UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 204 *Numerical methods*

Approximating solutions to systems of 1st-order initial-value problems

Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com

Introduction

- In this topic, we will
  - Give a description of systems of 1st-order initial-value problems (IVPs)
  - Discuss their solutions, if they can be found
  - Describe how operations translate to vector-space operations of vector addition and scalar multiplication
  - Look at converting our 1st-order solvers to solving systems of 1st-order IVPs
  - Go over some examples

Systems of initial-value problems

- Suppose we have a system of two 1st-order IVPs

$$y^{(1)}(t) = -y(t) - 2z(t) \qquad y(0) = 1$$
$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$

  – This has a unique solution:

$$y(t) = e^{-t}\left(\cos(2t) - 2\sin(2t)\right)$$
$$z(t) = e^{-t}\left(2\cos(2t) + \sin(2t)\right)$$

- Problem: this system has no known solution:

$$y^{(1)}(t) = -ty(t) - 2z(t) \qquad y(0) = 1$$
$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$

Systems of initial-value problems

- Here is another such problem:

$$x^{(1)}(t) = 0.02x(t) - 0.1x(t)y(t) \qquad x(0) = 5233$$

$$y^{(1)}(t) = -0.04y(t) + 0.02x(t)y(t) \qquad y(0) = 323$$

- Here is another problem that began a revolution in mathematics:

$$x^{(1)}(t) = 10\left(y(t) - x(t)\right) \qquad x(0) = 1$$

$$y^{(1)}(t) = x(t)\left(28 - z(t)\right) - y(t) \qquad y(0) = 1$$

$$z^{(1)}(t) = x(t)y(t) - \frac{8}{3}z(t) \qquad z(0) = 1$$

The Lorenz equations

- This second equation is responsible for the term *the butterfly effect*

Dan Quinn via Wikipedia

Vector-valued functions of a real variable

- We would like to approximate solutions to such systems
  - What approach do we use?

- Let's create a vector-valued function:

$$\mathbf{u}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} = \begin{pmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{pmatrix}$$

  - The derivative is as follows:

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} x^{(1)}(t) \\ y^{(1)}(t) \\ z^{(1)}(t) \end{pmatrix} = \begin{pmatrix} u_1^{(1)}(t) \\ u_2^{(1)}(t) \\ u_3^{(1)}(t) \end{pmatrix}$$

Vector-valued functions of a real variable

- We can define such a function:

$$\mathbf{u}(t) = \begin{pmatrix} t + \sin(t) \\ \cos(t) - 1 \\ t\sin(t) \end{pmatrix}$$

  - In this case, we have

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} 1 + \cos(t) \\ -\sin(t) \\ \sin(t) + t\cos(t) \end{pmatrix}$$

Vector-valued functions of a real variable

- A 3-dimensional vector-valued function of a real variable could represent:
  - The x, y and z coordinates of a drone at time t
  - The voltages at three nodes in a circuit at time t
  - The temperature readings from three sensors at time t

- You could have fifty sensors, resulting in a 50-dimensional vector-valued function of a real variable

- In each case, the derivative gives you the instantaneous rate-of-change of that vector-valued function
  - E.g., how fast the drone is moving north, west and vertically up

Vector-valued functions of a real variable

- For example, you may have a drone flying in a figure eight over an area:

$$\mathbf{u}(t) = \begin{pmatrix} 10\sin(t) \\ 10\cos(3t) \\ 100 \end{pmatrix}$$

- The direction of travel is given by:

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} 10\cos(t) \\ -30\sin(3t) \\ 0 \end{pmatrix}$$

- The speed is given by

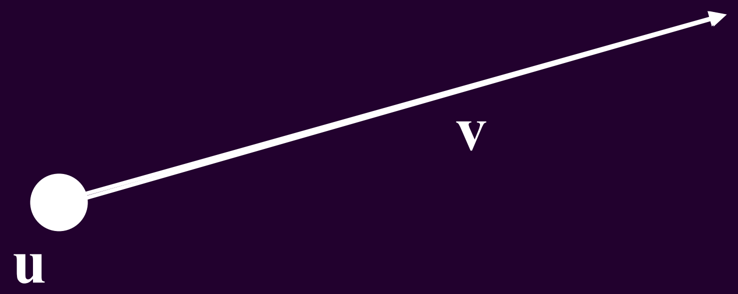
$$\left\|\mathbf{u}^{(1)}(t)\right\|_2 = \sqrt{100\cos^2(t) + 900\sin^2(3t)}$$

  – The speed varies between 5 m/s and 31 m/s

Approximating movement

- Notice that if we have a position vector **u** and velocity vector **v**,
  - Then assuming the velocity is constant, we can estimate the position one time step into the future

$$\mathbf{u} + h\mathbf{v}$$



Systems of 1st-order initial-value problems

- Now, can't we do the same with differential equations?

$$x^{(1)}(t) = 0.02x(t) - 0.1x(t)y(t) \qquad x(0) = 5233$$

$$y^{(1)}(t) = -0.04y(t) + 0.02x(t)y(t) \qquad y(0) = 323$$

$$\mathbf{u}(t) = \begin{pmatrix} u_1(t) \\ u_2(t) \end{pmatrix} \qquad \mathbf{u}^{(1)}(t) = \begin{pmatrix} u_1^{(1)}(t) \\ u_2^{(1)}(t) \end{pmatrix}$$

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} 0.02u_1(t) - 0.1u_1(t)u_2(t) \\ -0.04u_2(t) + 0.02u_1(t)u_2(t) \end{pmatrix} \qquad \mathbf{u}(t_0) = \mathbf{u}_0 = \begin{pmatrix} u_1(t_0) \\ u_2(t_0) \end{pmatrix} = \begin{pmatrix} 5233 \\ 323 \end{pmatrix}$$

$$= \mathbf{f}(t, \mathbf{u})$$

Systems of 1st-order initial-value problems

- Now, can't we do the same with differential equations?

$$y^{(1)}(t) = -ty(t) - 2z(t) \qquad y(0) = 1$$
$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} -tu_1(t) - 2u_2(t) \\ -u_2(t) + 2u_1(t) \end{pmatrix} \qquad \mathbf{u}(0) = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$
$$= \mathbf{f}(t, \mathbf{u})$$

Systems of 1st-order initial-value problems

- Similarly, this system of three initial-value problems can b

$$x^{(1)}(t) = 10\left(y(t) - x(t)\right) \qquad x(0) = 1$$

$$y^{(1)}(t) = x(t)\left(28 - z(t)\right) - y(t) \qquad y(0) = 1$$

$$z^{(1)}(t) = x(t)\,y(t) - \frac{8}{3}z(t) \qquad z(0) = 1$$

$$\mathbf{u}(t) = \begin{pmatrix} u_1(t) \\ u_2(t) \\ u_3(t) \end{pmatrix} \quad \mathbf{u}^{(1)}(t) = \begin{pmatrix} u_1^{(1)}(t) \\ u_2^{(1)}(t) \\ u_3^{(1)}(t) \end{pmatrix}$$

$$\mathbf{u}^{(1)}(t) = \begin{pmatrix} 10\left(u_2(t) - u_1(t)\right) \\ u_1(t)\left(28 - u_3(t)\right) - u_2(t) \\ u_1(t)u_2(t) - \frac{8}{3}u_3(t) \end{pmatrix} = \mathbf{f}(t, \mathbf{u}) \qquad \mathbf{u}(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Euler's method

- Let's see how this works with Euler's method:

$$\mathbf{u}^{(1)}(t) = \mathbf{f}\left(t, \mathbf{u}(t)\right)$$

$$\mathbf{u}(t_0) = \mathbf{u}_0$$

$$\mathbf{u}(t_0 + h) \approx \mathbf{u}_0 + h\mathbf{f}(t_0, \mathbf{u}_0)$$

$$\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + h\mathbf{f}(t_k, \mathbf{u}_k)$$

Heun's method

- Let's see how this works with Heun's method:

$$\mathbf{u}^{(1)}(t) = \mathbf{f}\left(t, \mathbf{u}(t)\right)$$
$$\mathbf{u}(t_0) = \mathbf{u}_0$$

$$\mathbf{s}_0 \leftarrow \mathbf{f}\left(t_k, \mathbf{u}_k\right)$$

$$\mathbf{s}_1 \leftarrow \mathbf{f}\left(t_k + h, \mathbf{u}_k + h\mathbf{s}_0\right)$$

$$\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + h\frac{\mathbf{s}_0 + \mathbf{s}_1}{2}$$

4th-order Runge-Kutta method

- Let's see how this works with the 4th-order Runge-Kutta method:

$$\mathbf{u}^{(1)}(t) = \mathbf{f}\left(t, \mathbf{u}(t)\right)$$
$$\mathbf{u}(t_0) = \mathbf{u}_0$$

$$\mathbf{s}_0 \leftarrow \mathbf{f}\left(t_k, \mathbf{u}_k\right)$$
$$\mathbf{s}_1 \leftarrow \mathbf{f}\left(t_k + \tfrac{1}{2}h, \mathbf{u}_k + \tfrac{1}{2}h\mathbf{s}_0\right)$$
$$\mathbf{s}_2 \leftarrow \mathbf{f}\left(t_k + \tfrac{1}{2}h, \mathbf{u}_k + \tfrac{1}{2}h\mathbf{s}_1\right)$$
$$\mathbf{s}_3 \leftarrow \mathbf{f}\left(t_k + h, \mathbf{u}_k + h\mathbf{s}_2\right)$$
$$\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k + h\frac{\mathbf{s}_0 + 2\mathbf{s}_1 + 2\mathbf{s}_2 + \mathbf{s}_3}{2}$$

Implementation

- Recall our implementation of Euler's method:

```
std::tuple<double *, double *, double *>
euler( std::function<double ( double t, double y )> f,
       std::pair<double, double> t_rng, double y0, unsigned int n ) {
    double h{ (t_rng.second - t_rng.first)/n };

    double  *ts{ new double[n + 1] };
    double  *ys{ new double[n + 1] };
    double *dys{ new double[n + 1] };

     ts[0] = t_rng.first;
     ys[0] = y0;
    dys[0] = f( ts[0], ys[0] );

    for ( unsigned int k{0}; k < n; ++k ) {
         ts[k + 1] = ts[0] + h*(k + 1);
         ys[k + 1] = ys[k] + h*dys[k];
        dys[k + 1] = f( ts[k + 1], ys[k + 1] );
    }

    return std::make_tuple( ts, ys, dys );
}
```

Implementation

- Let's update it for vector-valued functions:

```
std::tuple<double *, vector *, vector *> euler( vector f( double t, vector y ),
                  std::pair<double, double> t_rng, vector y0, unsigned int n ) {
    double h{ (t_rng.second - t_rng.first)/n };

    double  *ts{ new double[n + 1] };
    vector  *ys{ new vector[n + 1] };
    vector *dys{ new vector[n + 1] };

     ts[0] = t_rng.first;
     ys[0] = y0;
    dys[0] = f( ts[0], ys[0] );

    for ( unsigned int k{0}; k < n; ++k ) {
         ts[k + 1] = ts[0] + h*(k + 1);
         ys[k + 1] = ys[k] + h*dys[k];
        dys[k + 1] = f( ts[k + 1], ys[k + 1] );
    }

    return std::make_tuple( ts, ys, dys );
}
```

Implementation

- All of this worked because for our vector class:
  - We defined all possible arithmetic operators on pairs of vectors
  - We also defined all possible arithmetic operators for scalar multiplication

- We can make similar changes to the Dormand-Prince algorithm
  - Only one additional trivial change is required for the adaptive algorithms:
    - We must compare two vectors with the norm:

```
double a{ eps_abs*h/(2.0*norm( y - z )) };
```

Implementation

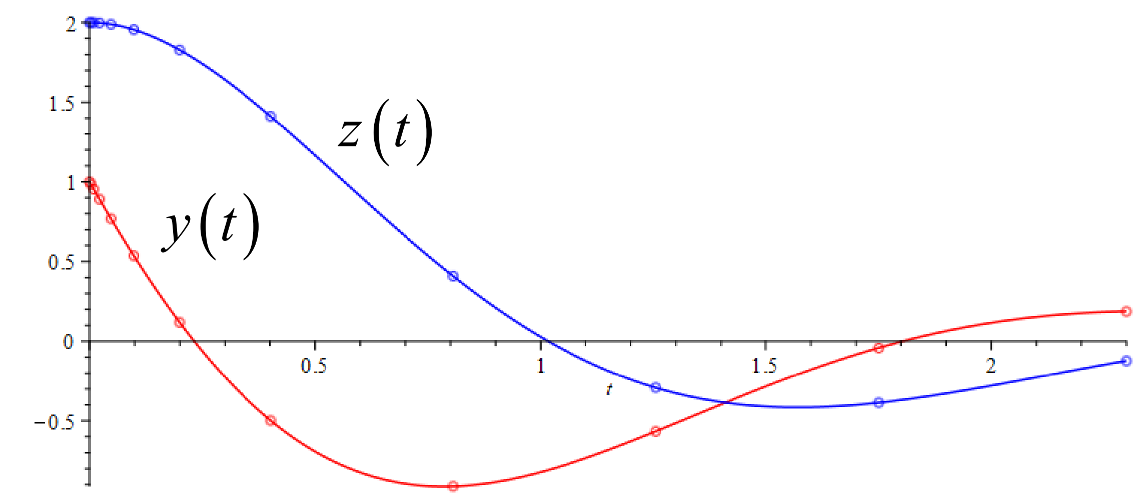
- Recall our problem and solution:

$$y^{(1)}(t) = -y(t) - 2z(t) \qquad y(0) = 1$$

$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$

$$y(t) = e^{-t}\left(\cos(2t) - 2\sin(2t)\right)$$

$$z(t) = e^{-t}\left(2\cos(2t) + \sin(2t)\right)$$



Implementation

- The greatest issue is writing down the functions

$$y^{(1)}(t) = -ty(t) - 2z(t) \qquad y(0) = 1$$
$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$

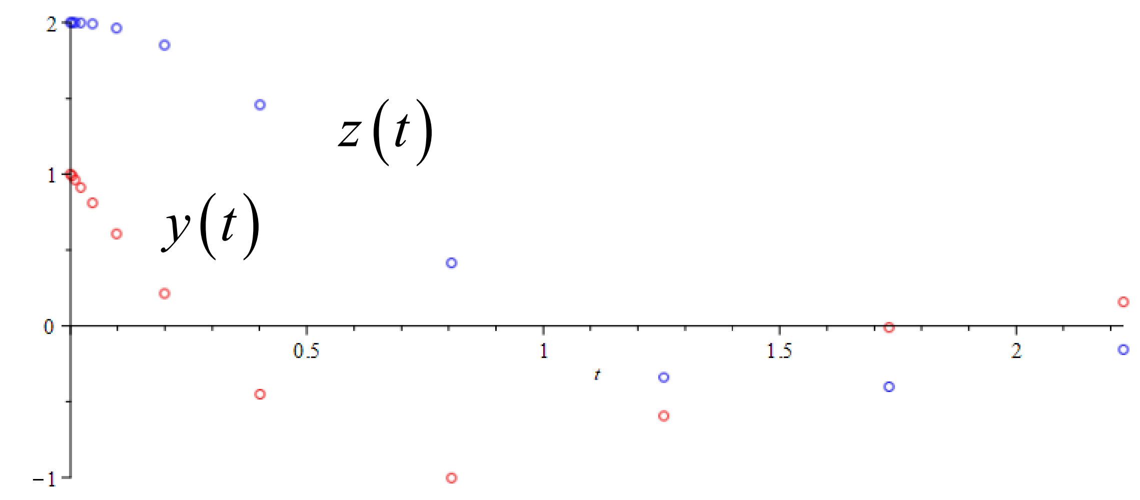
```
vector f( double t, vector u ) {
    return vector{ 2, (double[]){ -t*u(0) - 2.0*u(1),
                                  -u(1) + 2.0*u(0) } };
}

vector{ 2, (double[]){ 1.0, 2.0 } }
```

Implementation

- Recall that this last problem did not have a closed-form solution, however, we can still approximate it

$$y^{(1)}(t) = -ty(t) - 2z(t) \qquad y(0) = 1$$
$$z^{(1)}(t) = -z(t) + 2y(t) \qquad z(0) = 2$$



Implementation

- The greatest issue is writing down the functions

$$x^{(1)}(t) = 10\left(y(t) - x(t)\right) \qquad x(0) = 1$$

$$y^{(1)}(t) = x(t)\left(28 - z(t)\right) - y(t) \qquad y(0) = 1$$

$$z^{(1)}(t) = x(t)\,y(t) - \frac{8}{3}z(t) \qquad z(0) = 1$$

```
vector f( double t, vector u ) {
    return vector{ 3, (double[]){ 10.0*(u(1) - u(0)),
                                  u(0)*(28.0 - u(2)) - u(1),
                                  u(0)*u(1) - 8.0/3.0*u(2) } };
}

vector{ 3, (double[]){ 1.0, 1.0, 1.0 } }
```

Implementation

- Let’s go the opposite direction:

```
vector f( double t, vector u ) {
    return vector{ 3, (double[]){ -u(0) - u(1)*u(2) + t*t,
                                  -u(1)*u(2) + 2.5*t,
                                  -u(2) - u(0) + sin(t) } };
}
```

$$x^{(1)}(t) = -x(t) - y(t)z(t) + t^2$$

$$y^{(1)}(t) = -y(t)z(t) + 2.5t$$

$$z^{(1)}(t) = -z(t) - x(t) + \sin(t)$$

Summary

- Following this topic, you now
  - Have a deeper appreciation for systems of initial-value problems
  - Understand a system of initial-value problems can be converted to vector form
  - Know that this requires essentially no changes to the solvers
    - All of the arithmetic seamlessly translates to vector addition or scalar multiplication
  - Have an idea how to formulate such problems

References

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

Acknowledgments

Zizhou Wang for detecting an error in the indexing on Slide 23.

Colophon

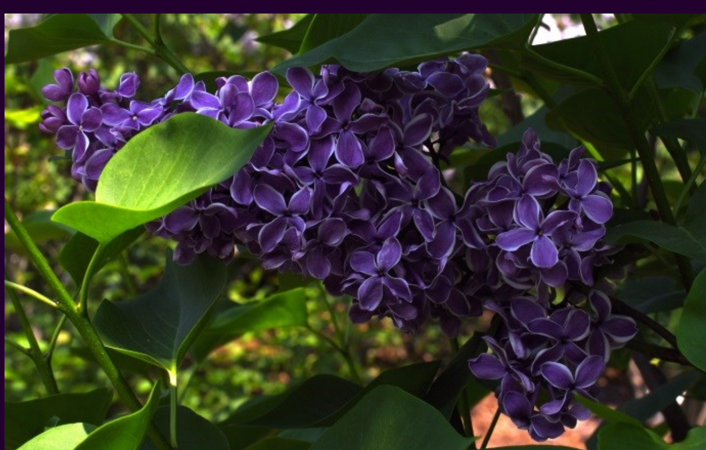
These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`. Mathematical equations are prepared in MathType by Design Science, Inc.

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