IMPRS GW Astronomy – Computational Physics 2025

Problem Set 1

Due May 26, 2025 by email to harald.pfeiffer@aei.mpg.de and takami.kuroda@aei.mpg.de

Harald Pfeiffer, Takami Kuroda

1 Floating Point Operations

As discussed in the lecture, round-off errors and other problems due to floating point operations can cause a lot of headache. Hence, it is of special importance to be careful. Please look at the python code below.

```
Delta=[]
t=0.
dt=0.1
for i in range(1,700000):
    t=t+dt
    Delta.append(t-i*dt)
Delta=np.array(Delta)

fig,ax=plt.subplots(1,3,figsize=(12,4))
ax[0].plot(Delta[:5000])
ax[1].plot(Delta[:70000])
ax[2].plot(Delta);
fig.suptitle('Delta (three different zooms)');
```

Task FP-1: Execute the code yourself, and explain the features in the output. In particular, explain the magnitude of the peaks and their spacing.

Task FP-2: Please change the timestep to 0.099853515625. Execute the code agian and explain your findings. What caused the issue in task Task FP-1?

Another potential problem occurs when you have equalities and inequalities. Please have a look at the following C-code.

```
#include<stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
{
```

```
7    double a, b;
8
9    [...]
10
11
12   if (a==b)
13     printf("a is equal b \n");
14   else
15     printf("a is not equal b \n");
16
17   if ((1./a)==(1./b))
18     printf("1/a is equal 1/b \n");
19   else
20     printf("1/a is not equal 1/b \n");
21
22   return 0;
23 }
```

Task FP-3: Please fill the part [...] in such a way that we obtain the output: 'a is equal b' and '1/a is not equal 1/b'. To compile the code you have to install a C-compiler, e.g., gcc, and compile this code.

Another problem discussed in the lecture, was the summation or subtraction of two numbers of very different size. For the purpose of investigating this problem, please have a look at the following code, which is written in yet another programming languange: Fortan.

```
1  program addNumbers
2
3  ! This simple program adds two numbers
4     implicit none
5
6  ! Type declarations
7     real (kind=4) :: a1, b1, x1, y1, z1
8     real (kind=8):: a2, b2, x2, y2, z2
9     real (kind=16):: a3, b3, x3, y3, z3
10
11
12 !  Computation with single precision
13    a1 = 1.0
14    b1 = 1.E-13
15    x1 = a1 - sqrt(a1-b1)
16    y1 = b1/(a1+sqrt(a1-b1))
17    z1 = b1/(a1+sqrt(a1+b1))
18    print *, 'With single precision, we obtain ', x1, y1, z1
19
20    !  Computation with double precision
21    a2 = 1.0
22    b2 = 1.D-13
23    x2 = a2 - sqrt(a2-b2)
24    y2 = b2/(a2+sqrt(a2-b2))
25    z2 = b2/(a2+sqrt(a2+b2))
26    print *, 'With double precision, we obtain ', x2, y2, z2
27
28    !  Computation with quad precision
29    a3 = 1.0
30    b3 = 1.Q-13
```

```
31    x3 = a3 - sqrt(a3-b3)
32    y3 = b3/(a3+sqrt(a3-b3))
33    z3 = b3/(a3+sqrt(a3+b3))
34    print *, 'With quad   precision, we obtain ', x3, y3, z3
35
36 end program addNumbers
```

To compile this code, you have to install a fortran compiler on your machine, e.g., `gfortran`.

Task FP-4: Please run the code and explain the output. What do you see and how could you explain this. Is it surprising to you that for double and quadrupole precision even a wrong equation (z1/2/3) provides a more precise answer to the real result than the original equation (x1/2/3)?

Task FP-5: Please change line 22 to `b2 = 1.E-13`, and run the code again. How does the output change, and why?

2 Ordinary Differential Equations

2.1 Local and global error

Choose one of RK methods and explain its local and global error due only to the truncation. When we discuss the total error, which is defined by a summation of round-off and (global) truncation error, there appears a point where the total error is minimized when varying the step size. Using that the round-off error is roughly expressed as $\sim \epsilon/h$, with ϵ and h being the machine precision and step size, derive $h_{\min}$, which minimizes the total error. How does it depend on the order of truncated terms?

2.2 Convergence test

Let us consider the following system, which mimicks a chirp signal

$$A = a \sin \omega t. \tag{1}$$

Here $a(= t)$ and $\omega(= t^2/100)$ are the increasing amplitude and frequency, respectively.

- Make your own codes for solving the ODE $\dot{A}$ with the forward Euler (FE), Heun's method (Heun), and the Runge-Kutta with four stages (RK4).
- Solve the system with these three different methods and estimate the amplitude (A_{est}) evolution from $t_{\text{ini}} = 0$ to $t_{\text{end}} = 25$ with a time step of $\Delta t = (t_{\text{end}} - t_{\text{ini}})/N$. Here N is a parameter denoting the number of total steps. Compare the overall trend with the exact solution A_{exact} (Eq. 1). Here you can use $N = 10^3$.
- Evaluate the following accumulated error

$$error = \frac{1}{t_{\text{end}} - t_{\text{ini}}} \int dt |A_{\text{est}} - A_{\text{exact}}| \tag{2}$$

  and plot them for different resolutions, for instance for $N = 10^2$, 10^3, 10^4. Confirm that each method (FE, Heun, RK4) actually converges at a consistent convergence rate with the order of numerical accuracy.
- If your computational resource allows, increase the resolution, e.g., $N = 10^{5,6,7,8}$, and confirm that at the corresponding resolution $h_{\min}$ the total error is started to be contaminated by the round-off error that we discussed in the previous task. (Hint, you might get something like a following plot Fig. 1.)

2.3 Stability region

In lecture we have discussed how we can analyze the (absolute) stability region of each numerical method using a test function

$$y' = \lambda y, \tag{3}$$

with $\lambda \in \mathbb{C}$. Here we consider the stability region of Heun's method. Derive its stability function and also plot its stability region.

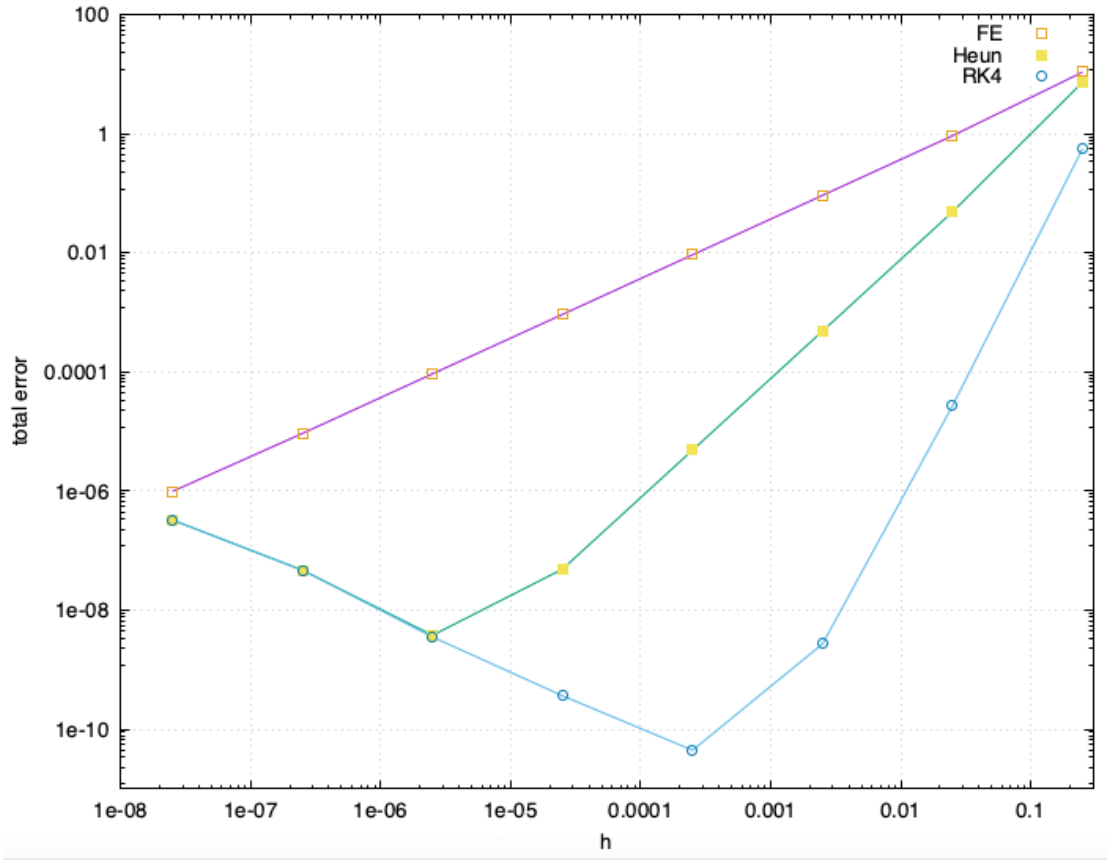

Figure 1: Convergence test. Here I plot the accumulated error $\frac{1}{t_{\rm end}-t_{\rm ini}}\int dt|A_{\rm est}-A_{\rm exact}|$, i.e., mean error, on y-axis against different resolutions on x-axis for three ODE solvers. Note that the root mean square error gives basically the same plot.