プログラミング演習 2

/work09/am/2023_PROGRAM/2023-10-12_15

復習

```
$ ift
```

台形公式

https://shintani.fpark.tmu.ac.jp/classes/information_processing_2/integration_2/trapezoidal.html

解析解

$$\int_0^{10} e^{-z} dz = \Big[-e^{-z} \Big]_0^{10} = -e^{-10} + 1 \simeq 1$$

誤差 $\simeq$ 8%

台形公式による数値積分

zの刻み幅1km

integ.f90

```
integer,parameter::km=11
real,dimension(km)::q,z
```

```
do k=1,km
z(k)=float(k-1)
q(k)=exp(-z(k))
end do!k

do k=km,1,-1
print *,k,z(k),q(k)
end do !k

sum=0.0
do k=1,km-1
sum=sum+(q(k)+q(k+1))*(z(k+1)-z(k))/2.0
end do

print *,'sum=',sum

end
```

```
ifort integ.f90 -o integ.exe
```

```
$ integ.exe
```

sum= 1.081928

zの刻み幅0.5km

積分の下端は0km, 上端は10kmのまま刻み幅を0.5 kmに変更する。

```
$ cp integ.f90 integ0.5.f90
```

integ0.5.f90

```
real,dimension(:),allocatable::q,z
! 配列の動的割付

zmax=10
dz=0.5
km=zmax/dz+1
allocate(q(k),z(k))

do k=1,km
z(k)=float(k-1)*dz
q(k)=exp(-z(k))
end do!k

do k=km,1,-1
print *,k,z(k),q(k)
end do !k

sum=0.0
do k=1,km-1
```

```
sum=sum+(q(k)+q(k+1))*(z(k+1)-z(k))/2.0
end do

print *,'sum=',sum

end
```

配列の動的割付

```
real,dimension(:),allocatable::q,z
```

```
zmax=10
dz=0.5
km=zmax/dz+1
allocate(q(k),z(k))
```

コンパイルと実行

```
$ ifort integ0.5.f90 -o integ0.5.exe
```

```
$ integ0.5.exe
```

シンプソン法

原理

https://shintani.fpark.tmu.ac.jp/classes/information_processing_2/integration_2/simpson.html

viエディタの置換機能

```
:%s/y/q/gc
```

integ_simpson.f90

```
program integ_simpson

real,allocatable::z(:),q(:)
integer km
real z0,z1,integ_fz

z0=0.0; z1=10.0; dz=1.0

km=(z1-z0)/dz

allocate (z(0:km),q(0:km))

do k=0,km
z(k)=z0+dz*float(k)
q(k)=f(z(k))
print *,z(k),q(k)
```

```
enddo

call simpson(q,km,dz,integ_fz)

print *,'integ_fz=',integ_fz

stop
end program

subroutine simpson(q,km,h,s)
integer,intent(in)::km
real,intent(in)::q(0:km),h
real,intent(inout)::s
real s1,s2

s1=0.0
do i=1,km-1,2
  s1=s1+q(i)
enddo

s2=0.0
do i=2,km-2,2
  s2=s2+q(i)
enddo
s=(q(0)+4.0*s1+2.0*s2+q(i))*h/3.0

end subroutine

real function f(z)
real,intent(in)::z

f=exp(-z)

end function
```

コンパイルと実行

```
$ ifort integ_simpson.f90 -o integ_simpson.exe
```

```
$ integ_simpson.exe
```

```
  0.0000000E+00   1.000000
   1.000000      0.3678795
   2.000000      0.1353353
   3.000000      4.9787067E-02
   4.000000      1.8315639E-02
   5.000000      6.7379470E-03
   6.000000      2.4787523E-03
   7.000000      9.1188197E-04
   8.000000      3.3546262E-04
   9.000000      1.2340980E-04
   10.00000      4.5399931E-05
 integ_fz=   1.004912
```

誤差0.4%に減少（台形公式の誤差は8%だった）

文法事項の解説

関数とサブルーチン

Fortranでは主プログラムと別に副プログラムを使用することができる。副プログラムには, **サブルーチン** (subroutine)と**関数** (function)の2種類ある

副プログラムは主プログラムとは分けて書く

主プログラム

```
program プログラム名
...
end program プログラム名
```

サブルーチン

```
subroutine サブルーチン名 （引数）
...
end subroutine サブルーチン名
```

関数

```
関数の型 function 関数名 （引数）
...
end function 関数名
```

Fortranにおける関数

関数の定義

```
real function f(z)
real,intent(in)::z

f=exp(-z)

end function
```

zの値を引数で指定してやると, 定義式$f(z) := \exp(-z)$を用いてfの値が計算される。

```
real function f(z)
```

上のzのことを仮引数と呼び, fのことを帰り値と呼ぶ。c言語の関数と同じ使い方。realは帰り値fの型を実数型と決めている。

関数呼び出し

関数の値を計算する

```
q(k)=f(z(k))
```

z(k)を上のfuncition f(z)の仮引数zに渡す。関数の帰り値fがq(k)に代入される。

```
real,intent(in)::z
```

intentを使うと，　その変数が入力用 (in)、出力用 (out)、両方 (inout)のどれなのかを明示することができる。プログラムがが見やすくなるのと、エラーの発生を防ぐのに役立つ。

Fortranにおけるサブルーチン

サブルーチンの定義

```
subroutine simpson(q,km,h,s)
integer,intent(in)::km
real,intent(in)::q(0:km),h
real,intent(inout)::s
real s1,s2

.....

end subroutine
```

```
subroutine simpson(q,km,h,s)
```

simpson: サブルーチン名

q,km,h,s: 仮引数

入力用の変数のリスト

```
integer,intent(in)::km
real,intent(in)::q(0:km),h
```

出力用の変数のリスト

```
real,intent(inout)::s
```

inoutにしておくとsの値を上書きすることができる。すなわち, 主プログラムにおけるsの値をサブルーチン内で書き換えることができる。今の場合は主プログラムにおけるsの値をサブルーチン内で引き継いでいないので、outでもよい。

計算の主要部

```
s1=0.0
do i=1,km-1,2
  s1=s1+q(i)
enddo

s2=0.0
do i=2,km-2,2
  s2=s2+q(i)
enddo
s=(q(0)+4.0*s1+2.0*s2+q(i))*h/3.0
```

下記URL参照。もしくは"シンプソン法"で検索する。

https://shintani.fpark.tmu.ac.jp/classes/information_processing_2/integration_2/simpson.html

当面シンプソン法の原理について理解しておくべき点は下記の通り

・与えられた関数$f(z)$をzの2次関数として近似している

・精度は刻み幅の4乗

その他具体的な計算手法については、計算結果に疑念が生じたときに調べればよい。

練習

プログラムを自分で打ち込んでみる

プログラム内にPRINT文を挿入して変数の値の変化を追う (特にDOループの中やif分の前)。

プログラム内の変数の値を変更して動作チェック。上記の例だとdzを変化させるとよい。

関数f(z)の関数形を変える。例えば関数形をsin(z)に変更して、積分の範囲を0から$\pi/2$に変更してみる。積分の値は1に近くなるはず

(https://www.geisya.or.jp/~mwm48961/electro/definite_integral3.htm)