COMP2421: Lecture 13

Exact Solutions and Errors

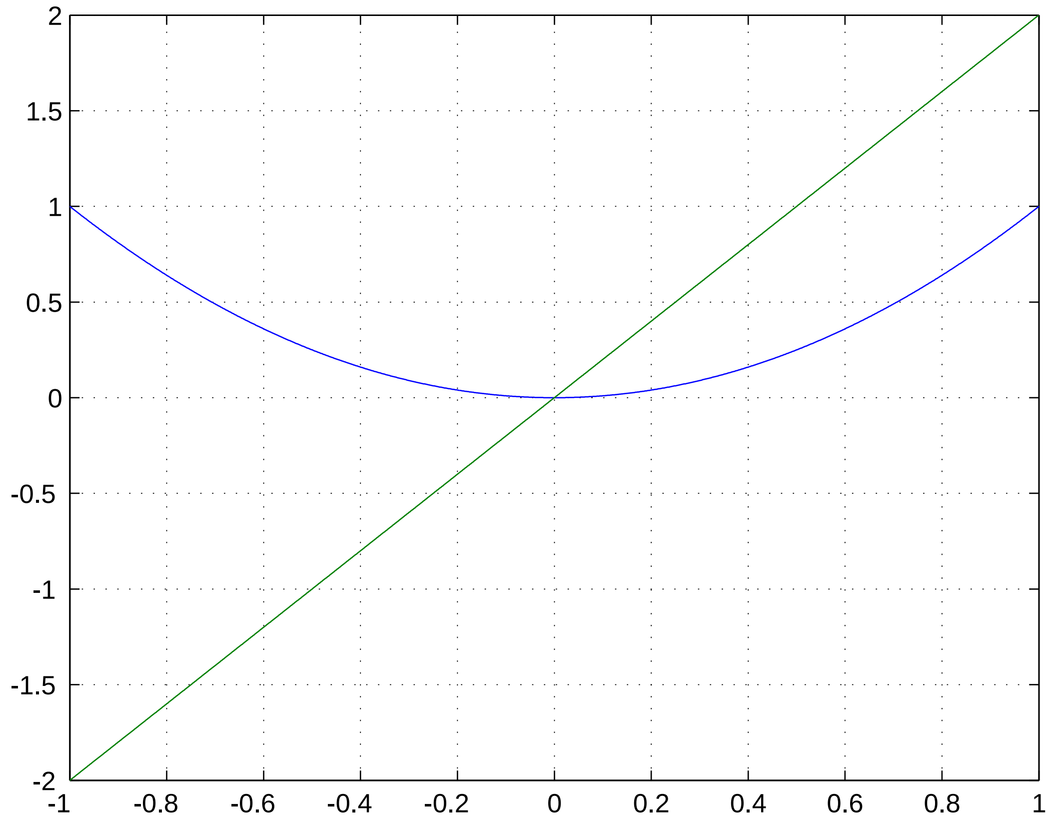
Exact Derivatives
An Exact Solution of a Differential Equation
Errors from Euler's Method
Improving Upon Euler's Method
The Midpoint Scheme

Exact Derivatives

- In some special cases it is possible to evaluate the derivative of a function exactly.
- Similarly, in some special cases it is possible to solve a differential equation exactly.
- In general however this is not the case and so computational methods are required – that is what this module is concerned with.
- The special, exact, cases are not what this module is about, however it is helpful to consider one or two examples.

Example 1

- Consider the function $y(t) = t^2$.
- We can plot an estimate of the graph of $y'(t)$ quite easily in this case:



Example 1 (cont.)

- In fact we may use the definition of $y'(t)$ to evaluate this function at any point:

$$y'(t) = \lim_{\mathrm{d}t \to 0} \frac{y(t + \mathrm{d}t) - y(t)}{\mathrm{d}t} \, .$$

- When $y(t) = t^2$ we know that

$$\begin{aligned} \frac{y(t + \mathrm{d}t) - y(t)}{\mathrm{d}t} &= \frac{(t + \mathrm{d}t)^2 - t^2}{\mathrm{d}t} \\ &= \frac{t^2 + 2 \times t \times \mathrm{d}t + (\mathrm{d}t)^2 - t^2}{\mathrm{d}t} \\ &= 2 \times t + \mathrm{d}t \, . \end{aligned}$$

- Now taking the limit as $\mathrm{d}t \to 0$ we see that, in this particular case, $y'(t) = 2t$.

Example 2

- Similarly, when $y(t) = t^3$ we have: $\frac{y(t+\mathrm{d}t)-y(t)}{\mathrm{d}t}$

$$
\begin{aligned}
&= \frac{(t+\mathrm{d}t)^3 - t^3}{\mathrm{d}t} \\
&= \frac{t^3 + 3\times t^2 \times \mathrm{d}t + 3\times t \times (\mathrm{d}t)^2 + (\mathrm{d}t)^3 - t^3}{\mathrm{d}t} \\
&= 3\times t^2 + 3\times t \times \mathrm{d}t + (\mathrm{d}t)^2 \,.
\end{aligned}
$$

- Now taking the limit as $\mathrm{d}t \to 0$ we see that, in this case, $y'(t) = 3t^2$.
- In general, we may show that when $y(t) = t^n$, then $y'(t) = nt^{n-1}$.

Example 3

- By working backwards from a known expression for $y(t)$ and $y'(t)$ we can make up our own differential equation that has $y(t)$ as a known solution.
- EG, when $\quad y(t) = t^3$:

$$y'(t) = 3t^2 = 3y(t)/t \quad \text{(for example)}.$$

- Hence we know the solution to the following equation and initial condition:

$$y'(t) = 3y(t)/t \quad \text{subject to } y(1) = 1.$$

- If we solve this for values of t between 1.0 and 2.0 (say) then we know the exact answer when $t = 2.0$ is $y(2) = 8$.

Euler's Method

- We can solve this problem using Euler's method and then look at the errors when $t = 2.0$.
- Recall that Euler's method stores the solutions in an array y as follows:

```
t = np.zeros([n+1,1])          #Initialise the arrays t and y
y = np.zeros([n+1,1])
t[0] = 1.0
y[0] = 1.0
dt = (tfinal - t0)/float(n)    #Calculate the size of each interval
for i in xrange(n):            #Take n steps of Euler's method
    y[i+1] = y[i] + dt * f(t[i],y[i])
    t[i+1] = t[i] + dt
return t,y
```

- In this particular case $f(t, y) = 3y/t$ and so the loop becomes:

```
for i in xrange(n):
    y[i+1] = y[i] + dt * 3y[i]/t[i]
    t[i+1] = t[i] + dt
```

Euler's Method – Results

- The following table shows computed results for the final solution, at $t = 2.0$, collected using the Python function `runEuler()` in `runNumerical.py`.

n	$\mathrm{d}t$	Computed solution	Error	Ratio
10	0.1	7.0000	1.0000	
20	0.05	7.4545	0.5455	0.5455
40	0.025	7.7143	0.2857	0.5238
80	0.0125	7.8537	0.1463	0.5122
160	0.00625	7.9259	0.0741	0.5062
320	0.003125	7.9627	0.0373	0.5031
640	0.0016125	7.9813	0.0187	0.5016

Euler's Method – Results (cont.)

- What is happening to the error as $\mathrm{d}t \to 0$?
  - It is decreasing.
  - Each time $\mathrm{d}t$ is halved the error is halved.
  - The error is proportional to $\mathrm{d}t$.

- What might we expect the computed solution to be if we halved $\mathrm{d}t$ one more time?

Big O Notation

- In considering algorithm complexity you have already seen this notation. For example:
  - Gaussian elimination requires $O(n^3)$ operations when n is large;
  - backward substitution requires $O(n^2)$ operations when n is large.
- For large values of n the *highest* powers of n are the most significant.
- For small values of $\mathrm{d}t$ however it is the *lowest* powers of $\mathrm{d}t$ that are the most significant:
  - when $\mathrm{d}t = 0.001$ then $\mathrm{d}t$ is much bigger than $(\mathrm{d}t)^2$ for example.

Big O Notation (cont.)

- We can make use of this “big O” notation in either case.
- For example, suppose

$$f(x) = 2x^2 + 4x^3 + x^5 + 2x^6 \ ,$$

  - then $f(x) = O(x^6)$ as $x \to \infty$;
  - and $f(x) = O(x^2)$ as $x \to 0$.
- In this notation we can say that the error in Euler’s method is $O(\mathrm{d}t)$.

Improving Upon Euler's Method

- Let's assume that the error in Euler's method is proportional to $\mathrm{d}t$.
- Then, halving $\mathrm{d}t$ will halve the error.
- Suppose the error in taking one step of size $\mathrm{d}t$ is E, then taking two steps of size $\frac{1}{2}\mathrm{d}t$ should yield an error of $E/2$:

$$y1 - y_{\text{exact}} = E \tag{1}$$

$$y2 - y_{\text{exact}} \approx E/2\ . \tag{2}$$

- Subtracting twice (2) from (1) gives:

$$y_{\text{exact}} \approx 2y2 - y1\ ,$$

which should be an improved approximation.

- On the next slides we use this to derive an improved computational algorithm...

Improving Upon Euler's Method (cont.)

- To get $y1$ take a single step of size $\mathrm{d}t$:

$$y1 = y(i) + \mathrm{d}t \times f(t(i), y(i))\ .$$

- To get $y2$ take two steps of size $\frac{\mathrm{d}t}{2}$:

$$\begin{aligned} k &= y(i) + \frac{\mathrm{d}t}{2} \times f(t(i), y(i)) \\ \mathrm{temp} &= t(i) + \frac{\mathrm{d}t}{2} \\ y2 &= k + \frac{\mathrm{d}t}{2} \times f(\mathrm{temp}, k)\ . \end{aligned}$$

Improving Upon Euler's Method (cont.)

- Combining $y1$ and $y2$ as suggested on the slide before last gives the following:

$$
\begin{aligned}
y(i+1) &= 2 \times y2 - y1 \\
&= 2k + \mathrm{dt} \times f(\mathrm{temp}, k) - y(i) - \mathrm{dt} \times f(t(i), y(i)) \\
&= 2y(i) + \mathrm{dt} \times f(t(i), y(i)) + \mathrm{dt} \times f(\mathrm{temp}, k) \\
&\quad -y(i) - \mathrm{dt} \times f(t(i), y(i)) \\
&= y(i) + \mathrm{dt} \times f(\mathrm{temp}, k) \ .
\end{aligned}
$$

- As a computational algorithm this gives:

```
for i in xrange(n):
        k = y[i] + 0.5*dt * f(t[i],y[i])
        temp = t[i] + 0.5*dt
        y[i+1] = y[i] + dt * f(temp,k)
        t[i+1] = t[i] + dt
```

The Midpoint Scheme

- The above algorithm is known as the *midpoint scheme* and it has been implemented in the Python function `midpoint(rhs,t0,y0,tfinal,n)` in `numericalSolve.py`.
- The following table shows computed results for the final solution, at $t = 2.0$, collected using the Python function `runMidpoint()` in `runNumerical.py`.

n	$\mathrm{d}t$	Computed solution	Error	Ratio
10	0.1	7.9351	0.0649	
20	0.05	7.9825	0.0175	0.2689
40	0.025	7.9955	0.0045	0.2591
80	0.0125	7.9988	0.0012	0.2545

The Midpoint Scheme (cont.)

- For this new scheme we see that the error *quarters* each time the interval dt is *halved*.
- That is the error is approximately proportional to $(\mathrm{d}t^2)$.
- Equivalently, the error is $O(\mathrm{d}t^2)$ as $\mathrm{d}t \to 0$.
- This is a significant improvement on Euler's method:
  - we say that the midpoint scheme is "second order";
  - whilst Euler's method is just "first order".

Example

- Take two steps of the midpoint rule to approximate the solution of

$$y'(t) = y(1-y) \quad \text{with } \textit{initial condition } \; y(0) = 2$$

for $0 \le t \le 1$.

- For this example we have:
  - $n = 2$
  - $t_0 = 0$
  - $y_0 = 2$
  - $t_{\text{final}} = 1$
  - $\mathrm{d}t = (1-0)/2 = 0.5$
  - $f(t, y) = y(1-y)$.

Summary

- In some special cases exact solutions of differential equations can be found – this is not true in general however.
- Computational modelling is required for most problems of practical interest (and will of course work just as well even if an exact solution could be found).
- Comparison with a known solution shows that Euler's method leads to an error that is proportional to $\mathrm{d}t$.
- The midpoint scheme's error is proportional to $(\mathrm{d}t)^2$ but requires about twice the computational work per step.
- Only 2 computational schemes introduced here – there are many more that we don't consider...