Phys 512 ODE’s

ODEs

- usual case: we have $y(x_0)$, find $y(x_0+h)$ given $dy/dx=f(x,y)$
- Harder than integration, because we don't know how to evaluate dy/dx along the path - that depends on the (unknown) value of y.
- We can still apply many of the Taylor series tricks we've learned.
- NB - we could have a system of equations just as well as a single equation.
- NB 2 - we can also have higher order equations recast into a system. variables are then y,y',y"…

1st order

- We could just take $y(x+h)=y(x)+h\, dy/dx = y(x)+hf$
- How will error scale with size of h?
- This isn't very good…
- Can we do better?

RK2

- I could take a trial step, then evaluate derivative. Maybe combine with first derivative to do better?

- Taylor expand $y(x+h)$ to 2nd order.

- Take $k_1=hf(x,y)$, $k_2=hf(x+\alpha h,y+\beta k_1)=hf(x+\alpha h,y+\beta fh)$

- Let answer be $y(x+h)=ak_1+bk_2$.

- Solve for a,b,α,β to make agree with Taylor.

- Answer underconstrained - have freedom to pick. Usual is $\alpha=\beta=1$, $a=b=1/2$, or $\alpha=\beta=1/2$ and $a=0,b=1$.

RK2 Continued

- Taylor expansion: $y(x+h)=y(x)+hf+h^2/2[\partial f/\partial x+f\partial f/\partial y]+O(h^3)$
- $k_1=hf$
- $k_2=hf(x+\alpha h,y+\beta hf) = h(f+\alpha h\ \partial f/\partial x + \beta hf\ \partial f/\partial y)=hf+h^2(\alpha\ \partial f/\partial x + \beta f\ \partial f/\partial y)$
- Set $y(x+h)=y(x)+ak_1+bk_2$.
- $y(x+h)=y(x)+(a+b)hf + bh^2(\alpha\ \partial f/\partial x + \beta f\ \partial f/\partial y)$
- matching with Taylor gives $a+b=1$, $\alpha b=\beta b=1/2$

RK4

- Can extend to 4th order. Mathy, but not very illustrative.

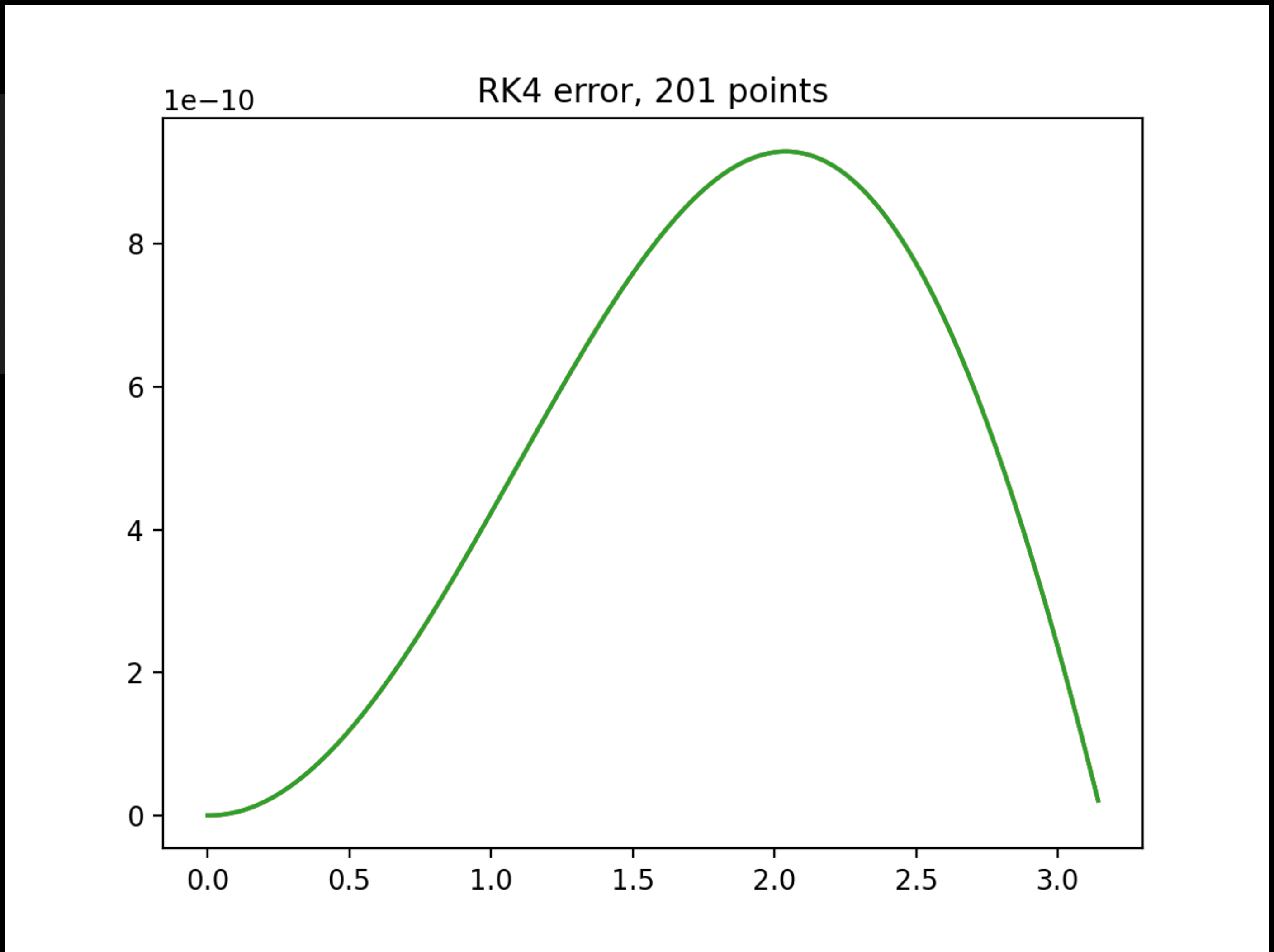
- $k_1=hf(x,y)$
  $k_2=hf(x+h/2,y+k_1/2)$
  $k_3=hf(x+h/2,y+k_2/2)$
  $k_4=hf(x+h,y+k_3)$
  $y(x+h)=y(x)+(k_1+2k_2+2k_3+k_4)/6$

- Accurate to 4th order - ODE equivalent of Simpson's rule.

- Adaptive stepsize *highly* recommended.

```python
def f(x,y):
    dydx=np.asarray([y[1],-y[0]])
    return dydx
```

```python
def rk4(fun,x,y,h):
    k1=fun(x,y)*h
    k2=h*fun(x+h/2,y+k1/2)
    k3=h*fun(x+h/2,y+k2/2)
    k4=h*fun(x+h,y+k3)
    dy=(k1+2*k2+2*k3+k4)/6
    return y+dy
```




```python
npt=201
x=np.linspace(0,np.pi,npt)
y=np.zeros([2,npt])
y[0,0]=1 #starting conditions
y[1,0]=0 #if I start at peak, then first derivative =0
for i in range(npt-1):
    h=x[i+1]-x[i]
    y[:,i+1]=rk4(f,x[i],y[:,i],h)
truth=np.cos(x)
print(np.std(truth-y[0,:]))
```

Bulirsch-Stoer

- We saw we could take multiple not-very accurate integrals, and combine to get high accuracy by estimating error terms.
- Can do the same with ODEs - Romberg equivalent is called Bulirsch-Stoer.
- If you need high accuracy, have smooth function, try this!

Stability

- let $y'=-cy$. Answer should be $\exp(-cx)$
- Solve the stupid way: $y(x+h)=y(x)+hf(x,y)= y(x)-hcy(x)$
- $y(x+h)=y(x)(1-hc)$. $y(x+nh) = y(x)(1-hc)^n$.
- What happens if h is to large? For $|1-hc|>1$, this *grows* exponentially. Large steps have made our answer *unstable*.

Stiff Equations

- Very often in systems, one equation (or eigenmode) has a large c, other has a small c.
- Solution for large c converges very quickly, and stays at solution while we wait for small c to evolve.
- If we aren't careful and take steps for small c, then large c becomes unstable, and solution blows up.
- Shrinking time step enough to track large c may be impractical.
- Such systems are called *stiff*. Solutions presented here are simplistic, but knowing you have a stiff set is most of the battle.

U238 Decay

- Radioactive decay common situation.
- U238 takes 4 billion years to go to Th234. Po214 takes 160 microseconds to go to Pb210.
- To solve naively, takes (4 billion years/160 microsconds) $=10^{21}$ timesteps.

	Half-Life	Time unit	Emitter
Uranium-238	4,468	billion of years	alpha
Thorium-234	24,10	days	beta -
Protactinium-234	6,70	hours	beta -
Uranium-234	245 500	years	alpha
Thorium-230	75380	years	alpha
Radium-226	1 600	years	alpha
Radon-222	3,8235	days	alpha
Polonium-218	3,10	minutes	alpha
Plomb-214	26,8	minutes	beta -
Bismuth-214	19,9	minutes	beta -
Polonium-214	164,3	microseconds	alpha
Plomb-210	22,3	years	beta
Bismuth-210	5,015	years	beta
Polonium-210	138,376	days	alpha
Plomb-206	Stable		

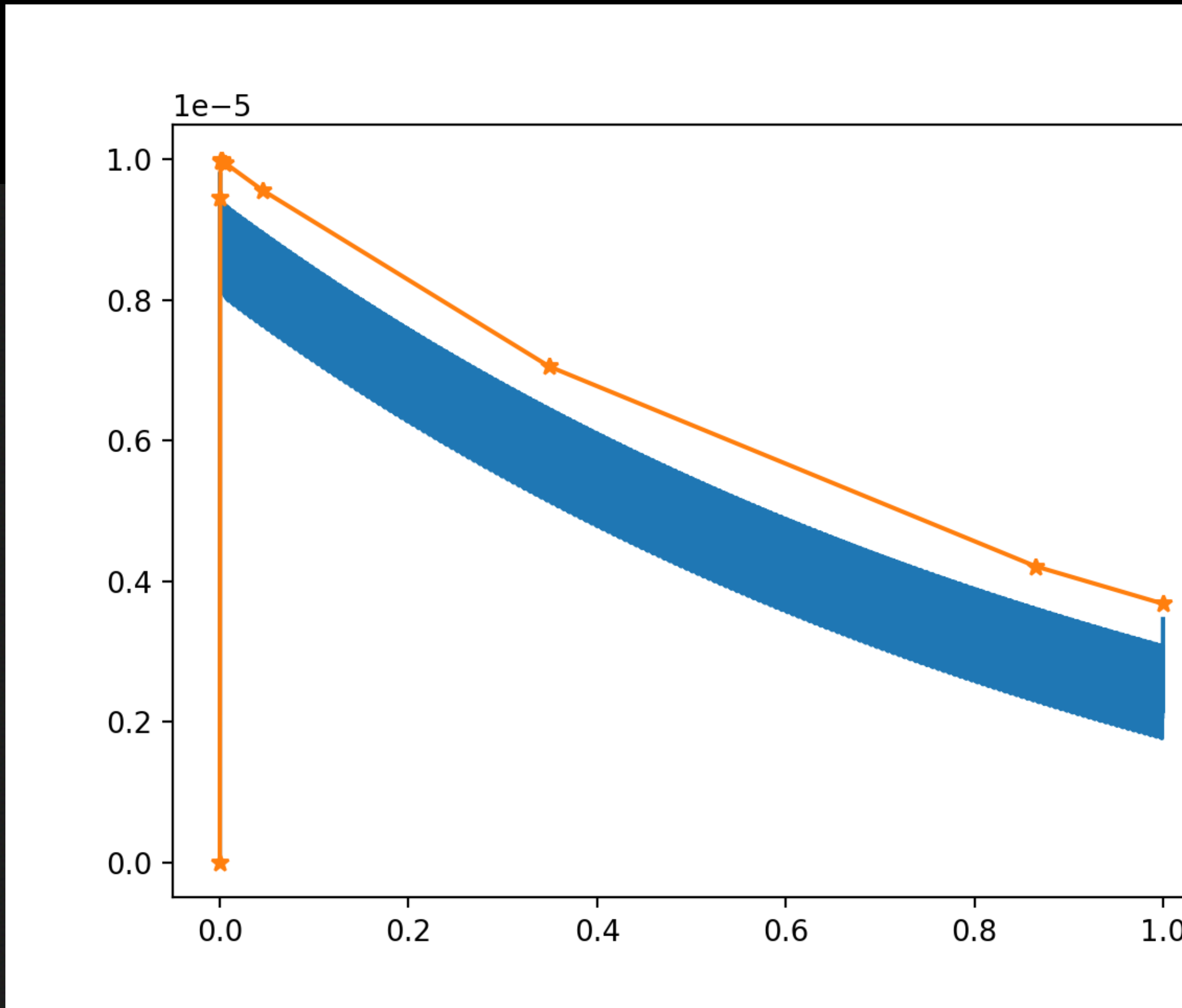
Implicit

- Common solution - use derivative at end of interval.
- In constant coefficient case $y_{n+1}=y_n-hcy_{n+1}$.
- Solve: $y_{n+1}(1+hc)=y_n$, $y_{n+1}=y_n/(1+hc)$. (reminder - old way was $y_{n+1}=y_n(1-hc)$. Agrees to 1st order but not 2nd.)
- I can now make h very large and remain stable. I may not be accurate, but I can crank up h.
- This is just an introduction, much better ways to handle stiff equations exist, but at least you know to look for them…

Scipy ODE Driver

```
import numpy as np
from scipy import integrate
def fun(x,y,half_life=[1,1e-5]):
    #let's do a 2-state radioactive decay
    dydx=np.zeros(len(half_life)+1)
    dydx[0]=-y[0]/half_life[0]
    dydx[1]=y[0]/half_life[0]-y[1]/half_life[1]
    dydx[2]=y[1]/half_life[1]
    return dydx


y0=np.asarray([1,0,0])
x0=0
x1=1
ans_rk4=integrate.solve_ivp(fun,[x0,x1],y0)
ans_stiff=integrate.solve_ivp(fun,[x0,x1],y0,method='Radau')
print('took ',ans_rk4.nfev,' evaluations to solve with RK4.')
print('took ',ans_stiff.nfev,' evaluationg to solve implicitly')
print('final values were ',ans_rk4.y[0,-1],' and ',ans_stiff.y[0,-1],'
```




```
[>>> exec(open('stiff.py').read())
 took  212618  evaluations and  3.127746820449829  seconds to solve with RK4.
 took  72  evaluations and  0.0035657882690429688  seconds to solve implicitly
 final values were  0.3678794411714445  and  0.3678803705878816  with truth  0.367879441171442
[>>> plt.clf();plt.plot(ans_rk4.t,ans_rk4.y[1,:]);plt.plot(ans_stiff.t,ans_stiff.y[1,:],'*-')
```