Modelli numerici in fisica (I)

Modulo 1 (3 crediti)

1. Introduzione al linguaggio Fortran 77
2. Rappresentazione finita dei numeri
3. Integrazione numerica
4. Numeri random e pseudorandom
5. Metodi Monte Carlo per simulazione di processi random
6. Metodi MC per integrazione
7. Importance sampling

Testo: R.H. Landau e M. J. Paez "Computational Physics"

Modelli numerici in fisica (II)

Modulo 2 (3 crediti)

1. Applicazione dei metodi MC alla simulazione di sistemi fisici
2. Algoritmo alla Metropolis
3. Simulazione MC del modello di Ising
4. Generazione di frattali
5. Simulazione MC di un sistema fluido

Computer in Fisica

- Entrano sotto vari aspetti:

✓ **analisi numerica**

✓ **raccolta e analisi dati**

✓ **visualizzazione dati**

✓ **manipolazione algebrica**

✓ **calcoli approssimati (integrazioni)**

✓ **simulazione di modelli**

Uso del computer

1) Capire se per il problema è utile una procedura numerica.
2) Eventualmente spezzare il problema in parti più semplici o sottoproblemi.
3) Per il problema o i sottoproblemi determinare una traccia della possibile procedura numerica e valutarne la fattibilità.
4) Elaborare l'algoritmo.

Algoritmo (I)

Un algoritmo è una serie di regole che permettono di risolvere un problema.

Questa definizione è generica: anche una ricetta di cucina è un algoritmo (?). Per usarlo con un computer bisogna che soddisfi ad alcune proprietà:

- **finitezza**: **numero istruzioni deve essere finito**
- **definitezza**: **le istruzioni non devono essere ambigue**
- **effettuabilità**: **deve esistere un computer in grado di eseguire le regole in tempo finito**
- **risolvibilità**: **deve produrre un risultato o almeno una diagnostica**

Algoritmo (II)

Inoltre deve essere:

Robusto: non dipende dai dati di input

Efficiente

Può dipendere dal tipo di calcolatore e dal sistema operativo che scegliamo (o siamo costretti a scegliere!).

Esempio per un algoritmo

Dobbiamo moltiplicare due numeri complessi:

$z=a+ib \qquad w=c+id$

Nel computer: $z=(a,b)$ $w=(c,d)$

Regola dell'algoritmo: $t=z \bullet w$ sarà $t=(ac-bd,ad+bc)$

Algoritmo in C

```
...
typedef struct {
        float real,imag;
                } complex;   /* definizione del nuovo tipo dati */

/* definizione della function che implementa l' algoritmo di moltiplicazione */

complex cmplx_mul( complex a, complex b)
{
   complex t;
   t.real = a.real*b.real - a.imag*b.imag;
   t.imag = a.real*b.imag + a.imag*b.real;
   return (t);
}

/* esempio di utilizzo */
...
int main()
{
   complex t, w, z;
...

  t = cmplx_mul(z,w);
...
```

Algoritmo in Fortran

```
...
complex ::   t,w,z
...
t = z * w
...
```

In Fortran le operazioni sui complessi sono definite in modo intrinseco come per i numeri reali.

Fortran

Un po' di storia:

FORmula TRANslation sviluppato da IBM dal 1954-57

Il primo sistema operativo che consentiva di scrivere le istruzioni non in linguaggio macchina, ma con formule simili ad equazioni algebriche

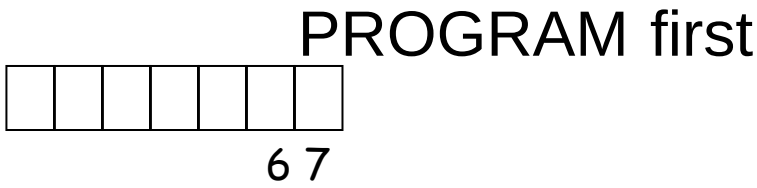
Versione più recente di largo uso: Fortran 77

Fortran 90 ultimo aggiornamento con sviluppo verso la programmazione strutturata (non ancora di largo uso)

Alcuni fondamenti di Fortran

Le istruzioni vanno scritte da colonna 7 a colonna 72

Residuo storico dell'uso delle schede, è stato superato dal Fortran 90, ma per compatibilità è meglio osservarlo.



Da notare che emacs per es. aiuta nella scrittura

Compilatore su Linux: *g77*

Fortran: tipi di variabili

Intere: numero senza cifre decimali

Varia tra

$$-2^{n-1} \leq I \leq 2^{n-1} - 1$$

con n numero di bit

n=32 -2 147 483 648 ≤ I ≤ 2 147 483 647

n=16 -32 768 ≤ I ≤ 32 767

Reali

Vengono rappresentati come: mantissa x 2^{esp}

Precisione singola: 4 bytes = 32 bits REAL o REAL*4

24 per mantissa + 8 per esponente

$10^{-44} \leq X \leq 10^{38}$ cifre decimali: 6÷7

Precisione doppia: 8 bytes = 64 bits REAL*8 o DOUBLE PRECISION

53 per mantissa + 11 per esponente

$10^{-322} \leq X \leq 10^{308}$ cifre decimali: 15÷16

Precisione della "macchina"

Si può definire come il max numero che si può aggiungere ad 1 senza cambiarne il valore

$$1 + \varepsilon = 1$$

$\varepsilon \approx 10^{-7}$ in singola precisione

$\varepsilon \approx 10^{-16}$ in doppia precisione

limiti di overflow e underflow

proveremo i limiti della macchina

Dichiarare le variabili

Default tradizionale del Fortran:

tutte le variabili con iniziali I, J, K, L, M, N sono considerate intere

E' meglio iniziare con IMPLICIT NONE

e dichiarare tutte le variabili con

```
REAL :: x,y
REAL*8 :: x,y
INTEGER :: g,m,v
```

oppure

```
REAL  x,y
REAL*8  x,y
INTEGER  g,m,v
```

Ma usare :: è più moderno !!

Caratteri

Una variabile stringa racchiusa da ' .....'

'amico' 'inserisci il valore'

oppure va dichiarata con

```
CHARACTER :: nome
CHARACTER*3 :: nome_amico

nome_amico='Bob'
```

Operazioni sulle variabili

- a+b addizione
- a-b sottrazione
- a*b moltiplicazione
- a/b divisione
- a**b elevamento a potenza (nuovo rispetto al C)

Gerarchia: come in C ma l'elevamento a potenza viene effettuato prima

Attenzione alle divisioni con interi

3/2=1 3./2=1.5 3./2.=1.5

Appendice A: confronto Fortran e C

Fortran: esempio 1

```
      PROGRAM area
c     calcolo dell'area del cerchio
c     dati in input: raggio del cerchio
c     produce in output l'area del cerchio

c     qui inizia la sezione dichiarativa
      IMPLICIT none          ! nel fortran moderno si preferisce
                             ! dichiarare
                             ! espicitamente tutte le variabili
      REAL*8 pi, r, A        ! REAL per variabili reali
                             ! REAL*8 per variabili double prec.

c   qui inizia la sezione esecutiva

c   input ed assegnazione valori

      PRINT*,' Assegna il raggio'
      READ(*,*) r
      pi=4*atan(1.d0)

c   esecuzione
      A=pi*r**2

c     output
      WRITE(*,*) ' raggio=',r,' Area=',A

c   conclusione
      STOP
      END
```

Calcolo dell'area del cerchio

Formattazione I

Per controllare il formato di entrata e/o di uscita dei dati si usa l'istruzione FORMAT:

```
READ(*,10) a,b,c
```

Oppure

```
Print 10, a,b,c
10    FORMAT(...)
```

Un programma di computer fa quello che gli dici, non quello che vuoi.
(III Legge di Greer)

Formattazione II

Iw Converte i successivi w caratteri in un numero intero

Fw.d Converte i successivi w caratteri in un numero reale con d decimali

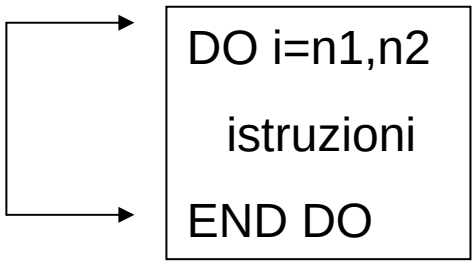
Ew.d come sopra e in notazione esponenziale

nX salta i successivi n caratteri

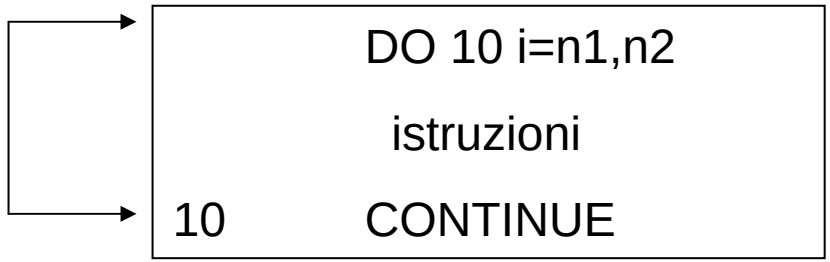
ES.

```
 10    FORMAT(I4,I4,3X,F10.3)
 10    FORMAT(I4,I4)  !equivale a scrivere 2I4
```

Cicli di DO



```
DO i=n1,n2
   istruzioni
END DO
```



```
      DO 10 i=n1,n2
         istruzioni
10    CONTINUE
```

Il ciclo va da n1 a n2 interi positivi o negativi ma n1 < n2

Lo step si incrementa di 1 a meno di non chiedere dei salti

Si possono fare anche salti

```
DO i=1,10,2
   istruzioni
END DO
```

Esegue i=1,3,5,7,9

Test di precisione, under e over flow

ALGORITMO1: eps=1.

ciclo DO fino a N

eps=eps/2.

one=1.+eps

scrivere formattati loopnumber, eps e one

fine ciclo

ALGORITMO2: under=1.

over=1.

ciclo DO fino a N

under=under/2.

over=over*2.

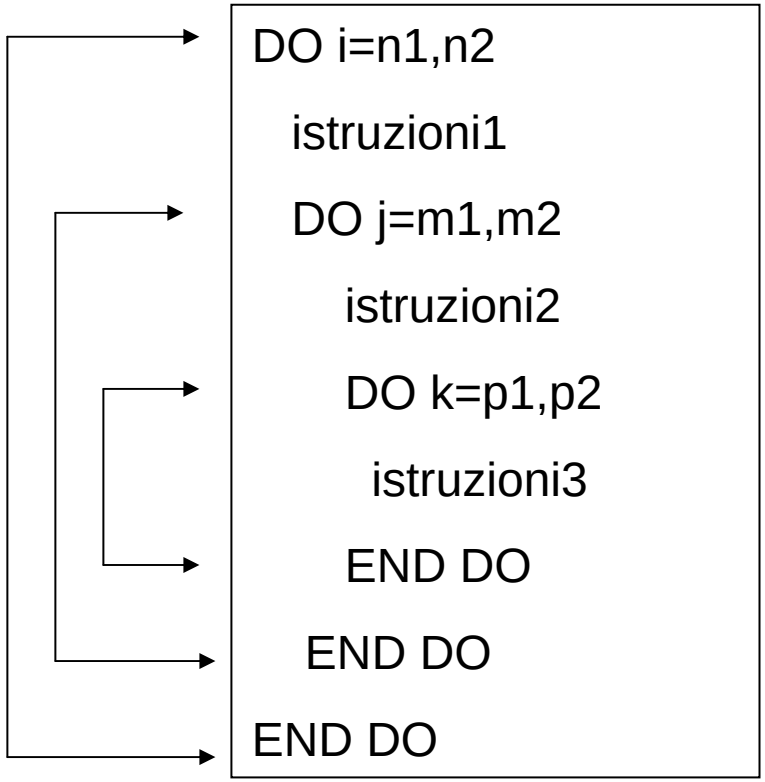
scrivere formattati loopnumber, under e over

fine ciclo

Test della precisione

```
ccccccccccccccccccccccccccccccccccccccccccc
      Program limit
      Implicit none
c
c determine the machine precision
c
      Integer I, N
      Real*8 eps, one
c number of iterations N
      N=60
c set initial values
      eps = 1
      one = 1
c add eps to one and print result
      Do 15, I = 1, N
         eps = eps / 2
         one = 1 + eps
         Write (*,*) I, one, eps
 15   Continue
      Stop 'limit'
      End
```

Cicli di DO inseriti



```
DO i=n1,n2
  istruzioni1
  DO j=m1,m2
    istruzioni2
    DO k=p1,p2
      istruzioni3
    END DO
  END DO
END DO
```

```
      DO 10 i=n1,n2
        istruzioni1
        DO 20 j=m1,m2
          istruzioni2
          DO 30 k=p1,p2
            istruzioni3
30        CONTINUE
20      CONTINUE
10    CONTINUE
```

Variabili logiche

Sono definite da:

LOGICAL :: test1,test2

test1=.true.

test2=.false.

In generale possono assumere solo due valori, TRUE o FALSE

ES:

```
      LOGICAL READY
      ITER=0
      READY=.FALSE.
50    CONTINUE
      IF (READY) RETURN
      ...
```

Operatori logici e istruzione IF

```
IF (espressione logica) THEN
    istruzioni
END IF
```

Esegue le istruzioni se l'espressione logica è vera

```
IF (espressione logica) THEN
    istruzioni
ELSE
    istruzioni alternative
END IF
```

```
IF (espressione logica1) THEN
    istruzioni1
ELSE IF (espress. logica2) THEN
    istruzioni2
ELSE
    istruzioni3
END IF
```

Se l'espressione logica è falsa esegue le istruzioni alternative

Operatori logici

Fortran moderno		*"Tradizionale"*
a == b	uguale	a .EQ. b
a /= b	diverso	a .NE. b
a > b	maggiore	a .GT. b
a >= b	maggiore o uguale	a .GE. b
a < b	minore	a .LT. b
a <= b	minore o uguale	a .LE. b

Tipi di errori

- **errori di approssimazione**
- **errori di rounding off**

Errori di approssimazione: derivano dal fatto che le operazioni che possiamo fare sono in numero finito. Per esempio

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \approx \sum_{n=0}^{N} \frac{x^n}{n!}$$

Detto anche errore di troncamento o errore dell'algoritmo

Cancellazione di cifre significative a causa della sottrazione

Il numero X è rappresentato nel calcolatore da x tale che

$$x = X(1+\Delta X)$$

allora $$A = B - C \Rightarrow a = b - c = B(1+\Delta B) - C(1+\Delta C)$$

$$\frac{a}{A} \approx 1 + \frac{B}{A}(\Delta B) - \frac{C}{A}(\Delta C)$$

Nel caso $$B, C >> A \Rightarrow \Delta A \approx \frac{B}{A}\left(\Delta B - \Delta C\right)\Big)$$

l'errore potrebbe essere anche grande

Errori di rounding off:

derivano dal fatto che i numeri sono rappresentati nel calcolatore con un numero limitato di bits

Esempio potrebbe succedere (dipende dal calcolatore:

$$2\left(\frac{1}{3}\right)-\frac{2}{3}=0.6666666-0.6666667=-0.0000001$$!!!!

In alcuni casi questi errori possono diventare *pericolosi*.

Precisione del calcolatore (macchina):

$$\Delta_m \approx 10^{-7} \text{ singola}$$

$$\Delta_m \approx 10^{-15} \text{ doppia}$$

Bisogna tener conto del numero di cifre significative e per questo è spesso preferibile usare la doppia precisione.

Errore totale

L'errore totale nella stima di una quantità calcolata con un algoritmo sarà la somma

$$\Delta_{tot} = \Delta_{RO} + \Delta_{APP}$$

Se l'algoritmo è composto di N steps in genere si può far vedere che

$$\Delta_{RO} \prec \sqrt{N} \qquad \Delta_{APP} \prec N^{-\gamma}$$

dove γ dipende dall'algoritmo usato, per es. $\gamma \approx 2$

L'errore è determinato principalmente dal RO ma deriva da un delicato bilanciamento fra i due termini.

Esempio : equazione di 2do grado

Sviluppare un algoritmo per risolvere l'equazione

$ax^2+bx+c=0$

Codifica preliminare

1) leggere dati di input a,b,c
2) calcolare il discriminante discr=b**2-4.*a*c
3) verifica del segno di discr con operatore IF

```
IF (discr > 0) THEN
   sq=sqrt(discr)
   scrivere che ci sono due radici reali
    x1=(-b + sq)/(2*a)
    x2=(-b - sq)//2*a)
    scrivere x1, x2
ELSE IF (discr=0)
     etc.
ELSE
     etc.
END IF
```

Esempio: radici equazione

Terminare la codifica dell'algoritmo e scrivere il programma

```
      program radici
c
c calcola le radici dell'equazione
c  a*x**2 + b*x + c = 0
c
      implicit none
c
c dichiarazione delle variabili
c
      real :: a
      real :: b
      real :: c
      real :: discriminante
      real :: sr
      real :: parte_immag
      real :: parte_reale
      real :: x1,x2
!  parte di input dei dati
      write(*,*) 'Per trovare le radici dell''equazione'
      write(*,*) 'a*x**2 + b*x + c=0'
      write(*,*) 'Introduci i valori a,b,c'
      read(*,*) a,b,c
c
      write(*,*) 'Sono stati introdotti i valori'
      write(*,*) ' a=',a,' b=',b,' c=',c
c
c
! calcolo del discriminante
      discriminante= b**2 - 4. *a*c
! controlla il segno del discriminante
      if (discriminante > 0. ) then
         sr = sqrt(discriminante)
         x1=( -b+ sr)/(2. *a)
         x2=( -b- sr)/(2. *a)
         write(*,*) 'Ci sono due radici reali'
         write(*,*) 'x1=',x1
         write(*,*) 'x2=',x2
      else if ( discriminante == 0. ) then
         x1=-b/(2.*a)
         write(*,*) 'Le due radici reali coincidono'
         write(*,*) 'x1=x2=',x1
      else
         parte_reale= -b/(2.*a)
         sr=sqrt(abs(discriminante))
         parte_immag=sr/(2.*a)
         write(*,*) 'Ci sono due radici complesse'
         write(*,*) '(parte reale,parte immag.)'
         write(*,*) 'x1=','(',parte_reale,
     &                    parte_immag,')'
         write(*,*) 'x2=','(',parte_reale,
     &                       -parte_immag,')'
        end if
        stop
        end
```

Esempio di errore sottrattivo: le soluzioni dell'equazione di secondo grado

```
IF (discr > 0) THEN
   sq=sqrt(discr)
   scrivere che ci sono due radici reali
    x11=-2c/(b+sq)
    x22=-2c/(b-sq)
    x1=(-b + sq)/(2*a)
    x2=(-b - sq)//2*a)
    scrivere x11, x22,x1,x2
END IF
```

Confrontare le radici quando $b^2 >> 4ac$

Se b>0 allora l'errore sottrattivo è su x1 e x22.

Se b<0 su x2 e x11

Sono errori dovuti al fatto che la radice e il termine che la precede quasi si cancellano

FILE INPUT/OUTPUT

- **Apertura file:**

OPEN (lab, FILE='nome_file', STATUS='stat' ,FORM= 'fmt')

lab=label →numero

stat
- unknown non specificato (meglio per file di output)
- new file nuovo di output (non riscrive su file precedente)
- old file di input già esistente

fmt
- formatted indica che i dati sul file sono formattati
- unformatted indica che i dati non sono formattati
- ometterlo indica la formattazione standard.

- **Scrittura su file**

  WRITE(lab, lab_fmt), lista_var

- **Lettura da file**

  READ(lab,lab_fmt), lista_var

  lab_fmt=label format , * indica la formattazione standard

  lab_fmt FORMAT=(lista di comandi che esprime la formattazione con cui si intende scrivere il file di output)

  Chiusura file →CLOSE (lab)

Calcolo di una serie

$$e^{-x} = \sum_{n=0}^{\infty} (-1)^n \frac{x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

Vogliamo calcolarla con un errore di una parte in 10^8

Assumiamo che l'errore nel calcolo della somma si possa approssimare con l'ultimo termine della somma.

Fermiamo il calcolo quando per l'ultimo termine calcolato (term) abbiamo rispetto alla somma accumulata (sum) :

$$\left| \frac{\text{term}}{\text{sum}} \right| < 10^{-8}$$

Schema del programma

- PROGRAM ----
- IMPLICIT NONE
- DICHIARAZIONE VARIABILI
- APERTURA FILES DI INPUT/OUTPUT
- INIZIALIZZAZIONE VARIABILI
  - MAX=10
  - STEP=0.1
- LOOP SU 0< X<max CON PASSO = STEP
  - CALCOLO DI e^{-x}
  - OUTPUT PERIODICO DELLA SIMULAZIONE
- CHIUSURA FILE DI OUTPUT
- STOP
- END

ALGORITMI PER IL CALCOLO DELLA SERIE e^{-x}

EXP_BAD

$$e^{-x} \approx \sum_{n=0}^{N} \frac{(-x)^n}{n!} = \sum_{n=0}^{N} \frac{up}{down}$$

```
sum=1
eps=10.0**(-8)
Loop per 1<i<N
    up=1
    down=1
    loop per 1<j<i
          up=-up*x
          down=down*j
    fine loop su j
    term = up / down
    sum=sum+term
    controllo se abs(term/sum)< eps
          write output
          esco dal loop
    fine controllo
Fine loop
```

ESEMPIO :

N = 100 x = 10.0

Up = 10^{100} Down ~ 10^{157}

N-simo termine della somma :

term = up /down

N = 200 down ~ $10^{374} > 10^{308}$!!!

EXP_GOOD

$$e^{-x} \approx \sum_{n=0}^{N} \frac{(-x)^n}{n!} = \sum_{n=0}^{N} \frac{(x)^{n-1}}{(n-1)!} \frac{(-x)}{n} = \sum_{n=0}^{N} term$$

```
sum = 1
term = 1
eps=10**(-8)
Loop per 1<i<N
  term = -term * x/i
  sum = sum + term
  controllo se abs(term/sum)< eps
          write output
          esco dal loop
    fine controllo
Fine loop
```

ESEMPIO :

N = 100 x = 10.0

N-simo termine della somma :

term = 1 · (-10/1) · (10/2) · …. · (10/100)

```
  PROGRAM EXP_GOOD
  implicit none
c dichiarazione variabili
   REAL*8 term,eps,max,step,sum,x,conf
   INTEGER n
c definizione ed apertura file ext. che vengono utilizzati
   OPEN (16,FILE='EXP_GOOD.dat',STATUS='UNKNOWN')
   OPEN (17,FILE='CONFRONTO_GOOD.dat',STATUS='UNKNOWN')
c inizializzazione variabili
  eps=1.0E-8
  step=0.1
  max=20.0
c main loop
   DO 10 x=0,max,step
        sum=1.0
        term=1.0
        DO 20 n=1,10000
             term=term*(-x)/n
              sum=sum+term
c  controllo la precisione della somma
              IF (sum.NE.0.AND.abs(term/sum).lt.eps) THEN
                   WRITE (16,100) x,sum,dexp(-x)
                   conf=dabs((sum-DEXP(-x))/sum)
                   WRITE (17,101) x,conf
                   goto 10
              ENDIF
```

NB: DEXP etc.

```
20    CONTINUE
10    CONTINUE
100   FORMAT(F5.1,3X,F17.15,3X, F17.15)
101   FORMAT(F5.1,3X,F17.15,3X)
CLOSE (16)
CLOSE (17)
STOP 'DATA SAVED IN EXP-GOOD.dat'
 END
```

```
      PROGRAM EXP_BAD
      implicit none
c     dichiarazione variabili
      REAL*8 down,eps,max,step,sum,up,x,conf,term
      INTEGER i,n
c     definizione ed apertura file ext. che vengono utilizzati
      OPEN (16,FILE='EXP_BAD.dat',STATUS='UNKNOWN')
      OPEN (17,FILE='CONFRONTO_BAD.dat',STATUS='UNKNOWN')
c     inizializzazione variabili
      eps=1.0E-8
      step=0.1
      max=20.0
c     main loop
      DO 10 x=0,max,step
         sum=1.0
         DO 20 n=1,10000
            up=1.0
            down=1.0
            DO 30 i=1,n
               up=-up*x
               down=down*i
 30         CONTINUE
            sum=sum+up/down
c     controllo la precisione della somma
            term=(up/down)/sum
            IF(sum.NE.0.AND.abs(term).lt.eps) THEN
               WRITE (16,100) x,sum,dexp(-x)
               conf=dabs((sum-DEXP(-x))/sum)
               WRITE (17,101) x, conf
               goto 10
            ENDIF
 20      CONTINUE
 10      CONTINUE
 100     FORMAT(F5.1,3X,F17.15,3X, F17.15)
 101     FORMAT(F5.1,3X,F17.15,3X)
         CLOSE (16)
         CLOSE (17)
         STOP 'DATA SAVED IN EXP-BAD.dat'
         END
```

Esercizi con la serie

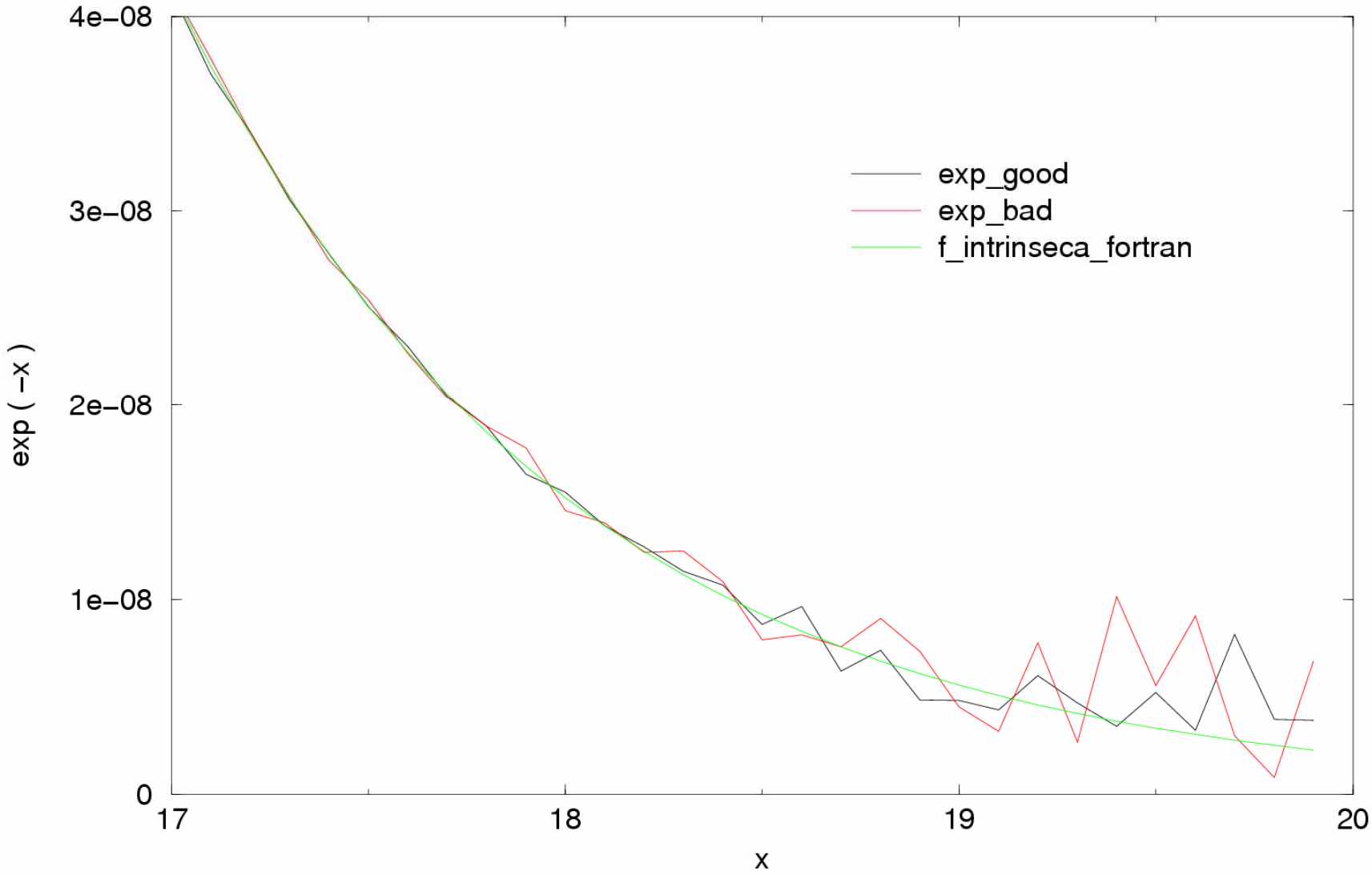
- Scrivere i programmi che calcolano con un errore di una parte in 10^8 la serie e^{-x} con i due algoritmi BAD e GOOD.
- Confrontare il risultato delle simulazioni con la funzione intrinseca fortran (dexp (-x)) : considerare il range di x da 0 a 20 (loop su x con step = 0.1) fino ad ottenere deviazioni forti rispetto a dexp(-x)
- Produrre con entrambi i programmi i seguenti files di OUTPUT :
  - ✓ EXP_GOOD.dat (EXP_BAD.dat) contenenti :

    x sum dexp(-x)
  - ✓ CONFRONTO_GOOD.dat (CONFRONTO_BAD.dat) :

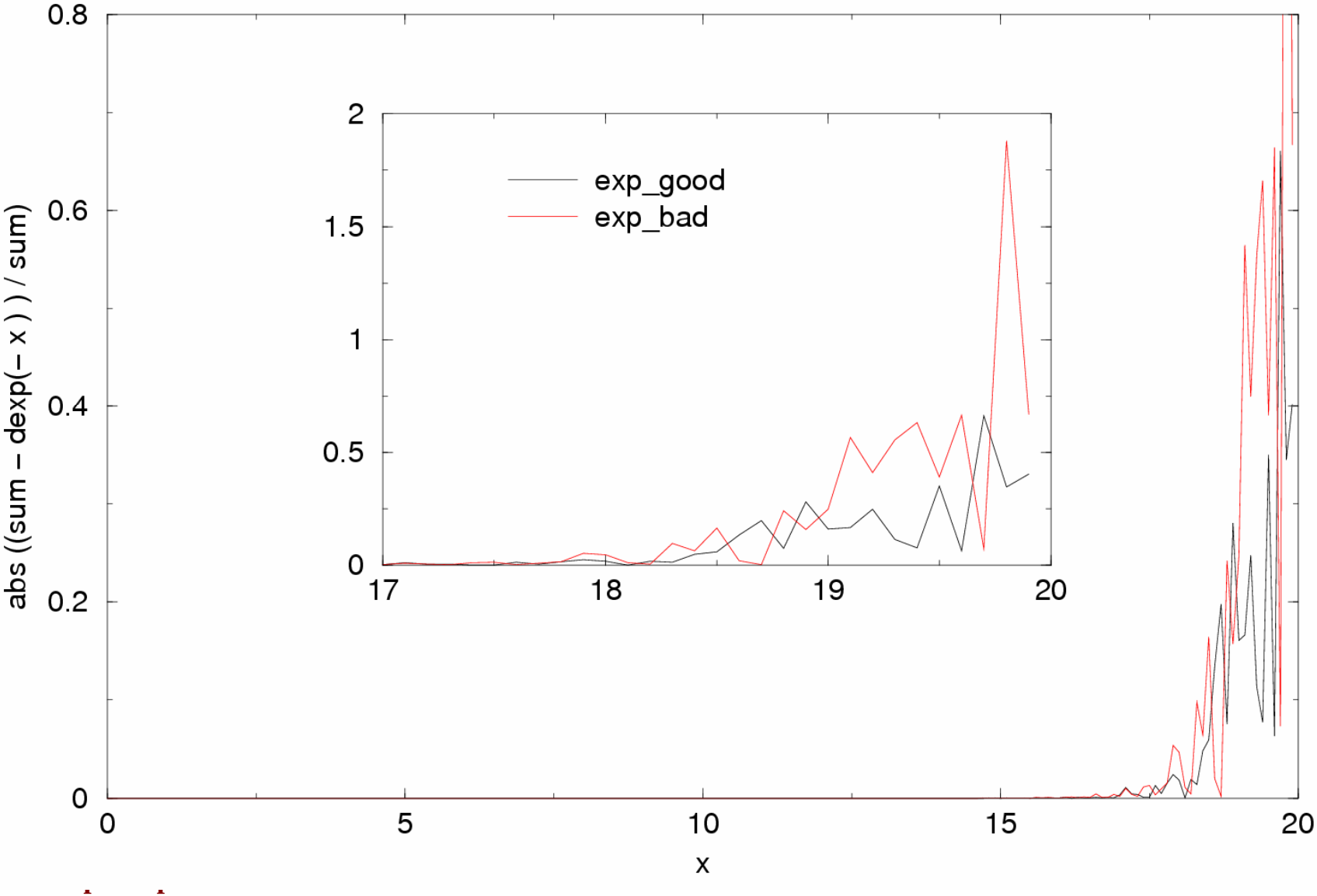
    x abs((sum-dexp(-x))/sum)
- Fare un grafico e confrontarlo con exp(-x) come calcolato da xmgr
- Vedere la differenza nei tempi di esecuzione:

  time ./name.x

RISULTATI CALCOLO DELLA SERIE



RISULTATI : ERRORE RELATIVO



Operatori logici

Operator	Meaning
.EQ.	Equal
.NE.	Not equal
.GE.	Greater than or equal
.LE.	Less than or equal
.LT.	Less than
.GT.	Greater than

and the following logical operators:–

Operator	Meaning
.AND.	and
.OR.	or
.NOT.	not

Es. if (a.eq.b .and. b.ne.c) stop

if(.not. a.lt.b) stop

Variabili vettoriali

Le variabili vettoriali vanno dichiarate all'inizio

Ci sono varie maniere per farlo

```
REAL :: X(100)
REAL :: Y(100,50)
```

```
REAL X,Y
DIMENSION X(100)
DIMENSION Y(100,50)
```

E' preferibile indicare primo e ultimo indice

```
REAL :: X(0:100)
REAL :: Y(0:100,0:50)
```

```
REAL :: X(-20:32)
REAL :: Y(-15:21,-6:82)
```

Operazioni con i vettori I

Azzeramento iniziale

```
REAL X(-1:5)
.....
DO I = -1,5
 X(I)=0.
END DO
```

Si può fare così

Oppure così

```
REAL X(-1:5)
.....
DATA X /7*0/
```

```
REAL X(-1:5,0:5)
.....
DATA X /42*0/
```

```
REAL X(-1:5)
.....
DATA X /7*1.0/
```

Qui viene inizializzato a 1

Operazioni con i vettori II

Operazioni varie

```
REAL X(-1:5),Y(-2,7),Z(-2,9)
```

Assegnati 7 valori a X e 10 valori a Y .

....

```
DO  I = -1,5
 Z(I)=X(I) + Y(I)
END DO
```

Fare attenzione alla corrispondenza degli indici!

Notare che in Z avra' definito solo i valori da –1 a 5

Sfondare l'indice max è un errore molto grave!

```
REAL X(100),Y(200), Z(200)
...
DO I=1,200
X(I)=  Y(I)+Z(I)
END DO
```

Functions

Nel programma una function si chiama con

nome(lista di argomenti)

Funzioni intrinseche: ce ne sono molte definite, per esempio:

ABS(X) COS(X) EXP(X)

Altre possono essere definite dall'utente

Main program

```
PROGRAM test
REAL*8 x,y,ft.
.....
y=ft(x)
print*, x,y
```

```
FUNCTION ft(x)
implicit none
Real*8 x,ft
ft = x*cos(x)-1.
RETURN
END
```

Function	Action	Example
INT	conversion to integer	J=INT(X)
REAL	conversion to real	X=REAL(J)
ABS	absolute value	X=ABS(X)
MOD	remaindering remainder when I divided by J	I=MOD(K,L)
MAX	maximum value	X=MAX(A,B,C,D)
	(at least 2 arguments)	I=MAX(K,L)
MIN	minimum value	X=MIN(A,B,C,D)
	(at least 2 arguments)	I=MIN(K,L)
SQRT	square root	X=SQRT(Y)
EXP	exponentiation	Y=EXP(X)
LOG	natural logarithm	X=LOG(Y)
LOG10	common logarithm	X=LOG10(Y)
SIN	sine	X=SIN(Y)
COS	cosine	X=COS(Y)
TAN	tangent	X=TAN(Y)
ASIN	arcsine	Y=ASIN(X)
ACOS	arccosine	Y=ACOS(X)
ATAN	arctangent	Y=ATAN(X)
ATAN2	arctangent(a/b)	Y=ATAN2(A,B)

Subroutines

Una subroutine è un sottoprogramma come la function ma può far ritornare al main program più valori

Dal main program si chiama con CALL

CALL nome_subroutine (argomenti in input, valori di ritorno)

```
PROGRAM test
assegna valori a s1,s2,s3
CALL rapporti(s1,s2,s3,r1,r2)
usa valori r1,r2
STOP
END
```

```
SUBROUTINE rapporti(a,b,c,x,y)
Implicit none
real a,b,c,x,y,t1,t2
t1=3.0*c**2
t2=5.0*t1/t1
x= a*b/(c-1.0)*t1
y= b*c/(a**2+b**2)*t2
RETURN
END
```

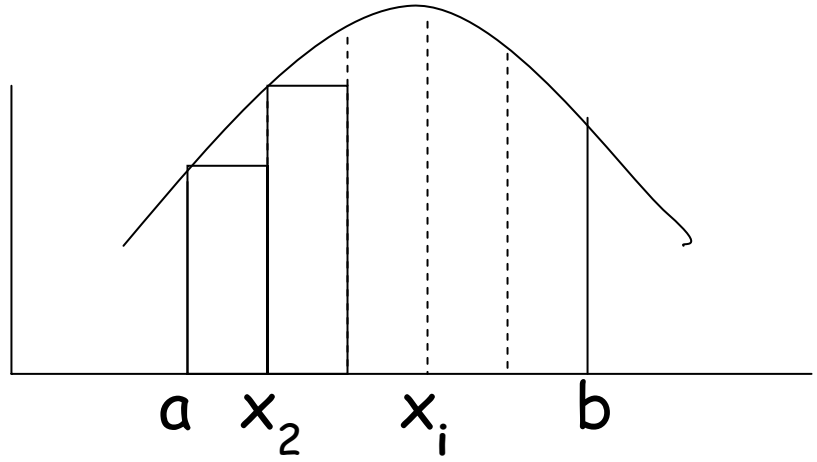
NB: t1 e t2 non vengono salvati dopo il RETURN

Vettori nelle subroutines

```
REAL X,SUM
DIMENSION X(100)
.
CALL ADD(X,SUM)
.
END
SUBROUTINE ADD(A,TOTAL)
REAL A,TOTAL
DIMENSION A(100)
INTEGER I
```

```
REAL X,SUM
DIMENSION X(100)
CALL ADD(X,SUM,100)
.
SUBROUTINE ADD(A,TOTAL,N)
REAL A,TOTAL
INTEGER N
DIMENSION A(N)
TOTAL=0.0
DO 1 I=1,N
.
```

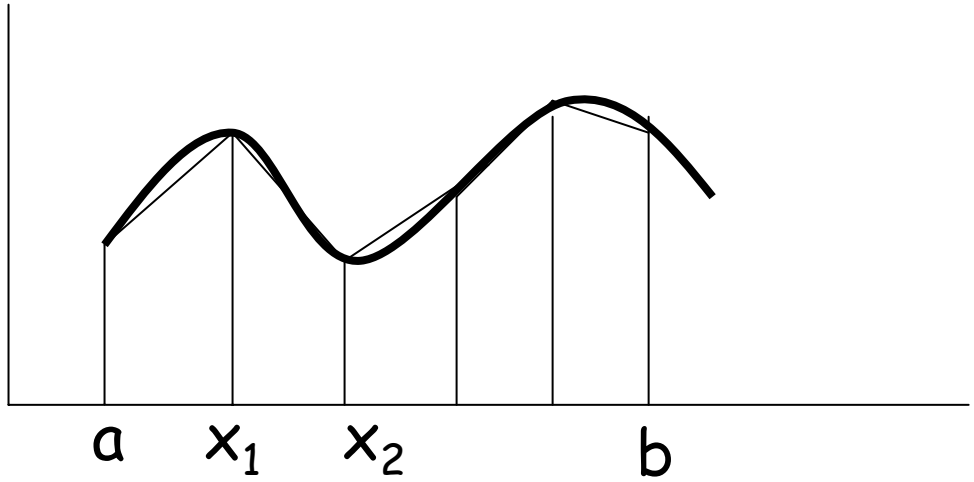
Metodi di integrazione semplice



$$\int_a^b f(x)dx = \lim_{h\to 0}\left[h\sum_{i=1}^{n-1} f(x_i) \right] \qquad h = \frac{(b-a)}{n-1}$$

$$\approx \frac{(b-a)}{n}\sum_{i=1}^{n-1} f(x_i) \qquad x_i = a + (i-1)h$$

Regola del trapezio



$$h = \frac{(b-a)}{n-1}$$

$$x_i = a + (i-1) * h \qquad i = 1, n$$

$$\int_{x_1}^{x_2} f(x)dx \cong \frac{h(f_1 + f_2)}{2} = \frac{1}{2}hf_1 + \frac{1}{2}hf_2$$

$$\int_a^b f(x)dx \approx h\left(\frac{1}{2}f(a) + f(x_2) + f(x_3) + \ldots + f(x_{n-1}) + \frac{1}{2}f(b)\right)$$

```
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
cccccccc
          program int_trap
          implicit none
c declarations
          real*8 res,test
          real*8 t0, vmin, vmax
          integer n
c
          vmin=0.0
          vmax=1.0
          open(8, File='trapez.dat',
Status='Unknown')
c
        t0=1.0-exp(-1.0)
        do 50 n=5, 501 , 2
          call trapez(n, vmin, vmax,res)
          test=abs(res-t0)/t0
          write(8,100) n,res,test
 50     continue
 100    format(i8,2E15.6)
          close(8)
          stop 'data saved in trapez.dat'
          end
c
c
          function f(x)
          implicit none
          real*8 f, x
          f=exp(-x)
          Return
          End
```

```
c trapezoid rule
          subroutine trapez(n, a, b,res)
          implicit none
          integer n,j

          real*8 f, a, b, h,sum, x,res
          sum=0
          h = ((b-a) / (n-1))
c sum the midpoints
          do 10 j=2, (n-1)
          x = h * (j-1)
          sum = sum + f(x)*h
 10     continue
c add the endpoints
          res = sum+0.5*(f(a)+f(b))*h
        return
          end
```

Regola di Simpson

Approssimiamo la funzione f(x) con una parabola

$$f(x) = \alpha x^2 + \beta x + \gamma$$

L'area di ciascuna sezione è l'integrale della parabola

$$\int_{x_i}^{x_i+h} (\alpha x^2 + \beta x + \gamma) dx = \frac{\alpha x^3}{3} + \frac{\beta x^2}{3} + \gamma x \Big]_{x_i}^{x_i+h}$$

Regola di Simpson

Per mettere in relazione α, β e γ con la funzione

$$\int_{-1}^{1}(\alpha x^2+\beta x+\gamma)dx=\frac{2\alpha}{3}+2\gamma$$

Notiamo che:

$$f(-1)=\alpha-\beta+\gamma \qquad f(0)=\gamma \qquad f(1)=\alpha+\beta+\gamma$$

$$\alpha=\frac{f(1)+f(-1)}{2}-f(0) \qquad \beta=\frac{f(1)+f(-1)}{2} \qquad \gamma=f(0)$$

$$\int_{-1}^{1}(\alpha x^2+\beta x+\gamma)dx=\frac{f(-1)}{3}+\frac{4f(0)}{3}+\frac{f(1)}{3}$$

$$\int_{x_i-h}^{x_i+h}f(x)dx=\int_{x_i}^{x_i+h}f(x)dx+\int_{x_i-h}^{x_i}f(x)dx=\frac{h}{3}f_{i-1}+\frac{4h}{3}f_i+\frac{h}{3}f_{i+1}$$

Regola di Simpson

Integrale numerico con la regola di Simpson:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f(x_1) + 4f(x_2) + 2f(x_3) + \ldots + 4f(x_{n-1}) + f(x_n)\right)$$

$$h = (b-a)/(n-1) \qquad \text{con } n \text{ dispari}$$

Valutiamo $\int_0^1 e^{-x}dx$

e variamo n in modo da aumentare la precisione

C integ. Simpson

```
      program calc_int
c
      implicit none
      real*8 :: x(1000),y(1000)
      real*8 :: a,b,ff,dx,int,test
      integer :: i,n
      character(50) :: str
      a=0.0
      b=1.0
c n deve essere dispari
      n=1001
      str='Integrazione con Simpson'
      dx=(b-a)/(n-1)   !step di integrazione
      do i=1,n
         x(i)=a+(i-1)*dx
         y(i)=ff(x(i))
      end do
      call simps(n,x,y,int)


     print*,str

      print*,' integrale=',int

      test=int/(1.0 - exp(-1.0))

      print*,' test=',test
      stop
      end
c
      real*8 function ff(x)
      implicit none
      real*8 x
      ff=exp(-x)

      return
      end

      subroutine simps(n,x,f,res)
c use simpson's formula, n must be odd
      implicit none
      integer :: n,j
      real*8 :: x(n),f(n)
      real*8 :: a,b,del,res,sump,sumd
      a = x(1)
      b = x(n)
      del = x(2)-x(1)
      res = (f(1)+f(n))/3.
      sump = 0.
      sumd = 0.
      do  j = 2,n-1,2
        sump = sump + 4.*f(j)/3.
      end do
      do  j = 3,n-2,2
        sumd = sumd + 2.*f(j)/3.
      end do
      res = del*(res+sump+sumd)
      return
      end
```

Confronto fra le due regole

1) Scrivere il programma per calcolare l'integrale con la regola del trapezio e di Simpson

2) calcolare

$$\varepsilon = \left| \frac{numerico - vero}{vero} \right|$$

3) fare un plot di

$$\log_{10} \varepsilon \text{ vs } \log_{10} n$$

con regola di Simpson e regola del trapezio per

n = 2,10,20,40,.....