MATH60082
Examples Sheet 2

Contents

2.1 Euler Formula

TASKS:

Consider an initial value ODE of the following form

$$\frac{dy}{dx} = f(x, y),$$

$$x \in [a, b] \text{ and } y(a) = \alpha$$

write a function to return the value of y at x=b given

$$f(x, y) = xe^{3x} - 2y,$$

$$a = 0, \quad b = 1, \text{ and } \alpha = 0$$

For this example we are going to implement the Euler method as given by

$$w_0 = \alpha$$

$$x_i = a + ih, \text{ and } h = \frac{b-a}{n}$$

$$w_{i+1} = w_i + hf(x_i, w_i), \text{ for } i = 0, 1, \ldots, n-1$$

where w at the nth step gives an estimate for the value of y at x=b.

As with all programs we start by thinking about what are the parameters and local variables in the problem. It is clear from the specification here that the parameters are

a, b, n, as well as the function to be integrated itself although as we are not interested in writing a generic algorithm we can ignore the last one. Start with an empty program with libraries for input/output to screen/file and mathematical functions.

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

int main()
{
  return 0;
}
```

download

Compile and run the program to check you have everything set up ok.

Next shall declare the external parameters first, followed by the local parameters and then initialise any values at the start of the algorithm.

```
  // parameters
  int n=10;
  double a=0.,b=1.,alpha=0.;
  // local variables
  double h,x,w;
  // intialise values
  h=(b-a)/(double)(n);
  x=a;
  w=alpha;
```

download

Finally we can add in the rest of the algorithm in a couple of lines and output results to the screen at the same time.

```
  // implement Euler's method
  for(int i=0;i<n;i++)
  {
    x = a + i*h; // update value of x to x_i
    cout << x << " " << w << endl; // output x_i and w_i to screen
    w = w + h*(x*exp(3*x)-2.*w); // update w to w_{i+1}
  }
  cout << b << " " << w << endl; // output x_n and w_n to screen
```

download

The output from this program should give `y(x=1)=2.7609`. Now that the program is running we should check the value of the result for different values of n, and see if the results appear to converge to a constant value as the number of steps n increases.

Now we can move this code into its own function, so that we can be more flexible with obtaining results. If we use the external parameters as arguments to the function and then copy/paste the function in from "main" it will look like:

```
double eulersMethod(int n,double a,double b,double alpha)
```

```
{
  // local variables
  double h,x,w;
  // intialise values
  h=(b-a)/(double)(n);
  x=a;
  w=alpha;
  // implement Euler's method
  for(int i=0;i<n;i++)
  {
    x = a + i*h; // update value of x to x_i
    cout << x << " " << w << endl; // output x_i and w_i to screen
    w = w + h*(x*exp(3*x)-2.*w); // update w to w_{i+1}
  }
  cout << b << " " << w << endl; // output x_n and w_n to screen
  return w;
}
```

download

and can be used inside "main" like

```
  cout << " y(b) ~ " << eulersMethod(1000,0.,1.,0.) << endl;
```

download

If you want to output the results of x against w to a file for plotting then we can use the flexibility of the stream variable by including it as an argument to the function. We add this argument as `ostream& output` (must use a non constant reference) and then where `cout` appears in the function it can be replaced by `output`. The final code could look something like:

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

double eulersMethod(int n,double a,double b,double alpha,ostream& output)
{
  // local variables
  double h,x,w;
  // intialise values
  h=(b-a)/(double)(n);
  x=a;
  w=alpha;
  // implement Euler's method
  for(int i=0;i<n;i++)
  {
    x = a + i*h; // update value of x to x_i
    output << x << " , " << w << endl; // output x_i and w_i to screen
    w = w + h*(x*exp(3*x)-2.*w); // update w to w_{i+1}
  }
  output << b << " , " << w << endl; // output x_n and w_n to screen
```

```
  return w;
}

int main()
{
  // output to screen
  eulersMethod(1000,0.,1.,0.,cout);
  // output to file
  ofstream myFileStream("test.csv");
  eulersMethod(1000,0.,1.,0.,myFileStream);
  myFileStream.close();
  return 0;
}
```

download

TASKS:

(i) Run the program and check the value with $n = 10$.

(ii) Print out the final value for $n = 10, 20, 40, 80, ...$, what happens?

(iii) Calculate the difference between the results

$$\Delta_{n,m} = w_n - w_m$$

with $n = 10$ and $m = 20$, $n = 20$ and $m = 40$ and so on

(iv) Can you predict how $\Delta_{n,m}$ behaves for large n and m?

2.2 Euler Method: Convergence and Accuracy

TASKS:

Can you assess the accuracy of your solution **without** using the analytic solution?

Next we want to analyse the accuracy of the method. In this case we do have an analytic solution to compare against but often we don't, so what can we do in that case? First, assume that if a method that converges at the rate c the following holds true

$$w_n = y(b) + \frac{A}{n^c} + O(n^{-(c+1)}).$$

Then we can say that the convergence of the method is smooth if A is a constant for all n. To make an empirical estimate of the convergence we take numerical estimates using n, kn and k^2n steps where k is an integer. The ratio of differences between the estimates R, written as

$$R = \frac{w_{kn} - w_n}{w_{kkn} - w_{kn}},$$

gives the resulting formula for the convergence rate

$$c = \frac{\log(R)}{\log(k)}.$$

Methods that demonstrate smooth convergence have two advantages, firstly once a method is shown to be convergent you know that the scheme must be stable in some sense, and secondly that extrapolation techniques can be utilised to improve accuracy.

To generate an empirical estimate for the convergence rate and confirm accuracy of the method we need a slimmed down version of the previous function. Create a new project with the `eulerMethod` function as follows

```
double eulersMethod(int n,double a,double b,double alpha)
{
  // local variables
  double h,x,w;
  // intialise values
  h=(b-a)/(double)(n);
  x=a;
  w=alpha;
  // implement Euler's method
  for(int i=0;i<n;i++)
  {
    x = a + i*h; // update value of x to x_i
    w = w + h*(x*exp(3.*x)-2.*w); // update w to w_{i+1}
  }
  return w;
}
```

download

Now we want to run a loop over different values of n to generate a table of results showing the convergence rate. To calculate the ratio R you will need to keep track of the difference between subsequent results. To do this you need to store the old version of value and difference **outside** the loop. The common way to denote these is `valueOld` and `diffOld`. Inside the loop store the value and difference using current value of n. The algorithm should look a bit like this

```
  // evaulate the convergence of the method
  // using an empirical method, relies on there
  // being smooth convergence
  // ratio between number of steps at each stage
  int k=4;
```

```
  // store previous values and differences
  double valueOld=1.,diffOld=1.;
  for(int i=1;i<=10;i++)
  {
    int n=pow(k,i);
    // calculate value with n
    double value = eulersMethod(n,0.,1.,0.);
    // and difference from last time
    double diff=value-valueOld;
    // output stage, steps, value, ratio R, convergence rate c and error
    cout << i << " " << n << " " << value << " " << diffOld/diff <<endl;

    // store old values
    valueOld = value;
    diffOld = diff;
  }
```

download

You might want to add in a bit of extra information to you outputs, the following code shows the convergence rate tending to 1 for the Euler method. Take this code and implement the midpoint rule or Runge-Kutta methods and check on the rate of convergence.

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;

double eulersMethod(int n,double a,double b,double alpha)
{
  // local variables
  double h,x,w;
  // intialise values
  h=(b-a)/(double)(n);
  x=a;
  w=alpha;
  // implement Euler's method
  for(int i=0;i<n;i++)
  {
    x = a + i*h; // update value of x to x_i
    w = w + h*(x*exp(3.*x)-2.*w); // update w to w_{i+1}
  }
  return w;
}

int main()
{

  // evaulate the convergence of the method
  // using an empirical method, relies on there
  // being smooth convergence
```

```
  // ratio between number of steps at each stage
  int k=4;

  // store previous values and differences
  double valueOld=1.,diffOld=1.;
  for(int i=1;i<=10;i++)
  {
    int n=pow(k,i);
    // calculate value with n
    double value = eulersMethod(n,0.,1.,0.);
    // and difference from last time
    double diff=value-valueOld;
    // calculate R
    double R=diffOld/diff;
    // can show that R = k^c where c is convergence rate
    double c=log(R)/log(k);
    // use c to estimate A
    double A=pow(n,c)/(1-pow(k,c))*diff;
    // output stage, steps, value, constant A, ratio R, convergence rate c
       and error
    cout << i << " " << n << " " << value << " " << A << " " << R << " " << c
        << " " << fabs(A/pow(n,c)) <<endl;

    // store old values
    valueOld = value;
    diffOld = diff;
  }
//    output from this code
//    1 4 2.09213 11.8537 0.915638 -0.063575 12.9459
//    2 16 2.93241 -4.73586 1.29973 0.189106 2.80345
//    3 64 3.14744 -4.41281 3.90769 0.983158 0.0739528
//    4 256 3.20119 -4.58885 4.00079 1.00014 0.0179111
//    5 1024 3.21462 -4.59139 4.00162 1.00029 0.00447473
//    6 4096 3.21798 -4.58666 4.00049 1.00009 0.00111896
//    7 16384 3.21882 -4.58472 4.00013 1.00002 0.000279766
//    8 65536 3.21903 -4.58409 4.00003 1.00001 6.99432e-05
//    9 262144 3.21908 -4.58391 4.00001 1 1.74859e-05
//    10 1048576 3.21909 -4.58385 4 1 4.37148e-06
  return 0;
}
```

download

2.3 Coursework Example

NOTE THIS IS AN EXAMPLE AND THE FORMULA FOR h HERE WILL NOT BE CORRECT IN YOUR REAL COURSEWORK

A trader has asked you to calculate the value of an interest rate derivative contract using a non-standard model. The value of the derivative contract $V(r,t)$ can be found by

solving

$$V(r,t) = P(r,t,T)N(h) \tag{1}$$

where N is the cumulative standard normal distribution. You are given explicit functions for P and h. They are:-

$$\begin{aligned} P(r,t,T) =& \exp\left[\frac{1}{2}k^2(t,T) - n(r,t,T)\right], \\ h(r,t,T) =& f(r,t,T) + v^2(t,T) \end{aligned}$$

where

$$\begin{aligned} f(r,t,T) =& m(r,t,T) - q(t,T), \\ v^2(t,T) =& \frac{\sigma^2}{2\kappa}(1 - e^{-2\kappa(T-t)}) \\ m(r,t,T) =& e^{-\kappa(T-t)}r + (1 - e^{-\kappa(T-t)})\theta, \\ n(r,t,T) =& (T-t)\theta + (r-\theta)(1 - e^{-\kappa(T-t)})/\kappa, \\ k^2(t,T) =& \frac{\sigma^2}{2\kappa^3}(4e^{-\kappa(T-t)} - e^{-2\kappa(T-t)} + 2\kappa(T-t) - 3), \\ q(t,T) =& \frac{\sigma^2}{2\kappa^2}(1 - e^{-\kappa(T-t)})^2. \end{aligned}$$

TASKS:

(i) Write a code to find the value $V(r_0, t = 0, T)$ of the financial contract. The parameters are $r_0 = 0.05$, $T = 1$, $\kappa = 0.2$, $\theta = 0.06$ and $\sigma = 0.04$.

(ii) Using the parameters from Task (i), write a program to calculate bond price P and the option price V for different interest rates and plot P against r and V against r on the same figure.

To solve this problem you are going to want to make a function to mirror the mathematical formulation. So this will mean making functions for q, k^2, n, m, v^2, f, h, P and V. To make keeping track of parameters easier I have set all functions take in κ, θ and σ as arguments, as well as r (if needed) t and T.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
```

```
using namespace std;

double normalDistribution(double x)
{
  return 0.5*erfc(-x/sqrt(2.));
}
double qFunc(double t,double T,double kappa,double theta,double sigma)
{
    return 0.;
}
double kSquaredFunc(double t,double T,double kappa,double theta,double sigma)
{
    return 0.;
}
double nFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}
double mFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}
double vSquaredFunc(double t,double T,double kappa,double theta,double sigma)
{
    return 0.;
}
double fFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}
double hFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}
double PFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}
double VFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return 0.;
}

int main()
{
```

```
    double r0=0.05,t=0,T=1.,kappa=0.2,theta=0.06,sigma=0.04;
}
```

download

Now you need to carefully go through each function writing out the mathematics. You should check values at each stage that they seem reasonable. For instance I would be able to guess that $q \approx 0.02 \times 0.2 \times 0.2 = 0.0008$, so anything close to that and I'm happy. Check your calculation in the main function.

```
double qFunc(double t,double T,double kappa,double theta,double sigma)
{
    return sigma*sigma/2./kappa/kappa*(1-exp(-kappa*(T-t)))*(1-exp(-kappa*(T-
        t)));
}
```

download

The actual value we output is 0.000657171, which seems close enough to our estimate. Now fill in the rest of the functions.

```
double kSquaredFunc(double t,double T,double kappa,double theta,double sigma)
{
    return sigma*sigma/2./kappa/kappa/kappa*(4.*exp(-kappa*(T-t)) - exp(-2.*
        kappa*(T-t)) + 2.*kappa*(T-t) - 3.);
}
double nFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return (T-t)*theta + (r-theta)*(1-exp(-kappa*(T-t)))/kappa;
}
double mFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return r*exp(-kappa*(T-t)) + theta*(1-exp(-kappa*(T-t)));
}
double vSquaredFunc(double t,double T,double kappa,double theta,double sigma)
{
    return (sigma*sigma)/(2.*kappa)*(1-exp(-2.*kappa*(T-t)));
}
double fFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return mFunc(r,t,T,kappa,theta,sigma)-qFunc(t,T,kappa,theta,sigma);
}
double hFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return fFunc(r,t,T,kappa,theta,sigma)+vSquaredFunc(t,T,kappa,theta,sigma)
        ;
}
```

```
double PFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return exp(0.5*kSquaredFunc(t,T,kappa,theta,sigma) -nFunc(r,t,T,kappa,
        theta,sigma));
}
double VFunc(double r,double t,double T,double kappa,double theta,double
    sigma)
{
    return PFunc(r,t,T,kappa,theta,sigma)*normalDistribution(hFunc(r,t,T,
        kappa,theta,sigma));
}
```

download

All we need to do now is output to file, the values over some range of r. Let's choose $r \in [-0.1 : 0.1]$, and loop through r outputting r, V and P.

```
int main()
{
    double r0=0.05,t=0,T=1.,kappa=0.2,theta=0.06,sigma=0.04;
    double rMin=-0.1,rMax=0.1;
    int n=100;
    double dr = (rMax-rMin)/n;
    ofstream output("test.csv");
    for(int i=0;i<=100;i++)
    {
        double r = rMin + i*dr;
        output << r << " , " << PFunc(r,t,T,kappa,theta,sigma);
        output << " , " << VFunc(r,t,T,kappa,theta,sigma) << endl;
    }
}
```

download

Finally your figure should look something like this:

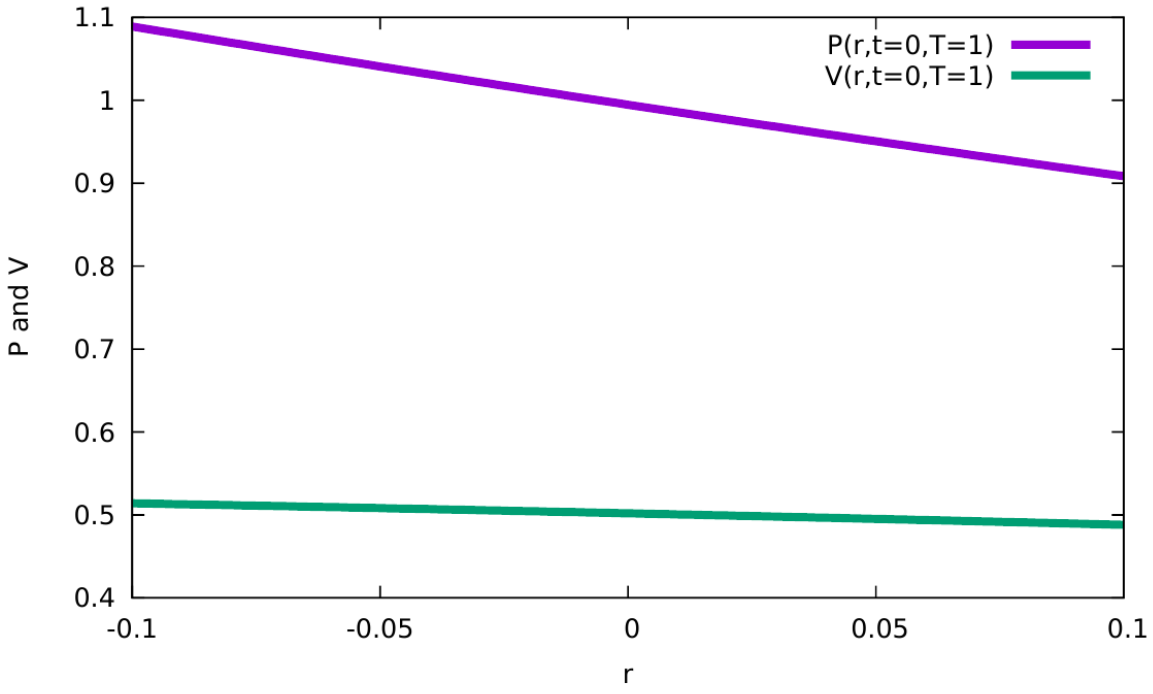

Figure 1: A plot of the function P and V against r.