Exercises Lecture V
Numerical Integration in 1D

1. **Equispaced points: comparison trapezoidal-Simpson rules**
   Consider the definite integral :

$$I = \int_0^1 e^x \, dx = e - 1 = 1.718282\ldots$$

   Write a code (e.g. `int.f90`) to calculate the integral using the (1) trapezoidal rule or (2) the Simpson rule. In general, we indicate with F_n the estimate of the integral from x_0 to x_n using a discretisation in n intervals (even for the Simpson algorithm) of width $h = \frac{x_n - x_0}{n}$. Therefore:

$$\int_{x_0}^{x_n} f(x)dx = F_n^{trap} + \mathcal{O}(h^2) = F_n^{Simpson} + \mathcal{O}(h^4)$$

   where

$$F_n^{trap} = h\left[\frac{1}{2}f_0 + f_1 + \ldots + f_{n-1} + \frac{1}{2}f_n\right]$$

   and

$$F_n^{Simpson} = h\left[\frac{1}{3}f_0 + \frac{4}{3}f_1 + \frac{2}{3}f_2 + \frac{4}{3}f_3 + \ldots + \right.$$
$$\left. + \frac{4}{3}f_{n-3} + \frac{2}{3}f_{n-2} + \frac{4}{3}f_{n-1} + \frac{1}{3}f_n\right]$$

   (a) Which is the dependence on n of the error $\Delta_n = F_n - I$? You can choose $n = 2^k$ (with $k = 2, \ldots 8$, at least) in order to have equispaced points when doing a log-log plot. You should find $\Delta_n \approx 1/n^2$ for the *trapezoidal rule* and $\Delta_n \approx 1/n^4$ for the *Simpson rule*.

2. **Monte Carlo method: generic sample mean and importance sampling**

   (a) Write a code to compute the numerical estimate F_n of $I = \int_0^1 e^{-x^2} dx = \frac{\sqrt{\pi}}{2} erf(1) \approx 0.746824$ with the MC *sample mean* method using a set $\{x_i\}$ of n random points uniformly distributed in [0,1]:

$$F_n = \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

   (b) Write a code (a different one, or, better, a unique code with an option) to compute F_n using the *importance sampling* with a set $\{x_i\}$ of points generated according to the distribution $p(x) = Ae^{-x}$ (*Notice that* erf *is an intrinsic fortran function; useful to compare the numerical result with the true value*). Remind that in the *importance sampling* approach:

$$\int_a^b f(x)dx = \left\langle \frac{f(x)}{p(x)} \right\rangle \int_a^b p(x)dx \approx \frac{1}{n} \sum_{i=1}^{n} \frac{f(x_i)}{p(x_i)} \int_a^b p(x)dx = F_n$$

   with $p(x)$ which approximates the behaviour of $f(x)$, and the average is calculated over the random points $\{x_i\}$ with distribution $p(x)$.
   Notes: pay attention to:
   - the normalization of $p(x)$;
   - the exponential distribution: `expdev` *provides random numbers* `x` *distributed in* `[0,+∞[`*; here we need* `x` *in* `[0,1]` ...

   (c) Compare the efficiency of the two sampling methods (uniform and importance sampling) for the estimate of the integral by calculating the following quantities: F_n, $\sigma_n = (< f_i^2 > - < f_i >^2)^{1/2}$, $\sigma_n/\sqrt{n}$, where $f_i = f(x_i)$ in the first case, and $f_i = \frac{f(x_i)}{p(x_i)} \int_a^b p(x)dx$ in the second case (make a log-log plot of the error as a function of n: what do you see?).

3. **MC Method: acceptance-rejection**
   Using the acceptance-rejection method, calculate $\pi = 4I$ with $I = \int_0^1 \sqrt{1-x^2}dx$. The numerical estimate of the integral is $F_n = \frac{n_s}{n}$ where n_s is the number of points under the curve $f(x) = \sqrt{1-x^2}$, and n the total number of points generated. An example is given in `pi.f90`. Estimate the error associated, i.e. the difference between F_n and the true value. Discuss the dependence of the error on n.
   (*Notice that many points are needed to see the $n^{-1/2}$ behavior, which can be hidden by stochastic fluctuations; it is easier to see it by averaging over many results (obtained from random numbers sequences with different seeds*))

4. **MC method–sample mean (generic); error analysis using the "average of the averages" and the "block average"** NOTE: THIS EXERCISE IS VERY IMPORTANT !!!

(a) Write a code to estimate the same integral of previous exercise, $\pi = 4I$ with $I = \int_0^1 \sqrt{1-x^2}dx$, using the MC method of sample mean with uniformly distributed random points. Evaluate the error $\Delta_n = F_n - I$ for n=10^2, 10^3, 10^4: it should have a $1/\sqrt{n}$ behaviour.

(b) Choose in particulat $n = 10^4$ and consider the corresponding error Δ_n. Calculate $\sigma_n^2 =< f^2 > - < f >^2$. You should recognize that σ_n CANNOT BE CONSIDERED A GOOD ESTIMATE OF THE ERROR (it's much larger than the actual error. . .)

(c) In order to improve the error estimate, apply the following two different methods of variance reduction: 1) "average of the averages": do m =10 runs with n points each, and consider the average of the averages and its standard deviation:

$$\sigma_m^2 =< M^2 > - < M >^2$$

where

$$< M >= \frac{1}{m}\sum_{\alpha=1}^{m} M_\alpha \quad e \quad < M^2 >= \frac{1}{m}\sum_{\alpha=1}^{m} M_\alpha^2$$

and M_α is the average of each run. You should recognize that σ_m is a good estimate of the error associated to each measurement (=each run) and $\sigma_m \approx \sigma_n/\sqrt{n}$ is the error associated to the average over the different runs.

(d) 2) Divide now the $n = 10,000$ points into 10 subsets. Consider the averages f_s within the individual subsets and the standard deviation if the average over the subsets:

$$\sigma_s^2 =< f_s^2 > - < f_s >^2 .$$

You should notice that $\sigma_s/\sqrt{s} \approx \sigma_m$.

```
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
!     int.f90:
!     integrates f(x)=exp(x) in the interval [vmin,vamx]=[0,1]
!     using trapezoidal and Simpson rule
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

module intmod
  public :: f, trapez, simpson
contains

  ! function to be integrated
  !
  function f(x)
    implicit none
    real :: f
    real, intent(in) :: x
    f = exp(x)
    return
  end function f

  !  trapezoidal rule
  !
  function trapez(i, min, max)
    implicit none
    real :: trapez
    integer, intent(in) :: i
    real, intent(in) :: min, max
    integer :: n
    real :: x, interval
    trapez = 0.
    interval= ((max-min) / (i-1))
    ! sum over the internal points (extrema excluded)
    do n = 2, i-1
       x = interval * (n-1)
       trapez = trapez + f(x) * interval
    end do
    ! add extrema
    trapez = trapez + 0.5 * (f(min)+f(max)) * interval
    return
  end function trapez

  !  Simpson rule
  !
  function simpson(i, min, max)
    implicit none
    real :: simpson
```

```
    integer, intent(in) :: i
    real, intent(in) :: min, max
    integer :: n
    real :: x, interval
    simpson = 0.
    interval = ((max-min) / (i-1))
    ! loop EVEN points
    do n = 2, i-1, 2
       x = interval * (n-1)
       simpson = simpson + 4*f(x)
    end do
    ! loop ODD points
    do n = 3, i-1, 2
       x = interval * (n-1)
       simpson = simpson + 2*f(x)
    end do
    ! add extrema
    simpson = simpson + f(min)+f(max)
    simpson = simpson * interval/3
    return
  end function simpson

end module intmod

program int
  use intmod
  !
  ! variable declaration
  !     accuracy limit
  !     min and max in x
  !
  implicit none
  real :: r1, r2, theo, vmin, vmax, t0, t1
  integer :: i, n
  ! exact value
  vmin = 0.0
  vmax = 1.0
  theo = exp(vmax)-exp(vmin)
  print*,' exact value =',theo
  open(unit=7,file='int-tra-sim.dat',status='unknown')
  !
  write(7,*)"# N, interval, exact, Trap-exact, Simpson-exact"
  call cpu_time(t0)
  do i = 2,8
     n = 2**i
     r1 = trapez(n+1, vmin, vmax)
```

```
    r1 = (r1-theo)
    r2 = simpson(n+1, vmin, vmax)
    r2 = (r2-theo)
    write(7,'(i4,4(2x,f10.6))') n, 1./n, theo, r1, r2
  end do
  call cpu_time(t1)
  print*," total time spent:",t1-t0
  close(7)
  print*,' data saved  in int-tra-sim.dat (|diff from exact value|)'
  stop
end program int
```

```
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
!c       pi.f90: Calculates pi using MC
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
Program pi
  Implicit none
  integer, dimension(:), allocatable :: seed
  real, dimension(2) :: rnd
  Real :: area, x, y
  Integer :: i, max, pigr, sizer
  call random_seed(sizer)
  allocate(seed(sizer)
    print*,' enter max number of points='
  read*, max
  print*,' enter seed (or type /) >'
  read*, seed
  call random_seed(put=seed)
  !       open data file, initializations
  Open(7, File='pigr.dat', Status='Replace')
  pigr=0
  ! points generated within a square of side 2
  ! count how many fall within the circle x*x+y*y <= 1;
  Do i=1, max
     call random_number(rnd)
     x = rnd(1)*2-1
     y = rnd(2)*2-1
     If ((x*x + y*y) <= 1) then
        pigr = pigr+1
     Endif
     area = 4.0 * pigr/Real(i)
     if (mod(i,10)==0) Write(7,*) i, abs(acos(-1.)-area)   !write every 10 points
  end do
  Close(7)
  Stop 'data saved in pigr.dat '
End program pi
```