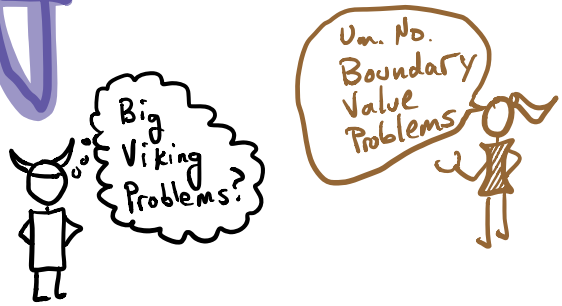
Solving BVPs

in Python



BVP

Boundary value problems are a kind of Differential equation where we know properties (value, derivative, etc) at different places.

In contrast, for an initial value problem we know all the properties at the same place.

IVP

$$u_1' = f(x, u_1, u_2)$$
$$u_2' = g(x, u_1, u_2)$$
$$u_1(x_0) = \alpha$$
$$u_2(x_0) = \beta$$

↑ same place

BVP

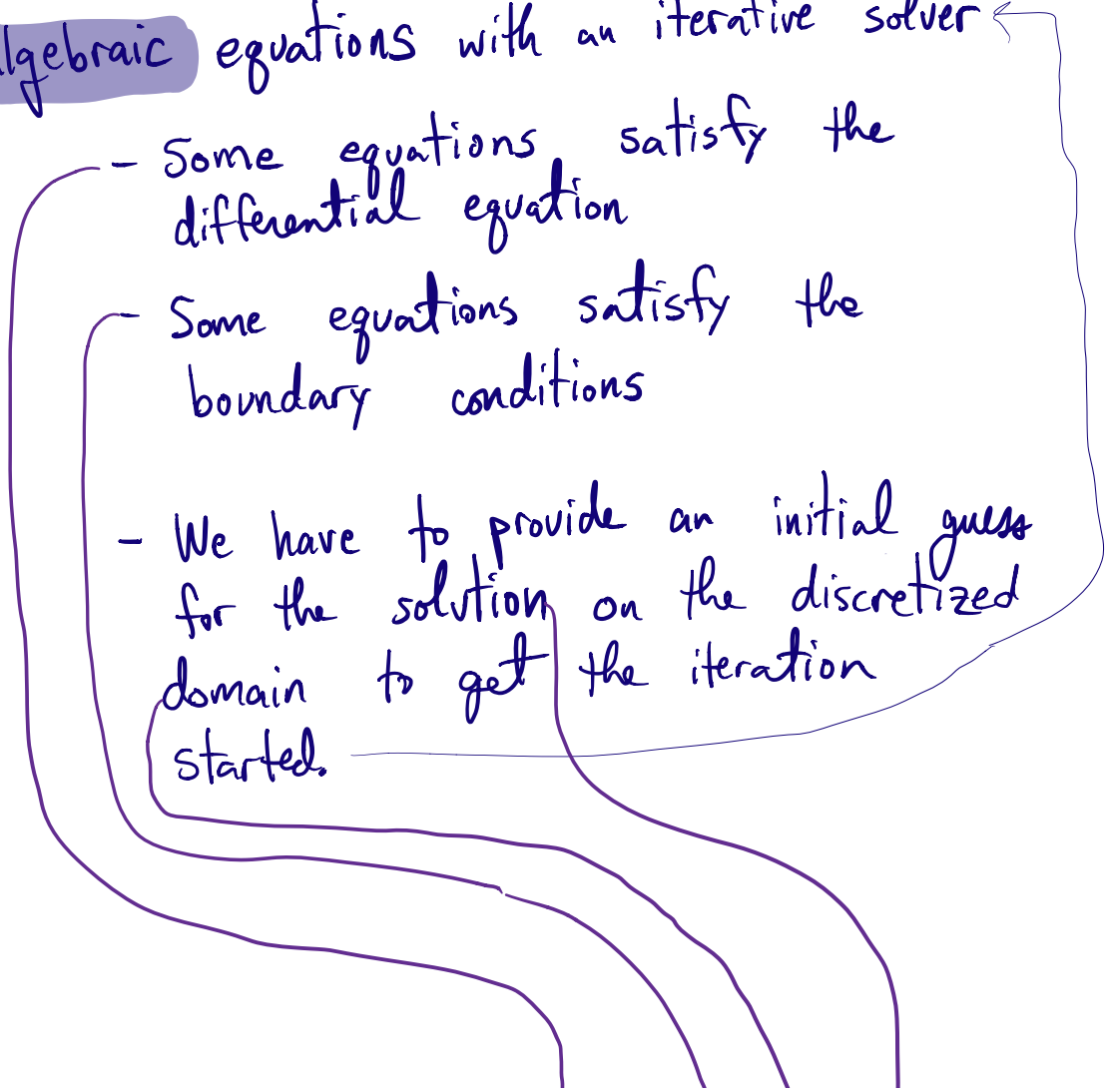
$$u_1' = f(x, u_1, u_2)$$
$$u_2' = g(x, u_1, u_2)$$
$$u_1(x_0) = \alpha$$
$$u_2(x_1) = \beta$$

↑ two different places

With an IVP, we know everything we need at one place, so we can integrate forward.

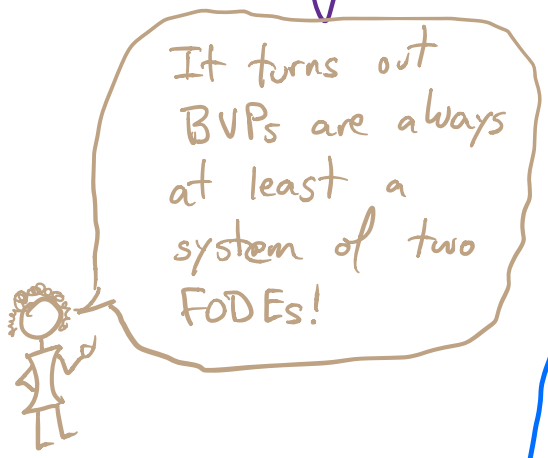
With a BVP, we discretize the domain and solve a set of approximate algebraic equations with an iterative solver

- Some equations satisfy the differential equation
- Some equations satisfy the boundary conditions
- We have to provide an initial guess for the solution on the discretized domain to get the iteration started.



sol = solve_bvp(fun, bc, X, Y)

↳ fun(X, Y) is a function defining the system of ODEs



X is an array of points you want the solution at.

Y is an array of solutions at each point in X.

fun(X, Y) returns an array of derivatives of Y at each X-point.

$$\underline{Y}' = \text{fun}(X, \underline{Y}) = \begin{bmatrix} \frac{dy_1}{dx} \\ \frac{dy_2}{dx} \\ \vdots \end{bmatrix}$$

→ The boundary conditions are represented by a function that will return zero for each condition at the solution.

$Z = bc(Y_A, Y_B)$

Y_A = the value of the solution at X[0]

Y_B = the value of the solution at X[-1]

→ These will both be arrays, one element for each solution component

→ bc returns an array of floats that will be zero at the solution.

e.g.

$u_1(0) = 4$

$u_1(1) = 5$

$\Rightarrow$

```
def bc(Ya, Yb):
    Y1a, Y2a = Ya
    Y1b, Y2b = Yb
    return [Y1a - 4,
            Y1b - 5]
```

these are zero at the solution

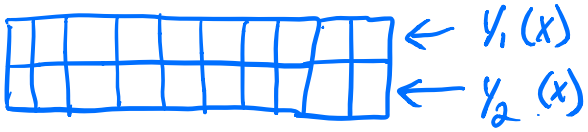
$u_1(0) = 4$

$u_1(1) = 5$

→ X is an array, usually from linspace, although it can be any increasing array of points.

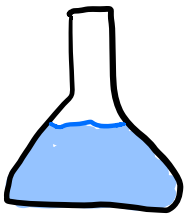
The finer (more closely spaced) the points, the more accurate the solution will be.

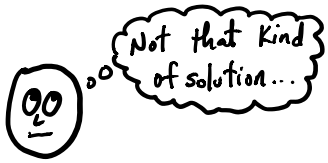
→ Y This is a guess for the solution at each x point. If we have 2 equations, and an x-array with 10 points, Y will have a shape of 2 x 10



The guess can be important; sometimes a bad guess will not converge.

The solution:





sol = solve_bvp(fun, bc, x, y)

↳ this is a data structure

sol.x has the discretized domain

sol.y has the solution at each point for each component

You should also check sol.message to ensure it succeeded.

You can evaluate the solution with:

s = sol.sol(newX)

this is an interpolated solution