One-Step Methods

General one-step methods

- initial value problem

$$\frac{du}{dt} = f(t, u)$$
$$u(0) = u_0$$

- equivalent integral equation

$$u(t) = u(0) + \int_0^t f(s, u(s))\, ds, \quad t \in [0, T]$$

- numerical grid for t:

$$0 = t_0 < t_1 < t_2 < \cdots t_n = T$$

  - uniform grid: $t_k = kh, \quad k = 0, \ldots, n$

- one-step method for approximation $u_k \approx u(t_k)$

$$u_{k+1} = u_k + (t_{k+1} - t_k)\phi(t_k, u_k), \quad k = 0, \ldots, n$$

Euler's method

- basic idea: approximate integral in

$$u(t_{k+1}) = u(t_k) + \int_{t_k}^{t_{k+1}} f(s, u(s))\, ds$$

- rectangle rule

$$\int_{t_k}^{t_{k+1}} f(u(s), s)\, ds \approx (t_{k+1} - t_k) f(t_k, u(t_k))$$

- Euler's method

$$u_{k+1} = u_k + (t_{k+1} - t_k) f(t_k, u_k)$$

  or for equidistant grid

$$u_{k+1} = u_k + h f(t_k, u_k)$$

  - Euler's method is the simplest one-step method with

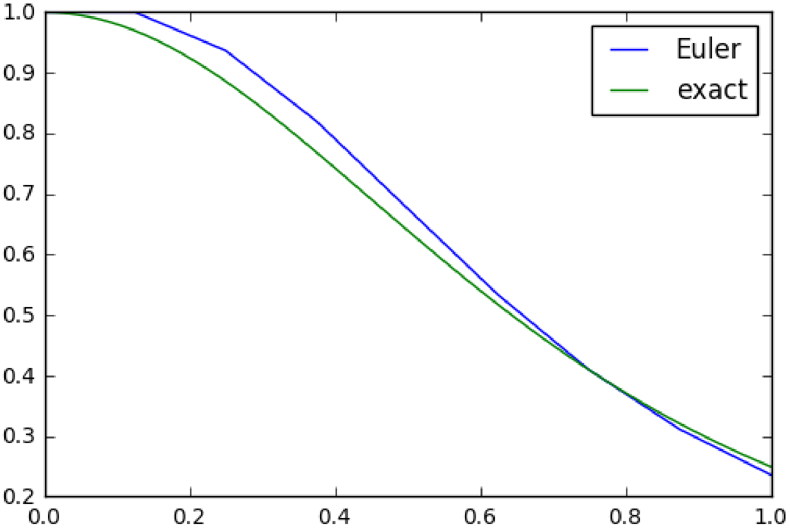
$$\phi(t, u) = f(t, u)$$

example

- IVP
$$\frac{du}{dt} = -4t(1+t^2)\,u^2, \quad u(0) = 1$$
- exact solution (by separation of variables, see ODE course)
$$u(t) = \frac{1}{(t^2+1)^2}$$

```
f = lambda t, u : -4*t*(1+t**2)*u**2
phi = f; n = 8; h = 1.0/n
th = np.linspace(0,1,n+1)
uh = np.zeros(n+1) # Euler
uh[0] = 1.0
for k,tk  in enumerate(th[:-1]):
    uh[k+1] = uh[k] + h*phi(tk,uh[k])
tex = np.linspace(0,1,129)
uex = np.zeros(129) # exact
for k,tk in enumerate(tex):
    uex[k] = 1.0/(tk**2+1)**2
```

```
plt.plot(th,uh,label='Euler');plt.plot(tex,uex,label='exact
```



recursion for error

- error at time t_k – definition

$$e_k = u_k - u(t_k)$$

- recursion: change in error = change in approximation minus change in exact value

$$e_{k+1} = e_k + (u_{k+1} - u_k) - (u(t_{k+1}) + u(t_k))$$

- Euler's method for approximation

$$u_{k+1} - u_k = hf(t_k, u_k)$$

- exact solution *almost* satisfies Euler's method

$$u(t_{k+1}) + u(t_k) = hf(t_k, u(t_k)) + hL(t_k, h)$$

where

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

is called the *local discretisation error* or *truncation error*

- substituting this into the formula for e_{k+1} gives

$$e_{k+1} = e_k + h(f(t_k, u_k) - f(t_k, u(t_k))) - hL(t_k, h)$$

interpretation and bound on error growth

- formula from last slide

$$e_{k+1} = e_k + h(f(t_k, u_k) - f(t_k, u(t_k))) - hL(t_k, h)$$

- the error e_{k+1} consists of three parts:
  - the previous error e_k at t_k
  - the effect of e_k on the Euler method: $h(f(t_k, u_k) - f(t_k, u(t_k)))$
  - the error generated by the rectangle rule approximation: $-hL(t_k, h)$
- assumption: $f(u, t)$ Lipschitz-continuous in u, i.e.

$$\|f(u, t) - f(v, t)\| \leq M\|u - v\|$$

- triangle inequality for error

$$|e_{k+1}| \leq (1 + hM)|e_k| + hL_k$$

where $L_k = |L(t_k, h)|$

a lemma

Lemma

If the $d_k > 0$ *satisfy, for some* $C > 1$ *and* $D > 0$

$$d_{k+1} \leq C d_k + D, \quad k = 0, 1, 2, \ldots$$

then

$$d_k \leq C^k d_0 + D \, \frac{C^k - 1}{C - 1}, \quad k = 0, 1, 2, \ldots$$

Proof

- by recursion
- similar to geometric series

error bound for Euler's method

Proposition

Let $T = nh$*,* M *Lipschitz constant for* $f(\cdot, t)$*,* $L = \max_{k=0,\ldots,n} L_k$ *and* e_k *be the error of Eulers method for* $du/dt = f(u,t)$*. Then*

$$|e_n| \leq \exp(TM)|e_0| + hL\frac{\exp(TM) - 1}{M}$$

- remark: often, $e_0 = 0$

Proof:

- apply bound for $|e_{k+1}|$
- use lemma with $C = 1 + hM$, $D = hL$ and $d_k = |e_k|$

example $f(t,u) = -u$

- $T = 1$, $M = 1$ and

$$L(t,h) = \frac{\exp(-(t+h)) - \exp(-t)}{h} + \exp(-t)$$
$$= \exp(-\tau) - \exp(-t), \quad \tau \in [t, t+h]$$

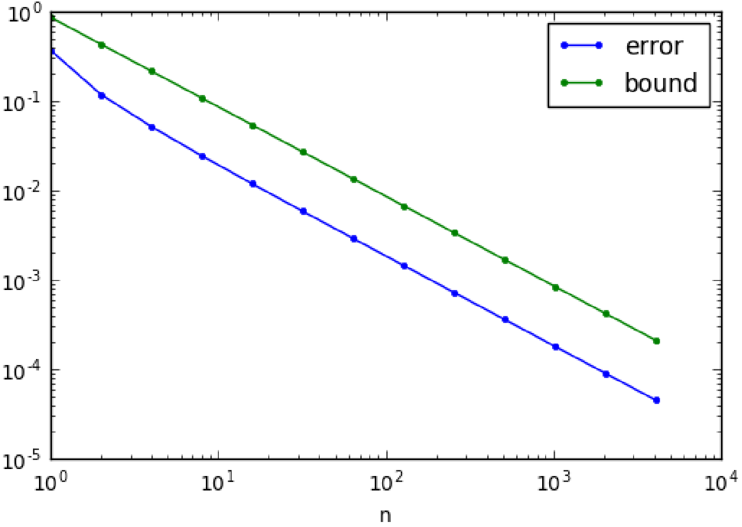
  and by mean value theorem $L = h/2$ thus
- (notice the usage of letter e in $e = 2.71\ldots$ and the error e_n)
- error bound

$$|e_n| \le h(e-1)/2$$

- remark: the error bound is a bit pessimistic but we will see how to get a better bound later

```python
f = lambda u,t : -u
phi = f;
nvec = (1,2,4,8,16,32,64,128,256,512,1024,2048,4096)
error = np.zeros(len(nvec)); bound = np.zeros(len(nvec))
for i,n in enumerate(nvec):
    h = 1.0/n
    th = np.linspace(0,1,n+1)
    uh = np.zeros(n+1) # Euler
    uh[0] = 1.0
    for k,tk  in enumerate(th[:-1]):
        uh[k+1] = uh[k] + h*phi(uh[k],tk)
    uex = np.exp(-th)
    eh  = uh - uex
    error[i] = abs(eh).max()
    bound[i] = (np.e-1)*h/2
```

```python
plt.loglog(nvec,error,'.-',label='error')
plt.loglog(nvec,bound,'.-',label='bound')
plt.xlabel('n')
plt.legend();
```



one-step methods

- methods of the form

$$u_{k+1} = u_k + h\phi(t_k, u_k)$$

- *Euler's method:*

$$\phi(t, u) = f(t, u)$$

- *Heun's method:*

$$\phi(t, u) = \frac{1}{2}(f(t, u) + f(t + h, u + hf(t, u))$$

- *midpoint method:*

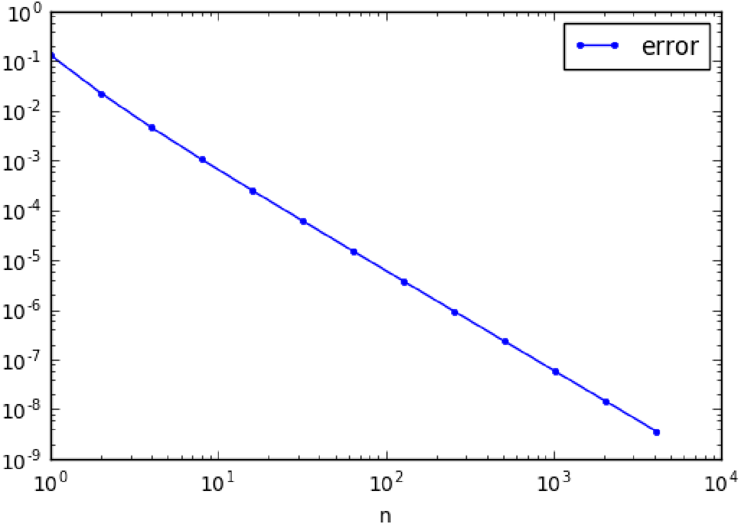
$$\phi(t, u) = f(t + h/2, u + hf(t, u)/2)$$

- these methods are referred to as *explicit methods*

example for Heun's method

```
f = lambda t, u : -u
phi = lambda t, u, h, f=f : 0.5*(f(t,u) \
                    + f(t+h,u+h*f(t,u)))
nvec = (1,2,4,8,16,32,64,128,256,512,1024,2048,4096)
error = np.zeros(len(nvec))
for i,n in enumerate(nvec):
    h = 1.0/n
    th = np.linspace(0,1,n+1)
    uh = np.zeros(n+1) # Heun
    uh[0] = 1.0
    for k,tk  in enumerate(th[:-1]):
        uh[k+1] = uh[k] + h*phi(tk, uh[k],h)
    uex = np.exp(-th)
    eh  = uh - uex
    error[i] = abs(eh).max()
```

```
plt.loglog(nvec,error,'.-',label='error')
plt.xlabel('n')
plt.legend();
```



fourth order Runge-Kutta method

- a classical method still some times used today
- four auxiliary functions

$$
\begin{aligned}
k_1 &= f(t, u) \\
k_2 &= f(t + h/2, u + hk_1/2) \\
k_3 &= f(t + h/2, u + hk_2/2) \\
k_4 &= f(t + h, u + hk_3)
\end{aligned}
$$

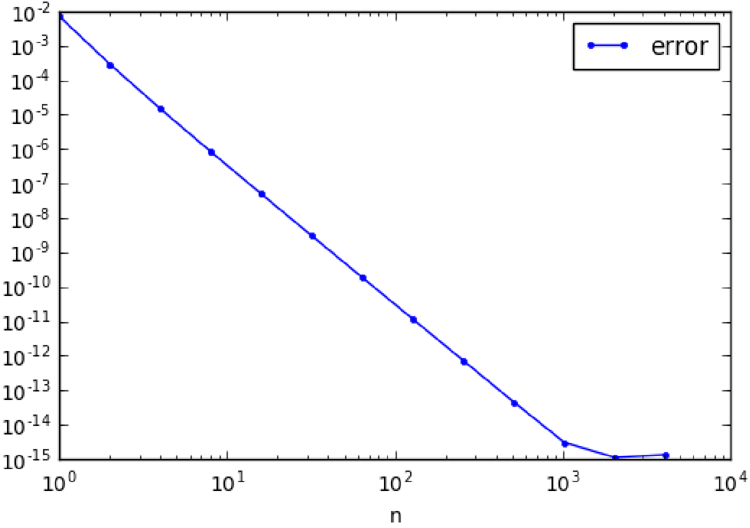
- the function $\phi(t, u)$

$$\phi(t, u) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

  - connection with Simpson's quadrature method

```
def phi(t,u,h,f=f):
    k1 = f(t,u)
    k2 = f(t+h/2,u+h*k1/2)
    k3 = f(t+h/2,u+h*k2/2)
    k4 = f(t+h,u+h*k3)
    return (k1+2*k2+2*k3+k4)/6.0
nvec = (1,2,4,8,16,32,64,128,256,512,1024,2048,4096)
error = np.zeros(len(nvec))
for i,n in enumerate(nvec):
    h = 1.0/n
    th = np.linspace(0,1,n+1)
    uh = np.zeros(n+1) # RK4
    uh[0] = 1.0
    for k,tk  in enumerate(th[:-1]):
        uh[k+1] = uh[k] + h*phi(tk, uh[k],h)
    uex = np.exp(-th)
    eh  = uh - uex
    error[i] = abs(eh).max()
```

```
plt.loglog(nvec,error,'.-',label='error')
plt.xlabel('n')
plt.legend();
```



local discretisation error of one-step method

- recall general formula for one-step method

$$u_{k+1} = u_k + h\phi(t_k, u_k)$$

- how well the exact solution satisfies the one-step method

$$L(t,h) = \frac{u(t+h) - u(t)}{h} - \phi(t, u(t))$$

Definition (consistency):

- *The one-step method is consistent if*

$$\lim_{h \to 0_+} \sup_t L(t,h) = 0$$

- *The one-step method is consistent of order p if*

$$L(t,h) = O(h^p)$$

  as $h \to 0$ uniformly in t
- $L(t,h)$ is $O(h^p)$ means here that there exists a $C > 0$ such that

$$|L(t,h)| \leq Ch^p$$

stability of one-step method

Definition (stability):
The one-step method defined by $\phi(t, u)$ is stable if $\phi(t, \cdot)$ is Lipschitz continuous, i.e.,

$$\|\phi(t, u) - \phi(t, v)\| \leq M\|u - v\|$$

for all $t \in [0, T]$

convergence theorem for one-step methods

Theorem

A one-step method which is stable and consistent is convergent.

- remark: converse holds as well (Lax equivalence theorem)

Proof

- Same as for Euler's method
- here we have

$$u(t_{k+1}) - u(t_k) = h\phi(t_k, u(t_k)) + hL(t_k, h)$$

and

$$\|\phi(t, u) - \phi(t, v)\| \leq M\|u - v\|$$

- as for Euler we get then

$$\|e_{k+1}\| \leq (1 + hM)\|e_k\| + hL_k$$

and thus

$$\|e_n\| \leq \exp(TM)\|e_0\| + L\frac{\exp(TM) - 1}{M}$$