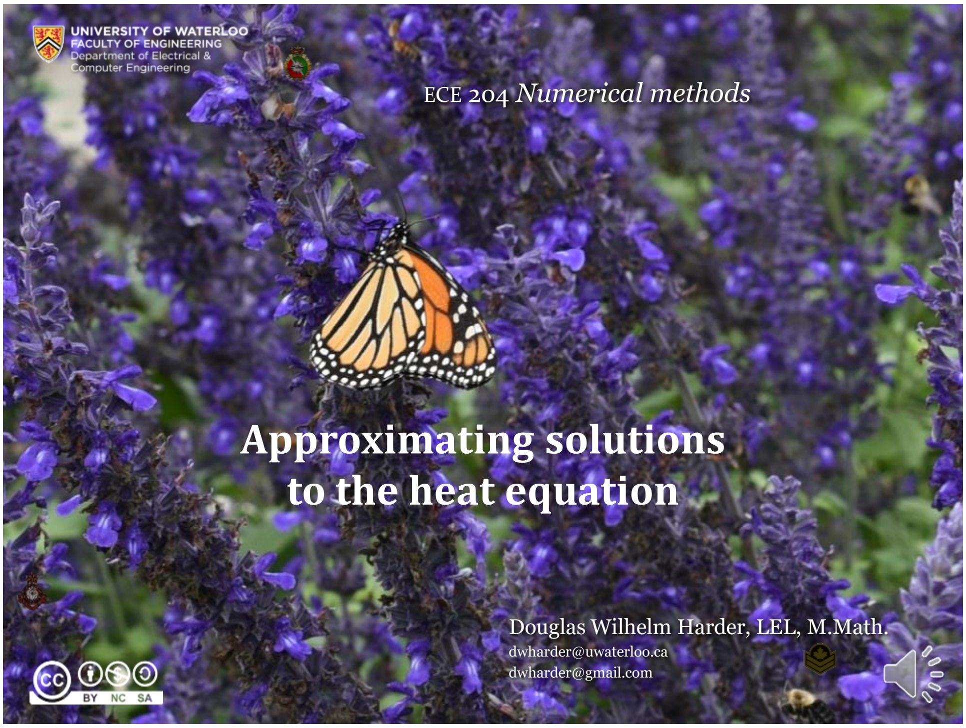


1

Introduction

- In this topic, we will
  - Introduce the heat equation
  - Convert the heat equation to a finite-difference equation
  - Discuss both initial and boundary conditions for such a situation in one dimension
  - Look at an implementation in MATLAB
  - Look at two examples
  - Discuss Neumann conditions and look at the necessary modifications required and additional examples

2

2

Partial differential equations

- The *heat equation* models the transfer of heat within a system

$$\frac{\partial}{\partial t} u(\mathbf{x}, t) = \alpha \nabla^2 u(\mathbf{x}, t)$$

  – The value α is the *diffusivity* coefficient, which is proportional to how quickly heat can travel throughout the medium

- If the heat transfer is restricted to one dimension, this simplifies to

$$\frac{\partial}{\partial t} u(x, t) = \alpha \frac{\partial^2}{\partial x^2} u(x, t)$$

  – This is the case if it is heat transfer along a wire

Partial differential equations

- In one dimension, this says:

$$\frac{\partial}{\partial t} u(x, t) = \alpha \frac{\partial^2}{\partial x^2} u(x, t)$$

  – The rate of change of the temperature over time is proportional to the concavity of the temperature in space
  – If the concavity is locally zero (the temperature is constant or linearly changing), there is no local change in temperature

Partial differential equations

- In one dimension, we can substitute our two approximations:

$$\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t}=\alpha\frac{u(x-h,t)-2u(x,t)+u(x+h,t)}{h^2}$$

  – Note we are only using the O(*h*) approximation

- We can rewrite this as follows:

$$u(x,t+\Delta t)=u(x,t)+\Delta t\alpha\frac{u(x-h,t)-2u(x,t)+u(x+h,t)}{h^2}$$

  – Compare this with Euler's method:

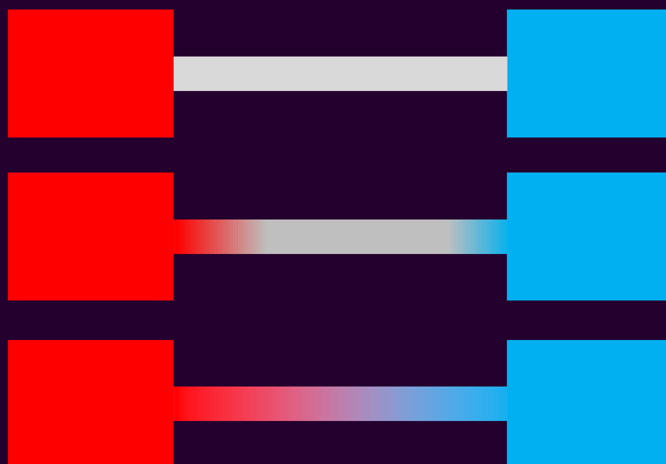
$$f(t+\Delta t)=f(t)+(\Delta t)f^{(1)}(t)$$

Partial derivatives

- Suppose we have a wire or other connection between two large bodies:
  – For example, with Dirichlet conditions, one end may be in contact with a body that is 100˚C while the other may be in contact with a cooling unit at 0˚C
  – When the system is turned on,
    the wire has a temperature at each point
    - Perhaps 20˚C
  – After a few seconds, depending on the material, the temperature near the heat source will increase, while the temperature near the heat sink will decrease, though more slowly

Approximating partial derivatives

- Suppose here we have our system:



Approximating partial derivatives

- Thus, represent the temperature of the bar by a function

$$u(x, t)$$

  - The spatial variable x must fall between the two end-points:

$$a \le x \le b$$

  - Suppose the end points are [0, 1], in which case $u(0.5, t)$ is the temperature at the mid-point at time t
    - If $t = 0$ s, then the temperature is the initial temperature 20˚C
    - Suppose $t = 10$ s
      - If the material is insulating (e.g., wood), it is unlikely the temperature will be very different
      - If the material conducts heat rapidly (aluminium), it may already be getting warm to the touch
    - After a long time, we expect the temperature in the middle to be the average of the boundary values 50˚C

Functions of a vector variable

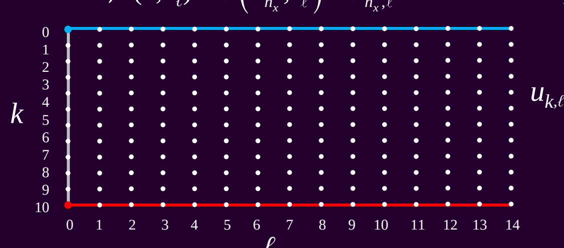
- We don't know what $u(x, t)$ is, so we will approximate it
  - First, divide the interval $[a, b]$ into n_x sub-intervals, each of width h
  - Thus, $x_k = a + kh$ so $x_0 = a$ and $x_{n_x} = b$
- Next, we cannot approximate the solution at each point in time, so we will break time into steps
  - Define $t_\ell = t_0 + \ell\Delta t$
- We will try to approximate $u(x_k, t_\ell)$
  - As before, $u(x_k, t_\ell) \approx u_{k,\ell}$

9

9

Functions of a vector variable

- To start, we have our initial conditions:
  - In this case, $u(x_k, t_0) \approx u_{k,0} = 20^\circ\text{C}$ for an $k = 1, 2, \ldots, n_x - 1$
- We also have two boundary conditions:
  - One side of the bar is in contact with a heat sink at 0˚C
    - Thus, $u(a, t_\ell) = u(x_0, t_\ell) = u_{0,\ell} = 0$ for $\ell = 0, 1, 2, 3, \ldots$
  - The other side is in contact with a heat source at 100˚C
    - Thus, $u(b, t_\ell) = u\left(x_{n_x}, t_\ell\right) = u_{n_x,\ell} = 100$ for $\ell = 0, 1, 2, 3, \ldots$



10

Functions of a vector variable

- So now what?

$$u(x,t+\Delta t) = u(x,t) + \Delta t\alpha \frac{u(x-h,t) - 2u(x,t) + u(x+h,t)}{h^2}$$

$$u(x_k,t_\ell+\Delta t) = u(x_k,t_\ell) + \Delta t\alpha \frac{u(x_k-h,t_\ell) - 2u(x_k,t_\ell) + u(x_k+h,t_\ell)}{h^2}$$

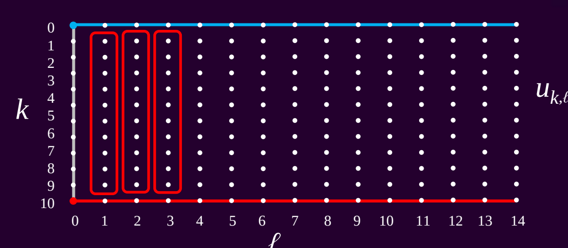
$$u(x_k,t_{\ell+1}) = u(x_k,t_\ell) + \Delta t\alpha \frac{u(x_{k-1},t_\ell) - 2u(x_k,t_\ell) + u(x_{k+1},t_\ell)}{h^2}$$

$$u_{k,\ell+1} = u_{k,\ell} + \Delta t\alpha \frac{u_{k-1,\ell} - 2u_{k,\ell} + u_{k+1,\ell}}{h^2}$$

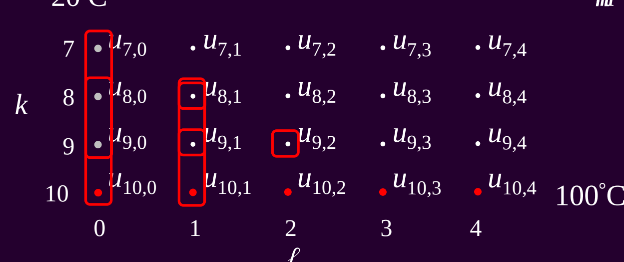
11

Functions of a vector variable

- Let's zoom in:



$$u_{k,\ell+1} = u_{k,\ell} + \Delta t\alpha \frac{u_{k-1,\ell} - 2u_{k,\ell} + u_{k+1,\ell}}{h^2}$$



12

Restrictions

- There is one restriction to this algorithm:

$$\frac{\Delta t \alpha}{h^2} < \frac{1}{2}$$

  - A reasonable strategy: given α and h, suppose we want to approximate the solution from t_0 to t_f
    - We want $n_t \Delta t = t_f - t_0$ so $\Delta t = \frac{t_f - t_0}{n_t}$
    - Thus, let's ensure $\frac{t_f - t_0}{n_t} \frac{\alpha}{h^2} \le \frac{1}{4}$
    - That is, $\frac{1}{n_t} \le \frac{h^2}{4\alpha\left(t_f - t_0\right)}$

$$n_t \ge \frac{4\alpha\left(t_f - t_0\right)}{h^2} \qquad n_t = \left\lceil \frac{4\alpha\left(t_f - t_0\right)}{h^2} \right\rceil$$

Implementation

```
function [xs, ts, Us] = heat( alpha, x_rng, t_rng, u_init, u_bndry, nx )
    h = (x_rng(2) - x_rng(1))/nx;

    nt = ceil( 4.0*alpha*(t_rng(2) - t_rng(1))/h^2 )
    dt = (t_rng(2) - t_rng(1))/nt

    xs = linspace( x_rng(1), x_rng(2), nx + 1 )';
    ts = linspace( t_rng(1), t_rng(2), nt + 1 );

    Us = zeros( nx + 1, nt + 1 );

    for k = 2:nx
        Us(k, 1) = u_init( xs(k) );
    end

    Us([1, nx+1], 1) = u_bndry( ts(1) );

    for ell = 1:nt
        for k = 2:nx
            Us(k, ell + 1) = Us(k, ell) ...
                     + alpha*dt*(Us(k-1, ell) - 2*Us(k, ell) + Us(k+1, ell))/h^2;
        end

        Us([1, nx+1], ell+1) = u_bndry( ts(ell+1) );
    end
end
```

alpha
x_rng
t_rng
u_init
u_bndry
nx

The diffusivity coefficient
A 2-dimensional vector $[a, b]$
A 2-dimensional vector $[t_0, t_f]$
A function of the spatial variable x that gives the initial state at that point
A function of the time variable t that returns a 2-dimensional vector giving the left and right boundary values at that point in time
The number of sub-intervals we will break the interval $[a, b]$ into

Implementation

- Why not just program this in C++?
  - It seems like a straight-forward translation
- MATLAB is an interpreted language, meaning it is, in general, slow
  - There are, however, functions, that simply call compiled routines
  - Calling a compiled routine can be as fast as authoring that function in C++
- Where can we accomplish such a speed up?

Implementation

- Many vector-based functions execute faster than a corresponding for loop:

```
for k = 2:nx
    Us(k, 1) = u_init( xs(k) );
end

Us(2:nx, 1) = u_init( xs(2:nx) );
```

  - This requires that `u_init` work on a vector-valued argument

```
u1_init = @(x)( 20.0 );
u1_init = @(x)( 20.0*ones( size( x ) );
```

Implementation

- Additionally, the operation of calculating $x_{k+1} - 2x_k + x_{k-1}$ is so common, there is a MATLAB function to repeated perform this:

```
diff( x );             # This has one fewer entries
      x(2) - x(1)
      x(3) - x(2)
             .
             .
             .
    x(end) - x(end-1)

diff( x, 2 );          # This has two fewer entries
      x(3) - 2*x(2) + x(1)
      x(4) - 2*x(3) + x(2)
                .
                .
                .
    x(end) - 2*x(end-1) + x(end-2)
```

17

17

Implementation

```
function [xs, ts, Us] = heat( alpha, x_rng, t_rng, u_init, u_bndry, nx )
    h = (x_rng(2) - x_rng(1))/nx;

    nt = ceil( 4.0*alpha*(t_rng(2) - t_rng(1))/h^2 )
    dt = (t_rng(2) - t_rng(1))/nt

    xs = linspace( x_rng(1), x_rng(2), nx + 1 )';
    ts = linspace( t_rng(1), t_rng(2), nt + 1 );

    Us = zeros( nx + 1, nt + 1 );

    Us(2:nx, 1) = u_init( xs(2:nx) );
    Us([1, nx+1], 1) = u_bndry( ts(1) );

    for ell = 1:nt
        Us(2:nx, ell + 1) = Us(2:nx, ell) + alpha*dt*diff( Us(:, ell), 2 )/h^2;
        Us([1, nx+1], ell+1) = u_bndry( ts(ell+1) );
    end
end
```

18

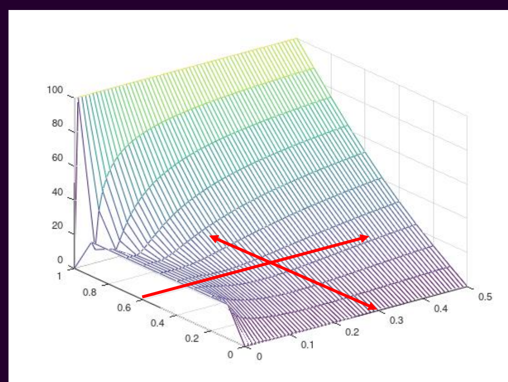
18

The heat equation

Example 1

- Consider this example:

```
>> u1_init = @(x)( 20.0*ones( size( x ) ) );
>> u1_bndry = @(t)( [0.0, 100.0]' );
>> [x1s, t1s, U1s] = heat( 0.3, [0, 1], [0, 0.5], u1_init, u1_bndry, 10 );
>> mesh( t1s, x1s, U1s );
```



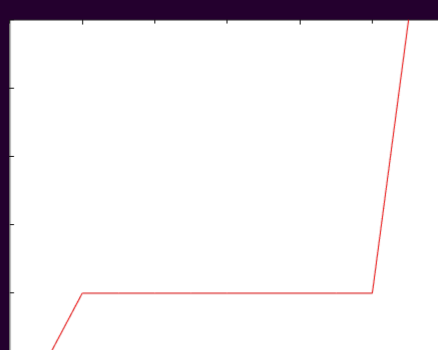
19

19

The heat equation

Example 1

- Recalling that $n_x = 10$, we see how the temperature changes over time



20

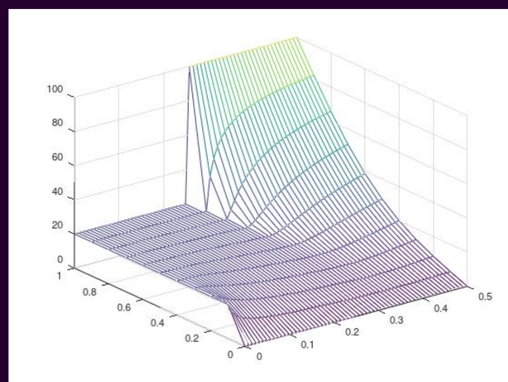
20

The heat equation

Example 2

- Consider this example:

```
>> u1_init = @(x)( 20.0*ones( size( x ) ) );
>> u2_bndry = @(t)( (t > 0.25)*[0.0, 80.0]' + [0.0, 20.0]' );
>> [x2s, t2s, U2s] = heat( 0.3, [0, 1], [0, 0.5], u2_init, u2_bndry, 10 );
>> mesh( t2s, x2s, U2s );
```



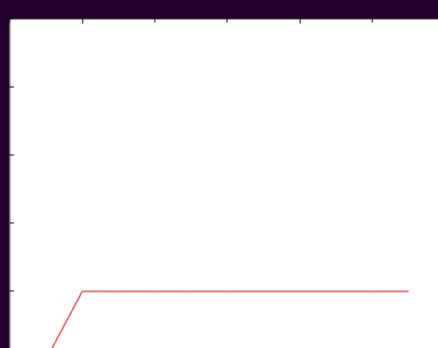
21

21

The heat equation

Example 2

- It starts to cool on the one side, but then the other side starts to heat up after 0.25 seconds



22

22

Error analysis

- Recall the formula using h was $O(h^2)$,
  but the formula using Δt was $O(\Delta t)$
  - Recall, however, that

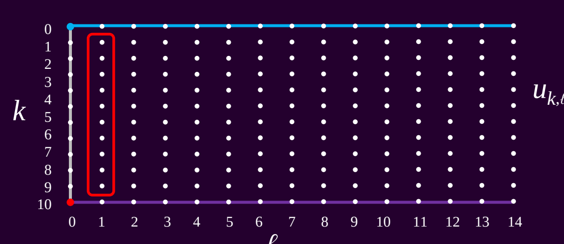
$$n_t \approx \frac{4\alpha(t_f - t_0)}{h^2} \qquad \frac{h^2}{4\alpha} \approx \frac{t_f - t_0}{n_t}$$

$$\Delta t = \frac{t_f - t_0}{n_t}$$

  - Thus, $\Delta t \approx \frac{h^2}{4\alpha}$ so if the error is $O(\Delta t)$, it is also $O(h^2)$

Neumann boundary conditions

- What happens if one boundary is insulated or has a Neumann boundary condition?
  - Recall, in our code, we
    - Calculated the next interior points
    - Then set the boundary conditions for that same ℓ



Neumann boundary conditions

- Recall from the last topic, we saw that if a boundary satisfied a Neumann condition, the following were true:

$$u_0 = -\frac{2}{3}u_a^{(1)}h + \frac{4}{3}u_1 - \frac{1}{3}u_2 \qquad u_n = \frac{2}{3}u_b^{(1)}h + \frac{4}{3}u_{n-1} - \frac{1}{3}u_{n-2}$$

- Suppose a boundary has a Neumann condition:
  - Calculated the next interior points
  - Calculate the boundary value based on the Neumann condition

25

25

Implementation

```
function [xs, ts, Us] = heat( alpha, x_rng, t_rng, u_init, u_bndry, u_dirichlet, nx )
    h = (x_rng(2) - x_rng(1))/nx;

    nt = ceil( 4.0*alpha*(t_rng(2) - t_rng(1))/h^2 )
    dt = (t_rng(2) - t_rng(1))/nt

    xs = linspace( x_rng(1), x_rng(2), nx + 1 )';
    ts = linspace( t_rng(1), t_rng(2), nt + 1 );

    Us = zeros( nx + 1, nt + 1 );

    Us(2:nx, 1) = u_init( xs(2:nx) );
    Us([1, nx+1], 1) = u_bndry( ts(1) );

    for ell = 1:nt
        Us(2:nx, ell + 1) = Us(2:nx, ell) + alpha*dt*diff( Us(:, ell), 2 )/h^2;

        Us([1, nx+1], ell+1) = u_bndry( ts(ell+1) );
    end
end
```

26

26

Implementation

```
function [xs, ts, Us] = heat( alpha, x_rng, t_rng, u_init, u_bndry, u_dirichlet, nx )
    # Initialization...

    dirichlet = u_dirichlet( ts(1) );
    boundary  =     u_bndry( ts(1) );

    if dirichlet(1)
        Us(1, 1) = boundary(1);
    else
        Us(1, 1) = -2.0/3.0*boundary(1)*h + 4.0/3.0*Us(2, 1) - 1.0/3.0*Us(3, 1);
    end

    if dirichlet(2)
        Us(nx+1, 1) = boundary(2);
    else
        Us(nx+1, 1) = 2.0/3.0*boundary(2)*h + 4.0/3.0*Us(nx, 1) - 1.0/3.0*Us(nx-1, 1);
    end

    # Populate the balance of the matrix 'Us'
end
```

$$u_0 = -\frac{2}{3}u_a^{(1)}h + \frac{4}{3}u_1 - \frac{1}{3}u_2$$

$$u_n = \frac{2}{3}u_b^{(1)}h + \frac{4}{3}u_{n-1} - \frac{1}{3}u_{n-2}$$

27

27

Implementation

```
    for ell = 1:nt
        Us(2:nx, ell + 1) = Us(2:nx, ell) + alpha*dt*diff( Us(:, ell), 2 )/h^2;

        dirichlet = u_dirichlet( ts(ell + 1) );
        boundary  =     u_bndry( ts(ell + 1) );

        if dirichlet(1)
            Us(1, ell+1) = boundary(1);
        else
            Us(1, ell+1) = -2.0/3.0*boundary(1)*h + 4.0/3.0*Us(2, ell+1) ...
                           - 1.0/3.0*Us(3, ell+1);
        end

        if dirichlet(2)
            Us(nx+1, ell+1) = boundary(2);
        else
            Us(nx+1, ell+1) = 2.0/3.0*boundary(2)*h + 4.0/3.0*Us(nx, ell+1) ...
                          - 1.0/3.0*Us(nx-1, ell+1);
        end
    end
end
```

28

28

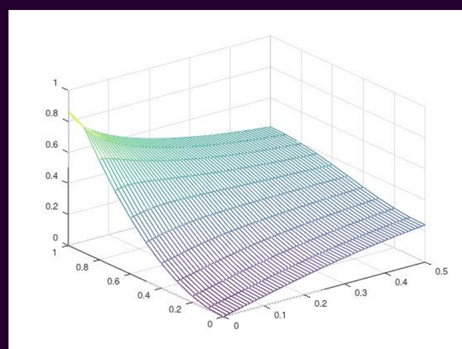
The heat equation

Example 3

- Consider this example:

$$\frac{1}{1-0}\int_0^1 x^2 dx = \frac{1}{3}$$

```
>> u3_in = @(x)(x.^2);
>> u3_by = @(t)( [0.0, 0.0]' );
>> u3_dt = @(5)( [false, false]' );
>> [x3s, t3s, U3s] = heat( 0.3, [0, 1], [0, 0.5], u3_in, u3_by, u3_dt, 10 );
>> mesh( t3s, x3s, U3s );
```



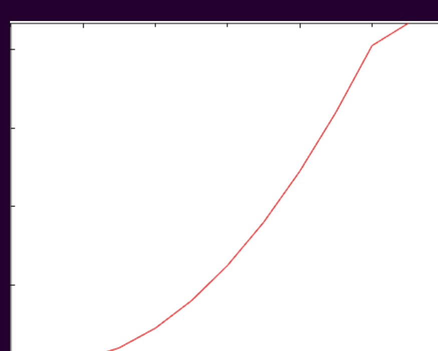
29

29

The heat equation

Example 3

- The temperature will approach the average temperature



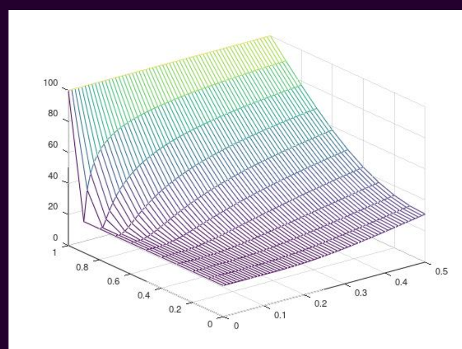
30

30

Example 4

- Consider this example:

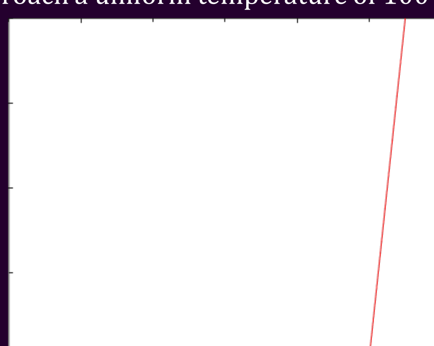
```
>> u4_in = @(x)( 20.0*ones( size( x ) ) );
>> u4_by = @(t)( [0.0, 100.0]' );
>> u4_dt = @(5)( [false, true]' );
>> [x4s, t4s, U4s] = heat( 0.3, [0, 1], [0, 0.5], u4_in, u4_by, u4_dt, 10 );
>> mesh( t4s, x4s, U4s );
```



The heat equation

Example 4

- Note the temperature heats up across the length
  - It will approach a uniform temperature of 100°C



Summary

- Following this topic, you now
  - Understand how to approximate the heat equation with a finite-difference equation
  - Have seen how to approximate the solution to the heat equation given both initial states and boundary values in one dimension
  - Are aware of how to implement such a solution in MATLAB
  - Have seen two examples
  - Understand how to deal with insulated boundary conditions with implementations and examples

References

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

Acknowledgments

None so far.

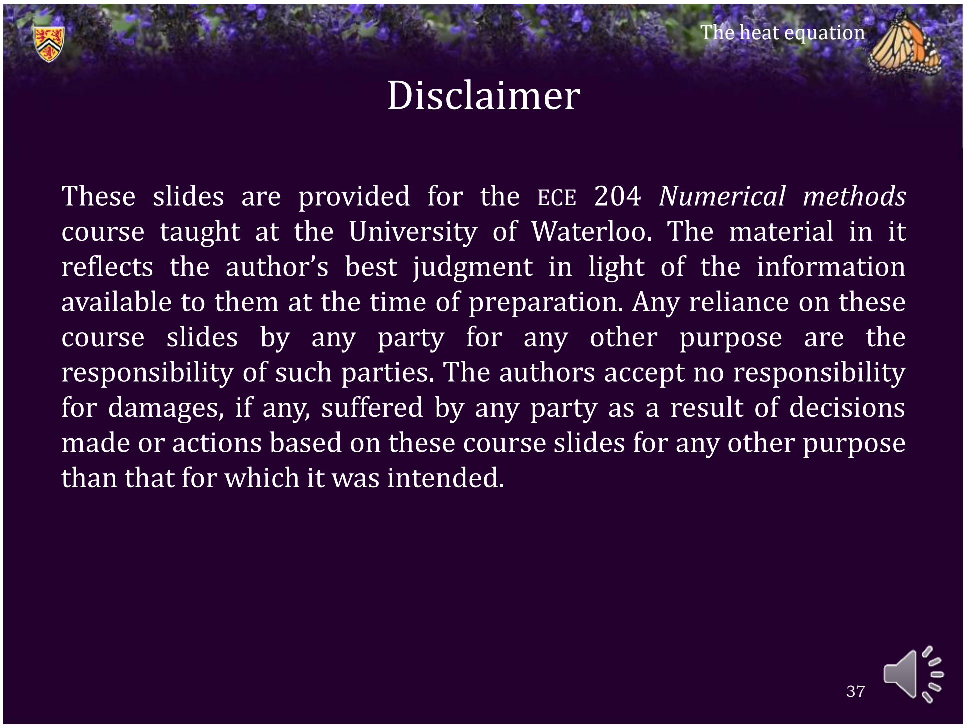
Colophon

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.


The heat equation
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.
37

37