# Modeling and Simulation.

## Practical Assignments 2 - Integration and Ordinary Differential Equations

For all of the assignments you are supposed to write a program in C, C++ or Python with suitable "general purpose" routines and a main routine to solve the particular problem in the assignment. Maintain a concise but comprehensive lab report to describe the results of your experiments with the code. Use double-precision floating point variables. The code should run on the student machines.
The assignments can again be made in pairs. You do not need to make a full report, but make notes of your results and any issues that you encounter (maintain a lab report). Demonstrate your code to the assistant and be ready to explain your code and your results.

For reference you may want to visit
`http://en.wikipedia.org/wiki/Numerical_integration`
`http://www.gnu.org/software/gsl/`
`https://wiki.python.org/moin/PyGSL`

## 1 Numerical integration

Write four routines to integrate a function over a specified interval, one using the rectangle rule, one using the trapezoidal rule, the third using Simpson and finally one using a two-point Gauss integration. All four should allow you to specify how many subdivisions/subintervals should be used on the interval.

- Rectangle rule: use a single point at the centre of each subinterval.
- Trapezoidal rule: use a point at the start and at the end of each subinterval; both points have equal weight. Obviously, the first point of the next interval corresponds with the last point of the current interval. So, avoid unnecessary additions in your code and treat only the first and last point in a range separately.
- Simpson: can be seen as a weighted sum of the above two rules: $(2 \times Rectangle + Trapezoidal)/3$. But try to implement this as a single routine.
- Two-point Gauss: if the subinterval runs from $x$ to $x + \Delta x$ put your sample points at $x + 0.5\Delta x(1 \pm 1/\sqrt{3})$

Test the accuracy of those routines for the following integrals (compute the correct result by hand and double the number of sub-intervals until at least one method gives a decent result):

$$\int_0^1 x^2 dx \qquad \int_0^1 e^{-x} dx$$

$$\int_0^1 x^3 dx \qquad \int_0^2 xe^{-x} dx$$

$$\int_0^1 x^5 dx \qquad \int_0^2 x^{-0.5} dx$$

## 2 Using the GNU Scientific Library (optional)

Now compute the above integrals using a function from the GNU Scientific Library.

## 3 Euler and Runge-Kutta in C

The aim of the Euler integration method and of both the second order and fourth order Runge-Kutta methods basically is, given $t0$ and the value of $y(t0)$ and $y'(t)$ as a function of $t$ and $y(t)$:

$$y'(t) = f(t, y(t))$$

to find the value $y(t0 + \Delta t)$. Repeated application allows one to find $y(t1)$ at $t1 = t0 + m\Delta t$. Here $y$ and $y'$ are vectors of $N$ values, and $f(t, y(t))$ a corresponding vector function returning the N-vector $y'$ at time $t$.
You are asked to write a library with three routines Euler, RungeKutta2 and RungeKutta4:

```
int Euler(double t0, double t1, double dt, double * y0, double * y1, int N,
          int f(double, double *, double *, void *), void * params);
int RungeKutta2(double t0, double t1, double dt, double * y0, double * y1,
          int N, int f(double, double *, double *, void *), void * params);
int RungeKutta4(double t0, double t1, double dt, double * y0, double * y1,
          int N, int f(double, double *, double *, void *), void * params);
/* t0      the start value t0 of t (input)
   t1      the final value of t (input)
   dt      the increment in t (input)
   y0[N]       the values of the functions y_n(t) at t0 (input)
   y1[N]       the values of the functions y_n(t) at t1 (output)
   N      the dimension of y
   int f(double t, double * y, double * dy, void * params)      computes
          the vector of derivatives dy[N] of y[N], given t, y[N] and params
   void * params      a pointer to a struct containing any parameters needed
          by f (input)
   All three functions return an estimate of y(t1) in y1.
   All functions return 0 upon success, or -1 when a computational error
   occurs (overflow, NaN or such)
*/
```

### Testing your functions

Write a separate file to test these routines for various functions f(), and various step sizes dt. Make sure that you use f-functions for which you know the correct result of the integration, so that you can evaluate the accuracy of your results and investigate how that depends on dt.
E.g. for N = 1
f = 1, with t0 and y0 starting at 0, integrating to t = 10;
f = t, with t0 and y0 starting at 0, integrating to t = 10;
f = y, with t0 starting at 0 and y0 at 1, integrating to t = 5;
f = y * y, with t0 starting at -1 and y0 at 1, integrating to t = 1 (this should not work, why?);
f = y * y, with t0 starting at 1 and y0 at 1, integrating to t = 10 (this should work);

# 4 Coupled ODEs

## The harmonic oscillator

Investigate the stability and accuracy of the three methods above for the coupled ODEs that describe the harmonic oscillator:

$$s' = v$$

$$v' = -ks$$

As this is a simple system with a known analytic solution, it provides a good example to study the performance of the three methods. Do the following:

a. Derive the correct analytic solution for a given value of $k$ and $s(0) = 1.0$, $v(0) = 0.0$. What is the period of oscillation?

b. For each of the three methods, integrate the system for ten periods, once with a time step equal to 10% of the period, and once with a time-step equal to 5% of the period.

c. Make various scatter plots for each solution, e.g. $s(t)$ vs. $t$ and $s(t) vs. v(t)$. The latter type of plot is called a “phase space plot” and can be quite illuminating.

d. Calculate the error by comparing to the analytic solution.

Which method is best? How effective is halving the time step in each case?

## The Lotka-Volterra model

From here on, we are focusing more on the ODEs and not so much on the integration method. So the remaining exercises should all be made using the RK4 method, but you may still need to find an optimal increment dt. Alternatively, you may want to have a look at the functions in the GNU Scientific Library (GSL, highly recommended).

$$x' = -ax + cdxy$$

$$y' = by - dxy$$

Use $a = 0.5$, $b = 1$,$c = 0.1$ and $d = 0.1$. Where is the stable point for x and y for these values? Perform one simulation for initial values near the stable point and one simulation with initial values that are much further from the equilibrium point.

## Gilpin’s model

This model is much like the Lotka-Volterra model, only now there are 2 prey species that compete with each other for food resources. Study the following system:

$$x' = x(1 - 0.001x - 0.001y - 0.01z)$$

$$y' = y(1 - 0.001y - 0.0015x - 0.001z)$$

$$z' = z(0.005x + 0.0005y - 1)$$

Make sure you run the simulation for quite a long time, e.g. until $t = 2000$ and with a sufficiently high accuracy.

### Influenza epidemic - the SEIR model

Influenza is a highly contagious disease that can spread quickly through a population of susceptible individuals, especially in a confined group, like a boarding school. In the lectures, the so-called SIR model has been discussed, where a single infected person can cause a chain reaction where susceptible persons are infected, become ill and infectious and eventually recover and become immune.
Here well add one more phase  after being infected and before becoming ill, a person will be in the exposed state (incubation period). Formulate a set of equations describing this system and implement a simulation. Use the following parameters:

- Total population: 1000, of which one initially in the exposed state.
- Each person encounters 50 people per day, on average.
- Probability of being infected: 0.04 per encounter with an ill individual.
- Average incubation time: 6 days.
- Average recovery time: 9 days.

Make a plot of the evolution of the disease.

#### Optional

Now also study the case where we split both the exposed and the ill phases in three consecutive sub-phases E1, E2 and E3, and I1, I2 and I3, with an average duration of two and three days respectively.

## 5 What to submit and what to demonstrate

- Submit your program code.
- Demonstration: Compare the performance of the various algorithms (attained accuracy for different step sizes dt or h) for several functions for which you know the exact solution (tables and discussion). Show the cost of each algorithm in terms of the number of evaluations of f() that is required to obtain a given high accuracy. For each of the coupled ODEs, clearly describe the parameters you have used, also for the integration, and the observed behaviour of the system. Prepare graphs, e.g. using GNUplot.