Notes from January 24 – Tuesday

- The ce108 students agreed that for an infinite series, all the terms must be related somehow because it is impossible to create an infinite number of things without some relationship between them. In fact, the so-called random number generators on the computers would generate numbers which will eventually repeat itself because the number of bits (binary digits) for the computer is finite. If you play Solitaire on the computer often enough, you will recognize that a game reminds you of another one you played days ago.

- For mathematical infinite series, they are related through a recurrence relationship because most mathematical functions are solution of an ordinary differential equation (ODE). See the document under “Support Material” on the class website with the link PDF Introduction to Infinite Series. From the derivation, you can see most infinite series in Calculus are results of an ordinary differential equation (math 245). The document includes the derivation of important functions such as \( \exp(x) \), \( \cos(x) \) and \( \sin(x) \).

- For the less experienced computer programmers, when they see an infinite series with factorials, they would immediately implement the program with calls to factorial functions. In matlab, \( \text{factorial}(4) \) returns the values of \( 4*3*2*1 \). But a much more efficient algorithm can be developed if one recognizes that the term previous to the present term has already a lower order factorial calculated. To calculate the present term, there is no need to calculate a new factorial. For example, if the previous term has 14! already calculated, it is easy to calculate 15! By simply multiplying 14! by 15. Using a modern computer, it is very easy to store values so that it wouldn’t have to be recalculated.

- If the first term of the series is established, the second term is related to the first term by an expression. Likewise the third term is related to the second term with a similar expression. Those expressions are a function of an index. If the series is an alternating series, the recurrence relationship is always negative.

- There are at least two ways to get those recurrence relations. (a) Solve the differential equation and observe the pattern between the terms. (b) Use an existing series and compare terms to observe the pattern of the ratio between terms.
In homework #2, 5 series are given along with the index to be used. Therefore, scheme (b) of the above paragraph must be used.

To obtain the recurrence relation between terms, it is VERY IMPORTANT not to multiply (or simplify) the factors before the pattern is recognized. For example, in the cosine series, you will see the patterns of (3 times 2), (5 times 4), (7 times 6), etc. It is very easy to recognize those patterns. But if you simplify the numbers to be (6), (20), (42), etc, it is much more difficult to recognize the pattern.

Remember, for alternating series the recurrence relationship must be negative.

When developing the recurrence relation, look for common patterns and write them down first, i.e., \( x^2 \), sign, etc.

The next step is to compare the term index with the numerical factors. If the pattern is still confusing, introduce another column as two times the index, sometimes, the indices are 1,3,5,7 or 2,4,6,8, therefore, a factor of 2 is involved.

Note: the recurrence relation is highly dependent on the index designed, so be careful about the use of the index (or indices). See the class note for the sine and cosine series when a different index was used:

[PDF Recurrence Relationship of Sine and Cosine](#)

The algorithm used for Home Work Number 1 can be adapted to calculate most series. Instead of repeating the same program statements over and over, it is easier to use a loop structure available in all high-level programming languages. For example, the code

```plaintext
i=0
term=1
sum=1
i=i+2
term=-term*x^2/(i*(i-1))
sum=sum+term
i=i+2
term=-term*x^2/(i*(i-1))
sum=sum+term
i=i+2
term=-term*x^2/(i*(i-1))
sum=sum+term
```

can be replaced by
i=0;
term=1;
sum=1;
for loop=1:3
  i=i+2;
  term=-term*x^2/(i*(i-1));
  sum=sum+term;
end

Semicolons were placed at the end of the executable statements to suppress printing of the intermediate results.

- The typical structure of this program is to initialize the first term so later terms could be calculated using an index and the independent variable (x in this case). Before the loop, the sum is initialized to be the first term, so later terms could be accumulated to it by fetch and store (overwrite previous value).

- In most cases, the index of the “for loop” could be used as the index of the program, so the program could be written as:

```matlab
term=1;
qsum=1;
for i=2:2:20
  term=-term*x^2/(i*(i-1));
  qsum=qsum+term;
end
```

In the above program, the index “i” is varied from 2 to 20 with an increment of 2. Therefore, for each loop, the index i would be 2, 4, 6, 8, ..., 20. At the end of the loop, the result of the series would be contained in the variable qsum, be sure to generate an output for that value, or else, it is a stupid program (one without output). Note: qsum was used instead of “sum” because the latter is a frequently used function in matlab.

- The same recipe could be used to sum most mathematical series, including the problems on midterm examination number one.

- For programmers who are conscious of wasteful computation, a variable x2=x*x would have been defined before the “for loop” and inside the loop x2 would be used in placed of x^2, thus saving one multiplication per loop.

- In many cases, several series could be summed using the same loop if the terms of the series share the same index. Just use different names for the sums and the terms (or perhaps an indexed array).