Numerical solutions of systems of differential equations (Chapter 7.4); Boundary-value problems (Chapter 11.1)

Last week we had studied the initial values problems for a single first order ordinary differential equation in the form

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big),\ x(t_0) = x_0,$$

where f is a scalar function of t and x.

Now we extend it to a system of n first-order differential equations:

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \begin{bmatrix} \frac{d\,x_1(t)}{dt} \\ \frac{d\,x_2(t)}{dt} \\ \vdots \\ \frac{d\,x_n(t)}{dt} \end{bmatrix} = \begin{bmatrix} f_1\big(t, x_1(t), x_2(t), \dots, x_n(t)\big) \\ f_2\big(t, x_1(t), x_2(t), \dots, x_n(t)\big) \\ \vdots \\ f_n\big(t, x_1(t), x_2(t), \dots, x_n(t)\big) \end{bmatrix} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big).$$

Euler's method

Consider the initial values problem

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big), \;\; \boldsymbol{x}(a) = \boldsymbol{x}_a, \;\; t \in [a, b].$$

The first two terms in the Taylor series of x_i is

$$x_i(t+h) \approx x_i(t) + hx_i'(t) = x_i(t) + hf_i\big(t, \boldsymbol{x}(t)\big).$$

We can estimate $\boldsymbol{x}(t+h)$ by the form

$$\boldsymbol{x}(t+h) \approx \boldsymbol{x}(t) + h\boldsymbol{x}'(t) = \boldsymbol{x}(t) + h\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big).$$

Euler's method can be easily rewritten by a simple change of a scalar into a vector.

Euler's method

Now split the interval $[a, b]$ into n subintervals with size $h = (b - a)/n$. Euler's method starts from $\boldsymbol{x}(a) = \boldsymbol{x}_a$ and after n steps

$$\boldsymbol{x}(a) \to \boldsymbol{x}(a + h) \to \boldsymbol{x}(a + 2h) \to \cdots \to \boldsymbol{x}(a + nh) = \boldsymbol{x}(b)$$

end with an approximation of $\boldsymbol{x}(b)$.

- Euler's method is still not good, but it shows how to extend other methods to solve ODE systems.

Warning

An important issue that we will **NOT** discuss in this course is how to weight different differential equations in a system.

Warning

An important issue that we will **NOT** discuss in this course is how to weight different differential equations in a system.

For example:

A differential equation in coulomb unit is typical with values around 10^{-19}, but an equation on relativistic speed is typical with values around 10^{8}.

Warning

An important issue that we will **NOT** discuss in this course is how to weight different differential equations in a system.

For example:

A differential equation in coulomb unit is typical with values around 10^{-19}, but an equation on relativistic speed is typical with values around 10^{8}.

Often we use normalization technique to deal with this issue.

Taylor series methods

In the vector form, Taylor expension is on a vector function with a single variable:

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + h\boldsymbol{x}'(t) + \frac{h^2}{2!}\boldsymbol{x}''(t) + \frac{h^3}{3!}\boldsymbol{x}^{(3)}(t) + \dots .$$

Since h is one-dimensional, it is the same as the case of $n = 1$. But when we use the chain rule to calculate the derivative of $\frac{d\,\boldsymbol{x}(t)}{dt}$, which equals to $\boldsymbol{f}\big(t, \boldsymbol{x}(t)\big)$, we will have

$$x_i''(t) = \frac{d\, f_i\big(t, \boldsymbol{x}(t)\big)}{dt} = \frac{\partial f_i}{\partial t}(t, \boldsymbol{x}) + \sum_{j=1}^{n} \frac{\partial f_i}{\partial x_j}(t, \boldsymbol{x}) x_j' = \dots .$$

After input all derivatives, Taylor series methods can be implemented as:

```
dx=f(t,x)                                % given by user
d2x=dfdt(t,x,dx)                         % given by user
x=x+h*dx                       %Euler's method; first order
x=x+h*(dx+1/2*h*(d2x))                   %       second order
```

Taylor series methods

When we use Taylor series method of order k, the terms up to order k in Taylor series of $\boldsymbol{x}(t)$ are included.

Hence, the local error is still $\mathcal{O}(1/n^{k+1})$ for each iteration, and the global error is $\mathcal{O}(1/n^k)$.

A 4th-order Runge-Kutta method in scalar form

For a differential equation

$$\frac{d\,x(t)}{dt} = f\big(t, x(t)\big),$$

we have a 4th-order Runge-Kutta formulas as

$$x(t+h) = x(t) + \frac{h}{6}\Big(K_1 + 2K_2 + 2K_3 + K_4\Big)$$

where

$$\begin{aligned} K_1 &= f(t, x) \\ K_2 &= f(t + {}^{h}\!/_{2}, x + {}^{h}\!/_{2} K_1) \\ K_3 &= f(t + {}^{h}\!/_{2}, x + {}^{h}\!/_{2} K_2) \\ K_4 &= f(t + h, x + hK_3). \end{aligned}$$

A 4th-order Runge-Kutta method in vector form

For a system of several differential equations

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big),$$

we have a 4th-order Runge-Kutta formulas for x_i as:

$$x_i(t+h) = x_i(t) + \frac{h}{6}\Big(K_{1,i} + 2K_{2,i} + 2K_{3,i} + K_{4,i}\Big)$$

where the only change comparing with the single equation case is on $K_{1,i}, K_{2,i}, K_{3,i}, K_{4,i}$, and they are calculated from the ith component of $\boldsymbol{f}$

$$\begin{aligned}
K_{1,i} &= f_i(t, \boldsymbol{x}) \\
K_{2,i} &= f_i(t + {}^h\!/_2, \text{'need all elements in } \boldsymbol{x} + {}^h\!/_2\boldsymbol{K}_1\text{'}) \\
K_{3,i} &= f_i(t + {}^h\!/_2, \text{'need all elements in } \boldsymbol{x} + {}^h\!/_2\boldsymbol{K}_2\text{'}) \\
K_{4,i} &= f_i(t + h, \text{'need all elements in } \boldsymbol{x} + h\boldsymbol{K}_3\text{'}).
\end{aligned}$$

A 4th-order Runge-Kutta method in vector form

For a system of several differential equations

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big),$$

we also can write the 4th-order Runge-Kutta formulas in a vector form:

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + \frac{h}{6}\Big(\boldsymbol{K}_1 + 2\boldsymbol{K}_2 + 2\boldsymbol{K}_3 + \boldsymbol{K}_4\Big),$$

where $\boldsymbol{K}_1, \boldsymbol{K}_2, \boldsymbol{K}_3, \boldsymbol{K}_4$ are defined as

$$\begin{aligned}
\boldsymbol{K}_1 &= \boldsymbol{f}(t, \boldsymbol{x}) \\
\boldsymbol{K}_2 &= \boldsymbol{f}(t + h/2, \boldsymbol{x} + h/2\boldsymbol{K}_1) \\
\boldsymbol{K}_3 &= \boldsymbol{f}(t + h/2, \boldsymbol{x} + h/2\boldsymbol{K}_2) \\
\boldsymbol{K}_4 &= \boldsymbol{f}(t + h, \boldsymbol{x} + h\boldsymbol{K}_3).
\end{aligned}$$

A 4th-order Runge-Kutta method in vector form

The derivation of the Runge-Kutta method of order 4 follows that the solution at $\boldsymbol{x}(t+h)$ agrees with the Taylor expansion up to and including the term of h^4. Hence, the local error is $\mathcal{O}\big(1/n^5\big)$.

Therefore, the global error is still $\mathcal{O}\big(1/n^4\big)$.

All methods in vector form are exact same as in scalar form.

Example

Consider a system of differential equations

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \begin{bmatrix} \frac{d\,x_1(t)}{dt} \\ \frac{d\,x_2(t)}{dt} \end{bmatrix} = \begin{bmatrix} f_1\big(t, x_1(t), x_2(t)\big) \\ f_2\big(t, x_1(t), x_2(t)\big) \end{bmatrix} = \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix} = \boldsymbol{f}\big(t, \boldsymbol{x}(t)\big).$$

and the initial condition $\boldsymbol{x}(0) = \big(x_1(0), x_2(0)\big) = (1, 0)$. The solution is

$$\boldsymbol{x}(t) = \big(\cos(t), \sin(t)\big).$$

More generally, $\boldsymbol{x}(t) = \big(a\cos(t), a\sin(t)\big)$ is the solution with the initial condition $\boldsymbol{x}(0) = (a, 0)$, and $\big(x_1(t), x_2(t)\big)$ with $t \in [0, 2\pi]$ are circles.

Euler's method is implemented in `MyEulerSystem.py` and the test problem is in `GrovEulerSystem.py`

Run the same test problem in `GrovEulerRK4System.py`. it clearly shows that the 4th-order Runge-Kutta method provides higher accuracy with the same amount of function evaluations.

Higher-order differential equation → system of first order

We can write $\frac{d^n x}{dt^n} = f\left(t, x, \frac{dx}{dt}, \ldots, \frac{d^{(n-1)}x}{dt^{(n-1)}}\right)$ as a system of n first-order differential equations by defining a vector:

$$\boldsymbol{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} x \\ \frac{dx}{dt} \\ \vdots \\ \frac{d^{(n-1)}x}{dt^{(n-1)}} \end{bmatrix}$$

$$\frac{d\boldsymbol{z}}{dt} = \begin{bmatrix} \frac{dz_1}{dt} \\ \frac{dz_2}{dt} \\ \vdots \\ \frac{dz_{n-1}}{dt} \\ \frac{dz_n}{dt} \end{bmatrix} = \begin{bmatrix} z_2 \\ z_3 \\ \vdots \\ z_n \\ f\left(t, z_1, z_2, \ldots, z_n\right) \end{bmatrix} = \boldsymbol{f}(t, \boldsymbol{z})$$

Higher-order differential equation $\rightarrow$ system of first order

Numerical solvers are **only** for systems of first-order differential equations.

A single differential equation of order n,

$$\frac{d^n x}{dt^n} = f\left(t, x, \frac{d\,x}{dt}, \ldots, \frac{d^{(n-1)}x}{dt^{(n-1)}}\right),$$

can be turned into a system of n first-order equations of the form

$$\frac{d\boldsymbol{z}(t)}{dt} = \boldsymbol{f}\left(t, \boldsymbol{z}(t)\right).$$

It requires n auxiliary conditions (e.g. initial conditions) in order to specify the solution precisely.

Boundary-value problems

For a system of differential equations (of first order)

$$\frac{d\,\boldsymbol{x}(t)}{dt} = \boldsymbol{f}\left(t, \boldsymbol{x}(t)\right),$$

we call it as

Initial-value problem:

- if all conditions are at the initial instant t_0; find $\boldsymbol{x}(t)$ for $t \geq t_0$.

Boundary-value problem:

- if conditions are at t_0 and t_s; find $\boldsymbol{x}(t)$ for $t_0 \leq t \leq t_s$.

Comparing with initial-value problems, the main difference is that boundary-value problems can have 0, 1, 2, more or even infinite different solutions.

A boundary-value problem

$$\frac{d^2x}{dt^2} = -x,\ x(0) = 1,\ x(\pi/2) = -3$$

The general solution of the differential equation is

$$x(t) = c_1 \sin(t) + c_2 \cos(t),\ t, c_1, c_2 \in \mathbb{R}.$$

Boundary conditions are

$$\begin{aligned} x(0) =& c_2 = 1 \\ x\left(\pi/2\right) =& c_1 = -3, \end{aligned}$$

Then, a unique solution is determined

$$x(t) = -3\sin(t) + \cos(t).$$

Shooting balls without air resistance

A cannon shoots from the origin, i.e. $\big(x(0), y(0)\big) = (0, 0)$, with initial speed $\big(x'(0), y'(0)\big) = (1, z)$.

It hits the point $(x, y) = (10, 0)$!

Based on the Newton's second law and a constant gravity g, we have

$$\frac{d^2 x}{dt^2} = 0, \qquad \frac{d^2 y}{dt^2} = -g.$$

These are two uncoupled differential equations. Let's start with the first one.

We have two conditions on x at $t = 0$. It is an **initial-value problem**, and has the solution $x(t) = t$.

To find the arrive time, we need find **a root of an equation**

$$\text{find } t \text{ such that } x(t) = t = 10.$$

Shooting balls without air resistance

For y, now we have a 2nd-order differential equation

$$\frac{d^2 y}{dt^2} = -g.$$

and the condition $y(0) = y(10) = 0$. This is a **boundary-value problem**. The general solution of the equation is

$$y(t) = -\frac{1}{2}gt^2 + at + b.$$

Boundary condition shows that

$$y(0) = b = 0$$
$$y(10) = -\frac{1}{2}g10^2 + a10.$$

So we obtain that $y'(0) = z = a = 5g$ and we also can get the parabola (recall that $x(t) = t$)

$$y(t) = -\frac{1}{2}gt^2 + 5gt = \frac{g}{2}t(10 - t) = \frac{g}{2}x(10 - x).$$

Shooting balls → Shooting method

If we are unable to determine the general solution, how can we get z?

Shooting balls $\rightarrow$ Shooting method

If we are unable to determine the general solution, how can we get z?
We can numerically solve the initial-value problem

$$\frac{d^2 y}{dt^2} = -g,\ y(0) = 0,\ \text{and}\ y'(0) = z,$$

for 'all' possible value of z and pick up the one that agrees with $y(10) = 0$.

Shooting balls $\rightarrow$ Shooting method

If we are unable to determine the general solution, how can we get z?
We can numerically solve the initial-value problem

$$\frac{d^2 y}{dt^2} = -g, \; y(0) = 0, \text{ and } y'(0) = z,$$

for 'all' possible value of z and pick up the one that agrees with $y(10) = 0$.

Or we can do it smarter:
The value of $y(10)$ is written as a function of z, which we call as $\varphi(z)$, and we try to find a solution such that

$$0 = \varphi(z) - y(10) = \varphi(z) - 0.$$

We need find a root of an equation!

Shooting balls $\rightarrow$ Shooting method

If we are unable to determine the general solution, how can we get z?
We can numerically solve the initial-value problem

$$\frac{d^2 y}{dt^2} = -g, \ y(0) = 0, \ \text{and} \ y'(0) = z,$$

for 'all' possible value of z and pick up the one that agrees with $y(10) = 0$.

Or we can do it smarter:
The value of $y(10)$ is written as a function of z, which we call as $\varphi(z)$, and we try to find a solution such that

$$0 = \varphi(z) - y(10) = \varphi(z) - 0.$$

We need find a root of an equation! Secant method can be applied here!

Shooting balls $\rightarrow$ Shooting method

If we are unable to determine the general solution, how can we get z?
We can numerically solve the initial-value problem

$$\frac{d^2 y}{dt^2} = -g,\ y(0) = 0,\ \text{and}\ y'(0) = z,$$

for 'all' possible value of z and pick up the one that agrees with $y(10) = 0$.

Or we can do it smarter:
The value of $y(10)$ is written as a function of z, which we call as $\varphi(z)$, and we try to find a solution such that

$$0 = \varphi(z) - y(10) = \varphi(z) - 0.$$

We need find a root of an equation! Secant method can be applied here!

NB: We know very little about $\varphi(z)$, and each value of $\varphi(z)$ is obtained by solving an initial-value problem!

Boundary-value problems with shooting method

For a boundary-value problem

$$\frac{d^2 x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x(b) = x_b$$

we implement a function that with a given z solves an initial-value problem

$$\frac{d^2 x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x'(a) = z,$$

and gives us the solution $x_z(t)$.

Let $\varphi(z) = x_z(b)$. By adjusting z we want to find a value $\bar{z}$ such that

$$\varphi(\bar{z}) - x(b) = 0.$$

Boundary-value problems with shooting method

For a boundary-value problem

$$\frac{d^2 x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x(b) = x_b$$

we implement a function that with a given z solves an initial-value problem

$$\frac{d^2 x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x'(a) = z,$$

and gives us the solution $x_z(t)$.

Let $\varphi(z) = x_z(b)$. By adjusting z we want to find a value $\bar{z}$ such that

$$\varphi(\bar{z}) - x(b) = 0.$$

If the differential equation is **linear**, we can show that $\varphi(z)$ is **linear** on z.

Boundary-value problems with shooting method

For a boundary-value problem

$$\frac{d^2x}{dt^2} = f(t, x, x'), \; x(a) = x_a, \; x(b) = x_b$$

we implement a function that with a given z solves an initial-value problem

$$\frac{d^2x}{dt^2} = f(t, x, x'), \; x(a) = x_a, \; x'(a) = z,$$

and gives us the solution $x_z(t)$.

Let $\varphi(z) = x_z(b)$. By adjusting z we want to find a value $\bar{z}$ such that

$$\varphi(\bar{z}) - x(b) = 0.$$

If the differential equation is **linear**, we can show that $\varphi(z)$ is **linear** on z.

Then, secant method only need one iteration. (-:

Boundary-value problems with shooting method

For a boundary-value problem

$$\frac{d^2x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x(b) = x_b$$

we implement a function that with a given z solves an initial-value problem

$$\frac{d^2x}{dt^2} = f(t, x, x'),\ x(a) = x_a,\ x'(a) = z,$$

and gives us the solution $x_z(t)$.

Let $\varphi(z) = x_z(b)$. By adjusting z we want to find a value $\bar{z}$ such that

$$\varphi(\bar{z}) - x(b) = 0.$$

Otherwise, we need several iterations. (Here, in fact we assume that $\varphi(z)$ is continuous and differentiable.)

Calculation involving the solution of differential equation

- In the example of shooting balls, we need solve the equation $x(t) = 10$.
- We might also need to calculate when we have

$$x(t) = y(t),$$

  where x and y are solutions of systems of differential equations.
- Or any other calculations involving the solutions of the differential equations.

- If the solutions x, y are obtained numerically, what would be the difficulty?
- What can we do?

Interpolation of discrete function values

Assume that by solving a differential equation we obtain a table of t and x values

t	t_0	t_1	$\dots$	t_n
x	x_0	x_1	$\dots$	x_n

With a given t we can calculate an approximation of $x(t)$ as follows:

- Find k such that $t_{k-1} \leq t < t_k$.
- Utilize $[t_{k-3}\ t_{k-2}\ t_{k-1}\ t_k\ t_{k+1}\ t_{k+2}]$ and $[x_{k-3}\ x_{i-k}\ x_{i-k}\ x_k\ x_{k+1}\ x_{k+2}]$ to estimate $x(t)$ by a *higher-order interpolation*.

We can use Numpy's build-in function `interp` to do this.

```
sol = solve_ivp(MinDiff, [0, 20], [0,0])
# interpolation of 3. component (NB 0-indexing)
f = lambda t: np.interp(t, sol.t, sol.y[2,:])
# print 3. component of the solution evaluated at t=7
print(f(7))
```

Summary

All differential equations can be written as a system of the first-order differential equations.

All numerical methods for solving systems of the first-order differential equations are the same as the ones for single equation, but in vector form.

In initial-value problems, we need n auxiliary conditions for a system of n first-order equations in order to determine a unique solution.

Boundary-value problems can be solved by shooting method, which include solving an initial-value problem, finding a root of an equation, and optionally using interpolation.

In next week it will be 4-hour exercise. Homework assignment need be handed in before **Nov. 14 10pm**!