Second Programming Project – Developing Multifunction Programs

Larry Caretto
Computer Science 106

Computing in Engineering and Science

April 18, 2006

Outline

- Goals for second project
- Specific tasks for second project
  - Evaluation of infinite series that has more complicated terms than the one for sin(x) in exercise six
  - Trial and error solution of one equation
  - Combination of first two tasks
- Code structure for project two tasks

Project Two Goals

- Develop and test code with multiple functions in a stepwise manner
- Compute more complex infinite series than those in exercise six
- Solve one equation in one unknown by trial and error
- View one parameter in an infinite series as an unknown

Project Two Overview

- Write and test a function to compute the temperature in a slab as a function of time, T(z,t)
- Write and test a function to solve an equation in one variable x = 3sin(x) + cos(x)
- Combine the first two functions to find the time when the temperature at a given z location in a slab achieves a certain value

Temperature Problem

- Have a slab that with large dimensions in x and y directions (relative to z)
- Look at one dimensional heat conduction in z direction ($-L \le z \le L$)
- Initially entire slab is at temperature T_0
- At t = 0 sides ($z = \pm L$) are set to T_B
- Want to find T(z, t)

Solution

$$\lambda_n = \frac{(2n+1)\pi}{2} \qquad \xi = \frac{z}{L} \qquad \tau = \frac{\alpha t}{L^2}$$

$$\frac{T(z,t) - T_0}{T_B - T_0} = 1 - 2\sum_{n=0}^{\infty} \frac{(-1)^n}{\lambda_n} e^{-\lambda_n^2 \tau} \cos(\lambda_n \xi)$$

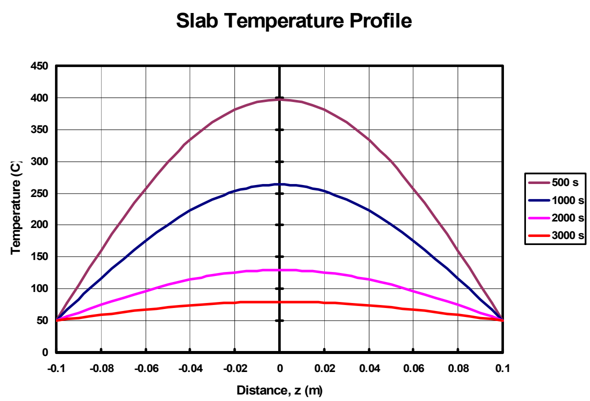
- T_0, T_B, L, z, t, and α (α = thermal diffusivity) are given as input data
- Find T(z, t) from infinite series, similar to exercise six

Summing Infinite Series

- Define allowed error and maximum terms
- Set the series sum to zero
- Loop while not converged and number of terms is less than maximum
  - Compute new term in series
  - Add new term to sum
  - converged = |term| <= allowErr * |sum|
- End loop

Task One

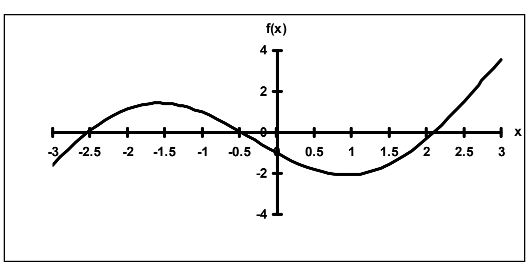
- Write `double temperature( double z, double time)` to compute T(z,t)
- Use global variables for T_0, T_B, L, and α
- Write main function that calls your temperature function in a nested loop to compute table for T(z, t)
  - Recall exercise six, tasks one and two
- Plot results to estimate t_{100}, defined as time such that $T(0,t_{100}) = 100$
- Sample results on next chart



Trial-and-error Solutions

- What is solution of 3x + 7 = 2x + 3?
- What is solution of x = 3 sin(x) + cos(x)?
- Second case has only one unknown, but cannot be solved explicitly
- Need a process for solving such equations
- General problem: find x so that f(x) = 0
- Here f(x) = x – 3 sin(x) – cos(x) = 0
  - Could also write f(x) = 3sin(x) + cos(x) – x = 0, but it would have the same solutions

f(x) = x – 3 sin(x) – cos(x)



- What values of x give f(x) = 0?

Task Two

- Write a function `double secant( double x )` that solves f(x) = 0 in general
- Algorithm in project description
- Upon entry, x is initial guess; final x (or error code) returned in function name
- Function will apply to any problem for solving f(x) = 0 if we write f(x) as a separate function

Secant Function Algorithm

- Uses two most recent guesses x_1 and x_2 to compute new guess, x_3, based on values of $f_1 = f(x_1)$ and $f_2 = f(x_2)$
- Uses linear interpolation to get $f(x_3) \approx 0$
- $x_3 = x_1 + [(x_2 – x_1)/(f_2 – f_1)] (0 – f_1)$
- Interpolation not exact; iterate to obtain small error
- Adjust x_3 to avoid large change
- Pseudocode in assignment

Task Two Code

- Three functions
  - Main function gets user input on initial guess, calls secant function and prints result
  - Secant function to solve f(x) = 0 calls function below to evaluate equation

```
double f( double x )
{
    return x - 3 * sin( x ) - cos( x );
}
```


Task Two Code

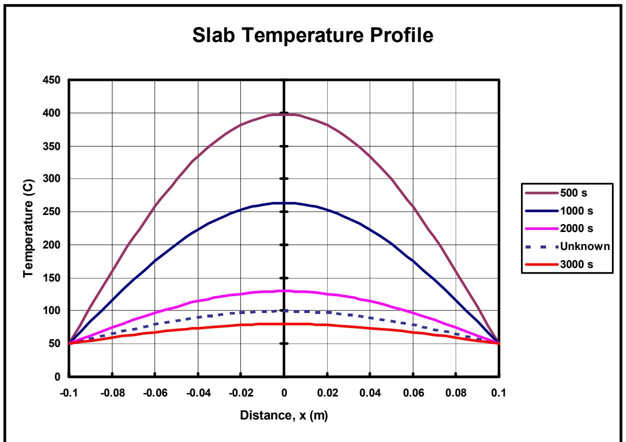
- Three functions
  - Main function gets user input on initial guess, calls secant function and prints result
  - Secant function to solve f(x) = 0 calls function below to evaluate equation

Note that this function gives same result as one on last slide

```
double f( double t )
{
    return t - 3 * sin( t ) - cos( t );
}
```


Task Three

- Combination of tasks one and two
  - In task one we found temperature for given time (and other variables)
  - Here we use secant method from task two to find (unknown) time to reach given temperature (all other data given)
  - Use temperature and secant functions from tasks one and two without change



Task Three Code

- Main function gets initial guess from user, calls secant function to get answer and prints answer
- Secant and temperature functions from tasks one and two used here
- Create new f(x) function that gives time so that f(time) = 0 when computed temperature equals desired temperature

f(time) function

```
const double ERROR_FLAG = -999.999;
double desiredT;
double f(double time)
{
    double const centerlineZ = 0;
    double temp =   temperature(
              centerlineZ, time );
    if( temperature == ERROR_FLAG )
        return ERROR_FLAG;
    else
        return temp - desiredT;
}
```

California State University
Northridge

19

Task Three Summary

- Main function (similar to task two)
  - get initial guess on time from user
  - call secant function to get answer
  - print result
- Temperature function from task one
- Secant function from task two
- New f(time) function from previous chart
- Download answers from web site and match them before submitting project

California State University
Northridge

20

Programming Guidelines

- Download from web site and use in last three assignments
- Designed to make programs easier to understand and modify
- Generally simple set of priciples
- Summary on next chart

California State University
Northridge

21

Programming Guidelines II

- **Good program structure:** program logic steps easy to follow
- **Effective use of comments:** help explain program but not excessive
- **Well-formatted output**
- **Neat appearance of code**
- **Meaningful variable names**
- **Portability and documentation**: Not considered in Comp 106

California State University
Northridge

22

Programming Guidelines II

- **Good program structure:** This means code that is inherently easy to read and understand. Well-structured programs have a control flow that can be easily seen by someone reading the code. Well-structured code does not have many levels of nesting within a single function. Instead, the overall code is broken up into several functions, each of which performs a small, distinct task. Some programmers believe that each function should have only one point where values are returned to the calling function. Others believe that more that one return point may be okay, if justified by the logic of the function. For example, a function that does an iterative calculation until convergence is achieved or some maximum number of iterations is exceeded may have two return points: one for the converged result, the second for a result that has not converged.
- **Effective use of comments:** These statements describe what the code is supposed to do so that another person can follow the code. It also makes it easy for the author to review the code at a later date. Each function should start with an introductory set of comments giving the programmers name and date and a statement of the purpose of the function. An overall description of the entire program is also helpful. Avoid comments, which are obvious from reading the code. Comment statements can be used to define the purpose of each variable. (In this course, comments are judged not only on their technical content but also on the quality of the writing, including spelling and grammar.)
- **Well-formatted output:** Your output should be easy to follow and understand. Although it is possible to use complicated output statements to develop elaborate output, it is also possible to use simple free-format output (e. g. cout <<, “x = “ << x;). This technique provides results that can be understood although they may not be perfectly formatted. Labeling of results so that they cam be understood is more important that perfection in the output formats.
- **Meaningful variable names:** Variable names should be chosen which make the meaning of the variables clear. For mathematical software it is usually convenient to use the symbols that are used in equations for variables. Thus Ohm's Law might be written as e = i * r. This is just as meaningful as writing voltage = current * resistance. Sometimes Greek letters are spelled out. For example, the common symbol for density is the Greek letter r. Computer programs can use density or rho as variable names for density.
- Long variable names are sometimes separated by an underscore such as average_grade; an alternative is to capitalize the start of the new word as in averageGrade or AverageGrade. Symbolic constants are often written in all capital letters such as PI or MAXIMUM_ELEMENTS.
- There are a variety of naming conventions that may be used to define variables. This usually involves a prefix that indicates the scope of the variable or the variable type. Such guidelines are used for the development of complex programs where it is useful to have such information and where professional programmers are familiar with the notational conventions.
- **Portability:** Always use standard C++; do not use special options for a particular compiler. This assures that programs written for one machine can be used on another machine with a minimum of conversion work. This should not be a problem for programs in this course.
- **Documentation:** This includes a discussion of how the overall code and the various functions work. It also includes a list of all the variables and their meaning. The intention of program documentation is different from that of user instructions. The latter are intended to allow another person to use your routine as a black box. Program documentation should allow another person to be able to understand how your program works and to modify it if necessary.
- **Neat Appearance of the code:** This includes the following: indenting structures such as loops and if blocks, using blank lines to separate comments and different portions of the code, using symbols such as rows of *'s or ='s to separate portions of the code, and using white space in statements. In general any process that makes the code easier to read and understand should be used. In the example below, the block on the right is easier to read than the block on the left, even though both blocks accomplish the same task. For additional examples look at the appearance of the codes provided in the various laboratory exercises you have been assigned.
- The two sets of statements below do exactly the same task. Which of these is easier for you to read and understand?
- if(x==3.0){z=y+w/x;y=x*x/cos(z);}else{z=y=3*x }if (x == 3.0){z = y + w / xy = x * x / cox(z)}else{z = y = 3 * x;}
- **Generality:** This means a program, which can handle as general a problem as possible with no modifications to the routine. Always use variable names to represent quantities, which may change in some future application of the program. Use symbolic constants, especially for array dimensions.
- The items in the above list are given in roughly the order of their importance. The major idea is that the code should be easy to understand when it is read by someone not familiar with the code. This includes the code’s author who is viewing the code after several months.
- Students enter this course with a wide variety of backgrounds in programming experience. Students who have no previous experience in programming will be working hard just to complete the assignments. Those of you with some programming experience should take more time to see if you can follow all of the guidelines presented here.
- All students should be concerned about simple items like meaningful variable names, the use of white space and indenting portions of the code to show its structure. These items will make your code easier to understand and to debug.

California State University
Northridge

23