ChEE 201
Computer Reading Number 3

You may be asking yourself, "Why am I learning all this computer programming stuff in a course that isn't about computer programming?" The simple answer is because many engineering problems that you will encounter just can't be solved analytically; there are problems that can't be manipulated until you just solve the equations. The types of problems that can be solved by a computer include differential equations, large systems of equations, and non-linear equations. We'll encounter the last two types of problems in this class, but will not discuss differential equations until later chemical engineering courses.

Computer programs are also particularly useful for automating some routine tasks. In Homework 2, you wrote a program that would report the middle of three numbers. It seems like a trivial task for writing a program, but now imagine that you had to find the middle of three numbers 10,000 times in a row…quickly. Instead of trying to do it by hand, you could embed your short code into another larger program to run through all of the number sets and it would only take the computer a minute or two...and it wouldn't make any mistakes, like you probably would after doing the first ten or a hundred.

You know you can solve two equations with two unknowns. You probably used substitution or a matrix manipulation tool, or possibly your calculator had a built in algorithm. Now imagine you had a set of 100 equations with 100 unknowns. There is no way you could solve them by hand, and most calculators don't allow you to put in an equation that long. Even the matrix manipulation methods would become too difficult for you to use for systems larger than five equations and five unknowns unless you know some special tricks for taking advantage of special situations. Computer programs routinely handle large problems like these.

These are just some of the reasons that we will be focusing on computer approaches for solving problems throughout the semester.

Learning Objectives:

1) Students will become aware of some programming and debugging tricks that will make their lives simpler later on
2) Students will learn about dimensioning variables and why this is important in calculations
3) Students will learn definitions of, and how to use, relative and absolute error

Students are expected to have worked through Readings 1 and 2 and the related HW assignments. If they have not, they will be sent to do these activities before help will be given on this section of reading.

Programming and debugging tips:

You should be becoming more comfortable using VBA and Excel by this point. However, if you are not, a good way to start any new program is to start with a program that you know is working and then modify it to become the new program that you wanted to have. You just cut out the lines that you don't need and add in the new ones that you do need.

Another good way to learn how to program in a new language is to look at examples of code that work, which is why these readings have many examples. Finally, a good place for information is to go online and look for code you can use that others have written. You need to be careful, though, because you may end up trying to use code that is too complex or too confusing to understand, getting yourself into a mess. A good rule of thumb is that you should only use the code you find online that you can understand the structure of and what it is doing.

Msgbox:

If you have written a program and you aren't sure it's doing what you think it should be doing, you should go back to using a msgbox here and there in the code to see where the computer is going and what numbers it has when it reaches certain points. An analogy is that you show all of your work on your homework so you can get partial credit. If you didn't do this (much like the computer programs also don't do), you would either get the right answer or no credit. Make your computer program show it's homework by having it report intermediate answers when it reaches certain points. This is particularly useful if you have worked out some of the numbers by hand for a simple case first.

General Tips:
Do not ever use a section of code that could get into an infinite loop. We encountered the Do...While loop earlier in Reading 2 and this is one example of a type of code that can cause you grief later on if you use it.

A good programming practice is to write a small section of code and then see how that piece works. Then write the next piece and check that one. Keep building up until the entire program is doing what you want it to do. Don't just sit down and try to write 100 perfect lines of code that solve the most complex problem since very few programmers are good enough to be able to do that.

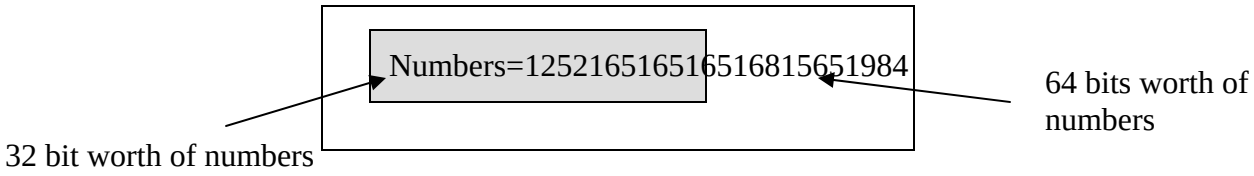
If you have a longer piece of code and you find that it doesn't seem to be working, try removing pieces of the code to see if those sections were where the problem was. You can always CTRL+Z to undo the removal to put the code back as long as you haven't saved the file over again. This is most useful when your program isn't given you any results at all or is reporting NAME or VALUE in the Excel spreadsheet.

Finally, the last advice is to test your code with numbers where you know what the final result should be. If you had written a program that found the square root of a number, you would probably test it with numbers like 4, 9, 16, 25...instead of 37934.12. You know what the answer should be for the first four, but not the last number. An alternative is to compare the numbers from another method or an example you found in another text or online if you can't easily think of an example to check by inspection.

Dimensioning Variables in VBA:
In Computer Reading One, you were introduced to some of the variable dimensioning methods. We used "Dim n as single", for example, and gave a brief description of what this would do. Dimensioning is important when we use Option Explicit since we are then telling the program that we are going to explicitly define all variables that we will be using and what types they will be. This is a great programming tip to keep our own minds straight about what we are doing. If we didn't do this type of explicit dimensioning, we could end up with a big hodgepodge of variables of types that are randomly decided by the first piece of information the code sees regarding that variable. We want to have more control over our variables and what we are doing so dimensioning makes sense.

There are only a few dimensioning statements that we will use throughout this class. We've already seen "Dim (variable) as single". This dimensionsing statement would set the variable to single precision. This means that the number carried along for the calculations would occupy only one storage position on the computer, typically 32 bits of information on a modern computer. Another type of dimensioning is double precision, which would hold 64 bits of information. This would be written as "Dim (variable) as double". The homework you will do for this assignment will explore the difference between using these two types of dimensioning statements.

Basically, if you have a number that takes up more than 32 bits of memory, it will be rounded at some point to fit within the 32 bit space. All the other numbers will be lost like in this figure here:



Double precision, then, would be better to use when you need to have a very accurate answer, like when you're landing a two billion dollar piece of equipment on Mars. You need to have a very, very accurate answer here because being off by a foot of two could cause the loss of your entire investment of money and time.

So, why wouldn't you always choose to use double precision if it would lead to more accurate results? The problem is that using double precision takes longer to run because you're carrying so much more baggage; even if your number is small, the computer is still writing and saving much more extra information. A good analogy is that you probably try to take only a few pieces of very full luggage when you travel by airplane. Now, imagine that you had the same amount of stuff, but you instead packed it in 100 bags. How much longer do you think it would take you

to check in at the airport? How about moving through the hallways? Carrying along a lot of empty space (unfilled luggage = unfilled memory bits) doesn't make sense because we're just moving a lot of air around the airport.

If we can get away with it, then, we want to use single precision. A good way to know whether you can get away with this time saving method is to use both types of dimensioning and see if the answer changes. If it does change, you will need to use double precision if you want a highly accurate answer.

The final dimensioning statement you might encounter in this class is "Dim (variable) as string". This sets the variable to a string of characters. You might do this if you were writing a program to keep track of address, telephone numbers, or other alphanumeric data. In class, a short exercise was done to show this a little more clearly and why it may be important.

Error Definitions:

There are several types of error that we will encounter in this class. We will go through a short list of definitions before we come back and look at each one in more detail.

$$\text{True error: } E_t = \text{true value - approximate answer}$$

$$\text{True fractional relative error} = \frac{\text{True error}}{\text{True value}}$$

$$\text{Approximate error: } E_a = \text{Current approximate value - Previous approximate value}$$

$$\text{Approximate relative error} = \frac{\text{Approximate error}}{\text{Current approximate answer}}$$

Sometimes true error is called the absolute error, as well. These are the four major errors we'll see and use. The first two errors are the true error because we are comparing our calculated answer to a value that is known and true. An example would be if you solved the following equation:

$$3x + 9 = 0$$

and you got x = -2 somehow by misentering information into your calculator or you made an algebraic mistake by hand. The true error would then be:

$$E_t = -3 -(-2) = -1$$

Here, we know what the final answer should be even though we didn't get it right. If we had gotten the correct answer, we would have $E_t = 0$. This is the type of error you are used to on most exams when you lose points; there your answer is being compared to a true answer the grading key has.

The question then becomes, when do we need to use the approximate relative error? We use this type of error when we don't know what the final correct answer is but we still want to be able to estimate an error. This is often done so we can decide when to stop adding corrections to our previous approximate answer and know that we are probably very close to the correct answer and is typically encountered in functions that are infinite sums.

Not too long ago, calculators and computers weren't around, and people did much of the manipulation of numbers by hand or on a sliderule. Some infinite sums were developed to help find approximate answers without doing a lot of work. Here are some of the common infinite sums that were used:

$$\frac{1}{1-x} = x^1 + x^2 + \ldots + x^n = \sum_{n=1}^{\infty} x^n \quad (\text{for } |x| < 1)$$

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots + \frac{x^n}{n!} = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots + \frac{x^n}{n} = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \qquad (\text{for } |x| < 1)$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$$

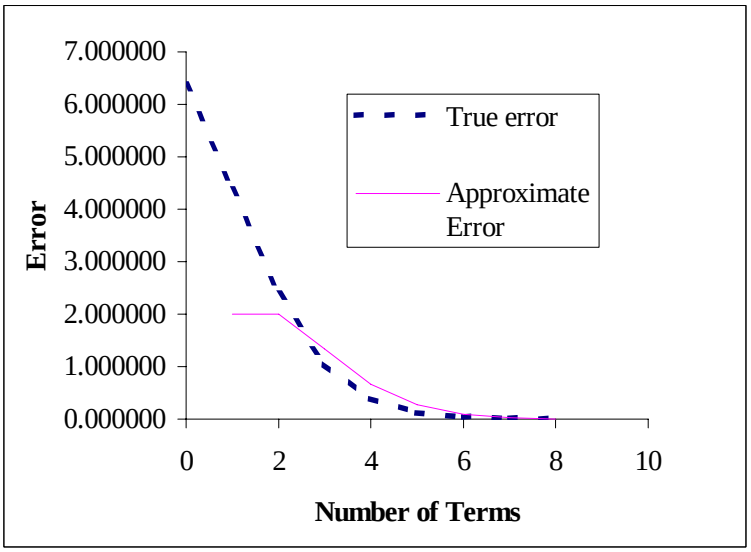
$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} \ldots$$

These functions will be similar to some of the Taylor series expansions that we are going to do in another section very soon in that they will all be summed up to an infinite number, n, to get the best answer possible. Other forms of infinite sums for functions are in your calculus books and other mathematics texts.

Let's turn now to how the infinite sum representation and approximate error can lead us to stopping a computer program when we are trying to find a value. We'll use the exponential function and try to find the exponential of 2. We can set up a table to have our number of terms, our true error, and our approximate error, along with their relative amounts. Keep in mind that the true value is e^2 = 7.389056099:

Number of Term	Additional term	Sum of Terms So Far	True error	True Relative Error	Approximate Error	Approximate Relative Error
0	1.000000	1.000000	6.389056	0.864665	N/A	N/A
1	2.000000	3.000000	4.389056	0.593994	2.000000	0.666667
2	2.000000	5.000000	2.389056	0.323324	2.000000	0.400000
3	1.333333	6.333333	1.055723	0.142877	1.333333	0.210526
4	0.666667	7.000000	0.389056	0.052653	0.666667	0.095238
5	0.266667	7.266667	0.122389	0.016564	0.266667	0.036697
6	0.088889	7.355556	0.033501	0.004534	0.088889	0.012085
7	0.025397	7.380952	0.008104	0.001097	0.025397	0.003441
8	0.006349	7.387302	0.001755	0.000237	0.006349	0.000859

Here, we've started off the first row and second column with the zeroth order term, which is 1. Then we have each next term as we move down the column in column two, calculating the individual contribution of each new term to the total final answer. The third column is the sum of all the terms up to and including the one for that row. Then, the errors are calculated in the last four rows using the formulas on page 3. Why are there no values for the approximate errors in the first row?



Let's look now at how the true error and approximate error change as the number of terms is increased. A plot would look like:

We see that both errors trend downward rapidly by the time we get to the third or fourth terms. And both errors get vanishingly small by the time we are out to seven or eight terms. This indicates there is convergence of our answer to a

final value as we add an infinite number of terms.

If you were using this approximation, how many terms would you add before you knew your answer was good enough? Well, the answer depends on how good your answer needs to be. If you can tolerate an answer that has some error, then you could stop sooner rather than later. Absolute comparisons like the ones shown in this plot might not help in deciding when your answer is good enough. This is why relative errors are defined and used often. Saying that my answer is off by 1.0 doesn't mean anything unless I know what the context of the problem is and what the units are on 1.0. On the other hand, saying that my answer is off by 1% (0.01 for fractional error) lets us know that we are very close to whatever the final answer is.

Let's return to the question now of when the approximate error is useful. In this case, in columns 4 and 5, we had the true error because we already know what the answer was and could compare to it. In columns 6 and 7, we don't need to know what the actual answer is because we base our improvement on the previous approximation instead. So, we'll use this error whenever we don't know what the real final answer should be. This will give us some stopping criteria in our computer programs so that we can know we have a good answer and not spend any more time calculating improvements to what we already have calculated. Typically, it is acceptable to have your computer program stop adding new terms when your approximate relative error becomes less than one to five percent. Sometimes, though, the tolerance needs to be made smaller when a very accurate answer is needed.