CS3911 Introduction to Numerical Methods with Fortran Exam 1 Solutions

1. **Fortran 90 Problems**
   Answer the following Fortran 90 related questions. **You should provide all computation details and elaborate computation steps. Otherwise, you will receive no point.**

   (a) [**5 points**] What are the results of `2**1/2` and `2**(1/2)`?
   Solution: Since operator `**` has a higher priority than `/` does, we have `2**1/2` = `(2**1)/2` = `(2**1)/2` = `2/2` = `1`. Since both operands in `1/2` are `INTEGER`s, `1/2` is `0`. As a result, `2**(1/2)` = `2**0` = `1`. ∎

   (b) [**5 points**] What is the result of `2**3**2`?
   Solution: Since `**` is right associative, `2**3**2` = `2**(3**2)` = `2**9` = `512`. ∎

   (c) [**5 points**] Suppose we have the following type specification:

   ```
   CHARACTER(LEN=4) :: A = "opqrst", B = "1234", C = "@#$"
   ```

   What is the content of `A` after the execution of `A(2:3) = C(3:)  // B(:3) // C(4:4)`? Use a box to indicate a space if your result has spaces.
   Solution: Since all character strings are of length 4, the content of `A`, `B` and `C` are `"opqr"`, `"1234"` and `"@#$□"`, respectively.

   - Since `C(3:)` means the substring of `C` from position 3 to the end (*i.e.*, `C(3:4)`), the result is `"$□"`.
   - Since `B(:3)` means from the beginning to position 3 (*i.e.*, `B(1:3)`), the result is `"123"`.
   - Since `C(4:4)` is position 4, the result is `"□"`.

   Thus, `C(3:)  // B(:3) // C(4:4)` yields `"$□" // "123" // "□"` = `"$□123□"`. However, since `A(2:3)` has a length of 2, when storing a length 6 string `"$□123□"` into `A(2:3)` the last 4 characters are truncated. Thus, `A(2:3)` only receives `"$□"`. Consequently, the assignment yields `"o$□r"`. ∎

   (d) [**5 points**] Suppose `x` is a `REAL` variable. Which one of `x**7` and `x**7.0` is better for evaluating x^7 in terms of efficiency? **Explain your calculations clearly. Providing only an answer and/or vague explanation receives zero point.**
   Solution: `x**7` is evaluated as `x*x*x*x*x*x*x` with six multiplications, while `x**7.0` is evaluated as $e^{7\log(x)}$, where log() is the natural logartithm. Therefore, `x**7` is more efficient as it does not involve transendantal function calls (*i.e.*, log() and exp()). **Question:** Can you evaluate `x**7` with less than 6, say 4, multiplications? ∎

   (e) [**5 points**] Suppose we have the following `READ(*,*)` statements:

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

Suppose the input files contains the following lines:

```
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
26 27 28 29 30
31 32 33 34 35
```

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

Solution: The first `READ(*,*)` reads `11` and `12` into `L` and `M`. The second `READ(*,*)` skips the second input line. The third `READ(*,*)` starts reading with the third line and reads `21` into `N`. Finally, the fourth `READ(*,*)` reads `26` on the fourth line into `K`. Thus, `L`, `M`, `N` and `K` receive `11`, `12`, `26` and `27`, respectively. ∎

2. **Accuracy and Reliability**

(a) **[15 points]** Let $\vec{\mathbf{X}} = (x_1, x_2, \ldots, x_n)$ be an n-dimensional non-zero vector. Its "normalized" version is computed as follows:

$$\text{normalized } \vec{\mathbf{X}} = \frac{\vec{\mathbf{X}}}{|\vec{\mathbf{X}}|}$$

where $|\vec{\mathbf{X}}| = (x_1^2 + x_2^2 + \cdots + x_n^2)^{1/2}$. Design a procedure for computing $|\vec{\mathbf{X}}|$ that can avoid as many numerical issues as possible. **You may assume all x_i's have been read in and describe your computation steps clearly and precisely. A complete Fortran 90 program is <u>not</u> required. Then, explain why your method is better. Providing only a simple or vague answer receives <u>zero</u> point.**

Solution: A potential problem of implementing $|\vec{\mathbf{X}}| = (x_1^2 + x_2^2 + \cdots + x_n^2)^{1/2}$ in a straightforward way is that larger $|x_i|$'s makes x_i^2 even larger, which, in turn, can make $x_1^2 + x_2^2 + \cdots + x_n^2$ very large. As a result, overflow may be a problem. To overcome this problem, we may take the maximum K of the $|x_i|$'s (*i.e.*, $K = \max(|x_1|, |x_2|, \ldots, |x_n|)$ and scale the input down like this x_1/K, x_2/K, …, x_n/K so that $|x_i/K| \leq 1$ for all i. Then, $|\vec{\mathbf{X}}|$ is computed as follows:

$$|\vec{\mathbf{X}}| = K\sqrt{\left(\frac{x_1}{K}\right)^2 + \left(\frac{x_2}{K}\right)^2 + \cdots + \left(\frac{x_n}{K}\right)^2}$$

The remaining is obvious. ∎

(b) **[15 points] (1)** What would the potential problem(s) be in evaluating the following expression if x is large? **(2)** Suggest a way that can evaluate this expression accurately, and **show that your way is better with a clear and to-the-point explanation. (3)** What is the result approximately if x is very large? **You are supposed to provide a convincing argument. Vague and meaningless (*e.g.*, prove-by-example) answers receive <u>zero</u> point.** Note that there are **<u>three</u>** questions.

$$\frac{1}{\sqrt{x+1} - \sqrt{x}}$$

Solution: Here are the answers:

i. A potential problem is that if x is large enough, $1+x$ is close to x due to rounding or truncation. As a result, cancelation will occur in $\sqrt{1+x}-\sqrt{x}$. In fact, if the computer being used has a precision of n digits and if $x > 10^n$, then adding 1 to x has no effect. For example, if a computer has a precision of 7 digits and $x = 1234567000.0 = 0.1234567 \times 10^{10}$, then $1+x = 12345670001.0$ and after truncation (back to 7 significant digits) $1+x$ is equal to x. In this case, the computed result is 0. Thus, the potential problem is either overflow if $\sqrt{1+x}-\sqrt{x}$ is non-zero but very small, or division by zero.

ii. The given expression can be transformed as follows:

$$\frac{1}{\sqrt{1+x}-\sqrt{x}} = \frac{1}{\sqrt{1+x}-\sqrt{x}} \times \frac{\sqrt{1+x}+\sqrt{x}}{\sqrt{1+x}+\sqrt{x}} = \sqrt{1+x}+\sqrt{x}$$

In this way, cancelation will not occur. Overflow and underflow will not happen. In fact, this approach is more efficient as it only involves two additions and two square roots rather than one addition, one subtraction, two square roots and one division.

iii. From the above transformation, if x is sufficiently large, $\sqrt{1+x}+\sqrt{x}$ and the given expression is approximately $2\sqrt{x}$.

It would be very helpful if a program is used to verify this result. The following table is computed with single precision. Due to loss of significant digits (*i.e.*, cancelation), $\sqrt{1+x}-\sqrt{x}$ is 0 or the same value if $x \geq 3333333$. Because $\sqrt{1+x}-\sqrt{x}$ is inaccurate, so does $1/(\sqrt{1+x}-\sqrt{x})$. The last two columns show essentially the same results for virtually all x.

x	$\sqrt{1+x}-\sqrt{x}$	$\frac{1}{\sqrt{1+x}-\sqrt{x}}$	$\sqrt{1+x}+\sqrt{x}$	$2\sqrt{x}$
1111111	4.8828125×10^{-4}	2048	2108.1855	2108.185
2222222	$3.6221093 \times 10{-}4$	2730.6667	2981.4243	2981.4238
3333333	2.4414062×10^{-4}	4096	3651.4839	3651.4836
4444444	2.4414062×10^{-4}	4096	4126.3701	4126.3701
5555555	2.4414062×10^{-4}	4096	4714.0449	4714.0449
6666666	2.4414062×10^{-4}	4096	5163.9775	5163.9775
7777777	0	FPE	5577.7334	5577.7334
8888888	2.4414062×10^{-4}	4096	5962.8477	5962.8477
9999999	0	FPE	6324.5552	6324.5552
123456789	0	FPE	22222.223	22222.223

With this simple example, you should be able to understand the impact of cancelation on evaluating very simple expressions. ∎

3. **Non-Linear Equations**

(a) **[15 points]** Use the bisection method to solve $f(x) = \log(x) - e^{-x} = 0$ in the range of $[1, 2]$. Note that $\log(x)$ is the natural logarithm function usually written as ln() in math. It is known that there is one and only one root in $[1, 2]$. **You do not have to solve this equation completely. You only do three iterations of the bisection method (*i.e.*, reducing the length $[1, 2]$ 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.**

Interval Length	a	b	$f(c)$ where $c = (a + b)/2$
1	1	2	0.18233493
0.5	1	1.5	-0.06336126
0.25	1.25	1.5	0.065614134
0.125	1.25	1.375	2.7873516×10^{-3}

(b) **[15 points]** Solve $f(x) = \log(x) - e^{-x} = 0$ with Newton method and initial value $x_0 = 2.0$. **You do not have to solve this equation completely. You only do 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	2	0.5578119
1	1.1220196	-0.21049117
2	1.294997	-0.0153903365
3	1.3097091	-9.351969×10^{-5}

Note the derivative of $f(x) = \log(x) - e^{-x}$ is $f'(x) = \frac{1}{x} + e^{-x}$. ∎

(c) **[15 points]** Solve $f(x) = \log(x) - e^{-x} = 0$ with fixed-point iteration and initial value $x_0 = 2.0$. **You do not have to solve this equation completely. You only do three fixed-point iterations (*i.e.*, computing x_1, x_2 and x_3) with a correct transformation which must be presented clearly. 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. If your transformation causes numerical errors/issues or diverges, you receive zero point.

i	x_i
0	2
1	1.1449206
2	1.3747188
3	1.2877682

A correct transformation is by taking the exponential of both sides. This yields $x = e^{e^{-x}} = e^{(e^{-x})}$ and $g(x) = e^{e^{-x}}$. Taking natural logarithm of both sides, the transformation becomes $\log(\log(x)) = -x$ and $g(x) = -\log(\log(x))$. You may perform one iteration, and the second will cause floating-point exception because the argument of log() is negative. ∎