# Math 211

Lecture #8

September 21, 2000



## Numerical Methods

- A numerical "solution" is not a solution.
- It is a discrete approximation to a solution.
- We make an error on purpose to enable us to compute an approximation.
- Extremely important to understand the size of the error.



## Numerical Methods

Numerical approximation to the solution of

$$y' = f(t, y) \quad \text{with} \quad y(a) = y_0$$

on the interval $[a, b]$.
Find a discrete set of points

$$a = t_0 < t_1 < t_2 < \cdots < t_{N-1} < t_N = b$$

and values $y_0$, $y_1$, $y_2$, $\ldots$, $y_{N-1}$, $y_N$ with $y_j$ approximately equal to $y(t_j)$.

# Numerical Methods

- We will discuss four solvers
  - ◇ Euler's method,
  - ◇ second order Runge-Kutta,
  - ◇ fourth order Runge-Kutta,
  - ◇ and `ode45`.
- Everything works for first order systems almost without change.



Euler's Method: $y' = f(t, y)$ with $y(t_0) = y_0$

- Fixed step size $h = (b - a)/N$.
- At each step approximate the solution curve by the tangent line.
- First step:
  - ◇ $y(t) \approx y(t_0) + y'(t_0)(t - t_0)$. $t_1 = t_0 + h$
  - ◇ $y(t_1) \approx y(t_0) + f(t_0, y_0)h$.
  - ◇ $y_1 = y(t_0) + f(t_0, y_0)h,$ so $y(t_1) \approx y_1$.

return



Euler's Method: $y' = f(t, y)$ with $y(t_0) = y_0$

Algorithm
  Input $t_0$ and $y_0$.
  for $k = 1$ to $N$
    $y_k = y_{k-1} + f(t_{k-1}, y_{k-1})h$
    $t_k = t_{k-1} + h$
Thus,
$y_1 = y_0 + f(t_0, y_0)h$ and $t_1 = t_0 + h$
$y_2 = y_1 + f(t_1, y_1)h$ and $t_2 = t_1 + h$
etc.

back

## MATLAB routine eulerdemo.m

- Demonstrates truncation error.
- Demonstrates how truncation error can propagate exponentially.
- Demonstrates how the total error is the sum of propagated truncation errors.

return



## Error Analysis

- Euler's approximation

$$y_1 = y_0 + f(t_0, y_0)h; \quad t_1 = t_0 + h$$

- Taylor's theorem

$$y(t_0 + h) = y(t_0) + y'(t_0)h + R(h)$$
$$|R(h)| \leq Ch^2$$

- $y(t_1) - y_1 = R(h)$



## Error Analysis

- The truncation error at each step is the same as the Taylor remainder, and $|R(h)| \leq Ch^2$.
- There are $N = (b - a)/h$ steps.
- $$\text{Max error} \leq C\left(e^{L(b-a)} - 1\right) h,$$
  where $C$ & $L$ are constants that depend on $f$.

back

## Error Analysis

$$\text{Max error} \le C \left( e^{L(b-a)} - 1 \right) h,$$

where $C$ & $L$ are constants that depend on $f$.

- Good news: the error decreases as $h$ decreases.
- Bad news: the error can get exponentially large as the length of the interval [i.e., b-a] increases.



## MATLAB routine `eul.m`

- Syntax
  `[t,y] = eul(derfile,`$[t_0, t_f], y_0, h$`);`
  - ◇ `derfile` - derivative m-file defining the equation.
  - ◇ $t_0$ - initial time; $t_f$ - final time.
  - ◇ $y_0$ - initial value.
  - ◇ $h$ - step size.

return



## Derivative m-file

The derivative m-file describes the differential equation.

- Equation: $y' = y^2 - t$
- Derivative m-file:

```
function ypr = george(t,y)

ypr = y^2 - t;
```

- Save as `george.m`.

back
return

## MATLAB routine eul.m

- Solve $y' = y^2 - t$.
- Use the derivative m-file george.m.
- Use $t_0 = 0$, $t_f = 10$, $y_0 = 0.5$, and several step sizes.
- Syntax
  `[t,y] = eul('george',[0,10],0.5,h);`

back



## Experimental Error Analysis

- IVP $y' = \cos(t)/(2y-2)$ with $y(0) = 3$
- Exact solution: $y(t) = 1 + \sqrt{4 + \sin t}$.
- Solve using Euler's method and compare with the exact solution.
- Do this for several step sizes.



Derivative m-file ben.m

```
function yprime = ben(t,y)

yprime = cos(t)/(2*y-2);
```

M-file batch.m

```
[teuler,yeuler]=eul('ben',[0,3],3,h);
t=0:0.05:3;
y=1+sqrt(4+sin(t));
plot(t,y,teuler,yeuler,'o')
legend('Exact','Euler')
shg
z=1+sqrt(4+sin(teuler));
maxerror=max(abs(z-yeuler))
```