
1



# Introduction

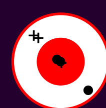
- In this topic, we will
  - Describe the shooting method
  - Explain how to convert a BVP into an IVP
  - Discuss the appropriate initial slopes to use for the IVPs
  - See how to solve such problems with our initial-value problem solver dp45
  - Explain how we can use the secant method to better approximation of the initial slope
  - Look at the special case where the ODE is linear



2

# The shooting method

- The approach we will use is commonly called the *shooting method*
  - Suppose you are aiming at a target
  - Unless you're firing a laser, the projectile follows a path affected by gravity, wind, air resistance, tumbling, imperfections, temperature, and the Coriolis effect



3

3



# Boundary-value problems

- Suppose we have a boundary-value problem:

$$u^{(2)}(x) = f\left(x, u(x), u^{(1)}(x)\right)$$
$$u(a) = u_a$$
$$u(b) = u_b$$

  - The solution must have a slope at $x = a$: $u^{(1)}(a) = u_a^{(1)}$
    - We don't know what this slope is…
  - Suppose, however, we converted the BVP into an IVP:

$$u^{(2)}(x) = f\left(x, u(x), u^{(1)}(x)\right)$$
$$u(a) = u_a$$
$$u^{(1)}(a) = s$$

4

4

# Initial-value problems

- This IVP has a solution $u_s(t)$:

$$u^{(2)}(x) = f\left(x, u(x), u^{(1)}(x)\right)$$
$$u(a) = u_a$$
$$u^{(1)}(a) = s$$

  - This solution has a value at $b$, but it is almost certain $u_s(b) \neq u_b$

- We would like to find that slope $s_b$ such that the solution to

$$u^{(2)}(x) = f\left(x, u(x), u^{(1)}(x)\right)$$
$$u(a) = u_a$$
$$u^{(1)}(a) = s_b$$

  also satisfies $u_{s_b}(b) = u_b$







# Initial-value problems

- Up to this point, we have formulated our questions in terms of a mathematical expression
  - For what value of $x$ does $e^{-2x}\sin(4x) + e^{-3x}\cos(2x) = 0.3$?

- This, too, is a numeric question,
  but one that is much more complex:
  - Find that slope $s$ such that the solution to the IVP

$$u^{(2)}(x) = f\left(x, u(x), u^{(1)}(x)\right)$$
$$u(a) = u_a$$
$$u^{(1)}(a) = s$$

    is such that $u_s(b) = u_b$
  - Given $s$, we must approximate the solution…

## Initial-value problems

- This is an equation in one variable *s* to which we don't know the solution:

$$u_s(b) = u_b$$

  - However, this is oddly enough, just a very complex root-finding problem

$$u_s(b) - u_b = 0$$





The shooting method

## Initial-value problems

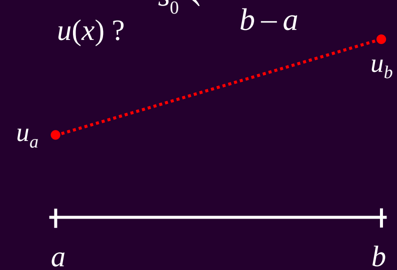
- Thus, given an initial slope *s*, we can calculate $u_s(b) - u_b$
  - If this is zero, we have found the appropriate slope
  - If $u_s(b) - u_b < 0$, $u_s(b) < u_b$, so we should choose a larger slope
  - If $u_s(b) - u_b > 0$, $u_s(b) > u_b$, so we should choose a smaller slope

- How much larger or smaller?
  - Newton's method won't work: we cannot differentiate it…
  - Let's use the secant method!

## First initial slope $s_0$

- The secant method requires two initial values:
  - What should we choose?

$$s_0 \leftarrow \frac{u_b - u_a}{b - a}$$





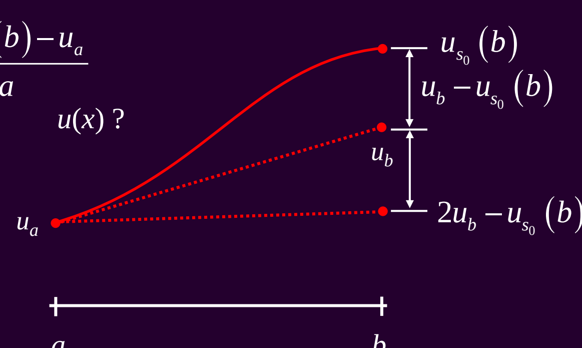





## Second initial slope $s_1$

- The secant method requires two initial values:
  - What should we choose for $s_1$?
  - Approximate to solution for the initial slope $u^{(1)}(a) = s_0$

$$s_1 \leftarrow \frac{2u_b - u_{s_0}(b) - u_a}{b - a}$$



# Subsequent initial slopes

- Now we iterate:
  - Given $s_k$, approximate the value of the solution at $x = b$

$$u_{s_k}(b)$$

  - Recall if we are finding the root of $g(x)$ and we have two approximations $x_{k-1}$ and $x_k$, the next approximation is:

$$x_{k+1} \leftarrow x_k - g(x_k)\frac{x_k - x_{k-1}}{g(x_k) - g(x_{k-1})}$$

  - We are finding the root of $g(s) \overset{\text{def}}{=} u_b - u_s(b)$, so substituting this in, we have

$$s_{k+1} \leftarrow s_k - \left(u_b - u_{s_k}(b)\right)\frac{s_k - s_{k-1}}{\left(u_b - u_{s_k}(b)\right) - \left(u_b - u_{s_{k-1}}(b)\right)}$$

$$= s_k - \frac{\left(u_b - u_{s_k}(b)\right)(s_k - s_{k-1})}{u_{s_{k-1}}(b) - u_{s_k}(b)}$$

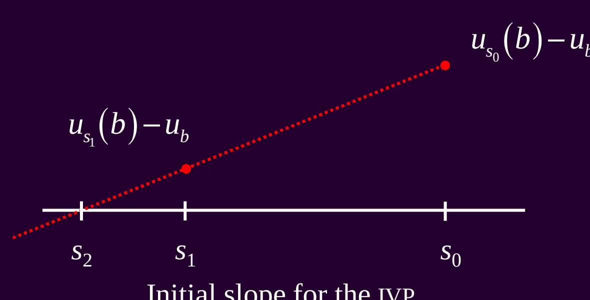
11

11

The shooting method

# Visualization

- Let's look at this visually:

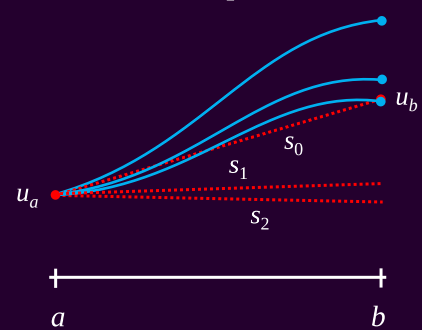



12

12

# Visualization

- Let's look this from another point-of-view:
  - We determine $s_0$ and approximate the solution
  - Based on this, we determine $s_1$ and approximate the solution
  - Based on this, we determine $s_2$ and continue










# Implementation

- How can we do this?

$$u^{(2)}(t) = f\left(x, u(x), u^{(1)}(x)\right) \qquad u^{(2)}(t) = f\left(x, u(x), u^{(1)}(x)\right)$$

$$u(a) = u_a \qquad u(a) = u_a$$

$$u(b) = u_b \qquad u^{(1)}(a) = s$$

$$w_0(x) = u(x)$$

$$w_1(x) = u^{(1)}(x)$$

$$\mathbf{w}(x) = \begin{pmatrix} w_0(x) \\ w_1(x) \end{pmatrix} \quad \mathbf{w}^{(1)}(x) = \begin{pmatrix} w_1(x) \\ f\left(x, \mathbf{w}(x)\right) \end{pmatrix} \quad \mathbf{w}_0 = \begin{pmatrix} u_a \\ s \end{pmatrix}$$

# Implementation

- For our implementation, suppose we have:

$$u^{(2)}(t) = u^{(1)}(x)u(x) + x + 1$$
$$u(0.0) = 1.3$$
$$u(5.0) = 2.9$$

  – Thus, we can define the values:

```
double a{ 0.0 };
double b{ 5.0 };
double u_a{ 1.3 };
double u_b{ 2.9 };
```

  – We also have the function

```
double f( double x, double u, double du ) {
    return du*u + x + 1.0;
}
```







# Implementation

- We, however, need a vector-valued function, so define

```
vector F( double x, vector w ) {
    return vector{ (double[]){
        w(1),
        f( x, w(0), w(1) )
    } };
}
```

- The initial guess for the slope is:

```
double s0{ (u_b - u_a)/(b - a) };
```

# Implementation

- We can now call dp45:

```
auto result{ dp45(
    F, std::make_pair( a, b ),
    vector{ (double[]){ u_a, s0 } },
    std::make_pair( 1e-4, 1e-2 ), 1e-5, true
) };

unsigned int n{ std::get<0>( result ) };
assert( std::get<1>( result )[n] == b );
double u0b{ std::get<2>( result )[n](0) };

double s1{ (2.0*u_b - u0b - u_a)/(b - a) };
```







# Implementation

- We can now call dp45 again:

```
result = dp45(
    F, std::make_pair( a, b ),
    vector{ (double[]){ u_a, s1 } },
    std::make_pair( 1e-4, 1e-2 ), 1e-5, true
);

n = std::get<0>( result );
assert( std::get<1>( result )[n] == b );
double u1b{ std::get<2>( result )[n](0) };

double s2{ s1 - (u_b - u1b)*(s1 - s0)/(u0b - u1b) };
```

## Implementation

```
// Determine and assign s0 and s1

for ( unsigned int k{0}; k < max_iterations; ++k ) {
    result = dp45(
        F, std::make_pair( a, b ),
        vector{ (double[]){ u_a, s1 } },
        std::make_pair( 1e-4, 1e-2 ), 1e-5, true
    );

    n = std::get<0>( result );
    u1b = std::get<2>( result )[n](0);

    if ( std::abs( u1b - u_b ) < eps_abs ) {
        // we are done: return or use the current solution
    }

    double s2{ s1 - (u_b - u1b)*(s1 - s0)/(u0b - u1b) };
    s0 = s1;
    s1 = s2;
}
```



19



## Humor

- The Wikipedia page says to use either the bisection method or Newton's method
  - The bisection method only works if we've bounded the root
  - Newton's method only works if can calculate the derivative…



20

## Linear ordinary differential equations

- One nice result:
  - If the ODE is linear, we are guaranteed we only have to perform one iteration of the secant method
- One issue:
  - If the ODE is not linear, the solution may not be unique







## Summary

- Following this topic, you now
  - Understand the idea behind the shooting method
  - Know how to convert the BVP into an IVP
  - Know how to determine two initial slopes
  - Understand how to apply the secant method to get a better approximation
  - Are aware that the secant method needs only one iteration if the ordinary differential equation is linear

# References

[1] https://en.wikipedia.org/wiki/Shooting_method





The shooting method

# Acknowledgments

None so far.

# Colophon

These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.
Mathematical equations are prepared in MathType by Design Science, Inc.
Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.







# Disclaimer

These slides are provided for the ECE 204 *Numerical methods* course taught at the University of Waterloo. The material in it reflects the author's best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.