Initial Value Problems

MATH2601: Control of Partial Differential Equations

A differential equation models a changing system.

Location: http://people.sc.fsu.edu/∼jburkardt/classes/control_2019/ivp/ivp.pdf

---

Differential equations describe changes in a quantity u which depends on a variable t. The differential equation is written in terms of the derivative $\frac{du}{dt}$, sometimes also written u_t or $\dot{u}$. A common form is $\frac{du}{dt} = f(t, u)$. Mathematically, solving the differential equation means determining an explicit formula for $u(t)$.

There are several classes of differential equation problems that can be posed. In the initial value problem or IVP, the problem is posed as seeking a solution $u(t)$ which satisfies the differential equation over the interval $[a, b]$, and which has the initial value $u(a) = ua$.

Except for textbook problems, differential equations cannot be solved. Hence, we employ computational techniques. Computationally solving a differential equation means determining a sequence of pairs of values (t_i, u_i) which we think of as approximate samples of the true solution. In this lab, we will look at some techniques for computing such approximations.

Computational Solution of Initial Value Problems

Given an IVP: $u' = f(t, y)$ *over* $[a, b]$, $u(a) = ua$

- *replace the continuous variables* t *by a sequence* t_i*;*
- *replace the differential equation for* t *and* u *by a sequence of discrete equations for* t_i *and* u_i*;*
- *solve the discrete equations for* (t_i, u_i)*;*
- *estimate the errors* $|u(t_i) - u_i|$*;*
- *plot the solution;*

1 The Euler Method

In order to apply a computational approach to the differential equation, we need to change it from a statement about a continuous variables to one about discrete variables. The obvious approach is to approximate the

derivative by a finite difference, so that we have:

$$\frac{\Delta u}{\Delta t} = \frac{u(t+\Delta t) - u(t)}{\Delta t} = f(t, u(t))$$

A reasonable computational solution would define $n+1$ equally spaced values $a = t_0 < t_1 < ... < t_n = b$, meaning that $\Delta t = \frac{b-a}{n}$, and try to estimate the corresponding sequence $u_0, u_1, ..., u_n$, where we hope that $u_i \approx u(t_i)$.

We start off knowing $u_0 = ua$. The finite difference equation can be used to relate u_0 and u_1:

$$\frac{\Delta u}{\Delta t} = \frac{u_1 - u_0}{\Delta t} = f(t_0, u_0)$$
$$u_1 = u_0 + \Delta t * f(t_0, u_0)$$

After determining u_1, we can write use a similar equation to relate u_1 and u_2, and proceed in this way until we have completed the calculation.

A MATLAB procedure for this process might look like this:

```
1
2  function [ t, u ] = ode_euler ( f, a, b, ua, n )
3
4    dt = ( b - a ) / n;
5    t = linspace ( a, b, n + 1 );
6    u = zeros ( n + 1, 1 );
7
8    u(1) = ua;
9    for i = 1 : n
10     u(i+1) = u(i) + dt * f ( t(i), u(i) );
11   end
12
13   return
14 end
```

To solve the problem $u' = -sin(t)$ over $[0, \pi]$ with $u(0) = 1$, we either first prepare an M file *f.m* that evaluates the right hand side:

```
1 function value = f ( t, u )
2   value = -sin(t);
3   return
4 end
```

or we can create an *anonymous function*:

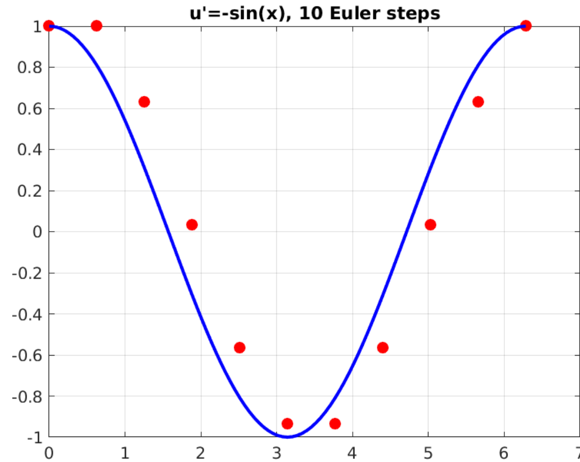
```
1   f = @(t,u) -sin(t);
```

Once **f** is defined, we can issue the commmands:

```
1   [ t, u ] = ode_euler ( f, 0, 2*pi, 1.0, 10 );
```

Plotting our 11 computed points versus the continuous solution curve $u = cos(t)$, we see that the method is able to provide a rough idea of the true solution.

```
1   plot ( t, u, 'markersize', 25 );
2   te = linspace ( 0, 2*pi, 101 );
3   ye = cos ( te );
4   hold ( 'on' );
5   plot ( te, ye );
```



2 Errors

Since we know the exact answer to the IVP, we can see that the results from the Euler computation are wrong. We can measure the error E by using the RMS norm: square the difference of each computed value and exact value, sum these terms, average them (divide by n) and then take the square root. We will write $E(n)$, because the number of steps will be an interesting parameter that can affect the error.

$$E(n) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(u_i - u(t_i))^2}$$

It seems reasonable to assume that the $E(n)$ will decrease if we increase n.

EXERCISE:

- Solve the IVP problem with $n = 10, 20, 40, 80, 160$;
- For each solution, compute $E(n)$
- What seems to happen to $E(n)$ each time we double n?

Although it is convenient for us to think in terms of the parameter n, error behavior is more often described in terms of the stepsiz. We are calling the stepsize dt; it is also often termed h. If we switch to this perspective, then we would represent the error norm as $E(dt)$. In general, we will see that the error norm behaves like a linear function of dt. We say that the error is roughly a linear function of stepsize $E \approx \alpha * dt$, or that the error is "of order dt" $E = O(dt)$.

3 The Backward Euler Method

The backward Euler method seems like a simple variation of Euler's method

$$\frac{\Delta u}{\Delta t} = \frac{u(t + \Delta t) - u(t)}{\Delta t} = f(t + \Delta t, u(t + \Delta t))$$

BLAHBLAH

A MATLAB procedure for this process might look like this:

```
function [ t, u ] = ode_euler_backward ( f, a, b, ua, n )

  dt = ( b - a ) / n;
  t = linspace ( a, b, n + 1 );
  u = zeros ( n + 1, 1 );

  u(1) = ua;
  for i = 1 : n
    v = u(i);
    for j = 1 : 10
      v = u(i) + dt * f ( t(i+1), v );
    end
    u(i+1) = v;
  end

  return
end
```

EXERCISE: Consider the IVP $u' = -\lambda * u$ over the interval $[0.0, 0.2]$ with u(0)=1. The exact solution is $u(t) = e^{-\lambda t}$. Suppose that $\lambda = 100$. Solve this problem for $dt = 0.1, 0.05, 0.025, 0.0125, 0.00625$ (which is $n = 2, 4, 8, 16, 32$); Make a table of your estimates for $u(0.2)$.

```
Estimated value of u(0.2):

dt       Euler         Backward Euler
-------------------------------------
0.1      ..........     ..........
0.05     ..........     ..........
0.025    ..........     ..........
0.0125   ..........     ..........
0.00625  ..........     ..........
```