Queens College, CUNY, Department of Computer Science

Numerical Methods

CSCI 361 / 761

Spring 2018

Instructor: Dr. Sateesh Mane

Midterm 1 Spring 2018

due Friday March 2, 2018, 11:59 pm

- **NOTE: It is the policy of the Computer Science Department to issue a failing grade to any student who either gives or receives help on any test.**
- This is an **open-book** test.
- **Any problem to which you give two or more (different) answers receives the grade of zero automatically.**
- This is a **take home exam.**
  Please submit your solution via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
  The file name should have either of the formats:

  `StudentId_first_last_CS361_midterm1_Feb2018`

  `StudentId_first_last_CS761_midterm1_Feb2018`

  Acceptable file types are txt, doc/docx, pdf (also cpp, with text in comment blocks).
- **In all questions where you are asked to submit programming code, programs which display any of the following behaviors will receive an automatic F:**
  1. Programs which do not compile successfully (compiler warnings which are not fatal are excluded, e.g. use of deprecated features).
  2. Array out of bounds.
  3. Dereferencing of uninitialized variables (including null pointers).
  4. Operations which yield NAN or infinity, e.g. divide by zero, square root of negative number, etc. *Infinite loops.*
  5. Programs which do NOT implement the public interface stated in the question.
- **In addition, note the following:**
  1. Programs which compile and run successfully but have memory leaks will receive a poor grade (but not F).
  2. All debugging and/or output statements (e.g. `cout` or `printf`) will be commented out.
  3. Program performance will be tested solely on function return values and the values of output variable(s) in the function arguments.
  4. In other words, program performance will be tested solely via the public interface presented to the calling application. (I will write the calling application.)

1 Question 1

- The **Catalan numbers** are a special case of the binomial coefficients, for integers $n \geq 0$:
$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{2n+1}\binom{2n+1}{n}. \tag{1.1}$$
- The **Fuss–Catalan numbers** are a generalization of the Catalan numbers, defined via
$$A(m,p,r) = \frac{r}{mp+r}\binom{mp+r}{m}. \tag{1.2}$$
- We shall consider only integers $m \geq 0$, $p \geq 0$ and $r \geq 0$.
- Note that $C(n) = A(n,2,1)$.
- **Write a C++ function to compute the value of $A(m,p,r)$.**

```
int FussCatalan(int m, int p, int r);
```

- Note the following:
  1. Return 0 if $m < 0$ or $p < 0$ or $r < 0$ or $mp + r < m$.
  2. If $m = 0$ then return 1.
  3. Else compute the value of $A(m,p,r)$ using **integer arithmetic only.**
- **Use your code to compute the value of $A(m,p,r)$ and fill the following table.**

  Solution.
  Go up to $r = 20$ just to demonstrate the reach of the algorithm using 4–byte integers on a 32–bit machine.

m	p	r	$A(m,p,r)$
10	2	10	10015005
10	2	11	15737865
10	2	12	24192090
10	2	13	36463440
10	2	14	53993940
10	2	15	78676884
10	2	16	112971936
10	2	17	160043576
10	2	18	223926516
10	2	19	309722116
10	2	20	423830264

- **Your code will be tested to verify that it returns the values you display in the above table. Your code will be also tested using other values for m, p and r.**

*** SEE NEXT PAGE ***

1.1 Analysis & efficient coding

- **The purpose of Question 1 is to test your ability to code the calculation in a way that guards against overflow.**
- **Recall LCM$(a, b) = ab/$gcd(a, b) and LCM$(a, b) = a(b/$gcd$(a, b))$.**
- **On a 32–bit computer, the above inputs will cause overflow if coded naïvely (but see below)**
- **However, on a 64–bit computer, inefficient algorithms will yield the correct numbers, without overflow.**
- ***** You will be graded on the quality of your coded algorithm, not only on the final numbers. *****
- **This class is very much about analysis not mere number crunching.**

1.2 Integer arithmetic

- This is a comment/explanation of the statement in Question 1:
  "Else compute the value of $A(m, p, r)$ using **integer arithmetic only.**"
- Like the binomial coefficients, the Fuss–Catalan numbers are integers (if m, p and r are integers).
- As stated above, the purpose of the question is to test your ability to compute the numbers by writing an algorithm which guards against overflow.
- **Therefore the use of floating–point numbers is disallowed.**
- **All calculations should be performed using `int` or `long` variables.**
- **Using `int` or `long` (4 bytes), on a 32–bit computer, the above inputs will cause overflow if coded naïvely.**
- **However, using `long long` (8 bytes) on a 32–bit computer, inefficient algorithm will still yield the correct numbers.** *(I know, I tried.)*
- **Therefore it is not enough to compute the correct values.**
  1. You will be graded on the quality of your algorithm.
  2. Solutions which require `long long` on a 32–bit computer (or which require a 64–bit computer to succeed), or which employ floating–point numbers, will receive a poor grade.
  3. **For any given computer architecture, find an algorithm which will return correct answers for input values as large as possible.**

Solution: comments

- **A solution which gets up to $r = 16$ using integer arithmetic with 4–byte iutegers is good enough to pass the exam question.**
- **This question seemed to inspire quite a bit of effort out of you.**
- **Many students came up with clever solutions which went beyond $r = 16$.**
- **I was pleased with their responses.**

Naïve factorials: failure

- Excluding invalid cases and also $m = 0$, the Fuss–Catalan number can be expressed as a finite product as follows.
- We first compute $n = mp + r$ and save it.
- Then we must compute $\binom{n}{m}$, where

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \,. \tag{1.3}$$

  1. **Note that $\binom{n}{m} = \binom{n}{n-m}$.**
  2. **Hence to avoid wasted computation, <u>if $m > n - m$ we swap $m = n - m$.</u>**
  3. **This is important.**

- The naïve solution now is to multiply the numerator terms and denominator terms and divide.

$$\binom{n}{m} = \frac{n(n-1)\dots(n-m+1)}{1 \times 2 \times \cdots \times m} \,. \tag{1.4}$$

```
int FussCatalan(int m, int p, int r)
{
  if ((m < 0) || (p < 0) || (r < 0) || (m*p+r < m)) return 0;
  if (m == 0) return 1;

  int n = m*p+r;
  if (m > n-m) m = n-m;

  int num = 1;
  int den = 1;
  for (int i = 1; i <= m; ++i) {
    num *= (n+1-i);
    den *= i;
  }
  int binom = num/den;

  return (r*binom)/n;
}
```

- **This leads to a large value for `num` and overflow.**
- **<u>This solution does not work, on a 32–bit computer using 4–byte integers.</u>**
- It may work on a 64–bit computer, for the inputs stated in the question.
- However, other algorithms will outperform it, on a 64–bit computer, for larger inputs.

Numerical algorithm: finite product

- Excluding invalid cases and also $m = 0$, the Fuss–Catalan number can be expressed as a finite product as follows.
- **We first compute $n = mp + r$ and save it.**
- Then we must compute $\binom{n}{m}$, where

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}\,. \tag{1.5}$$

  1. **Note that $\binom{n}{m} = \binom{n}{n-m}$.**
  2. **Hence to avoid wasted computation, if $m > n - m$ we swap $m = n - m$.**
  3. **This is important.**

- Then we are left with a finite product

$$A(m,p,r) = \frac{r}{n}\,\frac{n(n-1)\dots(n-m+1)}{1\times 2\times\cdots\times m} = \frac{r}{m}\,\frac{(n-1)\dots(n-m+1)}{1\times 2\times\cdots\times(m-1)}\,. \tag{1.6}$$

- **We cancel the factor of n to avoid a useless large number in the numerator.**
- Then we are left with a product of $m-1$ terms each in the numerator and the denominator.
  1. ***Do not multiply all the terms in the numerator product and divide by $(m-1)!$***
  2. **This will lead to a huge numerator product = overflow weakness.**
- Multiply the product this way:

$$\begin{aligned}
&\texttt{b} = (n-1)/1 = n-1\,,\\
&\texttt{b} = \texttt{b} * (n-2)/2 = (n-1)(n-2)/2!\,,\\
&\texttt{b} = \texttt{b} * (n-3)/3 = (n-1)(n-2)(n-3)/3!\,,\\
&\ \vdots\\
&\texttt{b} = \texttt{b} * (n-m+1)/(m-1) = (n-1)(n-2)(n-3)\dots(n-m+1)/(m-1)!\,.
\end{aligned} \tag{1.7}$$

- **At every step, the value of `b` is always an integer and is smaller than in the previous method.**
- The next step is to compute `r * b/m`.
- **If we do this, it gets up to $r = 17$ without overflow.**
- *Note the following:*
  1. **We cannot say `r * (b/m)` because in general m does not divide into b.**
  2. Some students made the mistake of computing `(b/m)`.
  3. *Sometimes m is not a divisor of b.*
  4. Hence some (but not all) of their numbers were wrong, for example when $r = 10$.
  5. They obtained the value **10015000** instead of **10015005**.

Code for finite product

The following solution (or equivalent) is good enough to pass the exam question.

```
int FussCatalan(int m, int p, int r)
{
  if ((m < 0) || (p < 0) || (r < 0) || (m*p+r < m)) return 0;
  if (m == 0) return 1;
  int n = m*p+r;
  if (m > n-m) m = n-m;
  int b = 1;
  for (int i = 1; i <= m-1; ++i) {
    b *= (n-i);
    b /= i;
  }
  return (r*b)/m;
}
```

Use of GCD to cancel factors

- **What follows below is "extra effort" to get up to the overflow limit `INT_MAX`.**
- **We pull out all the stops and divide as much as we can, to keep the intermediate numbers small.**
- Recall the loop

```
int b = 1;
for (int i = 1; i <= m-1; ++i) {
  b *= (n-i);
  b /= i;
}
```

- We need to work on this and keep the intermediate numbers small.

  1. We need to divide common factors between the numerator and denominator as soon as they appear.

```
int b = 1;
for (int i = 1; i <= m-1; ++i) {
  int j = i;
  int gcd1 = gcd_euclid(b,j);
  b /= gcd1;
  j /= gcd1;

  int t = n-i;
  int gcd2 = gcd_euclid(t,j);
  t /= gcd2;
  j /= gcd2;

  b *= t;
  b /= j;
}
```

  2. We cannot mess up the loop variable i so we define a local variable $j = i$.
  3. We factor out a common divisor `gcd1` between b and j.
  4. We define a local variable $t = n - 1$ and factor out a common divisor `gcd2` between t and j.
  5. Then after all the the above, we multiply the (reduced value of) b by t and divide by the (reduced value of) j.

- *But we are not done yet.*
- We can play the same trick with the last calculation `r*b/m`.

```
  int gcd3 = gcd_euclid(b,m);
  b /= gcd3;
  m /= gcd3;
  int gcd4 = gcd_euclid(r,m);
  r /= gcd4;
  m /= gcd4;
  return (r*b)/m;
```

- We divide out a common factor `gcd3` between b and m.
- We divide out a common factor `gcd4` between r and the reduced value of m.
- Then we return `r*b/m`.
- In fact, if you think about it, the final reduced value of m equals 1.

```
int FussCatalan(int m, int p, int r)
{
  if ((m < 0) || (p < 0) || (r < 0) || (m*p+r < m)) return 0;
  if (m == 0) return 1;

  int n = m*p+r;
  if (m > n-m) m = n-m;
  int b = 1;
  for (int i = 1; i <= m-1; ++i) {
    int j = i;
    int gcd1 = gcd_euclid(b,j);
    b /= gcd1;
    j /= gcd1;
    int t = n-i;
    int gcd2 = gcd_euclid(t,j);
    t /= gcd2;
    j /= gcd2;
    b *= t;
    b /= j;
  }
  int gcd3 = gcd_euclid(b,m);
  b /= gcd3;
  m /= gcd3;
  int gcd4 = gcd_euclid(r,m);
  r /= gcd4;
  m /= gcd4;
  return (r*b)/m;
}
```

- ***Do you see that* `gcd_euclid()` *is not just useless homework?***
- **Do you also see that it pays to do the homework assignments?**
- **The homework assignments give direct practice for the exams.**

- **This gets up to $r = 25$ without overflow.**
- **This is the best we can do.**
- **For $r = 26$, the value is larger than 2^{32} and cannot be stored using a 4–byte integer.**
- **Hence we have squeezed out as much as we can from the algorithm.**
- **It is fast and accurate.**

m	p	r	$A(m,p,r)$
10	**2**	**10**	**10015005**
10	**2**	**11**	**15737865**
10	**2**	**12**	**24192090**
10	**2**	**13**	**36463440**
10	**2**	**14**	**53993940**
10	**2**	**15**	**78676884**
10	**2**	**16**	**112971936**
10	**2**	**17**	**160043576**
10	**2**	**18**	**223926516**
10	**2**	**19**	**309722116**
10	**2**	**20**	**423830264**
10	**2**	**21**	**574221648**
10	**2**	**22**	**770755843**
10	**2**	**23**	**1025551163**
10	**2**	**24**	**1353412788**
10	**2**	**25**	**1772326270**

Alternative looping

- There is another way to compute the product, and that is to start from $(n-m+1)$ and go up to $(n-1)$.

$$\begin{aligned}
\texttt{b} &= (n-m+1)/1 = n-m+1\,,\\
\texttt{b} &= \texttt{b}*(n-m+2)/2 = (n-m+1)(n-m+2)/2!\,,\\
&\vdots\\
\texttt{b} &= \texttt{b}*(n-1)/(m-1) = (n-m+1)(n-m+2)(n-m+3)\dots(n-1)/(m-1)!\,.
\end{aligned} \tag{1.8}$$

- The reasoning is as follows.

  1. Consider the computation of $\binom{10}{10}$. Of course we know the answer equals 1.
  2. Suppose we multiply it out the bone–headed way. We obtain the following.

$$\begin{aligned}
\texttt{binom} &= 10/1 = 10\,,\\
\texttt{binom} &= \texttt{binom}*9/2 = 45\,,\\
\texttt{binom} &= \texttt{binom}*8/3 = 120\,,\\
\texttt{binom} &= \texttt{binom}*7/4 = 210\,,\\
\texttt{binom} &= \texttt{binom}*6/5 = 252\,,\\
\texttt{binom} &= \texttt{binom}*5/6 = 210\,,\\
\texttt{binom} &= \texttt{binom}*4/7 = 120\,,\\
\texttt{binom} &= \texttt{binom}*3/8 = 45\,,\\
\texttt{binom} &= \texttt{binom}*2/9 = 10\,,\\
\texttt{binom} &= \texttt{binom}*1/10 = 1\,.
\end{aligned} \tag{1.9}$$

  3. The final value is 1 but the product grows to 252 in the middle.
  4. Hence the product grows large at intermediate steps before becoming small again.
  5. This is an overflow weakness.
  6. If we loop over the numerator terms in the opposite direction we obtain the following.

$$\begin{aligned}
\texttt{binom} &= 1/1 = 1\,,\\
\texttt{binom} &= \texttt{binom}*2/2 = 1\,,\\
\texttt{binom} &= \texttt{binom}*3/3 = 1\,,\\
\texttt{binom} &= \texttt{binom}*4/4 = 1\,,\\
\texttt{binom} &= \texttt{binom}*5/5 = 1\,,\\
\texttt{binom} &= \texttt{binom}*6/6 = 1\,,\\
\texttt{binom} &= \texttt{binom}*7/7 = 1\,,\\
\texttt{binom} &= \texttt{binom}*8/8 = 1\,,\\
\texttt{binom} &= \texttt{binom}*9/0 = 1\,,\\
\texttt{binom} &= \texttt{binom}*1/10 = 1\,.
\end{aligned} \tag{1.10}$$

  7. **The final result is the same *but the intermediate numbers are smaller.***

Recurrence relation

- **Another possibility is to employ the recurrence relation for Fuss–Catalan numbers (see Wikipedia).**

$$A(m,p,r) = A(m,p,r-1) + A(m-1,p,p+r-1)\,. \tag{1.11}$$

- **This works, and avoids overflow.**
- **However, it is wasteful of memory and slow, if there are duplicate calculations.**

$$\begin{aligned} A(10,2,r) &= A(10,2,r-1) + A(9,2,r+1) \\ &= A(10,2,r-2) + \boldsymbol{A(9,2,r)} + \boldsymbol{A(9,2,r)} + A(8,2,r+2) \\ &= \dots \end{aligned} \tag{1.12}$$

- One should store an internal lookup table of Fuss–Catalan numbers, to avoid duplicated calculations.
- Since there are three indices m, p and r, this table will occupy a lot of memory, in general.

- **Nevertheless, slow and wasteful of memory though it may be, it gets up to $r = 25$ without overflow.**

m	p	r	$A(m,p,r)$
10	**2**	**10**	**10015005**
10	**2**	**11**	**15737865**
10	**2**	**12**	**24192090**
10	**2**	**13**	**36463440**
10	**2**	**14**	**53993940**
10	**2**	**15**	**78676884**
10	**2**	**16**	**112971936**
10	**2**	**17**	**160043576**
10	**2**	**18**	**223926516**
10	**2**	**19**	**309722116**
10	**2**	**20**	**423830264**
10	**2**	**21**	**574221648**
10	**2**	**22**	**770755843**
10	**2**	**23**	**1025551163**
10	**2**	**24**	**1353412788**
10	**2**	**25**	**1772326270**

A different approach

- One student came up with an alternative solution which took me by surprise.
- Build up and store an array of binomial coefficients.
- An unusual twist in the final line, *but necessary.*

```
int FussCatalan(int m, int p, int r) {
    int n = m*p + r;

    if (m < 0 || p < 0 || r < 0 || n < m)
        return 0;
    if (m == 0)
        return 1;

    // Binomial Coefficient (n, m)
    long binomial_coe[m+1];
    for (int i = 0; i < m + 1; i++)
        binomial_coe[i] = 0;
    binomial_coe[0] = 1;
    for (int i = 1; i <= n; i++) {
      for (int j = std::min(i, m); j > 0; j--)
            binomial_coe[j] += binomial_coe[j-1];
    }

    int result = (r * (binomial_coe[m] / n)) + (binomial_coe[m] % n * r / n);
    return result;
}
```

- **It gets up to $r = 23$, not all the way up to $r = 25$, but still good.**

2 Question 2

- You are given two arrays a and b of real numbers and a real number x.
- Both arrays have equal length n.
- You are required to sum the b_i over all values of i such that $a_i > x$:

$$S = \left[\sum_{i=1}^{n} b_i \right]_{a_i > x} . \tag{2.1}$$

- You are also given that the array a is sorted in ascending order.
- You are also given that the values of the elements a_i are distinct, so $a_i < a_j$ if $i < j$.
- **Write an efficient algorithm to compute the sum S in eq. (2.1).**
- The function signature is

```
double absum(int n, const double *a, const double *b, double x);
```

- Return the value of S in eq. (2.1).
- Return 0 if $n \leq 0$ and/or if $a_i \leq x$ for all values of i.
- **Recall that in C++ the indexing of arrays begins at 0.**

Remarks

- The total number of terms to sum is the same, in all the algorithms.
- The difference is in the number of comparisons.
- The naïve way to code is to loop from $i = 0$ to $n - 1$ and test `a[i]` every time.

```
double sum = 0;
for (int i = 0; i < n; ++i) {
  if (a[i] > x) {
    sum += b[i];
  }
}
return sum;
```

- **The above solution received no credit.**
- **It requires two comparisons per step (for i and `a[i]`), hence $2n$ overall.**
- We can do better than this.
- **Once we find a value of i such that `a[i] > x`, all the higher values of i will pass.**
- We sum from $j = i$ to $n - 1$ and stop testing the values of `a[i]`.

```
double sum = 0;
for (int i = 0; i < n; ++i) {
  if (a[i] > x) {
    for (int j = i; j < n; ++j) { sum += b[j]; }
    break;
  }
}
return sum;
```

- Alternative code: break out of the first loop when `a[i] > x`.

```
int i = 0;
while ((i < n) && (a[i] <= x)) { ++i; }
double sum = 0;
for (int j = i; j < n; ++j) { sum += b[j]; }
return sum;
```

- **On average there are $n/2$ terms in the sum.**
  1. Either way, there are two comparisons per step until we find `a[i] > x`, then one comparison per step.
  2. **Hence totally $2(n/2) + (n/2) = 3n/2$ comparisons, on average.**

- **Replace the first step (search for smallest `a[i] > x`) by a binary search.**
  1. Several students implemented binary searches.
  2. This is adapted from a (nice) student solution.

```
// binary search for smallest value of i such that a[i] > x
int i = 0;
if (a[0] <= x) {
  int left = 0;
  int right = n;
  int mid = (left+right)/2;
  while (left < mid){
    if (a[mid] > x){
      //we should start further left, so shrink right bracket
      right = mid;
    }
    else{
      //otherwise, we need to go further right, so shrink left bracket
      left = mid;
    }
    mid = (left + right)/2;
  }
  i = left+1;
}
double sum = 0;
for (int j = i; j < n; ++j) {
  sum += b[j];
}
return sum;
```

- **On average there are $n/2$ terms in the sum.**
  1. The binary search requires $\log_2(n)$ steps, and two comparisons per step.
  2. **Hence totally $(n/2) + 2\log_2(n)$ comparisons, on average.**
- ***Improving, we are getting there.***

- **Why loop forwards? Why not loop backwards?**
- *Who says we must drive in forward gear only?*
- *Put it in reverse and take off down the highway.*
- **Loop from the top down. Instead of looping forward and rejecting unwanted terms, we loop from the top down and sum the terms. As soon as one term fails, we break out of the loop. We do not loop over unwanted terms.**
- **Loop from $i = n - 1$ down to $i = 0$ and break as soon as $i < 0$ or `a[i] < x`.**

```
double sum = 0;
for (int i = n-1; (i >= 0) && (a[i] > x); --i) {
  sum += b[i];
}
return sum;
```

- **On average there are $n/2$ terms in the sum.**
- **Two tests per step, hence totally $2(n/2) = n$ comparisons, on average.**
- Note that we require two tests per step because we must test for $i \geq 0$.
  1. **Split the tests.**
  2. Test if `a[0] > x`. If yes, sum all the terms (no test for `a[i]`).
  3. Else loop from the top down, we only need to test `a[i] > x` (one comparison per step).

```
double sum = 0;
if (a[0] > x) {
  for (int i = 0; i < n; ++i) {
    sum += b[i];
  }
}
else {
  for (int i = n-1; a[i] > x; --i) {
    sum += b[i];
  }
}
return sum;
```

- **On average there are $n/2$ terms in the sum.**
- **One test per step, hence totally $n/2$ comparisons (maybe $1 + (n/2)$), on average.**
- **The number of comparisons equals the number of terms to sum.**
- ***How to beat that?***

3 Question 3

- The value of π can be computed via `const double pi = 4.0*atan2(1.0,1.0);`
- The Bessel function $J_0(x)$ can be computed by evaluating the following integral:

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta)\,d\theta\,. \tag{3.1}$$

- In this question we approximate $J_0(x)$ and compute the following sum and its derivative:

$$S(x) = \frac{1}{N}\sum_{j=0}^{N-1} \cos\Bigl(x\,\sin\frac{j\pi}{N}\Bigr)\,. \tag{3.2a}$$

$$S'(x) = \frac{dS}{dx} = -\frac{1}{N}\sum_{j=0}^{N-1} \sin\Bigl(x\,\sin\frac{j\pi}{N}\Bigr)\,\sin\frac{j\pi}{N}\,. \tag{3.2b}$$

- **Set $N = 20$ in this question.**
- **We shall employ the Newton–Raphson algorithm to compute a root of $S(x) = 0$.**
- **Take the first 4 and last 4 digits of your student id. Define η_a and η_b as follows:**

$$\eta_a = \frac{\text{first 4 digits of your student id}}{10^4}\,, \qquad \eta_b = \frac{\text{last 4 digits of your student id}}{10^4}\,. \tag{3.3}$$

- For example if your student id is 23054617, then

$$\eta_a = 0.2305\,, \qquad \eta_b = 0.4617\,. \tag{3.4}$$

- **Fill in the iterates in the tables below.**
- **Iterate until the value of x (solution for the root) converges to 3 decimal places.**
- You can state the values of $S(x_i)$ to two significant figures, in each row.

1. **Use $x_0 = 1.0 + \eta_a$ for the initial iterate.**

i	x_i	$S(x_i)$
0	$x_0 = 1.0 + \eta_a$	
1	x_1	
$\vdots$	$\vdots$	
	converged to 3 d.p.	

2. **Use $x_0 = 4.2 + \eta_b$ for the initial iterate.**

i	x_i	$S(x_i)$
0	$x_0 = 4.2 + \eta_b$	
1	x_1	
$\vdots$	$\vdots$	
	converged to 3 d.p.	

- *The function $J_0(x)$ has an infinite number of roots. These are approximations for the two smallest positive roots.*

Solution

- The difficulty in publishing a solution is that each student has a different id.
- Hence the starting point for the iteration is different for each student.
- I chose the values "$1.0+\eta_a$" and "$4.2+\eta_b$" because I looped through your student ids and Newton–Raphson iteration was stable for every student in the class.
- The iteration formula is

$$x_{i+1} = x_i - \frac{S(x_i)}{S'(x_i)}\,. \tag{3.5}$$

- The results below are for a student id of 50005000, $\eta_a = \eta_b = 0.5$.
- **Use $x_0 = 1.0 + \eta_a$ for the initial iterate.**

i	x_i	$S(x_i)$
0	**1.5**	**0.511828**
1	**2.41736**	**-0.00648926**
2	**2.40479**	**1.7267e-005**
3	**2.40483**	**1.19399e-010**

- **Use $x_0 = 4.2 + \eta_b$ for the initial iterate.**

i	x_i	$S(x_i)$
0	**4.7**	**-0.269331**
1	**5.66506**	**0.0485264**
2	**5.51714**	**-0.00100033**
3	**5.52008**	**-2.63405e-007**
4	**5.52008**	**-1.8463e-014**

- The results below are for a student id of 25007500, $\eta_a = 0.25$ and $\eta_b = 0.75$.
- **Use $x_0 = 1.0 + \eta_a$ for the initial iterate.**

i	x_i	$S(x_i)$
0	**1.25**	**0.645906**
1	**2.51494**	**-0.055781**
2	**2.40187**	**0.00153289**
3	**2.40482**	**9.3585e-007**
4	**2.40483**	**3.50781e-013**

- **Use $x_0 = 4.2 + \eta_b$ for the initial iterate.**

i	x_i	$S(x_i)$
0	**4.95**	**-0.193829**
1	**5.55279**	**0.0110943**
2	**5.51997**	**-3.69002e-005**
3	**5.52008**	**-3.62315e-010**
4	**5.52008**	**-3.33067e-017**

- ***Your numbers for the "converged values" might be (very slightly) different.***

4 Question 4

- We employ the same function as in Question 3.

$$S(x) = \frac{1}{N}\sum_{j=0}^{N-1} \cos\Bigl(x\,\sin\frac{j\pi}{N}\Bigr)\,. \tag{4.1}$$

- **Set $N = 20$ in this question.**
- **Compute the value of $S(x)$ to four decimal places and fill the table below.**
  Solution:

x	$S(x)$
0	**1**
1	**0.7651**
2	**0.2238**
3	**-0.2601**
4	**-0.3972**
5	**-0.1776**
6	**0.1506**
7	**0.3001**
8	**0.1716**
9	**-0.0904**
10	**-0.2460**

- **For each case, does the interval yield an initial bracket for the bisection algorithm?**
  Justify your answer in each case.
  This is part of the "analysis" part of this question.

  Solution. A short explanation is all that is required.

x_{low}	x_{high}	**initial bracket**	**reason**	**alternate expression**
0	2	**no**	$S(0)$ **and** $S(2)$ **same sign**	$S(0)S(2) > 0$
0	4	**yes**	$S(0)$ **and** $S(4)$ **opposite sign**	$S(0)S(4) < 0$
0	6	**no**	$S(0)$ **and** $S(6)$ **same sign**	$S(0)S(6) > 0$
0	8	**no**	$S(0)$ **and** $S(8)$ **same sign**	$S(0)S(8) > 0$
0	10	**yes**	$S(0)$ **and** $S(10)$ **opposite sign**	$S(0)S(10) < 0$
1	3	**yes**	$S(1)$ **and** $S(3)$ **opposite sign**	$S(1)S(3) < 0$
1	5	**yes**	$S(1)$ **and** $S(5)$ **opposite sign**	$S(1)S(5) < 0$
1	7	**no**	$S(1)$ **and** $S(7)$ **same sign**	$S(1)S(7) > 0$
1	9	**yes**	$S(1)$ **and** $S(9)$ **opposite sign**	$S(1)S(9) < 0$
4	6	**yes**	$S(4)$ **and** $S(6)$ **opposite sign**	$S(4)S(6) < 0$
4	8	**yes**	$S(4)$ **and** $S(8)$ **opposite sign**	$S(4)S(8) < 0$
4	10	**no**	$S(4)$ **and** $S(10)$ **same sign**	$S(4)S(10) > 0$
5	7	**yes**	$S(5)$ **and** $S(7)$ **opposite sign**	$S(5)S(7) < 0$
5	9	**no**	$S(5)$ **and** $S(9)$ **same sign**	$S(5)S(9) > 0$
6	8	**no**	$S(6)$ **and** $S(8)$ **same sign**	$S(6)S(8) > 0$
6	10	**yes**	$S(6)$ **and** $S(10)$ **opposite sign**	$S(6)S(10) < 0$

- **Use the bisection algorithm to compute the smallest of the roots bracketed in the above tables.**
  1. **Select initial values for x_{low} and x_{high} which yield the smallest bracket for the root.**
     (a) **You will lose credit if you do not use the smallest bracket which encloses the root.**
     (b) **This is part of the "analysis" part of this question.**
  2. Let x_0 be the initial value of x_{low}.
  3. Let x_1 be the initial value of x_{high}.
  4. Then $x_2 = (x_0 + x_1)/2$.
  5. Fill in the iterates in the table below.
  6. **Iterate until the value of x (solution for the root) converges to 3 d.p.**
  7. You can state the values of $S(x_i)$ to two significant figures, in each row.

- ***The above statements provoked utter panic and confusion.***
- This is what I had in mind.
  1. Read through the first table of numbers (values of $S(x)$ for $x = 0, 1, \ldots, 10$).
  2. Clearly $S(x)$ is a continuous function of x.
  3. Hence if the value of $S(x)$ changes sign, there is a root.
     (a) The value of $S(x)$ changes sign from $x = 2$ to 3.
     (b) *Hence there is at least one root between $x = 2$ and 3.*
     (c) The value of $S(x)$ changes sign from $x = 5$ to 6.
     (d) *Hence there at least one root between $x = 5$ and 6.*
     (e) The value of $S(x)$ changes sign from $x = 8$ to 9.
     (f) *Hence there at least one root between $x = 8$ and 9.*
- **This how we get started (find an initial bracket) for the bisection algorithm.**
  1. We compute the function value at a few arbitrarily selected points.
  2. In the above case I used equally spaced points.
  3. But in another context they could be powers of 2, for example $1, 2, 4, \ldots$ or $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$.
  4. The initial scan of points all depends on context.
- **Hence the first table shows you the approximate locations of the roots.**
  1. The root(s) between $x = 2$ and 3 are necessarily smaller than the roots between $x = 5$ and 6 or between $x = 8$ and 9.
  2. Think about it.
  3. **Hence the smallest root lies between $x = 2$ and 3.**
- **Therefore the smallest interval which brackets the smallest root is $(2, 3)$.**
  1. ***It has length 1, which we can read off from the first table.***
  2. **It is the shortest interval which brackets the smallest root.**

x	$S(x)$	
0	**1**	
1	**0.7651**	
2	**0.2238**	**positive to negative**
3	**-0.2601**	**change of sign: there is a root(s) between 2 and 3**
4	**-0.3972**	
5	**-0.1776**	**negative to positive**
6	**0.1506**	**change of sign: there is a root(s) between 5 and 6**
7	**0.3001**	
8	**0.1716**	**positive to negative**
9	**-0.0904**	**change of sign: there is a root(s) between 8 and 9**
10	**-0.2460**	

- In principle, there could be an even smaller root between $x = 0$ and 2, if we computed $S(x)$ using a finer scan of values of x.
- However, there is not enough data in the first table in Q4 to justify such a claim.
- **Bisection can locate a root only if the function value changes sign at the ends of an interval, and we have no such information using only $x = 0$, 1 and 2.**

- **Several students misunderstood and interpreted my instructions to mean "select an interval from the second table" in which case the answer is $x_0 = 1$ and $x_1 = 3$.**
  1. **Hence no points off for using $(1, 3)$ as the initial bracket.**
  2. ***But there will be a "frown of disapproval" for using any other initial bracket.***
- **Understand how to start off a bisection calculation.**
- **Solution using $x_0 = 2$ and $x_1 = 3$.**
  I let it run to 15 iterations.
  For 3 d.p. you can stop at 11 or 12 or 13 iterations.
- **The question does not ask which side is updated at every itertion, but I included the information anyway.**

i	x_i	$S(x_i)$	
0	**2**	**0.2238**	$S(x_0)$ **= positive**
1	**3**	**-0.2601**	$S(x_1)$ **= negative**
2	**2.5**	**-0.0483838**	$S(x_2)S(x_0) < 0$ **negative so update** x_{high}
3	**2.25**	**0.0827499**	$S(x_3)S(x_0) > 0$ **positive so update** x_{low}
4	**2.375**	**0.0155784**	$S(x_4)S(x_0) > 0$ **positive so update** x_{low}
5	**2.4375**	**-0.0168457**	$S(x_5)S(x_0) < 0$ **negative so update** x_{high}
6	**2.40625**	**-0.000739276**	$S(x_6)S(x_0) < 0$ **negative so update** x_{high}
7	**2.39062**	**0.00739379**	$S(x_7)S(x_0) > 0$ **positive so update** x_{low}
8	**2.39844**	**0.00332073**	$S(x_8)S(x_0) > 0$ **positive so update** x_{low}
9	**2.40234**	**0.00128909**	$S(x_9)S(x_0) > 0$ **positive so update** x_{low}
10	**2.4043**	**0.000274494**	$S(x_{10})S(x_0) > 0$ **positive so update** x_{low}
11	**2.40527**	**-0.000232494**	$S(x_{11})S(x_0) < 0$ **negative so update** x_{high}
12	**2.40479**	**2.09745e-005**	$S(x_{12})S(x_0) > 0$ **positive so update** x_{low}
13	**2.40503**	**-0.000105766**	$S(x_{13})S(x_0) < 0$ **negative so update** x_{high}
14	**2.40491**	**-4.23975e-005**	$S(x_{14})S(x_0) < 0$ **negative so update** x_{high}
15	**2.40485**	**-1.07119e-005**	$S(x_{15})S(x_0) < 0$ **negative so update** x_{high}

- *If you have done your work correctly, you should obtain the same answer as in Question 3.*
- **Solutions which employed multiple initial brackets and computed different roots suffered loss of credit.**
- **Recall the statement on the first page:**
  Any problem to which you give two or more (different) answers receives the grade of zero automatically.

- **A graph of $J_0(x)$ is shown in Fig. 1, up to $x = 20$.**
- **It is not part of the question, just for you to view.**
- **The Bessel function $J_0(x)$ oscillates forever, and cuts zero infinitely many times.**

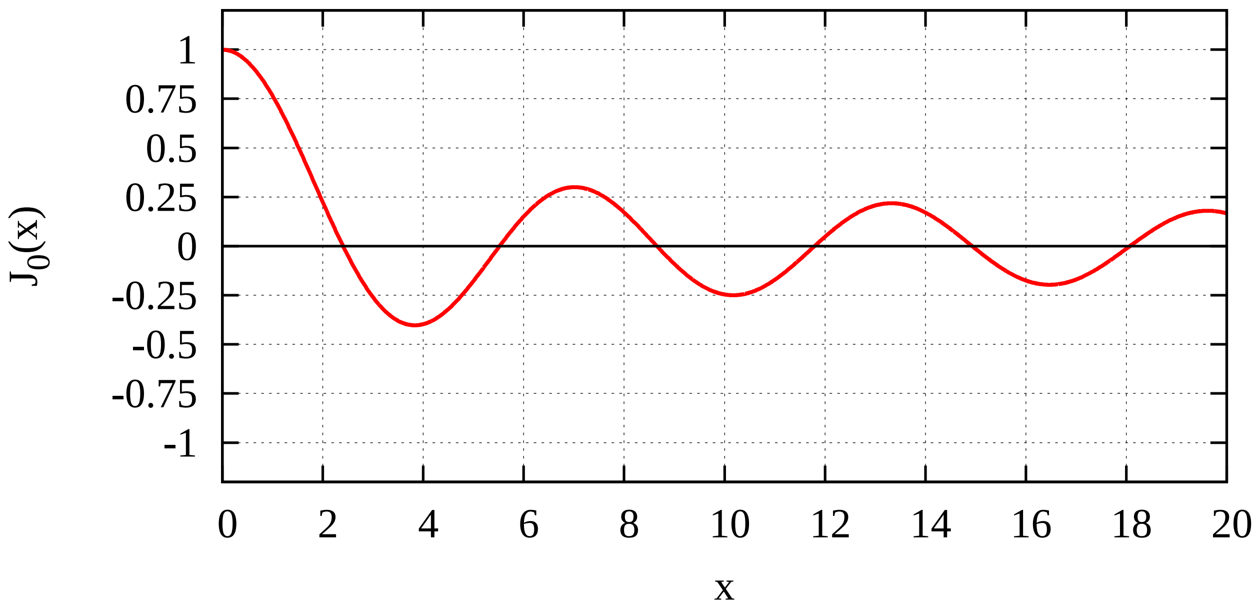

Figure 1: Graph of $J_0(x)$, up to $x = 20$.

5 Question 5

- Consider the function $f(x)$ graphed in Fig. 2 below.
- We are given the following information:
  1. The function $f(x)$ is continuous everywhere except at $x = 0$.
  2. The function $f(x)$ diverges at $x = 0$.
  3. The function $f(x)$ is negative and never equals zero for $x \leq x_0$.
  4. The function $f(x)$ is negative and never equals zero for $x \geq x_5$.
  5. The function $f(x)$ has three negative roots at α, β and γ, where $\alpha < \beta < \gamma < 0$.
  6. The function $f(x)$ ***equals zero but does not cross zero*** at $x = \beta$.
  7. The function $f(x)$ has one positive root at $x = \varepsilon$.
  8. The location of the divergence is indicated by δ, where $\delta = 0$.
- **For each case, does the interval yield an initial bracket for the bisection algorithm?** ***Justify your answer in each case.***

x_{low}	x_{high}	**initial bracket**	**reason**	**roots/discontinuity**
x_0	x_1	**yes**	$f(x_0)f(x_1) < 0$	$\{\alpha\}$
x_0	x_2	**yes**	$f(x_0)f(x_2) < 0$	$\{\alpha, \beta\}$
x_0	x_3	**no**	$f(x_0)f(x_3) > 0$	
x_0	x_4	**yes**	$f(x_0)f(x_4) < 0$	$\{\alpha, \beta, \gamma, \delta\}$
x_0	x_5	**no**	$f(x_0)f(x_5) > 0$	
x_1	x_2	**no**	$f(x_1)f(x_2) > 0$	
x_1	x_3	**yes**	$f(x_1)f(x_3) < 0$	$\{\beta, \gamma\}$
x_1	x_4	**no**	$f(x_1)f(x_4) > 0$	
x_1	x_5	**yes**	$f(x_1)f(x_5) < 0$	$\{\beta, \gamma, \delta, \epsilon\}$
x_2	x_3	**yes**	$f(x_2)f(x_3) < 0$	$\{\gamma\}$
x_2	x_4	**no**	$f(x_2)f(x_4) > 0$	
x_2	x_5	**yes**	$f(x_2)f(x_5) < 0$	$\{\gamma, \delta, \epsilon\}$
x_3	x_4	**yes**	$f(x_3)f(x_4) < 0$	$\{\delta\}$
x_3	x_5	**no**	$f(x_3)f(x_5) > 0$	
x_4	x_5	**yes**	$f(x_4)f(x_5) < 0$	$\{\epsilon\}$
x_3	$x_5 + 1000$	**no**	$f(x_3)f(x_5 + 1000) > 0$	
$x_0 - 1000$	x_4	**yes**	$f(x_0 - 1000)f(x_4) < 0$	$\{\alpha, \beta, \gamma, \delta\}$

- **For each case above which yields a valid initial bracket, state the set of roots (and/or discontinuity) to which the bisection algorithm would possibly converge, if an iteration were performed. Write your answer as "$\{\alpha, \beta\}$" or $\{\beta, \gamma, \delta\}$, etc.**
- ***Do not attempt to run a bisection iteration.***
- **Do NOT attempt to use a ruler to "measure" or estimate the values of** $x_0, \ldots, x_5$ **and/or** α, β, **etc.** *There will be NO credit (and considerable disrespect) for such foolishness.*

- **The concept of a "valid bracket" only requires that**
  the function values have opposite signs at the two ends.
- **It does not matter if the function is discontinuous in between.**
- **Bisection can converge to the location of a discontinuity: $\{\delta\}$ is a valid answer.**
- **Bisection can converge to any root or discontinuity in the initial bracket.**
  1. **I have explained this in class, more than once.**
  2. **Bisection can converge to β.**
  3. **For example if the initial bracket is (x_0, x_2) and $\beta = (x_0 + x_2)/2$.**
  4. **But bisection can converge to β only by accident, if an iterate equals β up to a tolerance.**
  5. **I did not specify a tolerance in this question, so an iterate would have to equal β exactly.**
  6. ***But that is possible, there is no reason why not.***
- ***The question warned specifically not to try to "measure" the graph.***
  1. ***Do not make assumptions as to which roots/discontinuity in the interval bisection can reach.***
  2. **Bisection can reach any of them.**
- **Conversely, bisection cannot converge to a root or discontinuity outside the initial bracket.**
- **This should be obvious.**

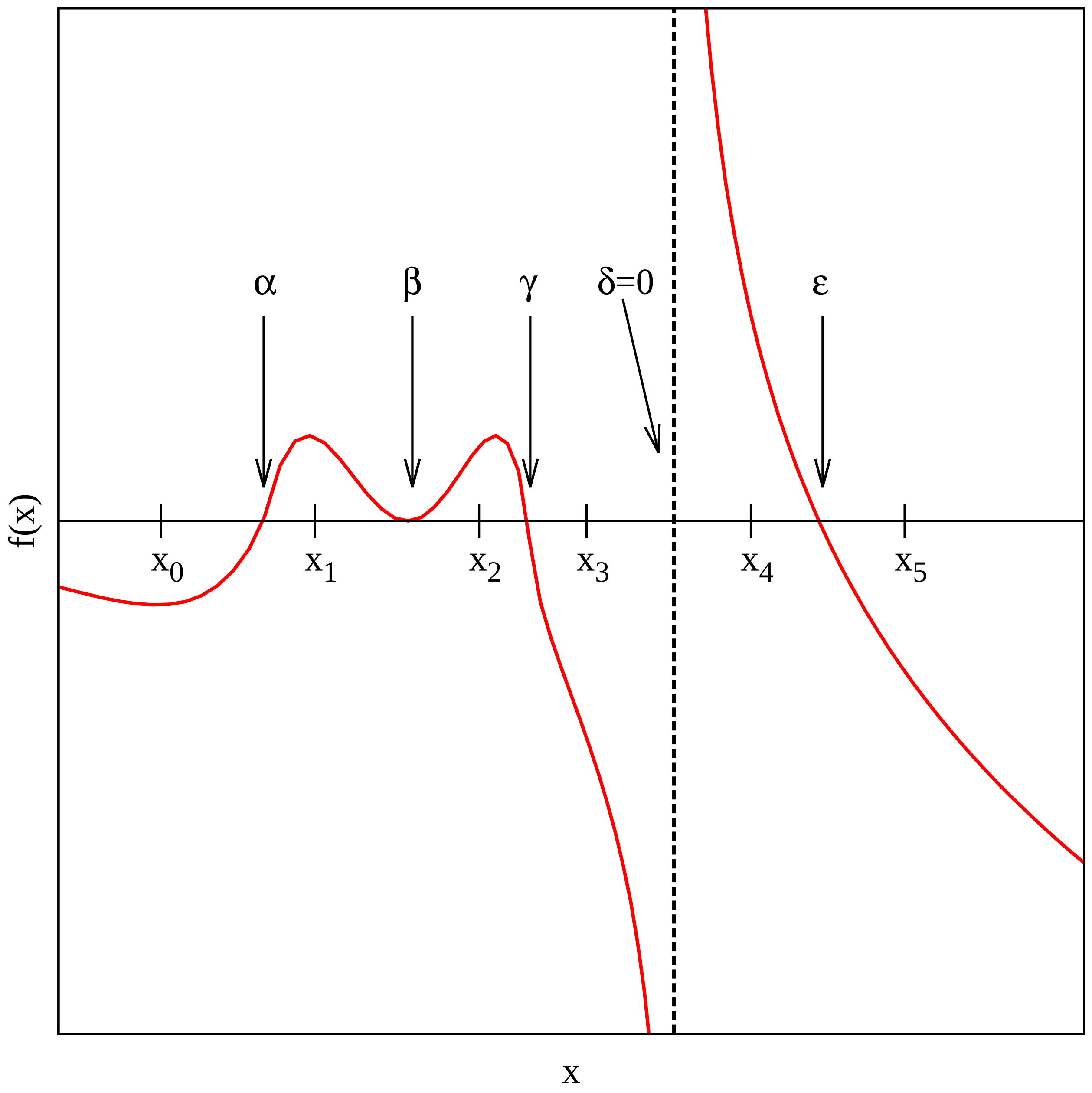

Figure 2: Graph of function $f(x)$ in Question 5.

6 Question 6

- In this question, we shall compute the convex hull of a set of points in a plane.
- The points are determined as follows:
  1. **Take the 8 digits of your student id and form four pairs as follows:**
  $$\begin{aligned}(x_1, y_1) &= (\text{digit } 1, \text{digit } 2)\,, \\ (x_2, y_2) &= (\text{digit } 3, \text{digit } 4)\,, \\ (x_3, y_3) &= (\text{digit } 5, \text{digit } 6)\,, \\ (x_4, y_4) &= (\text{digit } 7, \text{digit } 8)\,.\end{aligned} \tag{6.1}$$
  2. For example if your student id is 23054617, then
  $$\begin{aligned}(x_1, y_1) &= (2, 3)\,, \\ (x_2, y_2) &= (0, 5)\,, \\ (x_3, y_3) &= (4, 6)\,, \\ (x_4, y_4) &= (1, 7)\,.\end{aligned} \tag{6.2}$$
  3. **Form a set of 16 points as follows $(x_i \pm 0.2, y_i \pm 0.2)$ (loop $i = 1, 2, 3, 4$):**
  $$(x_i+0.2, y_i+0.2)\,, \quad (x_i-0.2, y_i+0.2)\,, \quad (x_i+0.2, y_i-0.2)\,, \quad (x_i-0.2, y_i-0.2)\,. \tag{6.3}$$
  4. If your student id is 23054617, then the points are
  $$(2\pm0.2, 3\pm0.2)\,, \quad (0\pm0.2, 5\pm0.2)\,, \quad (4\pm0.2, 6\pm0.2)\,, \quad (1\pm0.2, 7\pm0.2)\,. \tag{6.4}$$
  5. *If you have done your work correctly, you will have a set of sixteen distinct points.*
- **Compute the convex hull of the 16 points $(x_i \pm 0.2, y_i \pm 0.2)$, $i = 1, 2, 3, 4$.**
- **Plot a graph of the points and the convex hull.**
- ***** Revised question *** No need to submit code as part of your answer**
- **There is no way to display a meaningful solution.**
- **It is a different convex hull for each of you.**
  1. **For most students the convex hull had 7 points.**
  2. **For most others the convex hull had 6 points.**
  3. **There were a few for whom the convex hull had 5 points.**
  4. **There were a few for whom the convex hull had 8 points.**

7 Question 7: Mandatory for graduate students (optional for undergraduates)

- The Taylor series for a (sufficiently differentiable) function $f(x+a)$ is

$$f(x+a) = f(a) + xf'(a) + \frac{x^2}{2!}f''(a) + \cdots + \frac{x^n}{n!}f^{(n)}(a) + R_n\,. \tag{7.1}$$

- Here R_n is the remainder term.
- Hence for the trigonometric functions, we obtain

$$\sin(a+x) = \sin(a) + x\,\cos(a) - \frac{x^2}{2!}\sin(a) - \frac{x^3}{3!}\cos(a) + \dots \tag{7.2a}$$
$$\cos(a+x) = \cos(a) - x\,\sin(a) - \frac{x^2}{2!}\cos(a) + \frac{x^3}{3!}\sin(a) + \dots \tag{7.2b}$$

- **Set $a = 45^\circ = \pi/4$ and $x = 1^\circ = \pi/180$.**
- Note that $\tan(\theta) = \sin(\theta)/\cos(\theta)$ for any angle θ.
- **Using eqs. (7.2a) and (7.2b), compute the value of $\tan(46^\circ)$ to four decimal places.**

$$\begin{aligned}
\sin(46^\circ) &= \sin(45^\circ + 1^\circ) \\
&= \sin\left(\frac{\pi}{4} + \frac{\pi}{180}\right) \\
&= \frac{1}{\sqrt{2}}\left[1 + \frac{\pi}{180} - \frac{1}{2}\left(\frac{\pi}{180}\right)^2 - \frac{1}{6}\left(\frac{\pi}{180}\right)^3 + \dots\right] \\
&\simeq 0.707107\left[1 + 0.017453 - 1.52309\times10^{-4} - 8.86096\times10^{-7}\right] \\
&\simeq 0.71934\,, \\
\cos(46^\circ) &= \cos(45^\circ + 1^\circ) \\
&= \cos\left(\frac{\pi}{4} + \frac{\pi}{180}\right) \\
&= \frac{1}{\sqrt{2}}\left[1 - \frac{\pi}{180} - \frac{1}{2}\left(\frac{\pi}{180}\right)^2 + \frac{1}{6}\left(\frac{\pi}{180}\right)^3 + \dots\right] \\
&\simeq 0.707107\left[1 - 0.017453 - 1.52309\times10^{-4} + 8.86096\times10^{-7}\right] \\
&\simeq 0.69466\,, \\
\tan(46^\circ) &= \frac{\sin(46^\circ)}{\cos(46^\circ)} \simeq \frac{0.71934}{0.69466} \simeq 1.0355\,.
\end{aligned}$$

- We can also expand $\tan(a+x)$ directly in a Taylor series.
- To save time, you are given, for $a = \pi/4$:

$$\tan\left(\tfrac{1}{4}\pi + x\right) = 1 + 2x + 2x^2 + \frac{8}{3}x^3 + \dots \tag{7.4}$$

- **Using eq. (7.4), compute the value of $\tan(46^\circ)$ to four decimal places.**

$$\begin{aligned}
\tan(46^\circ) = \tan(45^\circ + 1^\circ) = \tan\left(\frac{\pi}{4} + \frac{\pi}{180}\right) \\
&= 1 + 2\frac{\pi}{180} + 2\left(\frac{\pi}{180}\right)^2 + \frac{8}{3}\left(\frac{\pi}{180}\right)^3 + \dots \\
&\simeq 1 + 0.034907 + 6.09235\times10^{-4} + 1.41775\times10^{-5} \\
&\simeq 1.035529997 \\
&\simeq 1.0355\,.
\end{aligned}$$

- *If you have done your work correctly, you should obtain the same answer in both cases.*
- **Using a calculator, or otherwise, obtain the value of $\tan(46^\circ)$ and estimate the value of the remainder term R_n for your calculation above.**

$$\tan(46^\circ) \simeq 1.035530314\,.$$

$$R_n \simeq 1.035530314 - 1.035529997 \simeq 3.16\times10^{-7}\,.$$

- *If you have done your work correctly, you should obtain $R_n = O(10^{-7})$.*
- ***Do not worry if you obtained a different (small) number for R_n.***
- **This was intended as a short simple exercise, do not worry over small details.**