# CS3911 Introduction to Numerical Methods with Fortran Exam 1 Solutions

1. **Fortran 90 Problems**

   In this problem, `L`, `M`, `N` and `K` are `INTEGER` variables. Answer the following questions. **Note that you should provide all computation details and elaborate computation steps. Otherwise, you will receive no point.**

   (a) [**3 points**] What are the results of `L**1/2` and `L**(1/2)`, where `L` is 2?

   **Solution:** Since operator `**` has a higher priority than `/`, we have `L**1/2` = `(L**1)/2` = `(2**1)/2` = `2/2` = `1`. Since both operands in `1/2` are `INTEGER`s, `1/2` is `0`. As a result, `L**(1/2)` = `L**0` = `2**0` = `1`. ∎

   (b) [**3 points**] What is the result of `L**M**N`, where `L`, `M` and `N` are 2, 3 and 2, respectively?

   **Solution:** Since `**` is right associative, `L**M**N` = `L**(M**N)` = `2**(3**2)` = `2**9` = `512`. ∎

   (c) [**3 points**] Suppose we have the following type specification:

   ```
   CHARACTER(LEN=4) :: A = "abcde", B = "+-*/", C = "xyz"
   ```

   What is the result of `A(2:3) = C(2:2) // B(2:3) // C(3:)`? Use a box to indicate a space if your result has spaces.

   **Solution:** `C(2:2)` means the substring of `C` from position `2` to position `2`, the result is `"y"`. `B(2:3)` consists of characters of `B` from position `2` to position `3`, which is `"-*"`. `C(3:)` means the substring of `C` from position `3` to the last. Since the length of `C` is 4 and since `C` is initialized with `"xyz"`, a space is added to the right and `C` is `"xyz□"`. Thus, `C(3:)` is `"z□"` with a space at the end. The concatenation `C(2:2) // B(2:3) // C(3:)` yields `"y" // "-*" // "z "` = `"y-*z□"`. However, since `A(2:3)` has a length of 2 characters, when storing a length 5 string `"y-*z "` into `A(2:3)` the last 3 characters are truncated. Thus, `A(2:3)` only receives `"y-"`. Note that `A`'s initial value is longer than 4 characters and is also truncated to `"abcd"`. As a result, the assignment yields `"ay-d"`. ∎

   (d) [**3 points**] Suppose we have the following `READ(*,*)` statements:

   ```
   READ(*,*) L, M
   READ(*,*)
   READ(*,*) N
   READ(*,*) K
   ```

   Suppose the input files contains the following lines:

   ```
   1 2 3 4
   5 6 7 8
   9 10 11 12
   13 14 15 16
   17 18 19 20
   ```

   What values `L`, `M`, `N` and tt K will have after the execution of the `READ(*,*)`'s?

**Solution:** The first `READ(*,*)` reads `1` and `2` into variables `L` and `M`, respectively. The second `READ(*,*)` skips the second input line. The third `READ(*,*)` starts reading with the third line and reads `9` into variable `N`. Finally, the fourth `READ(*,*)` reads `13` on the fourth line into `K`. In summary, `L`, `M`, `N` and `K` receive `1`, `2`, `9` and `13`, respectively. ∎

(e) [**3 points**] Declare a `REAL` variable that is good to hold at most 20 significant digits. Then, how do you write 3.1415926 in this type?

**Solution:** The following shows a possible type specification, where `X` is a `REAL` variable with the indicated precision:

```
INTEGER, PARAMETER :: LONG20 = SELECTED_REAL_KIND(20)
REAL(KIND=LONG20)  :: X
```

With the `KIND` of `LONG20` specified, a constant is written as `3.1415926_LONG20`. ∎

2. [**15 points**] If $x$ and $x^*$ are an approximation and the true result, respectively, then

$$-\lceil \log_{10} |x - x^*| \rceil$$

is approximately the number of accurate digits, where $\lceil x \rceil$ is the ceiling operator which rounds $x$ to the smallest integer larger than or equal to $x$. Prove this claim with a convincing argument. **A vague argument is considered incorrect.**

**Solution:** Since this is an absolute error, $x$ and $x^*$ are less than 1. As a result, we may assume they can be represented as $x = 0.a_1a_2\cdots a_{k-1}a_ka_{k+1}\cdots$ and $x^* = 0.b_1b_2\cdots b_{k-1}b_kb_{k+1}\cdots$. Note that we assume $x$ and $x^*$ are both positive. The case of both being negative is similar.

Let $k = -\lceil \log_{10} |x - x^*| \rceil$. Since $\lceil x \rceil$ rounds $x$ to the smallest integer that is larger than of equal to $x$, we have $x \le \lceil x \rceil$, and, hence, $\log_{10} |x - x^*| \le \lceil \log_{10} |x - x^*| \rceil$. Multiplying both sides by $-1$ yields $-\log_{10} |x - x^*| \ge -\lceil \log_{10} |x - x^*| \rceil = k$. Therefore, we have $-k \ge \log_{10} |x - x^*|$ and $10^{-k} \ge |x - x^*|$.

Without loss of generality, we assume $x - x^* > 0$. If $x - x^* < 0$, the use of $x^* - x > 0$ yields the same conclusion. Let us deal with the case of $x - x^* = 10^{-k}$ first. Note that $10^{-k} = 0.\underbrace{00\cdots0}_{k-1\ 0\text{'s}}1$. The diagram below shows the calculation of $x - x^* = 10^{-k}$. In this way, it is obvious that the first $k-1$ digits in $x$ and $x^*$ are identical.

$$\begin{array}{cccccccccc}
 & 0 & . & a_1 & a_2 & \cdots & a_{k-1} & a_k & a_{k+1} & \cdots \\
- & 0 & . & b_1 & b_2 & \cdots & b_{k-1} & b_k & b_{k+1} & \cdots \\
\hline
 & 0 & . & 0 & 0 & \cdots & 0 & 1 & & \\
\end{array}$$

Now consider the case of $x - x^* < 10^{-k}$. A value less than $10^{-k}$ has more than $k-1$ leading zeros after the decimal point. As a result, the subtraction becomes the following, where $x - x^* = 0.\underbrace{00\cdots0}_{k\ 0's}c_{k+1}c_{k+2}\cdots$ and $c_1$ may or may not be 0. In this case, $x$ and $x^*$ have at least $k$ identical digits.

$$\begin{array}{cccccccccc}
 & 0 & . & a_1 & a_2 & \cdots & a_{k-1} & a_k & a_{k+1} & \cdots \\
- & 0 & . & b_1 & b_2 & \cdots & b_{k-1} & b_k & b_{k+1} & \cdots \\
\hline
 & 0 & . & 0 & 0 & \cdots & 0 & 0 & c_{k+1} & \cdots \\
\end{array}$$

In summary, $-\lceil \log_{10} |x - x^*| \rceil$ is approximately the number of identical digits of $x$ and $x^*$. You may use a similar argument to show that $-\lceil \log_{10} (|x - x^*|/|x^*|) \rceil$ is approximately the number of identical digits for relative error. ∎

3. [**15 points**] Suppose you are given the following expression:

$$\ln \left[ \frac{\sqrt{1+x^2} - x}{\sqrt{1+x^2} + x} \right]$$

What are the major numerical problem when evaluating this expression if $x$ is very large? Do your best to rewrite this expression to avoid these problems and provide a convincing arguments showing your way is indeed trouble free. **Note that vague and/or no argument is considered incorrect.**

**Solution:** If $x$ is large, actually not very large, $1 + x^2$ is close to $x^2$ and $\sqrt{1+x^2}$ is close $x$. As a result, cancellation can occur in evaluating $\sqrt{1+x^2} - x$. This is *the* problem rather than overflow, because cancellation occurs well before overflow can happen.

Here is an elaboration. Without loss of generality, $x$ is assumed to have a form of $x = 10^p$. There is no problem with this assumption because one can always round $x$ to its closest power of 10. Since $x = 10^p = 1\underbrace{00\cdots0}_{p\ 0\text{'s}}$, $x^2 = (10^p)^2 = 10^{2p} = 1\underbrace{00\cdots0}_{2p\ 0\text{'s}}$ and $1 + x^2 = 1\underbrace{00\cdots0}_{2p-1\ 0\text{'s}}1$. Suppose the computer can only hold $k$ digits. If $1 + x^2$ has $2p + 1 > k$ digits, $1 + x^2$ is truncated to $1\underbrace{00\cdots0}_{2p\ 0\text{'s}}$, which is identical to $x^2$. As a result, $\sqrt{1+x^2} \approx x$, which makes the evaluation of function ln() to fail.

We know that the condition $2p + 1 > k$ causes truncation followed by cancellation. Since $2p+1 > k$, we have $p > (k-1)/2$, and the smallest $p$ that satisfies $2p+1 > k$ is $\lceil (k-1)/2 \rceil$. Therefore, if $k$ is the maximum number of digits a computer can hold, $x = 10^p$ will cause cancellation in evaluating $\sqrt{1+x^2} - x$, where $p = \lceil (k-1)/2 \rceil$. On the other hand, $10^{p-1}$ is the largest power of 10 without cancellation.

To calculate this expression properly without cancellation, apply the usual trick that was discussed in class as follows:

$$\frac{\sqrt{1+x^2} - x}{\sqrt{1+x^2} + x} = \frac{\sqrt{1+x^2} - x}{\sqrt{1+x^2} + x} \times \frac{\sqrt{1+x^2} + x}{\sqrt{1+x^2} + x} = \frac{\left(\sqrt{1+x^2}\right)^2 - x^2}{\left(\sqrt{1+x^2} + x\right)^2} = \frac{1 + x^2 - x^2}{\left(\sqrt{1+x^2} + x\right)^2} = \frac{1}{\left(\sqrt{1+x^2} + x\right)^2}$$

The given expression becomes

$$\ln \left[ \frac{\sqrt{1+x^2} - x}{\sqrt{1+x^2} + x} \right] = \ln \left[ \frac{1}{\left(\sqrt{1+x^2} + x\right)^2} \right] = -2 \ln \left( \sqrt{1+x^2} + x \right)$$

In this way, cancellation is no more a problem.

We may reduce the problem further. If the computation is performed on a $k$-digit computer, $x^2$ is a number of maximum $2k$ digits, and $1 + x^2 \approx x^2$. Therefore, $\sqrt{1+x^2} \approx x$, and

$\sqrt{1+x^2}+x \approx 2x$. The given expression has an approximation for large $x$ as follows:

$$\ln\left[\frac{\sqrt{1+x^2}-x}{\sqrt{1+x^2}+x}\right] = -2\ln(\sqrt{1+x^2}+x) \approx -2\ln(2x)$$

The table below shows results computed with the default `REAL` (*i.e.*, single precision) type. It is clear from the table that for $x \leq 1000$ both methods yield very similar results; but $-2\ln(2x)$ has large error because it is for large $x$. If $x$ is 10000 and larger, the numerator is nearly zero, and floating-point exception (FPE in the table) occurs in the `LOG()` function. On the other hand, the method without cancellation (column 5) still works fine, and the approximation $-\ln(2x)$ (column 6) delivers essentially the same results!

| $x$ | $\sqrt{1+x^2}-x$ | $\sqrt{1+x^2}+x$ | $\ln\left[\frac{\sqrt{1+x^2}-x}{\sqrt{1+x^2}+x}\right]$ | $-2\ln\left(\sqrt{1+x^2}+x\right)$ | $-2\ln(2x)$ |
|---|---|---|---|---|---|
| 1 | 0.41421353 | 2.4142136 | -1.762747 | -1.762747 | -1.386294 |
| 10 | 0.04987526 | 20.049875 | -5.996453 | -5.996445 | -5.991464 |
| 100 | 4.9972534E-3 | 200.005 | -10.597209 | -10.596684 | -10.596635 |
| 1000 | 4.8828125E-4 | 2000.0005 | -15.225522 | -15.201805 | -15.201805 |
| 10000 | $\approx 0$ | 20000 | FPE | -19.806974 | -19.806974 |
| 100000 | $\approx 0$ | 200000 | FPE | -24.412146 | -24.412146 |
| 1000000 | $\approx 0$ | 2000000 | FPE | -29.017315 | -29.017315 |
| 10000000 | $\approx 0$ | 20000000 | FPE | -33.622486 | -33.622486 |
| 100000000 | $\approx 0$ | 200000000 | FPE | -38.227657 | -38.227657 |
| 1000000000 | $\approx 0$ | 2000000000 | FPE | -42.832824 | -42.832824 |
| 10000000000 | $\approx 0$ | 20000000000 | FPE | -47.437995 | -47.437995 |

The cancellation showed above can be justified easily. Since single precision has about 7 to 8 significant digits (*i.e.*, $k = 8$), $p = \lceil (k-1)/2 \rceil = 4$ and $10^4 = 10,000$ causes cancellation.

With this simple example, I hope you understand the impact of cancellation. Thus, claiming that overflow could occur is unfounded because floating-point exception happens much earlier than overflow. More importantly, the cancellation problem occurs when $x \approx 10,000$, a small number! ∎

4. **[15 points]** A programmer tried to verify a formula he learned in calculus:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

As $n$ approaches infinity, this limit approaches $e = 2.71828\ldots$. So, he wrote a Fortran 90 program with a line like the following, where `N` are `X` are `INTEGER` and `REAL` variables, respectively.

```
X = (1+1/REAL(N))^N
```

Then, he used various `N` and printed the computed result `X`. The following table is a summary in which the first column shows the values of $n$ used, the second column is computed by Fortran 90 intrinsic function `EXP(REAL(n))`, the third column is computed by $(1 + 1/n)^n$, and the fourth column is the difference between the Fortran 90 and the computed results. Carefully study the table, he found a very odd phenomenon. The limit seems correct up to $n = 10,000$, then the computed results go up and down, and by $n = 10,000,000$ and larger, the computed result becomes 1. Of course, he knew there has to be some numerical issues in this computation. Do your best to pinpoint the major problem. **Note that you have to detail your findings. A vague answer such as "because of rounding, cancellation, etc" is not considered correct. You have to provide an accurate account of the problem.**

| $n$ | `EXP(REAL(n))` | $\left(1+\frac{1}{n}\right)^n$ | Difference |
|---|---|---|---|
| 10 | 2.7182817 | 2.5937431 | 0.1245387 |
| 100 | 2.7182817 | 2.7048113 | 0.0134704 |
| 1000 | 2.7182817 | 2.7170508 | 0.0012310 |
| 10000 | 2.7182817 | 2.7185969 | 0.0003152 |
| 100000 | 2.7182817 | 2.7219622 | 0.0036805 |
| 1000000 | 2.7182817 | 2.5952268 | 0.1230550 |
| 10000000 | 2.7182817 | 3.2939677 | 0.5756860 |
| 100000000 | 2.7182817 | 1.0000000 | 1.7182817 |
| 1000000000 | 2.7182817 | 1.0000000 | 1.7182817 |

**Solution:** If the result of $(1 + \frac{1}{n})^n$ becomes 1, it is because $1/n$ being small so that rounding/truncation causes $1+1/n$ to become 1. This is a correct observation. But, simply stating the cause is $n$ being so large is too vague and too general. In other words, you only have an "observation" or a hunch rather than an accurate assessment, because you did not answer the question "how large is large enough" or "is $n = 1,000,000,000$ large enough."

Suppose the above is carried out on a $k$-digit computer. If $n = 10^k$, then $1/n = 0.\underbrace{00\ldots0}_{k-1\ 0\text{'s}}1$ and $1+1/n = 1.\underbrace{00\ldots0}_{k-1\ 0\text{'s}}1$. After truncation to $k$ digits, $1+1/n$ is 1! As a result, $(1+1/n)^n$ is 1 for $n \geq 10^k$. The computation in the above table is performed with single precision which has 7 to 8 significant digits, and, hence this problem occurs around $100,000,000 = 10^8$. Note that $10^8$ is neither very large nor very small in floating-point representation. Claiming that very large $n$ caused the indicated problem is inaccurate and in fact is wrong. The real answer is truncation due to finite precision.

Some claimed that $1/n$ for large $n$ could cause underflow. This is true; but, it is a generic answer and is irrelevant to this question because the indicated problem occurs well before underflow can occur. Moreover, since all modern CPU's can represent floating-point numbers in the range $[10^{-64}, 10^{64}]$ and some CPU's have a wider range, $n = 1,000,000,000$ is actually not a large number at all. ∎

5. **[20 points]** Use the bisection method to solve $f(x) = e^{-x^2} - \sin(x) = 0$ in the range of $[0, 1]$. It is known that there is one and only one root in $[0, 1]$. **You do not have to solve**

**this equation completely. What you have to do is to carry out three iterations of the bisection method (*i.e.*, reducing the length $[a, b]$ from 1 to 0.125). Fill the computed values into the following table. Note that each value must have five or more significant digits. Otherwise, you will risk low grade.** Without doing so, you will receive **zero** point.

| Interval Length | $a$ | $b$ | $f(c)$ where $c = (a + b)/2$ |
|---|---|---|---|
| 1 | 0 | 1 | 0.2993752 |
| 0.5 | 0.5 | 1.0 | -0.1118559 |
| 0.25 | 0.5 | 0.75 | 0.0915365 |
| 0.125 | 0.625 | 0.75 | -0.0126277 |

Your results may be slightly different from the above due to the number of significant digits involved in computation. However, they should be very close. ∎

6. **[20 points]** The graph of equation $f(x) = e^{-x^2} - \sin(x) = 0$ shows that there is only one root in [0,1]. Hence, an initial value of $x_0 = 1$ is selected to find this root. Use Newton's method with initial value $x_0 = 1$ to solve this equation. **You do not have to solve this equation completely. What you have to do is to carry out three iterations of Newton's method (*i.e.*, computing $x_1$, $x_2$ and $x_3$). Fill $x_1$, $x_2$ and $x_3$ into the following table. Note that each value must have five or more significant digits. Otherwise, you will risk low grade.**

| $i$ | $x_i$ | $f(x_i)$ |
|---|---|---|
| 0 | 1 | -0.4735915 |
| 1 | 0.6288646 | 0.0851361 |
| 2 | 0.6802876 | $0.5074143 \times 10^{-3}$ |
| 3 | 0.6805981 | $\approx 0$ |