Ordinary Differential Equations I: Introduction and Euler's Method

Sam Sinayoko

Numercal Methods 2

Contents

```
# Setup notebook
import numpy as np
# Uncomment next two lines for bigger fonts
import matplotlib
LECTURE = False
try:
    %matplotlib inline
except:
    # not in notebook
    pass
if LECTURE:
    size = 20
    matplotlib.rcParams['figure.figsize'] = (10, 6)
    matplotlib.rcParams['axes.labelsize'] = size
    matplotlib.rcParams['axes.titlesize'] = size
    matplotlib.rcParams['xtick.labelsize'] = size * 0.6
    matplotlib.rcParams['ytick.labelsize'] = size * 0.6
import matplotlib.pyplot as plt
```

1 Learning Outcomes

After studying this notebook you should be able to

- Write the general form of a first order system of **explicit ordinary differential equations** (ODEs)
- Express a second order ODE into a first order ODE.
- Describe the difference between an initial value problem and a boundary value problem.
- Solve a first order initial value problem using **Euler's method**.
- Define what the order of accuracy of a method is and give the order of accuracy of **Euler's method**.

2 Introduction

Differential equations occur frequently in the solution of science and engineering problems to model devices, systems, and the world in which we live. You may already be familiar with a range of analytic techniques which can tackle a wide range of the most commonly occurring differential equations. In this section we show how computers can be also used to solve these equations and extend the range of equations for which we can obtain an accurate solution in reasonable time and also equations for which no closed form solution is possible. A good catalogue of differential equations is Zwillinger [1].

3 Examples and Analytical solutions

3.1 Example 1: first order ODE

Suppose we had £1000 and deposited in a bank account earning 10% interest compounded continuously per year, how much would we have after 5 years (from [2] Ex 9.3)? The differential equation governing the amount of money, y, is:

$$y'(t) = ry(t), \tag{1}$$
$$y(0) = y_0, \tag{2}$$

with \$r=0.1 \$ pounds per year and $y_0 = 1000$ pounds.

We can derive an explicit formula for this. The equation is linear and separable with solution:

$$y(t) = Ce^{rt}$$

where C is an arbitrary constant, but from the initial condition at t = 0, we have

$$y(t) = y_0 e^{rt}.$$

Thus at after $t_1 = 5$ years, we would have

$$y(t_1) = y_0 e^{rt_1} = 1000e^{0.1*5} \approx 1648.72 \text{ pounds.}$$

```
r = 0.1
y0 = 1000
t1 = 5.0
t = np.linspace(0, t1)
y = y0 * np.exp(r * t)
plt.figure(1)
plt.clf()
plt.plot(t, y)
plt.xlabel('Time [years]')
plt.ylabel(u'Savings [\T1\textsterling ]')
plt.savefig('fig02-01.pdf')
```

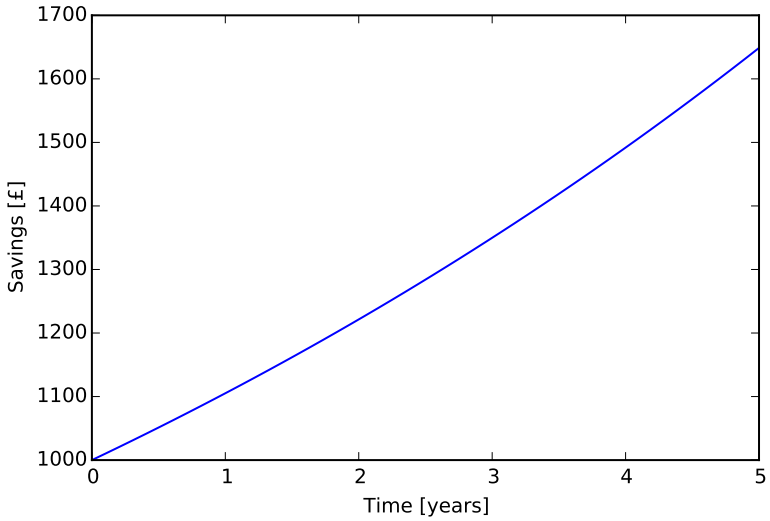

Figure 1: Savings (£) as a function of time (years) starting from 1000 £ in year 0.

3.1.1 Note

We can also easily find after how much time T our capital will double:

$$T = \frac{\ln 2}{r} \approx 7,$$

so every seven years our savings double! This ODE is ubiquitous in the

world and why governments are so excited about growth, and so worried about debt. It also applies to raw material or fossil fuel consumption: if the oil supply is finite and we consume a little bit more every year (say 1%) we will eventually exhaust the supply, hence the need for sustainable development [3].

3.2 Example 2: 1D second order ODE

A mass $m = 1\text{kg}$ initially at rest falling in a uniform gravity field $g = 9.8\text{m/s}^2$ from height $h = 10\text{m}$ satisfies Newton's second law:

$$y''(t) = -g, \tag{3}$$
$$y(0) = h, \tag{4}$$
$$y'(0) = 0. \tag{5}$$

If we integrate the ODE over time and use the initial condition $y'(0) = 0$, we get another ODE:

$$y'(t) = -gt, \tag{6}$$
$$y(0) = h. \tag{7}$$

We can solve it by integrating again and using $y(0) = h$:

$$y(t) = -\frac{1}{2}gt^2 + h. \tag{8}$$

```
m = 1
h = 10
g = 9.8
tmax = 1
t = np.linspace(0, tmax)
y = -0.5 * g * t**2 + h
plt.figure(2)
plt.clf()
plt.plot(t, y)
plt.xlabel('Time [s]')
plt.ylabel('Altitude [m]')
plt.savefig('fig02-02.pdf')
```

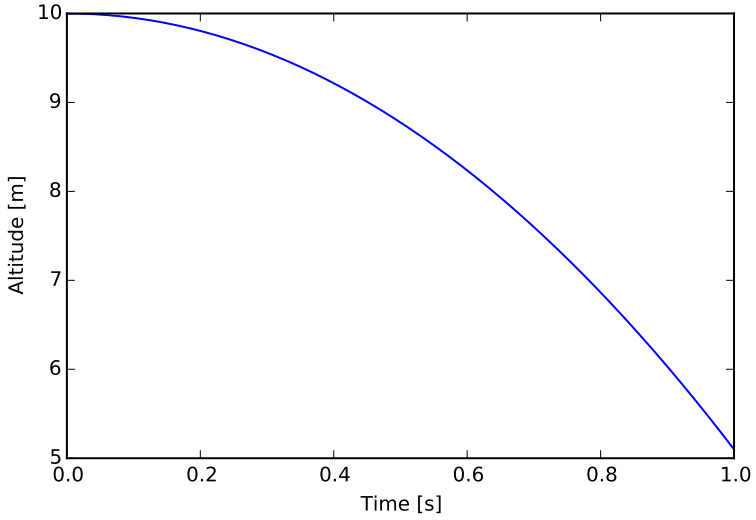

Figure 2: Altitute of a particle in free fall as a function of time starting 10 meters..

3.3 Example 3: 2D second order ODE

If the mass in Example 2 has an initial velocity in the horizontal direction $v_x = 1\text{m/s}$, then we get a two dimensional ODE:

$$y''(t) = -g, \tag{9}$$
$$x''(t) = 0, \tag{10}$$

with initial conditions

$$x(0) = 0, \qquad x'(0) = v_x \tag{11}$$
$$y(0) = 0, \qquad y'(0) = 0. \tag{12}$$

Integrating twice and using the initial conditions yields

$$y(t) = -\frac{1}{2}gt^2 + h \tag{13}$$
$$x(t) = v_x t \tag{14}$$

Note that we can rewrite this equation in vector form

$$\mathbf{Y}''(t) = \begin{pmatrix} x''(t) \\ y''(t) \end{pmatrix} = \begin{pmatrix} 0 \\ -g \end{pmatrix} = \mathbf{F} \tag{15}$$

Using such vector formulation is very useful for numerical solutions because numerical libraries tend to expect vectors.

4 General form for a first order ODE

4.1 Scalar ODE

The ODEs that appear in examples 1 and 2,

$$y'(t) = ry(t), \qquad y''(t) = -g, \qquad y'(t) = -gt \tag{16}$$

are scalar ODEs of the form

$$y'(t) = F(t, y). \tag{17}$$

These are called **first order explicit ODEs**.

Some scalar ODEs take the form

$$y'(t) = F(t, y, y') \tag{18}$$

and are called **first order implicit ODEs**.

4.2 Vector ODE – System of ODEs

More generally, systems of explicit ODEs can be expressed in terms of n scalar ODEs

$$\begin{cases} y_1'(t) = F_1(t, y_1, y_2, \cdots, y_n) \\ y_2'(t) = F_2(t, y_1, y_2, \cdots, y_n) \\ \quad \vdots \\ y_n'(t) = F_n(t, y_1, y_2, \cdots, y_n). \end{cases} \tag{19}$$

This is best expressed in vector form as

$$\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}), \tag{20}$$

where $\mathbf{Y}$, the vector containing the unkown dependent variables, and $\mathbf{F}$, the vector containing the known functions that define each scalar ODE, are given by

$$\mathbf{Y}(t) = \begin{pmatrix} y_1(t) \\ y_2(t) \\ \vdots \\ y_n(t) \end{pmatrix} \qquad \mathbf{F}(t, \mathbf{Y}) = \begin{pmatrix} F_1(t, y_1, y_2, \cdots, y_n) \\ F_2(t, y_1, y_2, \cdots, y_n) \\ \vdots \\ F_n(t, y_1, y_2, \cdots, y_n) \end{pmatrix} \tag{21}$$

Many equations that model important physical phenomena, such as the Navier-Stokes equation can be written in this form. See for example equations (2.99) in (2.94-2.98) in [4].

4.3 Higher order ODEs

Crucially, an ODE of order higher than 1 can always be expressed as a first order (vector) ODE. Thus, the second order scalar equation in Example 2 for a partical in free fall in a gravity field,

$$y''(t) = -g,$$

can be expressed as

$$\begin{cases} y'(t) = y'(t) \\ y''(t) = -g \end{cases} \tag{22}$$

so our ODE can be re-written as

$$\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}), \tag{23}$$

with

where $\mathbf{Y}$, the vector containing the unkown dependent variables, and $\mathbf{F}$, the vector containing the known functions that define each scalar ODE, are given by

$$\mathbf{Y}(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix} \qquad \mathbf{F}(t, \mathbf{Y}) = \begin{pmatrix} y'(t) \\ -g \end{pmatrix} \tag{24}$$

5 Boundary conditions

The above ODEs have a solution, and it is unique, provided that the function $\mathbf{F}$ is well behaved and that we have defined the right number of boundary conditions.

For a first order ODE of dimension n, we need n different boundary conditions. For each dependent variable y_i, we need to define either $y_i(t_i)$ or $y_i'(t_i)$ for some arbitrary value of the independent variable t_i.

5.1 Initial value problems

If all the boundary conditions are given at the same instant t_0, then the combination of the ODE with the boundary conditions is called an *initial value problem*. In that case, we define exactly how the problem starts (e.g. "exactly how to aim the gun") and solving the ODE will give us the solution for subsequent times.

5.2 Boundary value problems

If the solution is defined at different locations t_0, t_1, ... t_n, the combination of the ODE with the boundary conditions is called a *boundary value problem*. In that case, we don't know exactly how the solution starts, but we have some extra information (e.g. "when the arrow should hit the target"), so we can solve the ODE for all times. You can usually transform a boundary value problem into an initial value problem by using the shooting method [5, 6]. In this course we will therefore focus on initial value problems.

6 Euler's method

6.1 Description

Suppose we want to find an approximate solution to this initial value problem in Example 1. Let $[a, b]$ be an interval over which we want to find the solution to a well posed initial value problem $y'(t) = F(t, y)$ with $y(a)$ given. How might we approximate the solution? Let us construct a set of points $\{(t_k, y_k)\}$ that approximate the solution so $y(t_k) \approx y(t)$. We could chose mesh points and divide the interval up into M equal sub intervals and select mesh points

$$t_k = a + kh, \qquad h = \frac{b-a}{n}, \tag{25}$$

where h represents the step size and $k = 0, 1, \cdots, n$. We can now begin to solve

$$y'(t) = F(t, y), \tag{26}$$
$$y(t_0) = y_0, \tag{27}$$

for t in $[t_0, t_n] = [a, b]$.

Assuming that $y(t)$, $y'(t)$ and $y''(t)$ are continuous, we can use Taylor's theorem to expand $y(t)$ about $t = t_0$. This allows us to write that for every

value t there will be a value c_1 between t_0 and t such that

$$y(t) = y(t_0) + (t - t_0)y'(t_0) + \frac{1}{2}y''(c_1)(t - t_0)^2. \tag{28}$$

When $y'(t_0) = F(t_0, y_0)$ and $h = t_1 - t_0$ are substituted in, we have an expression for $y(t_1)$

$$y_1 = y(t_1) = y(t_0) + hF(t_0, y_0) + \frac{1}{2}y''(c_1)h^2. \tag{29}$$

If we assume that the step size h is small enough then we can ignore the term in h^2 so we obtain for our set of discretely chosen points:

$$y_1 \approx y_0 + hF(t_0, y_0). \tag{30}$$

This is known as **Euler's approximation** to the solution curve $y = y(t)$. We repeat this step by step to generate a sequence of points that can approximate the curve. The sequence is thus obtained iteratively using

$$t_{k+1} = t_k + h, \qquad\qquad y_{k+1} = y_k + hF(t_k, y_k), \tag{31}$$

for $k = 0, 1, \cdots, n$.

6.2 Implementation

A simple Euler solver based on [7]

```python
def euler(F, a, b, ya, n):
    """Solve the first order initial value problem
         y'(t) = F(t, y(t)),
          y(a) = ya,
    using Euler's method and return a tuple of made of two arrays
    (tarr, yarr) that approximate the solution on a uniformly spaced
    grid over [a, b]with n elements.

    Parameters
    ----------
    F : function
         A function of two variables of the form F(t, y), such that
         y'(t) = F(t, y(t)).
    a : float
         Initial time.
    b : float
         Final time.
    n : integer
         Controls the step size of the time grid, h = (b - a) / (n - 1)
    ya : float
         Initial condition at ya = y(a).


    """
    tarr = np.linspace(a, b, n)
    h = tarr[1] - tarr[0]
    ylst = []
    yi = ya
    for t in tarr:
        ylst.append(yi)
        yi += h * F(t, yi)

    yarr = np.array(ylst)
    return tarr, yarr
```

6.3 Example

6.3.1 Results

Let's solve Example 1 numerically and compare with the exact solution.

```
# Exact solution
r = 0.1
y0 = 1000
t0 = 0.0
t1 = 5.0  # change to 20
t = np.linspace(t0, t1)
y = y0 * np.exp(r * t)
y1 = y[-1]
print "Savings after %d years = %f" %(t1, y1)

# Numerical solutions with Euler's method
plt.figure(3)
plt.clf()
plt.plot(t, y, 'o-k', label='exact', lw=4)
plt.xlabel('Time [years]')
plt.ylabel(u'Savings [\T1\textsterling ]')
nbsteps = [5, 10, 20, 50, 100, 1000, 10000]
errlst = []
for n in nbsteps:
    teuler, yeuler = euler(lambda t, y: r * y, t0, t1, y0, n)
    yeuler1 = yeuler[-1]
    abs_err = abs(yeuler1 - y1)
    rel_err = abs_err / y1
    print ("%s steps: estimated savings = %8.3f, error = %8.3f (%5.3f %%)" \
           % (str(n).rjust(10), yeuler1, abs_err, 100 * rel_err))
    plt.plot(teuler, yeuler, label='n = %d' % n, lw=2)
    errlst.append(abs_err)
#plt.axis([4, 5, 1400, 1700])
plt.legend(loc=0)
plt.savefig('fig02-03.pdf')
```

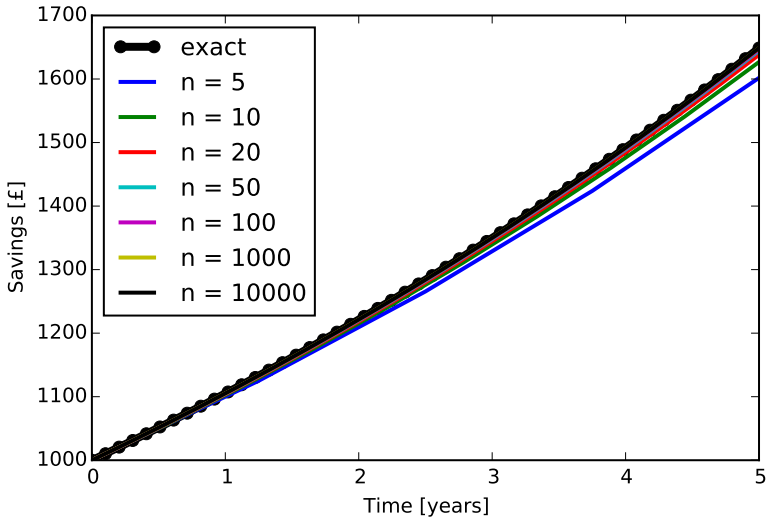

Figure 3: Numerical solution of example 1 using Euler's method.

```
# Plot the convergence of the error
plt.figure(4)
plt.clf()
plt.loglog(nbsteps, errlst)
plt.xlabel('Number of steps')
plt.ylabel(u'Absolute error (t = 5 years) [\T1\textsterling ]')
plt.savefig('fig02-04.pdf')
```

6.3.2 Discussion

Our solution converges to the exact solution: our estimate is within 2 pence of the exact result after 10000 steps. However, the convergence is slow. Looking at the logarithmic plot of the absolute error, we see that the error is divided by 10 when the number of steps is multiplied by 10 (or equivalently, when the step size is divided by 10):

$$\text{Euler's method:} \qquad h_2 = h_1/10, \qquad \text{error}_2 = \text{error}_1/10 \tag{32}$$

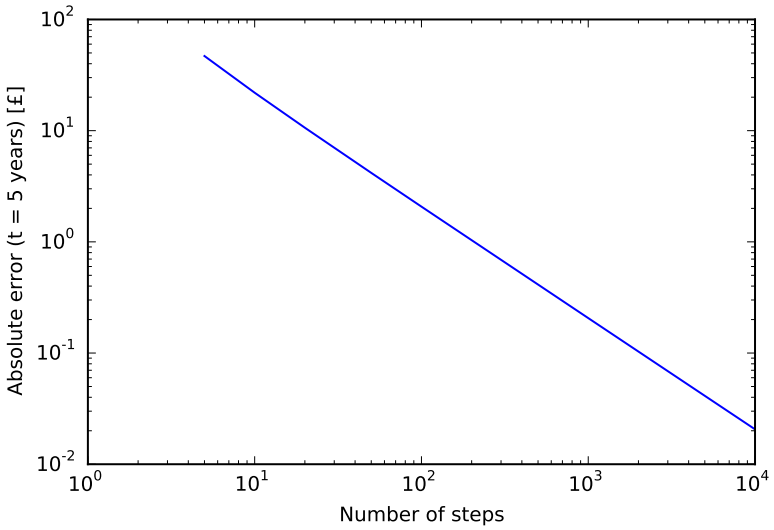

Figure 4: Absolute error as a function of the number of time steps between 0 and 5 years.

Ideally we'd like the error to decrease more rapidly as we refine the mesh. We'd like a method such that

$$\text{A more accurate method:} \qquad h_2 = h_1/10, \qquad \text{error}_2 = \text{error}_1/10^p, \qquad (33)$$

where p is an integer bigger than 2. **This integer p is called the order of the method**. The higher the order, the more accurate the method is and the steeper the slope in the logarithmic error plot becomes. Thus, we can see that Euler's method is of order 1.

An equivalent definition of the order of accuracy p of a numerical method is that the absolute error varies with h^p when h is the step size.

Note that this can also be seen by looking at the computed error for steps 10, 100, 1000 and 10000 (see Results section above):

number of steps	absolute error [£]	savings (5 years) [£]
10	21.937	1626.784
100	2.073	1646.648
1000	0.206	1648.515
10000	0.021	1648.701

It takes a very small step size to get to an accurate solution. This is because each successive iteration continues from the result of the previous

iteration, so small errors accumulate unless we proceed in very small steps (small h or equivalently, large n). This is best seen in the plot of the solution as a function of time: the error is initially small but increases with time.

The next lecture will introduce the Runge-Kutta method, which can be used to construct higher order methods and is a popular approach for solving ODEs.

7 Self study

1. Use Euler's method to the scalar initial value problem in Example 2.
2. Use Euler's method to the scalar initial value problem in Example 3.
3. * Revisit the "Description" part in the section on Euler's method and look at what happens when using the following equation based on Taylor's theorem

$$y(t) \approx y(t+h) - hy'(t+h) \tag{34}$$
$$\approx y(t+h) - hF(t+h, y(t+h)) \tag{35}$$

Is this equation explicit or implicit? Substitute $F(t, y(t)) = ry(t)$ (from Example 1) in the above equation and express $y(t+h)$ in terms of $y(t)$. Solve the equation numerically and compare with our Euler's method implementation above. This approach is another form of Euler method called Euler's backward method.

```
def solver(n, b=5, a=0, ya=1000, r=0.1):
    """Given the interest rate r, and the amount of savings ya=1000 at
    time a=0, return (tarr, yarr), where 'yarr' is an array of n
    elements giving the amount saved over time for a time grid tarr
    uniformly distributed between a and b.
    """
    # TODO
    pass
```

8 Conclusions

- The general form for an explicit ODE is $\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}(t))$.
- The general form for an explicit ODE is $\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}(t))$.

- Second order ODEs can be expressed as first order ODEs by introducing the first derivative of the solution as a dependent variable. For example, a scalar ODE $y''(t) = \mathbf{F}(t, y(t), y'(t))$ can be expressed as $\mathbf{Y}'(t) = \mathbf{F}(t, \mathbf{Y}(t))$, where

$$\mathbf{Y}(t) = \begin{pmatrix} y(t) \\ y'(t) \end{pmatrix}, \qquad \mathbf{F}(t, \mathbf{Y}) = \begin{pmatrix} Y_1(t) \\ F(t, Y_0(t), Y_1(t)) \end{pmatrix}. \tag{36}$$

- Initial value problems define the boundary condition at the same time (or location). Boundary value problems define the boundary conditions at different times (or location).

- Euler's method relies on the recursion equation

$$y(t_{k+1}) = y_{k+1} = y_k + hF(t_k, y_k),$$

where h is the step size.

- The order of accuracy of a numerical method is p when the absolute error varies with h^p.

9 References

1. Zwillinger, D. (1998). *Handbook of differential equations (Vol. 1)*. Gulf Professional Publishing.{google books}

2. Mathews, J.H. and Fink, K.D. *Numerical methods using Matlab: 3rd edition* Prentice-Hall. ISBN 0132700425. There is a 4th edition of this available (ISBN-13: 978-0130652485)

3. Al Bartlett. Lecture on *Arithmetic, Population and Energy* {youtube video}

4. Anderson. *Computational Fluid Dynamics. The Basics with Applications*, 1995. Section (2.10), page 85.

5. Wikipedia article on the Shooting Method {wikipedia article}

6. Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. (2007) *Numerical Recipes: The art of scientific computing*, Cambridge University Press, 3rd Edition. {link}

7. http://code.activestate.com/recipes/577647-ode-solver-using-euler-method/ (checked on 14 October 2015).