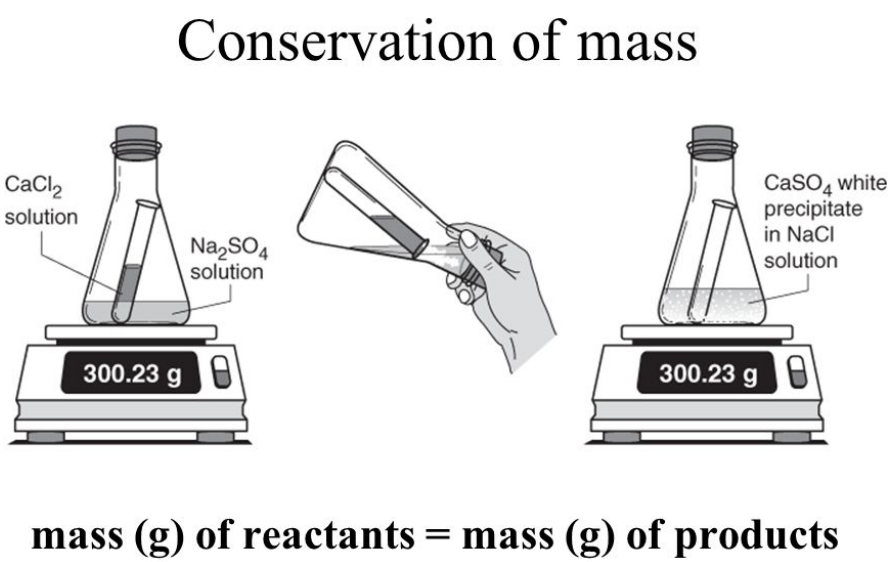
Conservation Laws for ODE's
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/ode_conservation/ode_conservation.pdf

---



"Conservation Laws"

- *Physical laws often require the conservation of some quantity $h(t)$;*
- *In such cases, the value of $h(t_0)$ should be unchanged over time;*
- *Conservation can be an important check if we don't know the exact solution;*

1 In real life, systems follow hidden laws

Mathematicians are perfectly happy making up differential equations to solve, even when they have little to do with the real world.

However, ODE's are important because they can be used to model, understand, and predict the behavior of electrical systems, engineering structures, astronomical bodies, and the absorption of medicines into the body. These systems often obey additional underlying laws that are worth observing.

One of the most useful concepts is that of **conservation**, the idea that although the observed system changes, there may be some underlying quantity whose value nonetheless stays the same. Physicists learn early on about conservation of mass, momentum, energy, and charge. If a hidden quantity is conserved in real life, we would hope that the same would be true for our ODE model.

Note that conservation is not the same as accuracy. If an ODE is meant to describe our location as we wander around the surface of the earth, then an accurate ODE solver might specify our location to within 100 feet,

but might actually place us 100 feet up in the air. On the other hand, a conservative ODE solver might be off by 100 feet, but it would specify a location that was exactly on the surface of the earth, conserving the value of $x^2 + y^2 + z^2$, our distance from the earth's center. There is a difference between being 100 feet up in the air, or 100 feet away from your desired location (but with your feet safely on the ground), even though both results have the same "accuracy".

2 The SIR ODE model of an epidemic

Earlier, we studied a spatial version of the SIR epidemic model. Now we will look at an ODE version in which our three variables are the number of Susceptible/Infected/Recovered people. We start with a population of P people, whose health is to be monitored over some time interval $t_0 \leq t \leq tstop$. At any time t, a person's status is denoted by one of three cases:

- **S**: susceptible, healthy but can catch the disease from an infected person;
- **I**: infected, can transmit the disease;
- **R**: recovered, had the disease, recovered, and is immune for some time;

Instead of monitoring the health of each person, we are simply going to keep track of three numbers at each time t, so that $S(t), I(t), R(t)$ count the number of individuals in each health class.

Rather than writing the differential equation immediately, let's begin by thinking of the situation as one in which we classify the health of our P individuals at time intervals of size Δt. Because we are modeling a real life system, there are several conditions that can observe:

- We must have $S(t), I(t), R(t)$ nonnegative;
- We must have $S(t) + I(t) + R(t) = P$;
- If $I(t)$ ever becomes zero, then the epidemic is over;
- If $dSdt, dIdt, dRdt$ approach zero, then the system approaches a "steady state".

Notice that the second condition is a conservation law. If we started with P people, our ODE solver must preserve this number.

We will make some very strong simplifying assumptions about how the disease spreads. Over a typical time interval Δt, we will assume that every pair of people can have an interaction that potentially transmits the disease. Here is a table of all the possible interactions:

```
S*S  S*I  S*R
I*S  I*I  I*R
R*S  R*I  R*R
```

The only "interesting" cases are S*I and I*S, in which a susceptible person meets an infected person. We really only need to count these encounters once, so we can assume that, at time t, over a typical time interval Δt, there are $S(t) * I(t)$ encounters in which the disease could be transmitted. Let α measure the probability that such an encounter will actually transmit the disease. Then both $S(t)$ and $I(t)$ will change:

$$
\begin{aligned}
S(t + \Delta t) &= S(t) - \frac{\alpha}{P} * S(t) * I(t) * \Delta t \\
I(t + \Delta t) &= I(t) + \frac{\alpha}{P} * S(t) * I(t) * \Delta t \\
R(t + \Delta t) &= R(t)
\end{aligned}
$$

But if this is our model, then people stay sick forever. In fact, we want to allow any sick person to recover over a time period Δt, with probability β. This means that our model becomes

$$\begin{aligned} S(t+\Delta t) &= S(t) - \frac{\alpha}{P} * S(t) * I(t) * \Delta t \\ I(t+\Delta t) &= I(t) + \frac{\alpha}{P} * S(t) * I(t) * \Delta t && -\beta * I(t) * \Delta t \\ R(t+\Delta t) &= R(t) && +\beta * I(t) * \Delta t \end{aligned}$$

Our final adjustment is made to account for the possibility that people who have recovered from the disease might catch it again. If that is the case, then there is a probability γ that any recovered person must move back into the susceptible class. And in that case, our SIR difference equation becomes:

$$\begin{aligned} S(t+\Delta t) &= S(t) - \frac{\alpha}{P} * S(t) * I(t) * \Delta t && && +\gamma * R(t) * \Delta t \\ I(t+\Delta t) &= I(t) + \frac{\alpha}{P} * S(t) * I(t) * \Delta t && -\beta * I(t) * \Delta t \\ R(t+\Delta t) &= R(t) && +\beta * I(t) * \Delta t && -\gamma * R(t) * \Delta t \end{aligned}$$

Now we can rearrange things, and take the limit as $\Delta t \rightarrow 0$ to transform our SIR difference equation into an SIR differential equation:

$$\begin{aligned} \frac{dS}{dt} &= -\frac{\alpha}{P} SI + \gamma R \\ \frac{dI}{dt} &= +\frac{\alpha}{P} SI - \beta I \\ \frac{dR}{dt} &= +\beta I - \gamma R \end{aligned}$$

There's actually a conservation law that these differential equations are hiding. Define the variable $P(t) = S(t) + I(t) + R(t)$ and note that if must follow that $\frac{dP}{dt} = \frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt}$. If you add up the left hand sides of the SIR ODE, and the right hand sides of the SIR ODE, you get the following differential equation:

$$\frac{dP}{dt} = 0$$

In other words, our SIR ODE model "knows" that the value of P, the total population, must be conserved. That's a mathematical statement. This does not guarantee that there is a corresponding computational statement of conservation! We will monitor the behavior of $P(t)$ over time, as a way to check whether our ODE solutions are "reasonable".

3 *sir_solve_ivp.py*: Population conservation

For our SIR problem, we will let set the population to $P = 100$ individuals. We will study the epidemic over the time interval $0 \leq t \leq 7500$. And we will use an initial condition of `[99, 1, 0]`, that is, initially there is just one infected person, and the rest are susceptible. Even though we are solving over a long time period, we can try a fairly small number of time steps, $n = 50$.

We also need to specify the values of the parameters α, β, γ. We choose $\alpha = 0.0075, \beta = 0.0050, \gamma = 0.0010$.

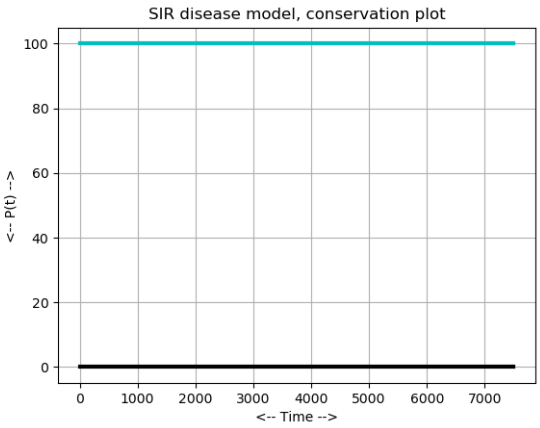
We evaluate the right hand side in a function *sir_dydt()*:

```
def sir_dydt ( t, y ):

  import numpy as np

  S = y[0]
  I = y[1]
  R = y[2]
  P = S + I + R

  alpha = 0.0075
  beta =  0.0050
  gamma = 0.0010

  dSdt = - alpha * S * I / P            + gamma * R
  dIdt =   alpha * S * I / P - beta * I
  dRdt =                       beta * I - gamma * R

  dydt = np.array ( [ dSdt, dIdt, dRdt ] )

  return dydt
```

Now we can call `solve_ivp()` to predict how this epidemic will evolve.

Here, it looks like, after some initial fluctuations, the epidemic becomes "endemic", that is, it seems to stay at a constant level, a persistent danger.

As far as conservation goes, we can check the value of the conserved quantity, and it looks excellent:



4 *sphere_ode*: Distance conservation

The "sphere" differential equation system describes changes in the position (p, q, r) of an object which is moving along a closed loop on the surface of the unit sphere:

$$a = 1.6, b = 1, c = \frac{2}{3}$$
$$\frac{\partial p}{\partial t} = (\frac{1}{c} - \frac{1}{b})rq$$
$$\frac{\partial q}{\partial t} = (\frac{1}{a} - \frac{1}{c})pr$$
$$\frac{\partial r}{\partial t} = (\frac{1}{b} - \frac{1}{a})qp$$

We can imagine that a friend is driving a car on some closed circuit over the face of the earth. We ignore the existence of oceans, assume the earth is perfectly spherical, and that we are using units of measurement so that the earth has a radius of 1.

Our right hand sidew function will be:

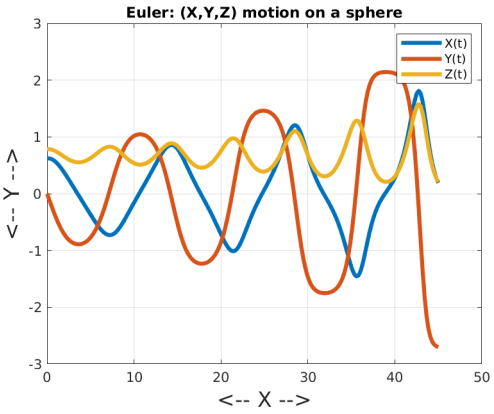
```
def sphere_dydt ( t, y ):
  import numpy as np

  a = 1.6
  b = 1.0
  c = 2.0 / 3.0

  p = y[0]
  q = y[1]
  r = y[2]

  dpdt = ( 1.0 / c - 1.0 / b ) * r * q
  dqdt = ( 1.0 / a - 1.0 / c ) * p * r
  drdt = ( 1.0 / b - 1.0 / a ) * q * p

  dydt = np.array ( [ dpdt, dqdt, drdt ] )

  return dydt
```

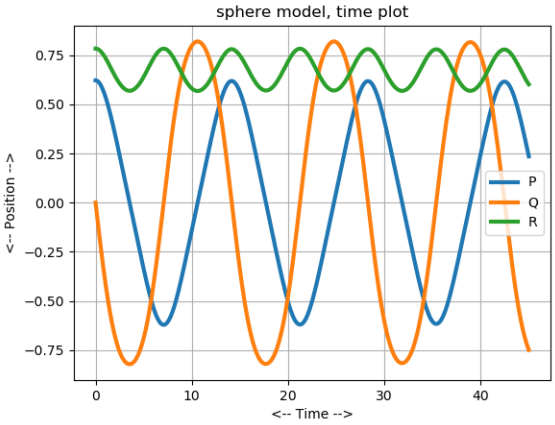
Because our position $(p(t), q(t), r(t))$ is always on the surface of the sphere, it must be the case that we conserve a quantity we can call $h(t)$, which we recognize as simply the square of our distance from the center of the sphere:

$$h(t) = p(t)^2 + q(t)^2 + r(t)^2 = 1 \text{ for all time } t$$

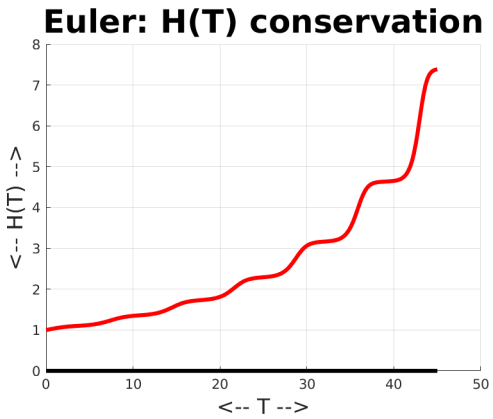
Suppose that at time $t0 = 0$ hours, we start our travel at $(p(0), q(0), r(0)) = (\cos(0.9), 0.0, \sin(0.9))$, and that we have enough fuel to travel until $tstop = 45$ hours.

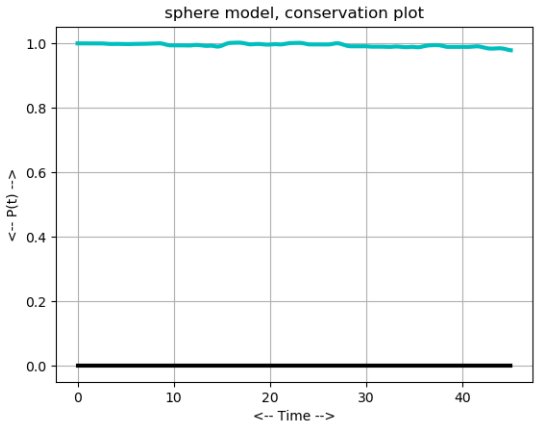
Let's consider approximating our itinerary by using $n = 200$ steps. If we use the Euler method in the usual way, we can plot the values of $p(t), q(t), r(t)$ over time, and we can compare with results from `solve_ivp()`. The Euler graphs look reasonably smooth and regular until we glance at the `solve_ivp()` results, which in fact show periodic behavior.





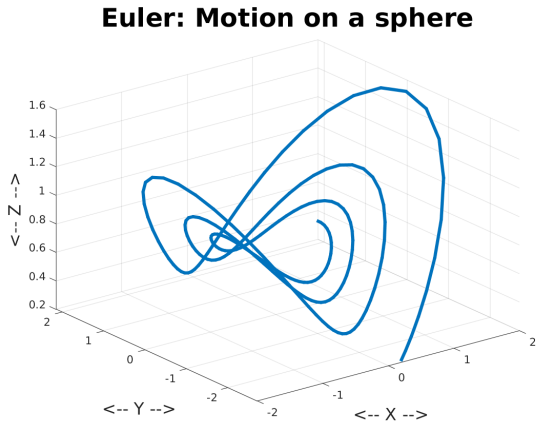
So let's compute the value of $h(t)$ over time and plot that. Remember that this quantity represents our distance from the center of the Earth, and it's rather important that this number always be equal to 1!

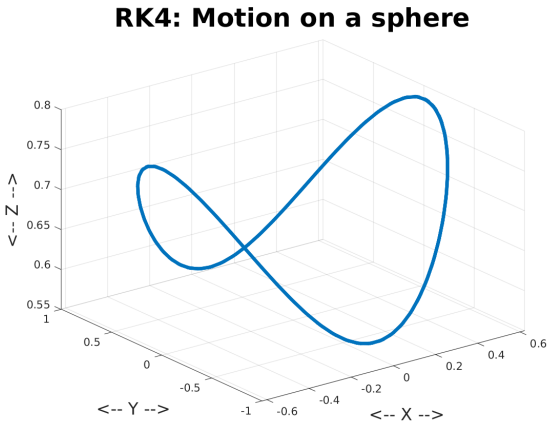




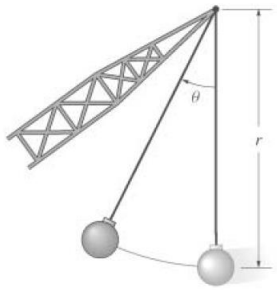
This plot reveals that our Euler solution is worthless. After 45 hours of driving around on the surface of the Earth, Euler thinks we are actually about 7 earth radii out in space!

As we know, when our ODE solution results are unsatisfactory, one thing to try is to take a smaller stepsize. An alternative is to seek a more accurate solver. Here, if we compare 3D plots of the sphere solution for the Euler method and the `solve_ivp()` solver, we see how the Euler solution is spinning off into space, while the other solution simply cycles around the same path, and stays on the Earth.





5 The pendulum



The deflection angle of a pendulum satisfies a second-degree ODE.

The pendulum problem models the back and forth swinging motion of a pendulum. The pendulum has length $l = 1$meter and mass $m = 1$kilogram, and is subject to the force of gravity whose strength is $g = 9.8$meters/second/second. The pendulum position at any time is reported by the value of $\theta(t)$, the angle that the pendulum makes with the downward direction, and can be written as

$$m\,l^2\,\frac{d^2\theta}{dt^2} = -m\,g\,l\,\sin(\theta)$$

We replace θ by the vector y: $y_0 = \theta, y_1 = \frac{d\theta}{dt}$, Assuming that θ is small, we can use the approximation $\sin(\theta) \approx \theta$ to simplify the right hand side, so that we are now working on the *linearized pendulum problem.* We will also find it convenient to write $\omega = \sqrt{\frac{g}{l}}$. Along with some algebraic simplifications, we arrive at the following pair of linear ODE's for our pendulum problem:

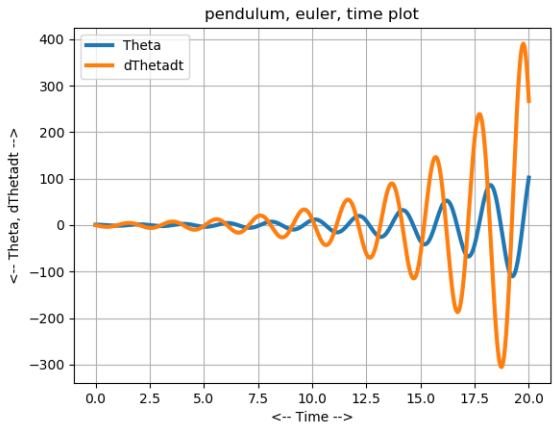
$$y_1' = y_2$$
$$y_2' = -\sqrt{\frac{g}{l}}\,y_1$$

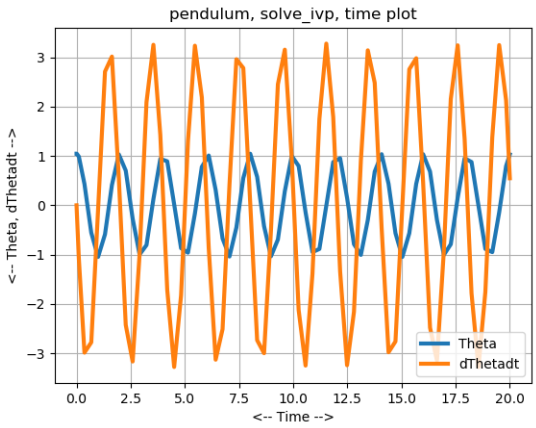
This pair of differential equations would be defined in a file *pendulum_dydt.py.*

We will use the following initial conditions:

$$y_1(0) = \frac{\pi}{3}$$
$$y_2(0) = 0$$

and we approximate the solution over the interval $0 \leq t \leq 10$. Computing the solution with both `euler_system()` and `solve_ivp()`, we are able to suspect that something is wrong with the Euler estimate:

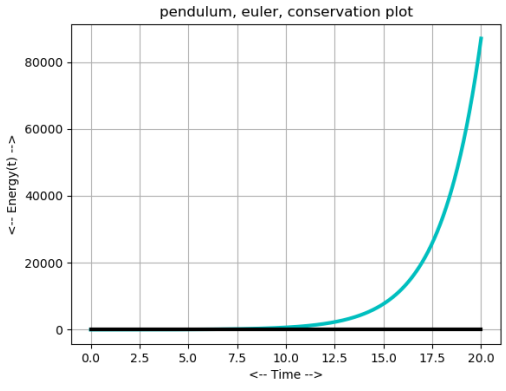


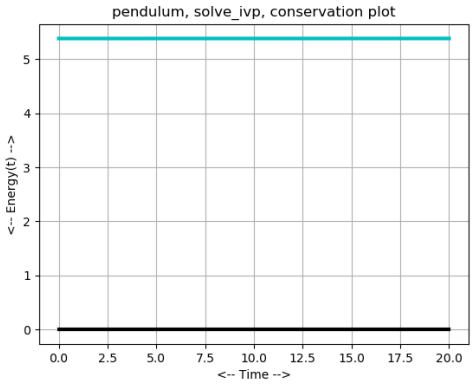


Because the pendulum is a physical system, at every time t it has an energy $E(t)$, which is the sum of its potential energy $m\,g\,h$ and kinetic energy $\frac{1}{2}mv^2$. The height h is $l\,(1-\cos\theta)$. Using a half angle formula, we see that $1-\cos(\theta) = 2\sin^2(\theta/2) \approx \frac{\theta}{2}$, and so we can write:

$$E(t) = \frac{1}{2}\frac{m\,g}{l}\,y_1^2(t) + \frac{1}{2}m\,y_2^2(t)$$

In the absence of friction or other disturbances, this energy should be constant. Since we know the solution at the starting time t_0, this means that the energy $E(t)$ at any time should equal that initial value $E(t_0)$. So now, rather than comparing the solutions directly, let us simply focus on the energy conservation. We try the Euler code with $n = 100, 500, 2000, 5,000, 10,000$. The first computations are a disaster. We compare the Euler result for $n = 10,000$ with *pendulum_solve_ivp()*.





If we did not have access to `solve_ivp()`, we might be excuse for thinking that the pendulum problem is unsolvable, or too difficult to get accurate answers!

6 A predator-prey conservation law

Recall the predator prey ODE: at any time t, we have populations of $u(t)$ prey and $v(t)$ predators, which might be rabbits and foxes. The ODEs that model the changes in population are:

$$\frac{du}{dt} = \alpha u - \beta uv$$
$$\frac{dv}{dt} = -\gamma v + \delta uv$$

where $\alpha, \beta, \gamma, \delta$ are positive parameters.

This system has a conserved quantity:

$$h(t) = \delta(u) - \gamma \ln(u) + \beta v - \alpha \ln(v)$$

In general, when $h(t)$ is a linear function (as in the SIR case), or quadratic (as in the pendulum problem), a good ODE solver can often produce almost exact conservation. However, because this conserved quantity includes logarithmic terms, we generally cannot predict perfect results. Here is how we compute $h(t)$:

```
def predator_prey_conserved ( t, y ):

  import numpy as np

  alpha = 2.0
  beta = 0.001
  gamma = 10.0
  delta = 0.002

  r = y[:,0]
  f = y[:,1]

  h = delta * r - gamma * np.log ( r ) + beta * f - alpha * np.log ( f )

  return h
```

We compare the conservation values for the Euler and `solve|ivp()` methods, we again see poor results for Euler, while `solve_ivp()` has automatically chosen stepsizes that result in near perfect conservation: