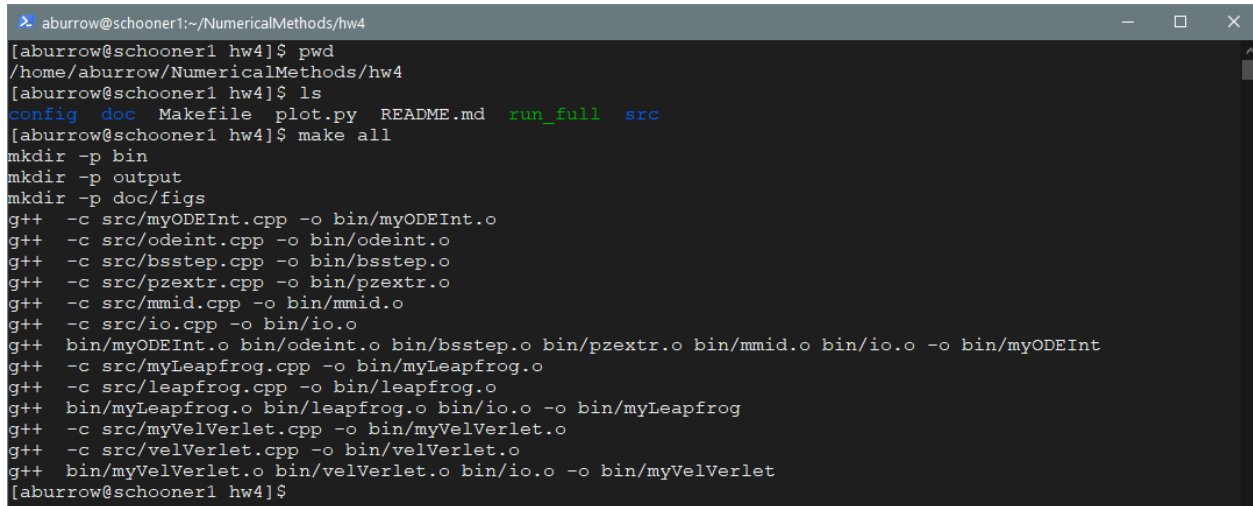


Assignment 4: Results/Description

Problem 1

For this problem I have written three programs (`myODEInt`, `myLeapfrog`, and `myVelVerlet`) to solve the following problems. These may be compiled all together with a `make all` as shown below.



```
aburrow@schooner1:~/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ pwd
/home/aburrow/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ ls
config doc Makefile plot.py README.md run_full src
[aburrow@schooner1 hw4]$ make all
mkdir -p bin
mkdir -p output
mkdir -p doc/figs
g++ -c src/myODEInt.cpp -o bin/myODEInt.o
g++ -c src/odeint.cpp -o bin/odeint.o
g++ -c src/bsstep.cpp -o bin/bsstep.o
g++ -c src/pzextr.cpp -o bin/pzextr.o
g++ -c src/mmids.cpp -o bin/mmids.o
g++ -c src/io.cpp -o bin/io.o
g++ bin/myODEInt.o bin/odeint.o bin/bsstep.o bin/pzextr.o bin/mmids.o bin/io.o -o bin/myODEInt
g++ -c src/myLeapfrog.cpp -o bin/myLeapfrog.o
g++ -c src/leapfrog.cpp -o bin/leapfrog.o
g++ bin/myLeapfrog.o bin/leapfrog.o bin/io.o -o bin/myLeapfrog
g++ -c src/myVelVerlet.cpp -o bin/myVelVerlet.o
g++ -c src/velVerlet.cpp -o bin/velVerlet.o
g++ bin/myVelVerlet.o bin/velVerlet.o bin/io.o -o bin/myVelVerlet
[aburrow@schooner1 hw4]$
```

This generates executables `myODEInt`, `myLeapfrog`, and `myVelVerlet` in the “./bin” directory, which may be run one at a time. There is also a `plot.py` in the root which may be run to generate plots from the output of the executables. However, to run all these programs at once, I have also included a `run_full` bash script that may be executed for convenience.

For the following problems, I am solving the second-order linear ordinary differential equation

$$\frac{d^2x}{dt^2} = a(x)$$

with acceleration $a(x) = -x$, which represents a harmonic oscillator with $k = \omega = m = 1$. This equation is solved with different methods below on the interval $0 \leq t \leq 10\pi$ (5 cycles). As solution for velocity is also simultaneously given in these methods, the kinetic energy is also given as $T(\dot{x}) = \frac{1}{2}m\dot{x}^2 = \frac{1}{2}v^2$ (with $m = 1$).

(a)

Below I run the program that demonstrates solving our differential equation using Numerical Recipes’ ODE integration using the Bulirsch-Stoer method:

```

aburrow@schooner1:~/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ pwd
/home/aburrow/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ ls
bin  config  doc  Makefile  output  plot.py  README.md  run_full  src
[aburrow@schooner1 hw4]$ ./bin/myODEInt
Reading from parameter file: ./config/params
x0 = 1
v0 = 0
mass = 1
No. points = 500
Relative accuracy = 1e-06
Minimum h = 0.0001 * h
Calculating solution...
Writing to ./output/odeint.dat
Complete.
[aburrow@schooner1 hw4]$

```

Here I solve for 500 points of the solution, with boundary conditions $x_0 = 1$ (initial position) and $v_0 = 0$ (initial velocity). The position solution is plotted in the left panel of Figure 1, and the kinetic energy of the system is plotted in the right panel. The residuals are also given below each panel. For this solution, we see that the calculated solution compares very well with the expected analytic solution $x(t) = x_0 \cos t$. The residuals show that it fits the solution on order of 10^{-11} , which is similar to what we found in Assignment 3. A similar result is found for the energy.

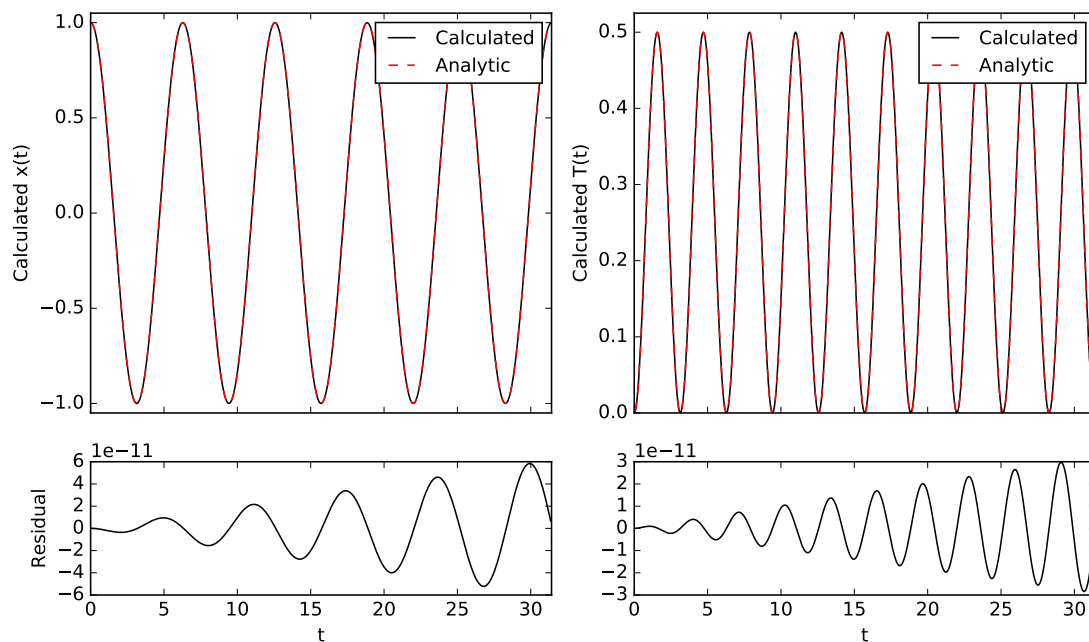


Figure 1: Solution using the ODEInt (BS) method. Left panel: Position vs. time; right panel: Kinetic energy vs. time.

Interestingly, it seems that the residuals tend to increase as a function of time between the maxima and minima.

(b)

Below I run the program that demonstrates solving the differential equation using the Leapfrog method, the algorithm for which was given to us in the assignment problem:

```
aburrow@schooner1:~/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ pwd
/home/aburrow/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ ls
bin  config  doc  Makefile  output  plot.py  README.md  run_full  src
[aburrow@schooner1 hw4]$ ./bin/myLeapfrog
Reading from parameter file: ./config/params
x0 = 1
v0 = 0
mass = 1
No. points = 500
Relative accuracy = 1e-06
Minimum h = 0.0001 * h
Calculating solution...
Writing to ./output/leapfrog.dat
Complete.
[aburrow@schooner1 hw4]$
```

Here I calculate the solution at the same 500 points with the same boundary conditions. The corresponding position and kinetic energy plot is displayed in Figure 2. We see that the solution is again fitting of the analytic solution, however this time this method produces residuals on order of 10^{-3} , and therefore it appears to be a weaker solution. And yet again, we see the residuals increase as a function of time.

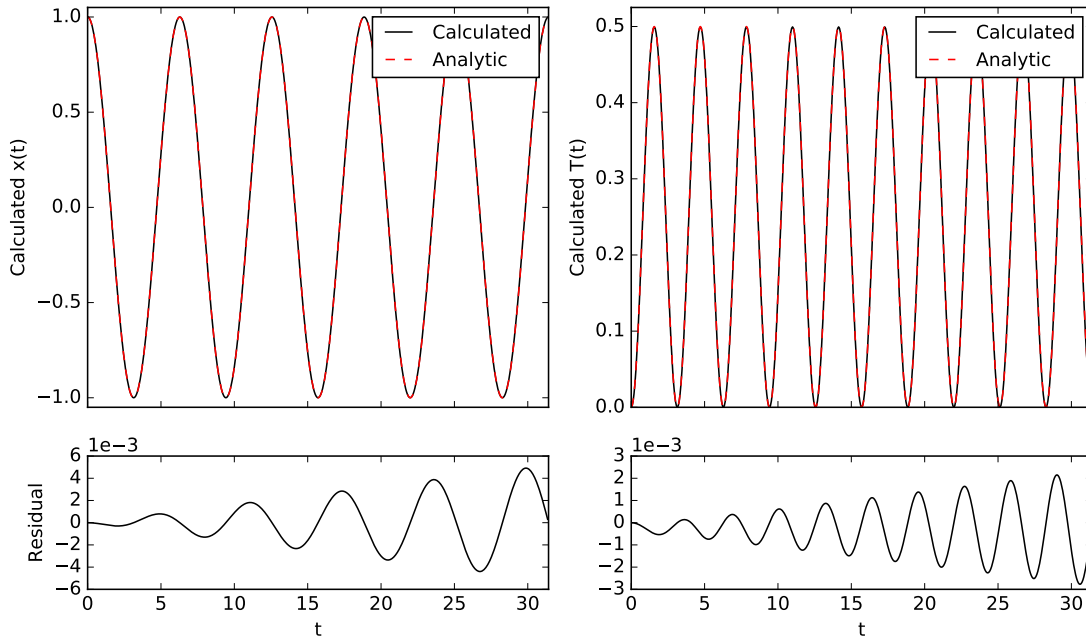
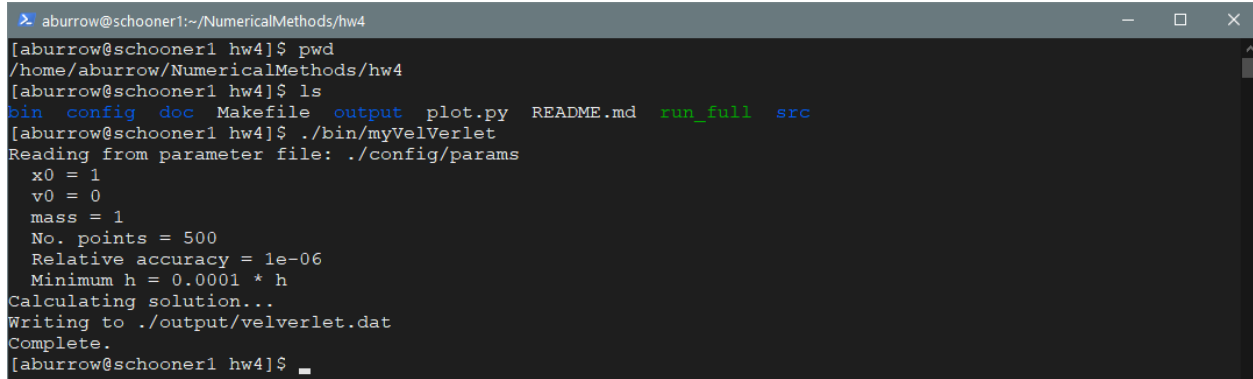


Figure 2: Solution using the Leapfrog method. Left panel: Position vs. time; right panel: Kinetic energy vs. time.

(c)

Below I run the program that demonstrates solving the differential equation using the Velocity Verlet method, the algorithm for which was given to us in the assignment problem:



```
aburrow@schooner1:~/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ pwd
/home/aburrow/NumericalMethods/hw4
[aburrow@schooner1 hw4]$ ls
bin  config  doc  Makefile  output  plot.py  README.md  run_full  src
[aburrow@schooner1 hw4]$ ./bin/myVelVerlet
Reading from parameter file: ./config/params
x0 = 1
v0 = 0
mass = 1
No. points = 500
Relative accuracy = 1e-06
Minimum h = 0.0001 * h
Calculating solution...
Writing to ./output/velverlet.dat
Complete.
[aburrow@schooner1 hw4]$
```

This algorithm is just the same as the Leapfrog method in this case. With $m = 1$, the code itself could be the exact same (where $F \rightarrow a$ and $p \rightarrow v$). However, I developed my code for this case to be more generalized just to compare and contrast the results. This is done with a force function that the user provides (it is in the myVelVerlet.cpp file).

I calculate again at the same 500 points with the same boundary conditions. The corresponding position and kinetic energy plot is in Figure 3. From the plot, it is apparent that the result is the exact same one as the leapfrog method (see Figure 2). However, in reality something in my code must have altered the results very slightly. Looking at the position column of the raw output, the last few digits of precision begin to deviate from those in the leapfrog solution over time t . This would likely be caused by the introduction of mass and the divisions necessary to transform from momentum/force to velocity/acceleration.

(d)

Looking at the plots, all three methods seem to do a fairly good job at calculating the solution to this differential equation, especially given the ease that the latter two methods took to code and perform. However, it is again abundantly clear that the Bulirsh-Stoer method is superior in calculating the solution by several orders of magnitude.

As stated before, it is also clear that Leapfrog and Velocity Verlet methods are effectively the same, however care must be taken when performing extra calculations (divisions, subtractions, etc.) to ensure the accuracy of the last few digits of precision in the values we get from them.

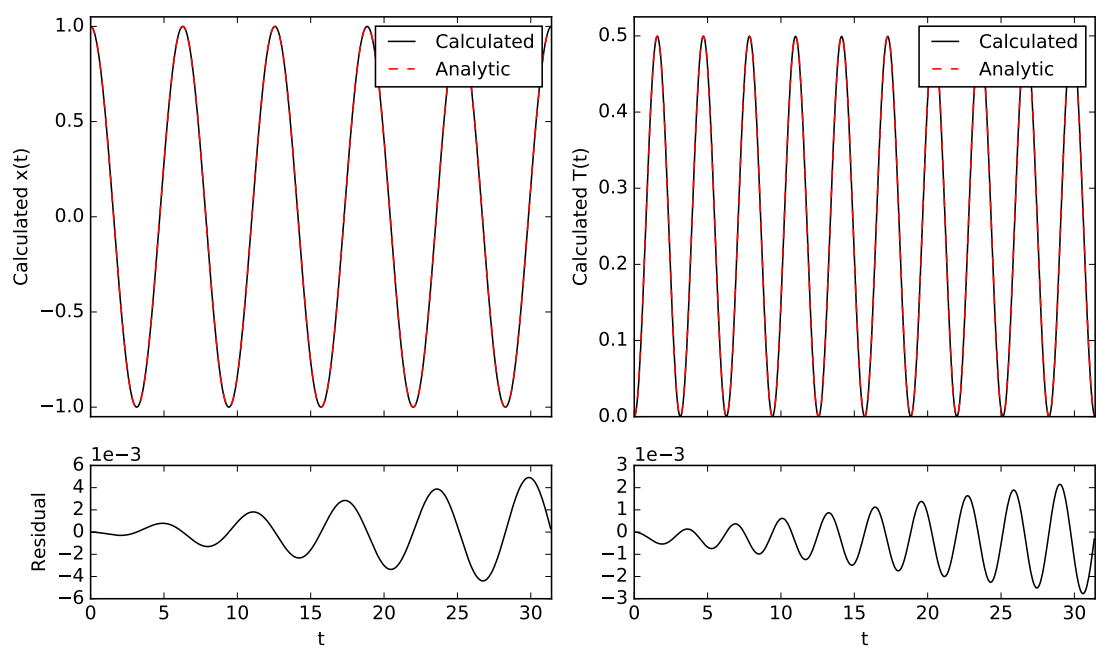


Figure 3: Solution using the Velocity Verlet method. Left panel: Position vs. time; right panel: Kinetic energy vs. time.