Lecture 2

Cooling Problem

Hai-Qing Lin

Beijing Computational Science Research Center

This PowerPoint Notes Is Based on the Textbook '***An Introduction to Computer Simulation Methods : Applications to Physical Systems***', 2nd Edition, Harvey Gould and Jan Tobochnik, Addison-Wesley(1996);
"A First Course in Computational Physics"; "Numerical Recipes";
"Elementary Numerical Analysis"; "Computational Methods in Physics and Engigering".

Chapter 2: Coffee Cooling

- Heat transfer phenomena => **solving first order ordinary differential/difference equation**.
- **The Euler algorithm and related computer program**.
- Modular programming.
- **Errors and Stability**.
- Seemingly unrelated physical systems can have the same formulation in terms of a computer algorithm.

Coffee Cooling?

- This is a very easy problem, yet some of you may not know (or forget) how to deal with it
- We take this as a warm up exercise for follow up computer simulation methods
- Students should take this opportunity to recall previous learned knowledge of physics, mathematics, and computer programming techniques

Question and Objective

- If we want to drink a cup of coffee in a hurry so we want the coffee to cool as soon as possible, is it better to add the cream immediately after the coffee is made, or should we wait for a while before we add the cream?
- This is a problem of heat transfer.
- Mathematically, this is a first order ordinary differential equation (ODE).

Background

Newton's law of cooling $\frac{dT}{dt} = -r(T - T_s)$

T : temperature
T_s : temperature of its surrounding
t : time
r : cooling constant (depend on physical system)

$$\frac{dT}{dt} = f(T)$$

$$= f(T_s) + f'(T_s)(T - T_s) + \frac{1}{2} f''(T_s)(T - T_s)^2 + \cdots$$

$$= -r(T - T_s) + \ldots$$

The Euler Algorithm

Change **differential** equation
into **difference** equation:

$$\frac{dy}{dx} = f(x, y) \rightarrow \frac{\Delta y}{\Delta x} = f(x, y), \quad |\Delta x| << 1.$$

Get the new x by adding Δx to the old x,
and approximate the new y by
the slope at the old x and old y:

$$x_1 = x_0 + \Delta x,$$

$$y_1 = y_0 + \Delta y \approx y(x_0) + f(x_0, y_0)\, \Delta x.$$

The Euler Algorithm

$$x_1 = x_0 + \Delta x,$$

$$y_1 = y_0 + \Delta y$$

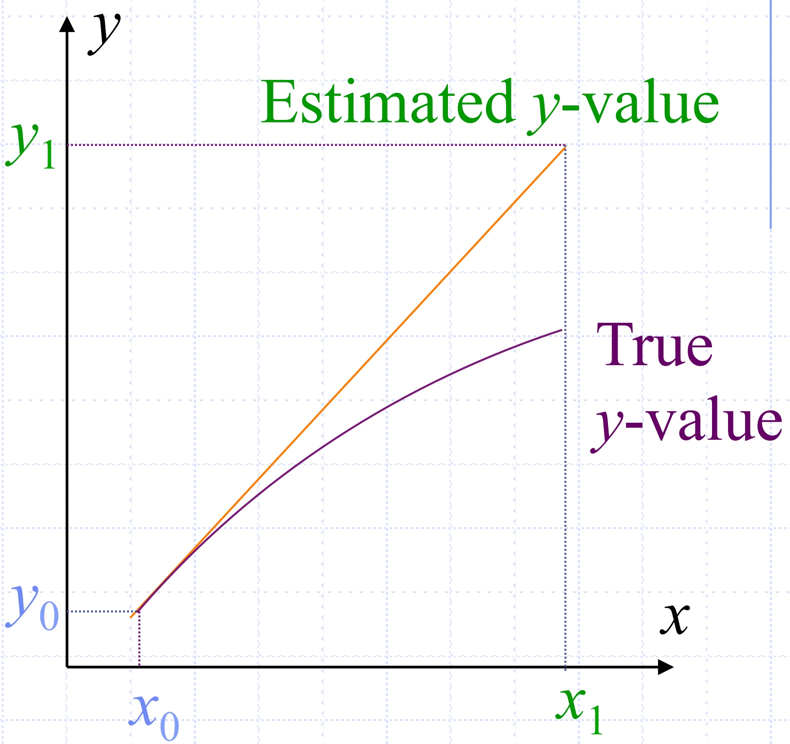
$$\approx y(x_0) + f(x_0, y_0)\, \Delta x.$$

$$\vdots$$

$$\vdots$$

$$x_{n+1} = x_n + \Delta x,$$

$$y_{n+1} = y_n + f(x_n, y_n)\, \Delta x.$$



Since we approximate the derivative by constant slope, it also called constant-slope method.

A Simple Example

- Solve the differential equation

$$\frac{dy}{dx} = 2x$$

with initial condition: $x_0 = 1.0, y_0 = 1.0$

- Exact solution: $y = x^2$
- Iterated solution with step size $\Delta x = 0.1$

$$\frac{dy}{dx} = 2x, \qquad \Delta x = 0.1$$

n	$x(n)$	$y(n)$	Slope $f(x,y)=2x$	$y(n+1)=y(n)+\text{slope}*dx$ $y(n)+f(x,y)*0.1$	Exact value $y_0(n)=x^2(n)$	Error $y(n)-y_0(n)$
0	1.00	1.00	2.00	1.00+2.00*0.10 = 1.20	1.00	0.00
1	1.10	1.20	2.20	1.10+2.20*0.10 = 1.42	1.21	0.01
2	1.20	1.42	2.40	1.42+2.40*0.10 = 1.66	1.44	0.02
3	1.30	1.66	2.60	1.66+2.60*0.10 = 1.92	1.69	0.03
4	1.40	1.92	2.80	1.92+2.80*0.10 = 2.20	1.96	0.04
5	1.50	2.20	3.00	2.20+3.00*0.10 = 2.50	2.25	0.05
6	1.60	2.50	3.20	2.50+3.20*0.10 = 2.82	2.56	0.06
7	1.70	2.82	3.40	2.80+3.40*0.10 = 3.16	2.89	0.07
8	1.80	3.16	3.60	3.16+3.60*0.10 = 3.52	3.24	0.08
9	1.90	3.52	3.80	3.52+3.80*0.10 = 3.90	3.61	0.09
10	2.00	3.90			4.00	0.10

Errors

- For each step, it is about $(\Delta x)^2$.
- For final result, it is about Δx.
- More error analysis later.

Computer Program for the Euler method

- Computer Algorithm:
  - a finite sequence of precise steps or rules that solve a problem, and then develop a computer program to implement the algorithm.

The Algorithm of the Euler Method:

1. Choose the initial condition, the step size, and the maximum value of x. Set x_0, y_0, x, x_{max}.
2. Determine y and the slope at the beginning of the interval. Get $y(x)$ and dy/dx from the differential equation.
3. Calculate y at the end of the interval by adding the change, the slope times the step size, to the value of y at the beginning of the interval and print the result. Obtain and print out new x and y.
4. Repeat steps 2 and 3 until the desired value of x is reached. Continue until $x = x_{max}$.

Example: Solving $\frac{dy}{dx} = 2x.$

```
PROGRAM example
    IMPLICIT NONE
    REAL :: x,y,xmax,delta_x

CALL initial(y,x,delta_x,xmax) ! Specify initial values and parameters

        write(6,1001)
1001 format(/3x, 'X',5x,'Y')
        write(6,1002) x,y
1002 format(2f6.3)

    DO WHILE ( x <= xmax )
        CALL Euler(y,x,delta_x)   ! use simple Euler algorithm
        write(6,1002) x,y
    ENDDO
END PROGRAM example
```

```
SUBROUTINE initial(y,x,delta_x,xmax)
    IMPLICIT NONE
    REAL :: x,y,xmax,delta_x

    write(6,*) 'Enter x_0, y_0, x_max, delta_x'
    read(5,*) x,y,xmax,delta_x
END SUBROUTINE initial

SUBROUTINE Euler(y,x,delta_x)
    IMPLICIT NONE
    REAL :: x,y,delta_x
    REAL :: slope,two
    DATA two/2.0/

    slope=two*x            ! depend on function form

    x = x + delta_x
    y = y + delta_x*slope
END SUBROUTINE Euler
```

The Coffee Cooling Problem

```
program cool   ! numerical solution of Newton's law of cooling
  use common
  real (selected_real_kind(15,307)) :: T_coffee,T_room,delta_t,tmax
  integer :: nshow,counter
  call initial(T_coffee,T_room,delta_t,tmax,nshow)
  counter = 0
  do
    if (t >= tmax) then
      exit
    end if
    call Euler(T_coffee,T_room,delta_t)
    counter = counter + 1      ! number of iterations
    if (modulo(counter,nshow) == 0) then
      call output(T_coffee,T_room)
    end if
  end do
end program cool
```

```
module common
   public :: initial, Euler, output
   integer, parameter, public :: double = 2
   real (selected_real_kind(15,307)), public :: r,t

contains

subroutine initial(T_init,T_room,delta_t,tmax,nshow)
   :
end subroutine initial

subroutine Euler(T_coffee,T_room,delta_t)
   :
end subroutine Euler

subroutine output(T_coffee,T_room)
   :
end subroutine output

end module common
```

```
subroutine initial(T_init,T_room,delta_t,tmax,nshow)
  real (selected_real_kind(15,307)), intent (out) ::
    T_init,T_room,delta_t,tmax
  integer, intent (out) :: nshow
  real (selected_real_kind(15,307)) :: tshow
  t = 0.0                    ! Time; could be read in
  T_init = 82.3              ! initial coffee temperature (C)
  T_room = 17.0              ! room temperature (C)
  print *, "cooling constant r ="
  read *, r
  print *, "time step dt ="
  read *, delta_t
  print *, "duration of run ="
  read *, tmax               ! minutes
  print *, "time between output of data ="
  read *, tshow
  nshow = int(tshow/delta_t)
  call output(T_init,T_room)
end subroutine initial
```

```
subroutine Euler(T_coffee,T_room,delta_t)
  real (selected_real_kind(15,307)), intent (in) :: T_room,delta_t
  real (selected_real_kind(15,307)), intent (in out) :: T_coffee
  real (selected_real_kind(15,307)) :: change
  change = -r*(T_coffee - T_room)*delta_t
  T_coffee = T_coffee + change
  t = t + delta_t
end subroutine Euler

subroutine output(T_coffee,T_room)
  real (selected_real_kind(15,307)), intent (in) :: T_coffee,T_room
  if (t == 0) then
    print *, ""
    print "(t7,a,t16,a,t28,a)", "time","T_coffee","T_coffee - T_room"
    print *, ""
  end if
  print "(f10.2,2f13.4)",t,T_coffee,T_coffee - T_room
end subroutine output
```

Accuracy and Stability

Error sources:

- *Round-off error*:
  finite number of digits for floating point numbers.
  (e.g. $3.21 \times 1.28 = 4.1088$)

- *Truncation error*:
  choice of algorithm. Programmer controllable, but no general prescription.

Why Error Analysis?

- Know how accurate results we will get
- Estimate computer resource needed
- Determine whether new algorithm must be introduced
- Extrapolation to “correct” answer: linear combination? Upper/lower bounds?

Accuracy and Stability

- **Questions to ask:**
  - How accurate?
  - How large an interval to be used?
  - What kind of computer?
  - How much computer time?
  - How much personal time?
- **Try and error:**
  practical way to determine accuracy
- **Stability of an algorithm:**
  small errors $\Rightarrow$ diverge.

Simple Plots

- Plot a set of data, e.g., 2D (x-y) plot. "**Quick and dirty**" mode: for debugging, understanding, etc.
- "**Presentation**" mode: for other people.
- *Use any software you like.*

Visualisation

- Visualise a physical system changing with time.
- Software:
  - Excel
  - Sigma plot
  - Others

Lecture 2 Review

- First-order differential equation.
- Difference equation.
- Euler algorithm (and modifications?).
- Modular programming.
- Program examples.
- Cooling program.
- Accuracy and stability.
- Data plotting.

Lecture 2 Review

We have seen that:

- Program statements are simple.
  Simple algorithm can yield complex behavior.
- Seemingly unrelated physical systems can have the same formulation in terms of a computer algorithm, e.g., nuclear decay, $dN/dt = -\alpha N$

Required for Lecture 2

- Heat transfer phenomena => **solving first order ordinary differential/difference equation**.
- **The Euler algorithm and related computer program**.
- Modular programming.
- **Errors and Stability**.
- Seemingly unrelated physical systems can have the same formulation in terms of a computer algorithm.