue')
  lines(c(x,x), c(0,fun(x)), lty=2, col='orange')

  m = dfun(x)
  c = fun(x) - m*x
  abline(c, m, col='red', lty=2)

  pausa()
  x0 = x
  x = x - alpha*m

  i = i+1
}

cat(sprintf("minimo in x=%f raggiunto in %d step\n", x, i))
text(-2.5, 30, sprintf("minimo in x=%f\nraggiunto in %d step\n", x, i), cex=1.5)
```

1.38 analizza_pi.R

```
# comandi per analizzare le prime 1000 cifre decimali di pi greco
```

```
# file generato con
#  echo "scale=10000; 4*a(1)" | BC_LINE_LENGTH=0 bc -l > pi1000.txt

pi <- read.table('pi1000.txt')   # Non funziona!

file.info('pi1000.txt')
fi <- file.info('pi1000.txt')
str(fi)

fileName = 'pi1000.txt'
readChar(fileName, file.info(fileName)$size)

pi1000 <- readChar(fileName, file.info(fileName)$size - 1)
ch.pi1000 <- strsplit(pi1000, split='')
str(ch.pi1000)

dec.pi = as.numeric( ch.pi1000[[1]][-(1:2)] )
barplot(table(dec.pi), col='cyan')
```

1.39 espressions_etc.R

```
# comandi vari con expression(), eval() e D()
# (non è uno script)
expression(3^2)
eval( expression(x^2), list(x=2) )
eval( expression(a*x^2), list(a=3, x=2) )
eval( expression(a*x^2), list(a=3, x=2:4) )
eval( expression(a*x^2), list(a=3:5, x=2) )
eval( expression(a*x^2), list(a=3:5, x=2:4) )
eval( expression(a*x^2), list(a=3, x=2:4) )

quad <- expression(a*x^2)
eval( quad, list(a=2, x=3))

x=1:10; plot(x,  eval( expression(a*x^2), list(a=3, x=x) )  )
x=1:10; plot(x,  eval( expression(-x+a*x^2), list(a=3, x=x) )  )
x=1:10; plot(x,  eval( expression(-10*x + a*x^2), list(a=3, x=x) )  )
der = D( expression(-10*x + a*x^2), 'x')
der
eval ( D( expression(-10*x + a*x^2), 'x'), list(a=3, x=x) )
point(x, eval ( D( expression(-10*x + a*x^2), 'x'), list(a=3, x=x) ) )
points(x, eval ( D( expression(-10*x + a*x^2), 'x'), list(a=3, x=x) ) )
points(x, eval ( D( expression(-10*x + a*x^2), 'x'), list(a=3, x=x) ), col=green )
points(x, eval ( D( expression(-10*x + a*x^2), 'x'), list(a=3, x=x) ), col='green' )

e1 = expression(x^2/sqrt(1-x^2))
e1
De1 = D(e1, 'x')
```

1.40 due_corpi.R

```
# Moto Terra-'Luna' (in realtà la sua massa può essere modificata),
# usando l'algoritmo che si trova su La Fisica di Feynman
# (Nota: i vettori sono pronti per varianti in 3D per prendere
# in considerazione un terzo corpo (-> moti non coplanari)
```

```
source("dueCorpi_fun.R")

#-- Costanti ------------------------------------------------------
mT  = 5.97e24; mL  = 7.34e22    # kg
dTL = 384e6                     # m   (valore medio)
h   = 3600        # s   (1 ora)
vL0  = 1022       # m/s v nominale (https://it.wikipedia.org/wiki/Luna)

#-- Valori da variare per avere diversi moti
vL0 = 1.01*vL0    # la modifichiamo per avere diverse orbite
#vL0 = 0.5*vL0
# m = mL            # massa della Luna
# m = mT
m = mL/1000        # 'trascurabile' rispetto alla Terra

dt = 1*h            #  discretizzazione
t.max = 12*30*24*h
iplot = 5           # ogni quanti step plottare i punti
CM = !TRUE          # per vedere il moto nel centro di massa del sistema

xlim=c(-1.1, 1.1)    # limiti del plot
ylim=c(-1.1, 1.1)

#-------------------------------------------------------------
# condizioni iniziali e velocità del centro di massa
# Terra
pT = c(0, 0, 0)       # vettori direttamente 3D in preparazione
vT = c(0, 0, 0)       # di eventuali n (ad es. n=3) corpi non coplanari
# Luna
pL = c(dTL, 0, 0)
vL = c(0, vL0, 0)
# Centro di massa del sistema
vCM = (mT*vT + m*vL) / (mT + m)
#-------------------------------------------------------------

# iniziamo la simulazione
i = 1
t = 0
tv  = 0
dst = dist(pL, pT)

# plot vuoto, con un punto al centro
plot(0, 0, xlim=xlim,  ylim=ylim,
     xlab='x / dTL', ylab='y / dTL', pch=4, cex=0.5,asp=1)

# Forza SULLA Luna dovuta alla Terra
F = forza(m, pL, mT, pT)
cat(sprintf("F = (%e, %e)\n", F[1], F[2]))

# accelerazioni sui due corpi
aL =  F/m
aT = (-F)/mT    # terzo principio, reso esplicito mediante le parentesi

cat("accelerazioni\n")
print(aL)
print(aT)

# calcoliamoci la velocità per t=-dt/2,
```

```
# al fine di usare lo stesso algoritmo di Feynman
vL = vL + aL * (-dt/2)
vT = vT + aT * (-dt/2)

cat("velocità\n")
print(vL)
print(vT)

cat("\nCominciamo: \n")

while(t < t.max) {
# velocità per t+dt/2 a partire a quelle per t-dt/2
  vL = vL + aL * dt
  vT = vT + aT * dt

  # posizioni per t+dt a partire da quelle per t
  pL = pL + vL * dt
  pT = pT + vT * dt

  if(i%%iplot == 0) { # plot ogni iplot passi per andare più veloci
  # posizioni plottate in unità di distanze Terra-Luna
  points( ( pL[1] + ifelse(CM, - vCM[1]*t, 0) ) / dTL,
          ( pL[2] + ifelse(CM, - vCM[2]*t, 0) ) / dTL,
           col='orange', cex=0.5)
  points( ( pT[1] + ifelse(CM, - vCM[1]*t, 0) ) / dTL,
          ( pT[2] + ifelse(CM, - vCM[2]*t, 0) ) / dTL,
         col='blue', cex=0.5)
  }
  # forze e accelerazioni nelle nuove posizioni
  F = forza(m, pL, mT, pT)
  aL = F/m
  aT = (-F)/mT

  # incrementiamo il tempo
  t = t + dt
  i = i+1
  tv[i] = t
  dst[i] = dist(pL,pT)
}
```

1.41 dueCorpi_fun.R

```
# distanza fra due punti definiti dai vettori posizione
dist <- function(p1, p2) {
   dp = p1 - p2
   sqrt( as.numeric( t(dp) %*% dp ) )
}

# forza SUL punto materiale di massa m1 situato in p1
# dovuta al punto materiale di massa m2 situato in p2
forza <- function(m1, p1, m2, p2) {
   G = 6.67e-11 # N m^2 / kg^2
   return( -G*m1*m2 / dist(p1,p2)^3 * (p1-p2) )
}
```

1.42 tre_corpi.R

```
# Trasformazione del sistema Terra-Luna in un 'tre corpi'

source("dueCorpi_fun.R")

#-- Costanti ------------------------------------------------------
mT  = 5.97e24; mL  = 7.34e22    # kg
dTL = 384e6                     # m   (valore medio)
h   = 3600        # s   (1 ora)
vL0  = 1022       # m/s (https://it.wikipedia.org/wiki/Luna)

#-- Valori da variare per avere diversi moti
# corpo 3 è quello che ha velocità diversa da zero
# come la Luna nel caso di due_corpi.R

m1 = mT
m2 = mT/100
m3 = mT/100

dt = 1*h            #  discretizzazione
t.max = 12*30*24*h
iplot = 10           # ogni quanti passi fi plottano i punti
CM = TRUE           # per vedere il moto nel centro di massa del sistema

xlim=c(-1.3, 1.3)    # limiti del plot
ylim=c(-1.3, 1.3)

#-------------------------------------------------------------
# condizioni iniziali e velocità del centro di massa
# Terra e altro corpo
p1 = c(0, 0, 0)
v1 = c(0, 0, 0)
p2 = c(-dTL, 0, 0)
v2 = c(0, -vL0/2, 0)

# 'Luna'
p3 = c(dTL, 0, 0)
v3 = c(0, vL0, 0)

# Centro di massa del sistema
vCM = (m1*v1 + m2*v2 + m3*v3) / (m1 + m2 + m3)
#-------------------------------------------------------------

# iniziamo la simulazione
i = 1
t = 0

# plot vuoto, con un punto al centro
plot(0, 0, xlim=xlim,  ylim=ylim,
     xlab='x / dTL', ylab='y / dTL', pch=4, cex=0.5,asp=1)

# Forza sui ciascun corpo dovuto agli altri due
F1 = forza(m1, p1, m2, p2) + forza(m1, p1, m3, p3)
F2 = forza(m2, p2, m1, p1) + forza(m2, p2, m3, p3)
F3 = forza(m3, p3, m1, p1) + forza(m3, p3, m2, p2)

# accelerazioni sui tre corpi
a1 =  F1/m1
```

```
a2 =  F2/m2
a3 =  F3/m3

# calcoliamoci la velocità per t=-dt/2,
# al fine di usare lo stesso algoritmo di Feynman
v1 = v1 + a1 * (-dt/2)
v2 = v2 + a2 * (-dt/2)
v3 = v3 + a3 * (-dt/2)


cat("\nCominciamo: \n")

while(t < t.max) {
  # velocità per t+dt/2 a partire a quelle per t-dt/2
  v1 = v1 + a1 * dt
  v2 = v2 + a2 * dt
  v3 = v3 + a3 * dt

  # posizioni per t+dt a partire da quelle per t
  p1 = p1 + v1 * dt
  p2 = p2 + v2 * dt
  p3 = p3 + v3 * dt

  if(i%%iplot == 0) { # plot ogni iplot passi per andare più veloci
     # posizioni plottate in unità di distanze Terra-Luna
     points( ( p1[1] + ifelse(CM, - vCM[1]*t, 0) ) / dTL,
             ( p1[2] + ifelse(CM, - vCM[2]*t, 0) ) / dTL,
            col='blue', cex=0.5)
     points( ( p2[1] + ifelse(CM, - vCM[1]*t, 0) ) / dTL,
             ( p2[2] + ifelse(CM, - vCM[2]*t, 0) ) / dTL,
            col='red', cex=0.5)
     points( ( p3[1] + ifelse(CM, - vCM[1]*t, 0) ) / dTL,
             ( p3[2] + ifelse(CM, - vCM[2]*t, 0) ) / dTL,
            col='orange', cex=0.5)
  }
  # forze e accelerazioni nelle nuove posizioni
  F1 = forza(m1, p1, m2, p2) + forza(m1, p1, m3, p3)
  F2 = forza(m2, p2, m1, p1) + forza(m2, p2, m3, p3)
  F3 = forza(m3, p3, m1, p1) + forza(m3, p3, m2, p2)
  a1 =  F1/m1
  a2 =  F2/m2
  a3 =  F3/m3

  # incrementiamo il tempo
  t = t + dt
  i = i+1
}
```

1.43 potenza_numeri_complessi_animazione.R

```
# Variante di numeri_complessi_3.R, con animazioni
#

# definizione delle funzioni ------------------------------------------
start.plot.im <- function(Z) {
   # Z rappresenta il max di |Re(z)| e |Im(z)| rappresentabile
   plot(NULL, xlim=c(-Z,Z), ylim=c(-Z,Z), xlab='Re( )', ylab='Im( )', asp=1)
   grid()
```

```
}

plot.im <- function(z, pch=19, col='blue', lwd=1) {
   points(z, pch=pch, col=col)
   lines( c(0, Re(z)), c(0, Im(z)), col=col, lwd=lwd)
}

cerchio <- function(C=c(0,0), r=1, ...) {
  th=seq(0, 2*pi, by=0.02)
  points(C[1] + r*cos(th), C[2] + r*sin(th), ty='l', ...)
}
#-----------------------------------------------------------------

rho  <- c(1, 1.05, 0.95)
cols <- c('blue', 'red', 'green')
lwds <- c(2, 2, 2)

nome.files <- "piano_complesso_animato"
png (file=paste("tmp/", nome.files, "%03d.png", sep=''), width=480, height=480)


for (i in 1:3) {
  for (k in 1:16) {
    start.plot.im(2)
    cerchio(col='grey')
    z = rho[i] * (0+1i)^(1/4)
    plot.im(z^k, lwd=lwds[i], col=cols[i])
    text(1.1, 1.7, sprintf("rho = %.2f \nk = %d",rho[i],k),
         col=cols[i], cex=1.7, pos=4)
  }
}

dev.off()

dt   <- 0.2
#-----------------------------------------------------------------
system (sprintf("convert -delay %d tmp/*.png %s.gif",
        as.integer(dt/0.01), nome.files))


#-----------------------------------------------------------
# alternativa (a riga di comando dalla shell):
#
# convert -set delay 20 -quality 100 tmp/*.png outputfile.mpeg
#
```

1.44 Buon Anno 'matematico'

(Soprattutto come esempio di `integrate()`)

```
cat(sprintf("Buon %d!\n",
            integrate(function(x) 1-x^2,-6,6)$value +
            10*sum(1/(2:10-(2:10)^2)) + 3*factorial(6) ))
```

---

2 Histories

(**Nota bene**: per loro natura potrebbero contenere errori)

2.1 Rhistory_27sett2018.R

```
2^5
1:20
a <- 1:10
a
b = pi/2
b
e
exp(1)
exp(1) -> e
ls()
ls.str()
a
e
rm(e)
ls()
rm(list=ls())
ls()
getwd()
?getwd
?setwd
?sqrt
?exp
?q
a
a^4
a=4
a
b=7
a*b
b^2
sqrt(b^2)
sqrt(b^2*a)
b^2*a
(b^2) * a
2^1023
log(2^1023)
log(2^1023, 2)
?log
log(2^1023, pi)
log(2^1023)
log2(2^1023)
2^1024
6 / 2
6 / pi
round( 6 / pi, 2)
round( 6 / pi, 4)
6 %% 4
15 %% 3
15 %% 5
15 %% 7
16 %/% 3
sin(pi/2)
sin(90 * pi/180)
sin.g <- function(x) sin(x * pi/180)
sin.g(180)
sin.g(90)
sqrt(9)
sqrt <- function(x) x^3
sqrt(9)
rm(sqrt)
sqrt(9)
sqrt(-1)
sqrt(-1 + 0i)
sqrt(-1 + 0i) -> a
a
a^2
a + 3
(a + 3 )^2
(a + 3 )^2 -> b
b
plot(a)
abline(h=0); abline(v=0)
plot(a, xlim=c(-10, 10), ylim=c(-10,10) )
grid()
abs(a)
abs(b)
plot(a, xlim=c(-2,2), ylim=c(-2,2) )
grid()
plot(b, xlim=c(-2,2), ylim=c(-2,2) )
plot(b, xlim=c(-10,10), ylim=c(-10,10) )
a = 2
b= 3
c = a
a == b
a == b -1
a == b -1  -> test
test
!test
! (!test )
x = 1:10
x
x^2
x > pi
x
q()
```

2.2 Rhistory_1ott2018.R

```
x <-  1:50
1:10  -> x
x
sum(x)
sum(x^3)
sum(log(x))
s = 0
x[2]
for(i in 1:10) s <- s + x[i]
s
for(i in 1:10) s <- s + x[i]
s
s=0; for(i in 1:10) s <- s + x[i]
```

```
s=0; for(i in 1:10) { s <- s + x[i] ; print(s) }
s=0; for(i in 1:10) s <- s + x[i] ; print(s)
xl = log(x)
xl+
xl
s=0; for(i in 1:10) s <- s + xl[i] ; print(s)
x <- 1:100
x
xl = log(x)
xl
s=0; for(i in 1:length(xl)) s<-s+xl[i]; print(s)
length(xl)
length(x)
s=0; k=0;  while(s < 300) { k=k+1; s<-s+xl[i] }
k
q()
```

2.3 Rhistory_4ott.R

```
getwd()
z <- (0+1i)^(1/4) * c(1, 1.05, 0.95)
z
source("plot_im.R")
a=1; b=-1; c=3
discr = b^2 - 4*a*c
source("eq2_0.R")
discr
source("eq2_0.R")
source("eq2_1.R")
a=1; b=-1; c=3
discr = b^2 - 4*a*c
print(discr)
if(discr <0) discr <- discr * (1+0i)
discr
source("eq2_2.R")
a=1; b=-1; c=-3
plot(function(x) a*x^2+b*x+c, -3, 3)
abline(h=0)
source("eq2_1.R")
sol
abline(v=sol[1])
abline(v=sol[2])
q()
```

2.4 Rhistory_10ott2018.R

```
e
e <- exp(1)
e
pi
print(pi)
print(pi, dig=10)
?print
outer(1:10, 1:10)
outer(1:10, 1:10, '+')
outer(1:10, 1:10, '-')
outer(1:10, 1:10, '/')
outer(c(F,T), c(F,T))
outer(c(F,T), c(F,T), '&')
outer(c(F,T), c(F,T), '|')
outer(c(F,T), c(F,T), 'xor')
v.cil2  <- function(r, h) {
    area<-pi*r^2; volume<-area*h; return(volume)}
ls()
ls.str()
str(v.cil2)
v.cil2(2, 3)
v.cil2
ls()
ls
sqrt <- function(x)  "boh"
sqrt(4)
base::sqrt(4)
rm(sqrt)
sqrt(4)
r4 <- function(x) x^(1/4)
r4(16)
x = 1:10
x = sqrt(1:10)
x
x[ x > pi]
x[ x > pi/2]
q()
```

2.5 Rhistory_11ott2018.R

```
1:10
1:10 -> a
a
1.3:10
1.3:10 -> b
a
b
ls.str()
c(4, -2, 3^2)
c(4, -2, 3^2) -> c
c
c(a, b ) -> d
d
( c(a, b ) -> d )
nf <- c(1,1)
nf
nf <- rep(1, 2)
nf
rep(d, 2)
seq(0, 100, by=1)
seq(0, 100, by=5)
seq(0, 100, len=100)
seq(0, 100, len=101)
length( seq(0, 100, len=101) )
x <- (1:100)^2
x
mean(x)
mean(x, digit=12)
```

```
sum(x) / length(x)
for(i in 1:length(x)) print(x)
for(i in 1:length(x)) print(x[i])
for(i in 1:20) print(x[i])
s= 0; for(i in 1:length(x)) s <- s+x[i]
s
s= 0; for(i in 1:length(x)) s <- s+x[i] ; s/length(x)
s= 0; for(xi in x) s <- s+xi ; s/length(x)
nomi <- c('giacomo', 'carlo', 'andrea')
nomi
for(i in i:3) print(nomi[i])
for(i in i:3) print(nomi[i])
for(i in 1:3) print(nomi[i])
for(chi in nomi) print(chi)
source("media.R")
ls()
ls.str()
media
media()
v = 1:3
v
media(c)
media(v)
media(v) -> ris
ris
media(v)
media
source("media.R")
media
media(x)
media(x^2)
q()
```

2.6 Rhistory_17ott2018.R

```
k <- 1:10
k
1/2^k
sum(1/2^k)
cumsum(1/2^k)
nf <- rep(1,)
nf
nf <- rep(1,2)
nf
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
N=10; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
nf
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
N=10; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf)
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf)
plot(nf, log='y')
N=20; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf, log='y')
grid()
grid(lwd=2)
grid(lwd=4)
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] +
plot(nf, log='y')
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
nf[51]/nf[50]
source("int_rette.R")
m1
m2
q1
q2
source("int_rette.R")
source("int_rette.R")
source("int_rette.R")
source("int_rette.R")
( p <- locator(2) )
p$x
p$y
q()
```

2.7 Rhistory_17ott2018a.R

```
source("rette_2p_interattivo.R")
rainbow(5)
rainbow(7)
sample( rainbow(7) )
sample( rainbow(7) )
sample( rainbow(7) )
sample( rainbow(7) )
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
sample( rainbow(7) )[1]
source("cerchi_2p.interattivo.R")
```

2.8 Rhistory_18ott2018.R

```
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf, log='y')
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
plot(nf, log='y', ylim=c(.9,1200) )
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
N=20; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf, log='y', ylim=c(.9,1200) )
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
plot(log10(nf), )
grig(lwd=1.5)
grid(lwd=1.5)
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf, log='y')
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
points(10^( 0.5* (1:N) ), col='blue', ty='l')
points(10^( 0.1* (1:N) ), col='blue', ty='l')
points(10^( 0.3* (1:N) ), col='blue', ty='l')
points(10^( log10(2) * (1:N) ), col='blue', ty='l')
points(10^( log10(exp(1)) * (1:N) ), col='blue', ty='l')
points(exp( log(2) * (1:N) ), col='green', ty='l')
points(exp( log(2) * (1:N) ), col='green', ty='l', lwd=2)
N=100; nf <- rep(1,2); for(k in 3:N) nf[k] <- nf[k-1] + nf[k-2]
plot(nf, log='y')
grid(lwd=4)
points(2^(1:N), col='blue')
points(10^(1:N), col='blue')
points(10^(1:N), col='red')
points(exp(1:N), col='green')
points(exp(1:N), col='green', ty='l', lwd=2)
points(exp( log(10) * (1:N) ), col='green', ty='l', lwd=2)
points(exp( log(2) * (1:N) ), col='green', ty='l', lwd=2)
phi = (1+sqrt(5))/2
phi
points(exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
1/sqrt(5)
1/sqrt(5) -> alpha
points(alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(100 * alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(100^-1 * alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(1000^-1 * alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(1000000^-1 * alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
```

```
points(1000000 * alpha * exp( log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(1000000 * alpha * exp( - log(phi) * (1:N) ), col='green', ty='l', lwd=2)
points(exp( (1:N) ) , col='red', ty='l', lwd=2)
points(exp( 0.001*(1:N) ) , col='red', ty='l', lwd=2)
points( 0.001* exp((1:N) ) , col='red', ty='l', lwd=2)
points( exp( -0.5 *  (1:N) ) , col='red', ty='l', lwd=2)
points( 10^5* exp( -0.5 *  (1:N) ) , col='red', ty='l', lwd=2)
points( 10^15* exp( -0.5 *  (1:N) ) , col='red', ty='l', lwd=2)
x <- 1:15
y1 <- c(16,27,45,74,122,201,331,546,900,1484,2447,4034,6651,10966,18080)
y2 <- c(6.77e+00, 9.16e-01, 1.24e-01, 1.68e-02, 2.27e-03, 3.07e-04,
        4.16e-05, 5.63e-06, 7.61e-07, 1.03e-07, 1.39e-08, 1.89e-09,
2.55e-10, 3.46e-11, 4.68e-12)
plot(x, y1)
plot(x, y1, log='y')
plot(x, y2, log='y')
plot(x, y2)
plot(x, y2, log='y')
locator(2)
source("prova.R")
par
 readline(prompt="da a, b, c separati da virgola \n")  -> linea
linea
strsplit(linea, ',')
 unlist( strsplit(linea, ',')  )
 as.numeric(  unlist( strsplit(linea, ',')  ) )
q()
```

2.9 Rhistory_24ott.R

```
n = 5
factorial(n)
gamma(n+1)
prod(1:n)
(1:n
)
prod(1:n)
f <- 1; for(i in n) f <- f * i
f
f <- 1; for(i in 1:n) f <- f * i
f
mqline <- function(m, q, ...) abline(q, m, ...)
source("fatt_rec.R")
ls()
ls.str()
fattoriale
fattoriale(5)
x <- 1:15
y1 <- c(16,27,45,74,122,201,331,546,900,1484,2447,4034,6651,10966,18080)
y2 <- c(6.77e+00, 9.16e-01, 1.24e-01, 1.68e-02, 2.27e-03, 3.07e-04,
        4.16e-05, 5.63e-06, 7.61e-07, 1.03e-07, 1.39e-08, 1.89e-09,
2.55e-10, 3.46e-11, 4.68e-12)
plot(x, y1)
plot(x, y2)
plot(x, y1, log='y')
plot(x, y2, log='y')
q()
```

2.10 Rhistory_25ott.R

```
plot(1:20, pch=1:20)
plot(1:20, pch=1:5)
plot(1:20, pch=1:20)
plot(1:20, pch='*')
plot(1:20, pch='X')
plot(1:20, pch='W')
plot(1:20, pch=0:20)
plot(0:20, pch=0:20)
plot(0:20, pch=0:20, col=rainbow(21), cex=2 )
plot(0:20, pch=0:20, col=rev(rainbow(21)), cex=2 )
plot(0:20, pch=0:20, col=rainbow(21), cex=2 )
plot(0:20, pch=0:20, col=rev(rainbow(21)), cex=2 )
seq(0,1, by=0.2) -> x
x
seq(0-1,1, by=len=11) -> y
seq(0-1,1, len=11) -> y
y
x
c(x, y)
c(x, y) -> z
c(x,y,z)
locator(1)
locator(5) -> p
p
str(p)
x
i <- 1:5
ls.str()
c(i, x)
c(i, x)  -> v
ls.str()
l <- list(2, 3, 'pippo')
l
l <- list(2, 3, 'pippo', x)
l
ls.str()
l
l[[1]]
l[[4]]
l[[3]]
t1 <- Sys.time()
( t1 <- Sys.time() )
( t2 <- Sys.time() )
t1
t2
t2 - t1
ls.str()
t2 - t1
( t3 <- Sys.time() )
t2 - t1
t3 - t1
as.numeric( t2 - t1 )
q()
```

2.11 Rhistory_8nov2018.R

```
seq(-6, -1, by=0.5)
x <- 10^seq(-6, -1, by=0.5)
x
plot(x, sin(x))
plot(x, sin(x), log='xy')
plot(x, x-sin(x), log='xy')
points(x, ( x-sin(x) ) / sin(x), col='red')
plot(x, x^2, log='xy')
points(x, x^3)
plot(x, x^2, log='xy')
points(x, x^3, col='red')
points(x, x^0.5, col='red')
points(x, 100*x^2, col='red')
points(x, 100*x^2, ty='l', col='red')
points(x, 10000*x^2, ty='b', col='red')
points(x, 0.01*x^2, ty='b', col='red')
points(x, 0.0001*x^2, ty='b', col='red')
points(x, x^3, ty='b', col='green')
points(x, 100*x^3, ty='b', col='green')
points(x, 1000000*x^3, ty='b', col='green')
q()
```

2.12 Rhistory_21nov2018

```
A = rbind( c(1,2,4), c(-1,2-8), c(7,0,1) )
A = rbind( c(1,2,4), c(-1,2,-8), c(7,0,1) )
A
det(A)
solve(A)
solve(A) %*%  A
solve(A) %*%  A  -> I
I
dig(I)
diag(I)
sum(diag(I))
A = cbind( c(1,2,4), c(-1,2,-8), c(7,0,1) )
A
A <- matrix( 1:9, c(3,3))
A
( A <- matrix( 1:9, c(3,3)) )
( A <- matrix( 1:6, c(2,3)) )
( A <- matrix( 1:6, c(3,2)) )
A
tA <- t(A)
tA
tA %*% A
A %*% tA
matrix(rep(NA, 9), c(3,3))
q()
```

2.13 Rhistory_22nov2018.R

```
n=3; A = matrix( runif(n*n), c(n,n))
A
```

```
n=3; A = matrix( round(runif(n*n), 3) , c(n,n))
A
iA = solve(A)
iA
iA %*% A
n=1000; A = matrix( round(runif(n*n), 3) , c(n,n))
iA = solve(A)
iA %*% A -> I
sum(diag(I))
A = rbind( c(1, 3), c(-3, 5) )
rm(list=ls())
A = rbind( c(1, 3), c(-3, 5) )
A
A[1,2]
A[1,]
A[2,]
A[,2]
A[,2] = - A[,2]
A
A
diag(A) = c(-7, 7)
A
D = matrix(1:6, nrow=2)
D
dim(D) = c(3,2)
D
dim(D) = c(1,6)
D
as.vector(D)
A <- matrix(c(4, 2, 1,
              5, 6, 7,
              1, 0, 3), nrow=3, byrow=TRUE)
A
print(A)
t(1:3) %*% 1:3
as.vector(t(1:3) %*% 1:3)
1:3 %*% t(1:3)
outer(1:3, 1:3)
outer(1:3, 1:3, '+')
q()
```

2.14 Rhistory_6dic2018.R

```
dati <- read.table("lancio.txt")
plot(dati)
ls.str()
plot(data$V1, data$V2)
plot(dati$V1, dati$V2)
plot(dati$V1, dati$V3)
plot(dati$V1, dati$V3, ty='l')
plot(dati$V2, dati$V3, ty='l')
dati <- read.table("lancio.txt")
plot(dati$V2, dati$V3, ty='l')
t <- dati$V1
x <- dati$V2
y <- dati$V3
plot(t, y, ty='l', xlab='t (s)', ylab='y (m)')
q()
dati <- read.table("t_x_v.txt")
str(dati)
dati
dati <- read.table("t_x_v.txt", header=TRUE)
str(dati)
dati
str(dati)
dati$t
dati$x
dati$v
plot(dati$t, dati$x)
points(dati$t, dati$v)
prompt="dai nome file (return per il precedente)"
nomeFile <- readline(prompt=prompt)
if (nomeFile != '') {
  nomeFileOld <- nomeFile
} else {
  nomeFile <- nomeFileOld
}
cat(sprintf("\nvado a caricare  %s\n", nomeFile))
lancio <- read.table(nomeFile, header=TRUE)
source("carica_file.R")
source("carica_file.R")
source("carica_file.R")
colnames(lancio)
source("analizza_lancio.R")
q()
```

2.15 Rhistory_13dic2018.R

```
runif(1)
runif(10)
runif(10, 0, 100)
?runif
pnorm(0.)
\begin{verbatim}
pnorm(3)
pnorm(0)
qnorm(0)
qnorm(0.5)
qnorm(0.99)
x = runif(100000)
mean(x)
sd(x)
summary(x)
hist(x)
hist(x, nc=50, col='cyan')
hist(x, nc=50, col='cyan', p=TRUE)
hist(x, nc=50, col='cyan', prob=TRUE)
sample(1:10)
sample(1:10)
sample(1:10)
sample( c('a', 'b', 'c', 'd') )
sample( c('a', 'b', 'c', 'd') )
sample( c('a', 'b', 'c', 'd') )
sample( letters )
```

```
sample( letters )
sample( letters )
sample( LETTERS )
sample( LETTERS )
sample( LETTERS[1:10] )
sample( LETTERS[1:10] )
sample( LETTERS[1:10] )
sample( LETTERS[1:10] )
sample(c(LETTERS, letters), 10)
sample(c(LETTERS, letters), 10)
sample(c(LETTERS, letters), 10)
n.estr <- sample(90)
n.estr
n.estr <- sample(90)
n.estr
for(n in n.estr) {cat(sprintf("Uscito %d\n", n)); Sys.sleep(1)}
sample(6, 100, rep=TRUE)
sample(6, 100, rep=TRUE)
sample(6, 100, rep=TRUE)
sample(6, 100, rep=TRUE)
sample(6, 100, rep=TRUE)
sample(0:36, 100, rep=TRUE)
sample(0:36, 100, rep=TRUE)
sample(c(letters, LETTERS, 0:9))
paste( sample(c(letters, LETTERS, rep(0:9,3)), 12), collapse='' )
paste( sample(c(letters, LETTERS, rep(0:9,3)), 12), collapse='' )
paste( sample(c(letters, LETTERS, rep(0:9,3)), 12), collapse='' )
paste( sample(c(letters, LETTERS, rep(0:9,3)), 12), collapse='' )
n <- sample(6, 10000, rep=TRUE)
hist(n, col='cyan')
( tn <- table(n) )
barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
n <- sample(6, 10000, rep=TRUE); tn <- table(n); barplot(tn, col='cyan')
citta = c("Roma", "Milano", "Napoli", "Torino", "Palermo")
mil.abi = c(2.9, 1.4, 0.97, 0.88, 0.67)
pie(mil.abi, citta)
pie(mil.abi, citta, col=rainbow(5))
barplot(mil.abi, names=citta, col=rainbow(5))
n1 <- sample(6, 1000, rep=TRUE)
n2 <- sample(6, 1000, rep=TRUE)
par(mfrow=c(2,1))
barplot(table(n1), col='cyan')
barplot(table(n2), col='cyan')
s = n1 + n2
d = n1 - n2
barplot(table(s), col='cyan')
barplot(table(d), col='cyan')
x <- rnorm(1000000)
hist(x, col='cyan', nc=100)
q()
rm(x)
y
q()
```