**NUMERICAL METHODS AND PROGRAMMING** **2024/2025**

**Error Analysis** **(PROBLEMS 2)**

**1.**— We want to evaluate with an absolute error of less than $1, cm^2$ the area of a square enclosure. To do this, one side of the enclosure is measured and the area is calculated by squaring it. It is known "a priori" that the enclosure is approximately $1\,m^2$. How accurate should the ruler used in the measurement be? Would it be possible to perform this calculation correctly on a computer using single precision?

**Sol. 1.** Given the side of the region $x$, the area can be obtained as $A = x\,x$. Only one operation is needed, thus the relative error will be:

$$r_A = 1\,r_x + 1\,r_x + r_A^A$$

We know that the relative error in $x$ can be decomposed as the inherent plus the storing error as $r_x = r_x^I + r_x^A$

Therefore:

$$r_A = 2(r_x^I + r_x^A) + r_A^A = 2r_x^I + (2r_x^A + r_A^A)$$

In order to bound the relative error we also know that the storing error is bounded by the Machine Epsilon:

$$\left|r_x^A\right| \leq r_M \qquad ; \qquad \left|r_A^A\right| \leq r_M$$

Thus:

$$|r_A| \;\leq\; 2\left|r_x^I\right| + \left(2\left|r_x^A\right| + \left|r_A^A\right|\right) \;\leq\; 2\left|r_x^I\right| + 3\,r_M$$

Since we want to evaluate the area with an absolute error smaller than 1 cm$^2$:

$$|E_A| \leq 1\ [\mathrm{cm}^2] \qquad ; \qquad \text{with } r_A = \frac{E_A}{A} \qquad \Rightarrow \qquad |r_A| \leq \frac{1\ [\mathrm{cm}^2]}{1\ [\mathrm{m}^2]} = 10^{-4}$$

The ruler precision $p$ determines the error of the measure of $x$ and thus its inherent error:

$$\left|r_x^I\right| \leq \frac{p}{1\ [\mathrm{m}]}$$

If the operations are made with infinite precision we could assume $r_M = 0$ and therefore the source of error in the area will come only from the inherent error:

$$|r_A| \leq \frac{2p}{1\ [\mathrm{m}]}$$

Which we want to be smaller than $10^{-4}$, so:

$$\frac{2p}{1\ [\mathrm{m}]} \leq 10^{-4} \ \Leftrightarrow\ p \leq (1/2)\, 10^{-4}\ [\mathrm{m}] \ = \ 0.05\ [\mathrm{mm}]$$

However, if the operations are made in single precision, we have that $r_M = (1/2)\ 2^{-24+2} = 2^{-23}$

$$\Rightarrow |r_A| \leq \frac{2p}{1} + 3\ (2^{-23}) \leq 10^{-4} \Leftrightarrow p \leq (1/2)\left(10^{-4} - 3\ (2^{-23})\right)\ [\mathrm{m}] = 0.0498\ [\mathrm{mm}]$$

Thus, for the operations using single precision in a computer, the ruler must allow measuring lengths with a maximum error of 0.0498 mm.

---

**2.**— We want to draw on a plotter the curve $y = f(x)$ for values of $x$ included in the interval $[a, b]$. For this purpose the following subroutine is created:

```
SUBROUTINE CURVE(A,B,N)
IMPLICIT REAL*4 (A-H,O-Z)
IMPLICIT INTEGER*4 (I-N)
DELTA=(B-A)/FLOAT(N)
X=A
Y=F(A)
CALL MOVE(X,Y)
DO I=1,N
  X=A+FLOAT(I)*DELTA
  Y=F(X)
  CALL DRAW(X,Y)
ENDDO
RETURN
END
```

where subroutine `MOVE(X,Y)` moves the pen without drawing to the $(x, y)$ coordinate point, subroutine `DRAW(X,Y)` moves the pen by drawing a straight line from the previous position to the $(x, y)$ point, and `F(X)` is a function of type `REAL*4 FUNCTION` that is written separately. According to the FORTRAN compiler manual, for variables of type `REAL*4`, $m = 24$ bits are allocated for storing the mantissa (sign included).

**a)** Explain very briefly how the subroutine works.

**b)** Reasonably find —in first approximation— a maximum value of $N$ beyond which the results are not improved, because the precision of the computer does not allow discriminating between two consecutive values of $x$.

**c)** Reasonably find —in first approximation— a maximum value of $N$ beyond which the results are not improved, because the precision of the computer does not allow discriminating between two consecutive values of $f(x)$.

**d)** Considering also that the plotter interprets the $(x, y)$ coordinates in centimeters, and that the maximum resolution of the pen is 0.1 millimeters, what is the maximum value of $N$ to be used in practice?

**NOTE:**Simplifications deemed reasonable may be made, since we speak of "first approximation", as long as they are justified.

**Sol. 2.a)**

The interval to be studied $[a, b]$ is discretized in $n + 1$ equally spaced points a distance $\delta = \dfrac{b-a}{n}$ and each point $(x_i, y_i)$ is obtained as $x_i = a + \delta\, i,\ y_i = f(x_i);\ i = 0, \dots, n$

The pen moves between the discrete points obtained. Graphically:

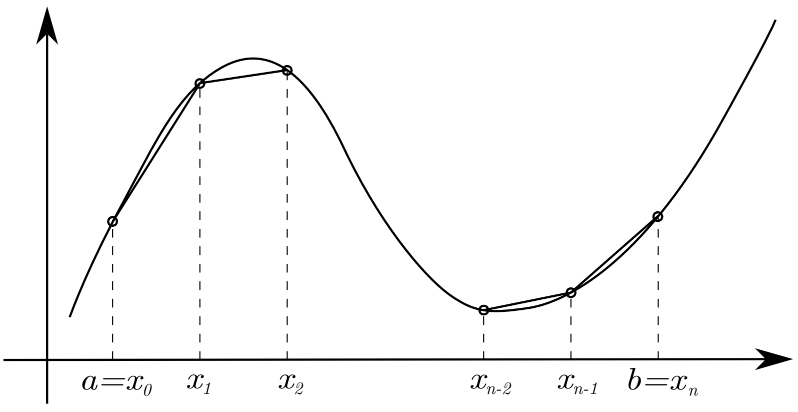

Figure 1: Plot by a piecewise polygonal.

The subroutine draws a piecewise polygonal (see figure 1).

**Sol. 2.b)**

Let $\delta$ be the space between two contiguous point

$$|x_{i+1} - x_i| = |\delta|$$

The error at point $i$ will be $E_{x_i} = x_i\, r_{x_i}$, but in first approximation we can disregard the error propagation for the determination of $x_i$, assuming that $r_{x_i} \approx r^A_{x_i}$.

We also know that:

$$|r^A_{x_i}| \le r_M = \frac{1}{2}\, 2^{-24+2} = 2^{-23}$$

Then:

$$|E_{x_i}| = |x_i\, r_{x_i}| \le |x_i|\, r_M \le \max_i |x_i|\ r_M$$

Searching the values of $\delta$ such that the computer is able to represent since it does not make any sense that $|E_{x_i}| > \delta$, so we try that:

$$\max_i |x_i|\ r_M \ \le\ |\delta|\ =\ \frac{|b-a|}{n}$$

thus:

$$\boxed{n \le N_x = \frac{|b-a|}{\max\limits_i |x_i|}\frac{1}{r_M};\quad \text{with } r_M\ =\ 2^{-23}}$$

**Sol. 2.c)**

We proceed the same way for the values of the function:

$$|y_{i+1}-y_i| = |\mu_i| = |f(x_{i+1}-f(x_i)| \approx \left|f'(x_i)\,(x_{i+1}-x_i)\right| = \left|f'(x_i)\right|\ |\delta|$$

The error will be: $E_{y_i} = y_i\, r_{y_i} = f(x_i)\, r_{y_i}$

If in first approximation we disregard the storing errors of the intermediate operations in the determination of $f(x) \Rightarrow$

$$r_{y_i} \approx \frac{f'(x_i)}{f(x_i)}\, x_i\, r_{x_i} + r^A_{y_i} \qquad ; \qquad r_{x_i} \approx r^A_{x_i} \qquad ; \qquad \left\{ \begin{array}{l} \left|f^A_{x_i}\right| \le r_M \\ \left|f^A_{y_i}\right| \le r_M \end{array}\right.$$

Therefore:

$$|E_{y_i}| \approx \left|f'(x_i)\, x_i\, r^A_{x_i} + f(x_i)\, r^A_{y_i}\right| \le \left|f'(x_i)\, x_i + f(x_i)\right|\ r_M \le \max_i \left|f'(x_i)\, x_i + f(x_i)\right|\ r_M$$

If we impose that:

$$\max_i \left|f'(x_i)\, x_i + f(x_i)\right|\ r_M \le \max_i |\mu_i| \approx \max_i \left|f'(x_i)\right|\ |\delta|$$

then:

$$\max_i \left|f'(x_i)\, x_i + f(x_i)\right|\ r_M \le \max_i \left|f'(x_i)\right| \frac{|b-a|}{n}$$

so:

$$\boxed{n \le N_y = \frac{|b-a|\ \max\limits_i |f'(x_i)|}{\max\limits_i |f'(x_i)\, x_i + f(x_i)|}\ \frac{1}{r_M}, \quad \text{with } r_M \ = \ 2^{-23}}$$

**Sol. 2.d)**

$$\left.\begin{array}{l} |x_{i+1}-x_i| = |\delta| \\ |y_{i+1}-y_i| = |\mu_i| \approx |f'(x_i)|\ |\delta| \end{array}\right\} \Rightarrow |\Delta\delta_i| = \sqrt{(x_{i+1}-x_i)^2 + (y_{i+1}-y_i)^2} \approx |\delta|\ \sqrt{1+(f'(x_i))^2}$$

We impose that:

$$0.01\text{cm} \le \max_i \sqrt{1+(f'(x_i))^2}\ |\delta| = \max_i \sqrt{1+(f'(x_i))^2}\ \frac{|b-a|}{n}$$

so:

$$\boxed{n \le N_\Delta = \frac{|b-a| \max\limits_i \sqrt{1+(f'(x_i))^2}}{0.01 \text{ cm}}}$$

In practice:

$$\boxed{n \ \le \ \min\{N_x,\ N_y,\ N_\Delta\}}$$

**3.**— Present several pathological examples of numerical operations in which it is demonstrated that in a digital computer:

**a)** "Similar" numbers must not be subtracted..

**b)** There must be no division by "small" numbers..

**c)** It is preferable to add the smallest numbers first.

**d)** The order of the factors alters the product.

Indicate what is the theoretical justification of these assertions. Show the examples in decimal system using a calculator.

---

We will use examples in decimal base using 3 significant digits. Let $\widehat{z}$ denote the obtained value corresponding to the exact value $z$.

**Sol. 3.a)**

Given two numbers $a$ and $b$ such that:

$$r_a = r_a^I + r_a^A \qquad \text{y} \qquad r_b = r_b^I + r_b^A$$

The operation subtraction and its relative error will be:

$$z = a - b \qquad \rightarrow \qquad r_z = \frac{a}{a-b} r_a + \frac{-b}{a-b} r_b + r_z^A \Rightarrow$$

$$\Rightarrow r_z = \left[ \frac{a}{a-b} r_a^I + \frac{-b}{a-b} r_b^I \right] + \left[ \frac{a}{a-b} r_a^A + \frac{-b}{a-b} r_b^A + r_z^A \right]$$

When the numbers are similar ($a \approx b$) the coefficients of error propagation are very high. The subtraction of similar numbers must be avoided since it amplifies the relative error.

Example 1: $\quad y = 1 - (1-x)(1+x) \qquad |x| << 1 \qquad$ ( 3 significant digits in base 10)
for $x = 0.100\ 10^{-2}$:

$$\begin{cases} \widehat{1-x} = 0.999\ 10^0 \\ \widehat{1+x} = 0.100\ 10^1 \\ (1-\widehat{x)(1+}x) = 0.999\ 10^0 \\ \widehat{y} = 0.1\ 10^{-2} \end{cases}$$

But actually $y = 0.1\ 10^{-5}$, then $r_y = \dfrac{y - \widehat{y}}{y} = -999 \approx -100000\%$
These problems can be avoided by simply operating the function:

$$y = 1 - (1-x)(1+x) = 1 - (1 - x^2) = x^2 \quad \Rightarrow \quad y = x^2$$

Example 2: $\quad y = \tan(x) - \sin(x) \qquad |x| << 1$
for $x = 0.100\ 10^0$:

$$\begin{cases} \widehat{\tan(x)} = 0.100\ 10^0 \\ \widehat{\sin(x)} = 0.100\ 10^0 \\ \widehat{y} = 0.000\ 10^0 \end{cases}$$

But actually $y = 0.000501255$, then $r_y = \dfrac{y - \widehat{y}}{y} = 1 \approx 100\%$

Again, these problems can be avoided operating the expression of the function:

$$y = \tan(x) - \sin(x) = \sin(x)\left(\frac{1}{\cos(x)} - 1\right) = \sin(x)\,\frac{1-\cos(x)}{\cos(x)} = \frac{\sin(x)\,\left(1-\cos^2(x)\right)}{\cos(x)\,\left(1+\cos(x)\right)} =$$

$$= \frac{\sin^3(x)}{\cos(x)\,\left(1+\cos(x)\right)}$$

An asymptotic equivalent expression could also be used:

$$\sin(x) \approx x - \frac{x^3}{3!} + \theta(x^5)$$

$$\tan(x) \approx x + \frac{x^3}{3} + \hat{\theta}(x^5)$$

$$\Rightarrow y \approx \frac{x^3}{2}$$

**Sol. 3.b)**

Given $a$ y and $b$ such that:

$$r_a = r_a^I + r_a^A \qquad \text{y} \qquad r_b = r_b^I + r_b^A$$

The operation division and its corresponding relative error are:

$$z = a/b \qquad \rightarrow \qquad r_z = \left(\frac{1}{1-r_b}\right) r_a + \left(\frac{-1}{1-r_b}\right) r_b + r_z^A \approx r_a - r_b + r_z^A \Rightarrow$$

and the absolute error:

$$E_z = z\, r_z, \qquad E_z \approx \left[\frac{a}{b}(r_a^I - r_b^I)\right] + \left[\frac{a}{b}(r_a^A - r_b^A + r_z^A)\right]$$

When $|b| << 1$ the coefficients of the absolute error propagation increase greatly.

However, in the results of the calculations, the absolute error is generally not very important. What is important is that the relative error is small (except in cases where the absolute errors must necessarily be small, as in the case of the computer drawing example where the absolute error in the coordinates of the points is what is important).

There is one case in which the absolute error is important: when the result is zero (since in this case the relative error is meaningless). For this reason it is normal that the convergence condition of an iterative algorithm is written in terms of relative error and maximum admissible absolute error in the shape:

$$\text{If } \left[|x_{k+1} - x_k| \;\leq\; \max\left(\varepsilon,\; r\,|x_{k+1}|\right)\right] \quad \rightarrow \quad \text{STOP}$$

It is preferable to program this condition in this way instead of

$$\text{If } \left[\left|\frac{x_{k+1} - x_k}{x_{k+1}}\right| \;\leq\; r\right] \quad \rightarrow \quad \text{STOP}$$

In order to avoid problems when $|x_{k+1}| \rightarrow 0$

**Sol. 3.c)**

It is preferable to add the small numbers first, so that their sum acquires a significant value before adding the large numbers..

Example: $$y = 0.1\ 10^{1} + \overbrace{0.1\ 10^{-2} + 0.1\ 10^{-2} + ... + 0.1\ 10^{-2}}^{1000 \text{ times}}$$

If the operations are performed in this order, $\hat{y} = 0.1\ 10^{1}$ is obtained, but actually $y = 0.2\ 10^{1}$. Thus, $r_y = \dfrac{y - \hat{y}}{y} = 0.5 = 50\%$

This error can be avoided by simply changing the order of operations, operating first the sum of the smallest terms and then adding it to the term of higher order.

$$y = \overbrace{0.1\ 10^{-2} + 0.1\ 10^{-2} + ... + 0.1\ 10^{-2}}^{1000 \text{ times}} + 0.1\ 10^{1}$$

**Sol. 3.d)**

The order of the factors alters the product, since the intermediate results (and their storing errors) are different.

Given $a$, $b$ and $c$ such that:

$$r_a = r_a^I + r_a^A \quad , \quad r_b = r_b^I + r_b^A \quad , \quad r_c = r_c^I + r_c^A$$

$$\Rightarrow \left\{ \begin{array}{lcl} z_1 = (a\,b)\,c & \rightarrow & r_{z_1} \approx (r_a + r_b + r_{ab}^A) + r_c + r_{z_1}^A \\ z_2 = a\,(b\,c) & \rightarrow & r_{z_2} \approx r_a + (r_b + r_c + r_{bc}^A) + r_{z_2}^A \end{array} \right.$$

It can be seen that $r_{ab}$ and $r_{bc}$ may be different, as well as $r_{z_1}^A$ and $r_{z_2}^A$.

Example:

$$\left\{ \begin{array}{lcl} z_1 = (0.5\,10^{0} \times 0.2\,10^{1}) \times\ 0.999\,10^{0} & \rightarrow & \hat{z}_1 = 0.999\,10^{0} \\ z_2 = 0.5\,10^{0} \times (0.2\,10^{1} \times\ 0.999\,10^{0}) & \rightarrow & \hat{z}_2 = 0.100\,10^{1} \end{array} \right.$$

**4.—** In a particular situation it is necessary to evaluate the following function:

$$f(x) = 1 - (1 - x)(1 + x) = x^2$$

for values of $x$ such that $|x| << 1$.

The computer used uses $m$ bits for the storage of the mantissa (including the sign) in floating point, and that rounds by approximation.

**a)** Should $f(x)$ be calculated as $1 - (1 - x)(1 + x)$ or as $x^2$? (In the latter case the calculation is done by multiplying the variable $x$ by itself).

**b)** Establish the relative error bounds in the two cases.

**c)** Which operation is best in terms of the values of $x$ ?

**d)** Present an example numerical of the convenience of using one or the other method when calculations are performed manually with the help of three significant decimal digits.

**Sol. 4.a)**

The function should be evaluated as $f(x) = x^2$, since the original expression $f(x) = 1-(1-x)(1+x)$ will amplify the relative errors when $|x| << 1$ because the term $(1-x)(1+x) \approx 1$ and a subtraction of similar numbers would be performed. For small values of $|x|$ we will get 0 as result.

**Sol. 4.b)**

$$f_1(x) = 1-(1-x)(1+x) \Rightarrow \begin{cases} A = 1-x & \rightarrow \quad r_A = \dfrac{1}{1-x}\cancel{r_1} + \dfrac{-x}{1-x} r_x + r_A^A \\ B = 1+x & \rightarrow \quad r_B = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x} r_x + r_b^A \\ C = A * B & \rightarrow \quad r_C = r_A + r_B + r_C^A \\ F1 = 1-C & \rightarrow \quad r_{F1} = \dfrac{1}{1-C}\cancel{r_1} + \dfrac{-C}{1-C} r_C + r_{F1}^A \end{cases}$$

$$\begin{aligned} \Rightarrow r_{F1} &= \frac{-(1-x)(1+x)}{1-(1-x)(1+x)} \left[ \frac{-x}{1-x} r_x + r_A^A + \frac{x}{1+x} r_x + r_B^A + r_C^A \right] + r_{F1}^A \\ &= \frac{-(1-x^2)}{x^2} \left[ \frac{-2x^2}{1-x^2} r_x + r_A^A + r_B^A + r_C^A \right] + r_{F1}^A \\ &= 2\, r_x - \frac{(1-x^2)}{x^2} [r_A^A + r_B^A + r_C^A] + r_{F1}^A \\ &= 2\, r_x^I + \left( 2\, r_x^A - \frac{(1-x^2)}{x^2} [r_A^A + r_B^A + r_C^A] + r_{F1}^A \right) \end{aligned}$$

We can bound the storing errors by the machine epsilon $r_M$:

$$\begin{aligned} |r_{F1}| &\leq 2 \left| r_x^I \right| + \left( 2 + \left| \frac{1-x^2}{x^2} \right| 3 + 1 \right) r_M \\ &= 2 \left| r_x^I \right| + 3 \left( 1 + \left| \frac{1-x^2}{x^2} \right| \right) r_M \end{aligned}$$

$$|x| << 1 \Rightarrow |r_{F1}| \leq 2 \left| r_x^I \right| + 3 \left( 1 + \frac{1-x^2}{x^2} \right) r_M = 2 \left| r_x^I \right| + \frac{3}{x^2} r_M$$

Therefore,

$$\boxed{|r_{F1}| \leq 2 \left| r_x^I \right| + \frac{3}{x^2} r_M \qquad \text{for } |x| << 1}$$

$$f_2(x) = x^2 \Rightarrow \{ \; F2 = x * x \quad \rightarrow \quad r_{F2} = r_x + r_x + r_{F2}^A$$

$$\Rightarrow r_{F2} = 2\, r_x + r_{F2}^A = 2 r_x^I + 2 r_x^A + r_{F2}^A$$

Bounding again the storing errors by the machine epsilon we can obtain a bound of the relative error:

$$\boxed{|r_{F2}| \leq 2 \; \left| r_x^I \right| + 3\, r_M}$$

**Sol. 4.c)**

Analyzing the relative error bounds for both cases it can be seen that it is always better the expression $f(x) = x^2$ because none of the terms amplifies the error highly. In contrast, the original expression greatly amplifies the storing errors of the intermediate results $A$, $B$ and $C$ for values $|x| << 1$.

**Sol. 4.d)**

$$x = 0.100\ 10^{-2} \Rightarrow \begin{cases} \hat{f}_2(x) = 0.100\ 10^{-5} & \text{valor calculado correcto} \\ \hat{f}_1(x) = 0.100\ 10^{-2} & \text{valor calculado incorrecto} \end{cases}$$

**5.—** The $n$ first powers $\varphi^1, \varphi^2, \ldots, \varphi^n$ of the golden ratio $\varphi = (-1+\sqrt{5})/2$ can be obtained without performing any multiplication, since the following relation is fulfilled:

$$\varphi^i = \varphi^{i-2} - \varphi^{i-1}.$$

Show that this way of performing calculations is numerically unstable, while repeated multiplication is numerically stable.

**Sol. 5.**

Unstable algorithm: $\qquad \varphi \ = \ (-1+\sqrt{5})/2 \ \approx \ 0.61803398875$

$$\begin{cases} x_0 = 1 & r_{x_0} = \cancel{r^I_{x_0}} + \cancel{r^A_{x_0}} = 0 \\ x_1 = \varphi & r_{x_1} = r^I_{x_1} + r^A_{x_1} \\ x_2 = x_0 - x_1 & r_{x_2} = \dfrac{x_0}{x_0 - x_1} r_{x_0} + \dfrac{-x_1}{x_0 - x_1} r_{x_1} + r^A_{x_2} = \dfrac{1}{\varphi^2} r_{x_0} - \dfrac{1}{\varphi} r_{x_1} + r^A_{x_2} \\ x_3 = x_1 - x_2 & r_{x_3} = \dfrac{x_1}{x_1 - x_2} r_{x_1} + \dfrac{-x_2}{x_1 - x_2} r_{x_2} + r^A_{x_3} = \dfrac{1}{\varphi^2} r_{x_1} - \dfrac{1}{\varphi} r_{x_2} + r^A_{x_3} \\ \quad\vdots & \quad\vdots \\ x_k = x_{k-2} - x_{k-1} & r_{x_k} = \dfrac{x_{k-2}}{x_{k-2} - x_{k-1}} r_{x_{k-2}} + \dfrac{-x_{k-1}}{x_{k-2} - x_{k-1}} r_{x_{k-1}} + r^A_{x_k} = \dfrac{1}{\varphi^2} r_{x_{k-2}} - \dfrac{1}{\varphi} r_{x_{k-1}} + r^A_{x_k} \end{cases}$$

To simplify the expression we change the notation $R_k = r_{x_k}$ and $r_{k-1} = r^A_{x_k}$, so that:

$$\Rightarrow \begin{cases} R_2 &= \varphi^{-2} R_0 - \varphi^{-1} R_1 + r_1 &= -\varphi^{-1} R_1 + r_1 \\ R_3 &= \varphi^{-2} R_1 - \varphi^{-1} R_2 + r_2 &= 2\varphi^{-2} R_1 - \varphi^{-1} r_1 + r_2 \\ R_4 &= \varphi^{-2} R_2 - \varphi^{-1} R_3 + r_3 &= -3\varphi^{-3} R_1 + 2\varphi^{-2} r_1 - \varphi^{-1} r_2 + r_3 \\ R_5 &= \varphi^{-2} R_3 - \varphi^{-1} R_4 + r_4 &= 5\varphi^{-4} R_1 - 3\varphi^{-3} r_1 + 2\varphi^{-2} r_2 - \varphi^{-1} r_3 + r_4 \\ \vdots & & \\ R_k &= \varphi^{-2} R_{k-2} - \varphi^{-1} R_{k-1} + r_{k-1} &= (-1)^{k-1} F_k\, \varphi^{-(k-1)} R_1 + \displaystyle\sum_{i=1}^{k-1} (-1)^{k-1-i} F_{k-i}\, \varphi^{-(k-1-i)} r_i \end{cases}$$

being $F_j$ the $j-$th term of the Fibonacci's sequence $= \{1, 1, 2, 3, 5, 8, ...\}$

So, if $R_1 = r^I_\varphi + r_0$, then:

$$R_k = (-1)^{k-1} F_k \, \varphi^{-(k-1)} r^I_\varphi + \sum_{i=0}^{k-1} (-1)^{k-1-i} F_{k-i} \, \varphi^{-(k-1-i)} r_i$$

$$\Rightarrow |R_k| \leq \varphi^{-(k-1)} \left[ F_k \, |r^I_\varphi| + \sum_{i=0}^{k-1} F_{k-i} \, \varphi^i \, |r_i| \right]$$

Considering that $r_i = r^A_{x_{i+1}} \to |r_i| \leq r_M$ then:

$$\boxed{|R_k| \leq \frac{1}{\varphi^{(k-1)}} \left[ F_k \, |r^I_\varphi| + \left( \sum_{i=0}^{k-1} F_{k-i} \, \varphi^i \right) r_M \right]}$$

Thus:

$$|R_k| \leq A_k \, |r^I_\varphi| + B_k \, r_M \begin{cases} A_k = \dfrac{F_k}{\varphi^{k-1}} \\ B_k = \left( \displaystyle\sum_{i=0}^{k-1} \frac{F_{k-i} \, \varphi^i}{\varphi^{k-1}} \right) = \displaystyle\sum_{i=0}^{k-1} \frac{F_{k-i}}{\varphi^{k-i-1}} = \sum_{i=0}^{k-1} A_{k-i} = \sum_{j=1}^{k} A_j \end{cases}$$

And it can be easily checked that the coefficients $A_k$ and $B_k$ grow highly when the value of $k$ is increased (table 1):

Table 1: Coefficients of the series.

| $k$ | $A_k$ | $B_k$ |
|---|---|---|
| 1 | 1 | 1 |
| 5 | ≈34 | ≈55 |
| 10 | ≈4181 | ≈6765 |
| 20 | ≈63 $10^6$ | ≈102 $10^6$ |

Then, both the inherent error in the initial data and the storage errors of the intermediate operations are amplified in an uncontrolled manner. ⇒ UNSTABLE.

NOTE:

it can be proved that the $k$-th term of the Fibonacci's sequence is:

$$F_k = \frac{(1+\varphi)^k - (-\varphi)^k}{1 + 2\,\varphi}, \qquad \text{with } \varphi = \frac{-1+\sqrt{5}}{2}$$

and that, by mathematical induction:

$$\begin{aligned} &F_1 = 1 \\ &F_2 = 1 \\ &\dots \\ &F_k = F_{k-2} + F_{k-1} \qquad \forall k \geq 3 \end{aligned}$$

Then,

$$A_k = \frac{1}{1+2\varphi}\left[\frac{(1+\varphi)^k}{\varphi^{k-1}} - \frac{(-\varphi)^k}{\varphi^{k-1}}\right] = \frac{\varphi}{1+2\varphi}\left[\left(\frac{1+\varphi}{\varphi}\right)^k - (-1)^k\right] = \frac{\varphi}{1+2\varphi}\left[\Big(\underbrace{1+\frac{1}{\varphi}}_{\approx 2.618}\Big)^k - (-1)^k\right]$$

So the term grows exponentially.

$$\begin{aligned}
B_k &= \sum_{j=1}^{k} A_j = \frac{\varphi}{1+2\varphi}\sum_{j=1}^{k}\left[\left(1+\frac{1}{\varphi}\right)^j - (-a)^j\right] \\
&= \frac{\varphi}{1+2\varphi}\left[\frac{\left(1+\frac{1}{\varphi}\right) - \left(1+\frac{1}{\varphi}\right)^{k+1}}{1-\left(1+\frac{1}{\varphi}\right)} - \frac{(-1)-(-1)^{k+1}}{1-(-1)}\right] \\
&= \frac{\varphi}{1+2\varphi}\left[\varphi\left(1+\frac{1}{\varphi}\right)\left(\left(1+\frac{1}{\varphi}\right)^k - 1\right) + \frac{1-(-1)^k}{2}\right] \\
&= \frac{\varphi}{1+2\varphi}\left[(1+\varphi)\left(\Big(\underbrace{1+\frac{1}{\varphi}}_{\approx 2.618}\Big)^k - 1\right) + \frac{1-(-1)^k}{2}\right]
\end{aligned}$$

Therefore the term also grows exponentially.

Stable Algorithm: $\varphi = (-1+\sqrt{5})/2 \approx 0.61803398875$

$$\left\{\begin{array}{ll}
x_0 = 1 & r_{x_0} = \cancel{r^I_{x_0}} + \cancel{r^A_{x_0}} = 0 \\
x_1 = \varphi & r_{x_1} = r^I_{x_1} + r^A_{x_1} \\
x_2 = x_1\ x_1 & r_{x_2} = r_{x_1} + r_{x_1} + r^A_{x_2} = 2\ r_{x_1} + r^A_{x_2} \\
x_3 = x_2\ x_1 & r_{x_3} = r_{x_2} + r_{x_1} + r^A_{x_3} = 3\ r_{x_1} + r^A_{x_2} + r^A_{x_3} \\
\vdots & \vdots \\
x_k = x_{k-1}\ x_1 & r_{x_k} = r_{x_{k-1}} + r_{x_1} + r^A_{x_k} = k\ r_{x_1} + \sum_{i=2}^{k} r^A_{x_i}
\end{array}\right.$$

Then:

$$r_{x_k} = k\ r^I_{x_1} + \left(k\ r^A_{x_1} + \sum_{i=2}^{k} r^A_{x_i}\right)$$

$$\Rightarrow |r_{x_k}| \le k\ \left|r^I_{x_1}\right| + k\ \left|r^A_{x_1}\right| + \sum_{i=2}^{k}\left|r^A_{x_i}\right|$$

$$\Rightarrow |r_{k_k}| \le k\ \left|r^I_{x_1}\right| + k\ r_M + (k-1)\ r_M$$

$$\Rightarrow \boxed{|r_{x_k}| \le k\ \left|r^I_{x_1}\right| + (2k-1)\ r_M}$$

In the worst case, the inherent error and the storage errors of intermediate operations grow linearly $\Rightarrow$ STABLE.

In addition, the upper bound will be very conservative, so that the relative error will be small in general, even after a large number of iterations.

**6.—** We want to approximate $e^x$ for various values of $x$ in the interval $[0, 10]$ by summing the first $n$ terms of the Taylor series. The calculation will be performed on a computer using 24 bits for the floating-point mantissa (including the sign). Carry out a simple study (without taking into account the effect of the propagation of the rounding error) that establishes a maximum value of the number of terms to be considered, above which no appreciable improvement in the approximation can be seen.

**Sol. 6.**

The Taylor's expansion of $e^x$ is:

$$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!} + R_n(x) \qquad ; \qquad R_n(x) = \frac{e^\xi}{(n+1)!} x^{n+1} \qquad \xi \in [0, x]$$

If we assume that we will not take into account the propagation of the error in the operations, we can approach the resolution of the problem by means of two criteria:

**1)** It does not make sense to keep adding terms when the new term is of the order of the Machine epsilon or even smaller:

$$\left| \frac{x^{n+1}}{(n+1)!} \right| \leq |e^x|\, r_M$$

$$x > 0 \quad \Rightarrow \quad \frac{e^{-x}\, x^{n+1}}{(n+1)!} \leq r_M$$

$$\max_{x\in[0,10]} \frac{e^{-x}\, x^{n+1}}{(n+1)!} = \begin{cases} \dfrac{e^{-(n+1)}\,(n+1)^{n+1}}{(n+1)!} & \text{for } x = n+1 \leq 10 \\[2ex] \dfrac{e^{-10}\, 10^{n+1}}{(n+1)!} & \text{for } x = 10 \leq n+1 \end{cases}$$

Then, there is no point in continuing when:

$$\boxed{\frac{e^{-Z_n} Z_n^{n+1}}{(n+1)!} \leq \frac{1}{2} 2^{-m+2} \qquad \text{with} \qquad \begin{cases} Z_n = n+1 & \text{if} \quad n \leq 9 \\ Z_n = 10 & \text{if} \quad n \geq 9 \end{cases}}$$

**2)** It does not make sense to keep adding terms when the truncation error is of the order of the Machine epsilon or even smaller:

$$\left| \frac{e^\xi\, x^{n+1}}{(n+1)!} \right| \leq |e^x|\, r_M$$

$$x > 0 \quad \Rightarrow \quad \frac{e^{\xi - x}\, x^{n+1}}{(n+1)!} \leq r_M$$

$$\max_{x\in[0,10],\, \xi\in[0,x]} \frac{e^{\xi - x}\, x^{n+1}}{(n+1)!} = \frac{10^{n+1}}{(n+1)!}$$

Then, there is no point in continuing when:

$$\boxed{\frac{10^{n+1}}{(n+1)!} \le \frac{1}{2}\,2^{-m+2}}$$

**7.**— In a calculus process it is required to repeatedly evaluate the function $f(x) = \sqrt{1+x} - 1$. It is known that for small values of $x$ , there exists, among others, the asymptotic approximation to the above function $f(x) \approx f_0(x) = x/2$. The calculations will be performed on a digital computer. Could the results obtained using the asymptotic approximation $f_0$ be better than the results of the calculations obtained using the function $f(x)$ itself? If yes: For what range of values of $x$?; what would this range be if single precision (24 bits for the mantissa, including the sign) is used? In the affirmative or negative case, present several numerical examples that corroborate the result, using the decimal base and performing floating point operations with three digits.

**Sol. 7.**

It is possible that formula $f_0(x)$ gives more accurate results than the original expression for very small values of $|x|$, because the truncation error of $f_0(x)$ is smaller as $|x| \to 0$, while the rounding error when operating $f(x)$ can be very large when $|x| \to 0$, since $1 + x \approx 1$. In fact for sufficiently small values of $|x|$ this expression will result in zero, which means 100% of relative error.

Analyzing the error propagation and the truncation errors:

**1)** $f_1(x) = \sqrt{1+x} - 1$

$$\begin{cases} Y = 1 + x & \to \quad r_Y = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x} r_x + r_Y^A \\ Z = \sqrt{Y} & \to \quad r_Z = \dfrac{1}{2}\, r_Y + r_Z^A \\ F1 = Z - 1 & \to \quad r_{F1} = \dfrac{Z}{Z-1} r_Z + \dfrac{-1}{Z-1}\cancel{r_1} + r_{F1}^A \end{cases}$$

$$\begin{aligned} \Rightarrow r_{F1} &= \frac{\sqrt{1+x}}{\sqrt{1+x}-1}\left(\frac{1}{2}\left(\frac{x}{1+x} r_x + r_Y^A\right) + r_Z^A\right) + r_{F1}^A \\ &= \left(\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\,\frac{1}{2}\,\frac{x}{1+x} r_x^I\right) + \left[\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\left(\frac{1}{2}\left(\frac{x}{1+x} r_x^A + r_Y^A\right) + r_Z^A\right) + r_{F1}^A\right] \end{aligned}$$

$$|r_{F1}| \le \left|\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\,\frac{1}{2}\,\frac{x}{1+x}\right| |r_x^I| + \left[\left|\frac{\sqrt{1+x}}{\sqrt{1+x}-1}\right|\left(\frac{1}{2}\left(\left|\frac{x}{1+x}\right| + 1\right) + 1\right) + 1\right] r_M$$

**2)** $f_0(x) = x/2$

$\{F0 = x/2 \quad \to \quad r_{F0} = r_x + \cancel{r_2} + \cancelto{0}{r_{F0}^A}$ (dividing by powers of 2 in binary modifies only the exponent and therfore does not add a new storing error term)

So that:

$$r_{F0} \;=\; r_x \;=\; r_x^I + r_x^A$$

But what we seek is not $r_{F0} = \dfrac{f_0(x) - \widehat{f}_0(x)}{f_0(x)}$.

We have to take into account the truncation error since we are using an approximation to the function we really want to evaluate. The total relative error is:

$$r^*_{F0} = \frac{f(x) - \widehat{f}_0(x)}{f(x)} = \frac{f(x) - f_0(x)(1 - r_{F0})}{f(x)} = \underbrace{\frac{f(x) - f_0(x)}{f(x)}}_{\text{truncation}} + \underbrace{\frac{f_0(x)}{f(x)}}_{\approx 1} \underbrace{r_{F0}}_{\text{rounding}}$$

Thus:

$$\begin{aligned} r^*_{F0} &= \frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)} + \frac{x/2}{(\sqrt{1+x}-1)}(r_x^I + r_x^A) \\ &= \frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)} + \frac{x/2}{(\sqrt{1+x}-1)} r_x^I + \frac{x/2}{(\sqrt{1+x}-1)} r_x^A \end{aligned}$$

$$|r^*_{F0}| \le \left|\frac{(\sqrt{1+x}-1) - x/2}{(\sqrt{1+x}-1)}\right| + \left|\frac{x/2}{(\sqrt{1+x}-1)}\right| |r_x^I| + \left|\frac{x/2}{(\sqrt{1+x}-1)}\right| r_M$$

In both cases when $|x| << 1 \Rightarrow \begin{cases} (1+x) \to 1 \\ (\sqrt{1+x}) \to 1 \\ (\sqrt{1+x}-1) \to x/2 \\ ((\sqrt{1+x}-1) - x/2) \to -x^2/8 \end{cases}$.

Then:

$$\begin{cases} |r_{F1}| \;\le\; R_1 \;\approx\; \left|\dfrac{1}{x/2}\,\dfrac{1}{2}\,x\right| \; |r_x^I| + \left[\left|\dfrac{1}{x/2}\right| \left(\dfrac{1}{2}\,(|x|+1) + 1\right) + 1\right] r_M \;=\; |r_x^I| + \left(2 + \dfrac{3}{|x|}\right) r_M \\[2ex] |r^*_{F0}| \;\le\; R_0 \;\approx\; \left|\dfrac{-x^2/8}{x/2}\right| + \left|\dfrac{x/2}{x/2}\right| \; |r_x^I| + \left|\dfrac{x/2}{x/2}\right| r_M \;=\; |x/4| + |r_x^I| + r_M \end{cases}$$

Evaluating the function as $f_0(x)$ will always be better as far as $R_0 \le R_1$, meaning that:

$$\left|\frac{x}{4}\right| + |r_x^I| + r_M \le |r_x^I| + \left[2 + \frac{3}{|x|}\right] r_M$$

$$\Rightarrow \left|\frac{x}{4}\right| \le \left[1 + \frac{3}{|x|}\right] r_M$$

Which will be satisfied when:

$$\left|\frac{x}{4}\right| \le \frac{3}{|x|} r_M$$

So that:

$$x^2 \leq 12\, r_M \;\Leftrightarrow\; |x| \leq 2\,\sqrt{3}\,\sqrt{r_M}$$

For values of $|x| \leq 2\,\sqrt{3}\,\sqrt{r_M}$ the expression $f_0(x)$ has an error bound smaller than that of the original expression $f(x)$ despite the truncation error.

Single precision $\Rightarrow$ $m = 24$ bits $\Rightarrow$ $r_M = 2^{-23}$. Thus, expression $f_0(x)$ will be better for values $|x| \leq 1.19604\, 10^{-3}$

In a calculator with 10 digits:

$$r_M = 1/2\ 10^{-(10+1)+2} = 1/2\ 10^{-9} \;\Rightarrow\; |x| \;\leq\; 7.74597\ 10^{-5}$$

$$\begin{cases} x = 0.0001 \rightarrow & f_1(x) = 0.4998750\ 10^{-4}, & f_0(x) = 0.5\ 10^{-4} \\ x = 0.00001 \rightarrow & f_1(x) = 0.5\ 10^{-5}, & f_0(x) = 0.5\ 10^{-5} \end{cases}$$

In a calculator with 3 digits:

$$r_M = 1/2\ 10^{-(3+1)+2} = 1/2\ 10^{-2} \;\Rightarrow\; |x| \;\leq\; 0.244949$$

---

**8.**— In a computational process it is necessary to evaluate the function $f(x) = e^{-1/x}$ for various values of $x \in [0.05, 0.10]$ with a relative error $r_f \leq 10^{-8}$. If the values of $x$ are known with relative error $r_x \leq 10^{-5}$, and assuming we can increase the computer's accuracy as far as necessary, can a satisfactory result be obtained? If not, with what precision would it be necessary to know the data?

In either case, what type of precision—single or double—should be used if the calculations are performed on a digital computer? Assume that in single and double precision 24 and 53 bits, respectively, are used to store the mantissa (including the sign).

---

**Sol. 8.**

If we assume that the computer has infinite precision this implies that all storage errors are zero and therefore $r_x = r_x^I$.

This way, the error when evaluating $f(x) = e^{-1/x}$ will be:

$$r_f = \frac{f'(x)}{f(x)} x\, r_x = \frac{e^{-1/x}(1/x^2)}{e^{-1/x}} x\, r_x = \frac{r_x}{x} = \frac{r_x^I}{x} \Rightarrow |r_f| = \frac{|r_x^I|}{|x|}$$

Since $\left|r_x^I\right| \leq 10^{-5}$ and $x \in [0.05,\ 0.10]$, the upper bound of the error is:

$$|r_f| \leq \frac{10^{-5}}{0.05} = 0.2\, 10^{-3}$$

So, we can state that $|r_f| \leq 10^{-8}$

Example.

$$\left.\begin{array}{l} x = 0.05 \;\rightarrow\; f(x) = 2.0611536\, 10^{-9} \\ \widehat{x} = 0.05\,(1 - 10^{-5}) \;\rightarrow\; f(\widehat{x}) = 2.0607414\, 10^{-9} \end{array}\right\} \Rightarrow \frac{f(x) - f(\widehat{x})}{f(x)} = 0.19998\ 10^{-3}$$

To achieve the desired accuracy we need the error in the initial data to be:

$$|r_f| = \frac{|r_x^I|}{|x|} \leq 10^{-8} \Rightarrow \left\{ \begin{array}{l} |r_x^I| \leq |x|\ 10^{-8} \\ x \in [0.05,\ 0.10] \end{array} \right\} \Rightarrow |r_x^I| \leq 0.05\,10^{-8} = 0.5\,10^{-9}$$

We now analyze the accuracy that we would have to use in a digital computer to obtain the desired accuracy:

$$\left\{ \begin{array}{lllllll} \text{Single precision} & \to & m = 24 & \to & r_M = 1/2\,2^{-24+2} & = & 2^{-23} & \approx & 0.119\,10^{-6} \\ \text{Double precision} & \to & m = 53 & \to & r_M = 1/2\,2^{-53+2} & = & 2^{-22} & \approx & 0.222\,10^{-15} \end{array} \right\}$$

Single precision will not be enough since we can not even store the value of $x$ accurately since $|r_x| \leq |r_x^I| + r_M$, and the machine epsilon alone is $r_M > 10^{-8}$.

We would need to use double precision. And we hope that double precision would be sufficient.

<u>Complete study</u>

$$\left\{ \begin{array}{lll} & r_x = r_x^I + r_x^A & ; \ |r_x^A| \leq r_M \\ Y = -1/X & r_y = 1\,\cancelto{0}{r_1} - 1\,r_x + r_y^A & ; \ |r_y^A| \leq r_M \\ F = EXP(Y) & r_f = \dfrac{g'(y)}{g(y)} y\,r_y + r_f^A = \dfrac{\cancel{e^y}}{\cancel{e^y}} y\,r_y + r_f^A & ; \ |r_f^A| \leq r_M \end{array} \right.$$

$$\begin{aligned} r_f &= \frac{-1}{x}\left(-(r_x^I + r_x^A) + r_y^A\right) + r_f^A \\ &= \frac{1}{x} r_x^I + \left(\frac{1}{x} r_x^A - \frac{1}{x} r_y^A + r_f^A\right) \end{aligned}$$

$$|r_f| \leq \frac{1}{|x|}|r_x^I| + \left(\frac{1}{|x|} + \frac{1}{|x|} + 1\right) r_M = \frac{1}{|x|}|r_x^I| + \left(\frac{2}{|x|} + 1\right) r_M$$

In double precision $r_M = 2^{-52} = 0.222\ 10^{-15}$ and considering that $x \in [005,\ 0.10]$ then the error can be bounded as:

$$|r_f| \leq \frac{|r_x^I|}{0.05} + \left(\frac{2}{0.05} + 1\right)\ 2^{-52}$$

We want to ensure that $|r_f| \leq 10^{-8}$, thus:

$$\boxed{|r_x^I| \leq 0.499999545\ 10^{-9}}$$

Then, indeed, the calculation is possible in double precision. Note that the precision required for the data $x$ is somewhat higher than that obtained above without considering storage errors in the operations.

**9.**— In a computational process it is necessary to evaluate several million times the function $f(x) = \ln(1+x)$, always for positive $(x > 0)$ and small $(x << 1)$ values of $x$. The FORTRAN compiler function available is very accurate, but requires a relatively high computation time (on the order of the equivalent to several tens of elementary operations).

To reduce the computational time, the possibility of approximating the function using the first two terms of its Taylor development is evaluated, this simplification being admitted when the relative error is less (in absolute value) than one hundred times the machine storage error. The following subroutines are prepared:

```
SUBROUTINE EXACT(X, Y1)      SUBROUTINEAPROX(X, Y2)
REAL * 4 X, Z, Y1            REAL * 4 X, T, Y2
Z = 1. + X                   T = 1. - X/2.
Y1 = ALOG(Z)                 Y2 = X * T
RETURN                       RETURN
END                          END
```

Knowing that the computer works in floating point rounding by approximation and allocating $m = 24$ bits to the mantissa (including 1 bit for the sign), and that the values of `X` are exact (in the sense that their inherent error is zero, although in general they will be affected by the corresponding storage error):

**a)** Analyze instruction by instruction the previous subroutines, obtaining the relative errors of the variables at each step.

**b)** Obtain the relative error of the value `Y1` calculated by the subroutine `EXACT` with respect to the exact value of $f(x)$. Obtain a bound of the error and explain what happens when $x$ tends to zero.

**c)** Obtain the relative error of the value `Y2` calculated by the subroutine `APPROX` with respect to the exact value of $f(x)$. Obtain a bound of the error and explain what happens when $x$ tends to zero.

**d)** Discuss for which range of values of `X` the subroutine `APPROX` can be used instead of the subroutine `EXACT`..

**e)** Is it possible that the `APPROX` subroutine provides more accurate results than the `EXACT` subroutine? If so, for what values of `X`?

**Note:** The Taylor expansion (with the Lagrangian remainder) of the function $f(x)$ is:

$$f(x) = -\sum_{i=1}^{n} (-1)^i \,\frac{x^i}{i} + R_n(x) \qquad ; \qquad R_n(x) = -1\left(\frac{-1}{1+\xi}\right)^{n+1} \frac{x^{n+1}}{n+1} \qquad \xi \in (0, x)$$

---

**Sol. 9.a)**

Given that the statement indicates that the inherent error in the data is null:

$$r_x^I = 0 \;\Rightarrow\; r_x = r_x^A \qquad \left|r_x^A\right| \le r_M$$

EXACT:

$$\begin{cases} Z = 1. + X & \rightarrow \quad r_z = \dfrac{1}{1+x}\cancel{r_1} + \dfrac{x}{1+x}\, r_x + r_z^A \\[2ex] Y1 = \ln(Z) & \rightarrow \quad r_{y1} = \dfrac{1/z}{\ln(z)}\, z\, r_z + r_{y1}^A \end{cases}$$

$$\Rightarrow \boxed{r_{y1} = \frac{1}{\ln(1+x)}\left[\frac{x}{1+x}\, r_x^A + r_z^A\right] + r_{y1}^A}$$

APPROXIMATED:

$$\left\{\begin{array}{lll} T = 1. - X/2 & \rightarrow & r_t = \dfrac{1}{1-x/2}\,\cancel{r_1} + \dfrac{-x/2}{1-x/2}\, r_x + r_t^A \\ Y2 = X * T & \rightarrow & r_{y2} = r_x + r_t + r_{y2}^A \end{array}\right.$$

$$\Rightarrow \boxed{r_{y2} = \left(1 - \frac{x/2}{1-x/2}\right) r_x^A + r_t^A + r_{y2}^A}$$

Note that in this error expression it has been considered that multiplying by integer powers of the numbering base ($B = 2$) does not introduce a new error neither in the division nor in the storage operation.

**Sol. 9.b)**

$$\begin{array}{lcl} \text{Known } r_{y1} & = & \dfrac{y_1 - \hat{y}_1}{y_1} \text{ with } \hat{y}_1 \text{ the calculated value} \\ \text{Seek } r_{y1}^* & = & \dfrac{f(x) - \hat{y}_1}{f(x)} = \underbrace{\dfrac{f(x) - y_1}{f(x)}}_{\text{truncation}} + \underbrace{\dfrac{y_1}{f(x)}}_{\approx 1} \underbrace{\dfrac{y_1 - \hat{y}_1}{y_1}}_{\text{rounding}} \end{array}$$

Since the expression is exact, $f(x) = y_1$, thus:

$$r_{y1}^* = r_{y1} = \frac{1}{\ln(1+x)}\left[\frac{x}{1+x}\, r_x^A + r_z^A\right] + r_{y1}^A$$

$$\left|r_{y1}^*\right| \leq \left[\frac{1}{|\ln(1+x)|}\left(\left|\frac{x}{1+x}\right| + 1\right) + 1\right] r_M \equiv R_1$$

$$\text{when} \left\{\begin{array}{l} x \rightarrow 0 \\ x > 0 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} 1 + x \rightarrow 1 \\ \ln(1+x) \rightarrow x \end{array}\right\} \Rightarrow R_1 \approx \left(2 + \frac{1}{|x|}\right) r_M \Rightarrow R_1 \rightarrow \infty$$

In practice, when $x \rightarrow 0$ the expression will result in zero, because $x$ is negligible compared to 1, so the error tends to 100% (not to $\infty$).

**Sol. 9.c)**

$$\begin{array}{lcl} \text{Known } r_{y2} & = & \dfrac{y_2 - \hat{y}_2}{y_2} \text{ with } \hat{y}_2 \text{ the calculated value} \\ \text{Seek } r_{y2}^* & = & \dfrac{f(x) - \hat{y}_2}{f(x)} = \underbrace{\dfrac{f(x) - y_2}{f(x)}}_{\text{truncation}} + \underbrace{\dfrac{y_2}{f(x)}}_{\approx 1} \underbrace{\dfrac{y_2 - \hat{y}_2}{y_2}}_{\text{rounding}} \end{array}$$

We know that $f(x) = \underbrace{x - \dfrac{x^2}{2}}_{y_2} + \underbrace{\left(\dfrac{1}{1+\xi}\right)^3 x^3/3}_{R_2(x)}$ with $\xi \in [0, x]$ (Taylor)

Then the relative truncation error is $\dfrac{f(x)-y_2}{f(x)} = \dfrac{R_2(x)}{f(x)}$, thus:

$$r^*_{y2} = \frac{\dfrac{1}{(1+\xi)^3}\dfrac{x^3}{3}}{\ln(1+x)} + \frac{(x-x^2/2)}{\ln(1+x)}\, r_{y2}$$

$$= \frac{1}{(1+\xi)^3}\frac{x^3/3}{\ln(1+x)} + \frac{(x-x^2/2)}{\ln(1+x)}\left[\frac{1-x}{1-x/2}\, r^A_x + r^A_t + r^A_{y2}\right]$$

$$= \frac{1}{(1+\xi)^3}\frac{x^3/3}{\ln(1+x)} + \frac{1}{\ln(1+x)}\left[x(1-x)\, r^A_x + x(1-x/2)\, r^A_t + x(1-x/2)\, r^A_{y2}\right]$$

$$\left|r^*_{y2}\right| \le \left|\frac{1}{(1+\xi)^3}\right| \frac{|x^3/3|}{|\ln(1+x)|} + \frac{|x|}{|\ln(1+x)|}\left[|1-x| + |1-x/2| + |1-x/2|\right] r_M$$

Considering $x>0$, $x<<1$ and $\xi \in [0,x]$:

$$\left|r^*_{y2}\right| \le \frac{x^3/3}{\ln(1+x)} + \frac{x(3-2x)}{\ln(1+x)}\, r_M \equiv R_2$$

$$\text{when}\left\{\begin{array}{c} x\to 0 \\ x>0 \end{array}\right\} \Rightarrow \{\ln(1+x)\approx x\} \Rightarrow R_2 \approx \frac{x^3/3}{x} + \frac{x(3-2x)}{x}\, r_M = \frac{x^2}{3} + (3-2x)\, r_M$$

$$R_2 \to 3\, r_M \qquad \text{when} \qquad x\to 0$$

**Sol. 9.d)**

As indicated, the validity of the subroutine APROX will be accepted when the relative error is less than one hundred times the machine error:

$$\frac{x^2}{3} + (3-2x)\, r_M \le 100\, r_M \Rightarrow x^2 \le 3(97+2x)\, r_M \Rightarrow \boxed{x \le \sqrt{291\, 2^{-23}} \approx 0.006}$$

**Sol. 9.e)**

Subroutine APROX will perform better than EXACT when:

$$\frac{x^2}{3} + (3-2x)\, r_M \le (2+\frac{1}{x})\, r_M \Leftrightarrow x^2 \le 3(2+\frac{1}{x} - 3 + 2x)\, r_M = 3(\frac{1}{x} + 2x - 1)$$

We can disregard $(2x-1)$ in comparison with $1/x \Rightarrow$

$$x^2 \le \frac{3}{x}\, r_M \Leftrightarrow \boxed{x \le \sqrt[3]{3\, 2^{-23}} \approx 0.007}$$

**10.**— A FORTRAN program includes the instruction `X=`$n$, where `X` is a real variable of type `REAL*4` (single precision) or of type `REAL*8` (double precision), and $n$ is a positive integer constant of type `INTEGER*4`.

Determine what is the value of $n$ at which storage errors can occur in single precision and double precision.

**Sol. 10.**

Statement `X=N` might produce storing errors when:

$$N \geq (\underbrace{10 \dots 01}_{m \text{ bits}})_2 = 2^{m-1} + 2^0 = 2^{m-1} + 1$$

where $m$ is the number of bits that are destined to the mantissa (including the sign), because from this value the number $N$ can have more significant bits than those that fit in the mantissa and it will be necessarily rounded.

If the first bit of the mantissa is not stored (which is most often the case) then one additional bit will fit and the storage errors will then be:

$$N \geq 2^m + 1$$

Therefore:

$$\begin{cases} \texttt{REAL}*4 & \rightarrow \quad m = 24 \quad \rightarrow \quad N \geq 16\,777\,217 \\ \texttt{REAL}*8 & \rightarrow \quad m = 53 \quad \rightarrow \quad N \geq 2^{53} + 1 \end{cases}$$

It can be observed that in double precision there will never be storage errors because the largest integer that can be considered is $N = 2^{e-1} - 1$ with $e = 32$, meaning $N = 2^{31} - 1$.

**11.** Develop a FORTRAN program that finds the solution to the above problem by numerical experimentation.

**Sol. 11.**

```
c==========================================================================
c=======================================================Program IntegerStore
c==========================================================================
c--------------------------------------------------------------------------
c This program finds the first integer that can not be stored exactly
c in a variable REAL*4.
c
c Note: The program must be compiled without optimization (option: -o0)
c       to prevent compilator corrections.
c--------------------------------------------------------------------------
implicit integer*4 (i-n)
implicit real*4 (a-h,o-z)
logical seguir
parameter (MXINT=2*(2**30-1)+1)   !we obtain (2**31-1) without overflow

i=0
seguir=.TRUE.
do while(seguir)
```

```
        i=i+1
        x=i        ! is equivalent to x=real(i)
        j=x        ! is equivalent to j=int(x)
        seguir=(i.lt.MXINT).and.(i.eq.j)
      enddo

      if(i.ne.j)then
        write(6,*) i,x
      else
        write(6,200)
      endif

  100 format(' Problem in term i=',i20,
     .       '                ----> x=',f30.9)
  200 format(' Every integer of type INTEGER*4'//
     .       ' fit in a real*4')

      read(5,'()')

      end
```

---

**12.—** In a digital computer the following sum is obtained

$$S_n = \overbrace{a + a + \ldots + a}^{\text{“}n\text{” times}},$$

where $a$ is any number whose inherent error is $r_a^I$. The operations are performed in floating point, allocating $m$ bits to the mantissa (including the sign):

**a)** Calculate the total error of the result obtained by performing the above operation. Indicate what part of the total error is due to the propagation of the inherent error of the data $a$ and what part is due to the propagation of the storage errors of the intermediate operations.

**b)** Analyze how the maximum error grows as the value of $n$ increases. Discuss to what extent this maximum value can be considered exaggerated.

**c)** Repeat the previous sections assuming that the calculation is performed in the form $S_n = n \cdot a$. Compare the results obtained in the two cases.

**d)** Relate these theoretical conclusions with the set-up of problem 8 of the previous problems sheet and make a theoretical prediction of the PK from which the problems referred to in that exercise can be produced. If necessary, compare the theoretical prediction with the "experimental" results obtained by numerical simulation.

---

**Sol. 12.a)**

Analyzing the error in each operation:

$$\begin{cases} S_1 = a & \rightarrow \quad r_{S_1} = r_a \\ S_2 = S_1 + a & \rightarrow \quad r_{S_2} = \dfrac{S_1}{S_1+a}\, r_{S_1} + \dfrac{a}{S_1+a}\, r_a + r^A_{S_2} \\ S_3 = S_2 + a & \rightarrow \quad r_{S_3} = \dfrac{S_2}{S_2+a}\, r_{S_2} + \dfrac{a}{S_2+a}\, r_a + r^A_{S_3} \\ S_4 = S_3 + a & \rightarrow \quad r_{S_4} = \dfrac{S_3}{S_3+a}\, r_{S_3} + \dfrac{a}{S_3+a}\, r_a + r^A_{S_4} \\ \vdots & \quad\ \vdots \\ S_n = S_{n-1} + a & \rightarrow \quad r_{S_n} = \dfrac{S_{n-1}}{S_{n-1}+a}\, r_{S_{n-1}} + \dfrac{a}{S_{n-1}+a}\, r_A + r^A_{S_n} \end{cases}$$

Considering $S_i = i \cdot a$ then:

$$\frac{S_i}{S_i+a} = \frac{i}{i+1} \qquad \text{y} \qquad \frac{a}{S_i+a} = \frac{1}{i+1}$$

Therefore:

$$\begin{cases} r_{S_1} &=& r_a \\ r_{S_2} &=& 1/2\ r_{S_1} + 1/2\ r_a + r^A_{S_2} &=& r_a + r^A_{S_2} \\ r_{S_3} &=& 2/3\ r_{S_2} + 1/3\ r_a + r^A_{S_3} &=& r_a + r^A_{S_3} + 2/3\ r^A_{S_2} \\ r_{S_4} &=& 3/4\ r_{S_3} + 1/4\ r_a + r^A_{S_4} &=& r_a + r^A_{S_4} + 3/4(r^A_{S_3} + 2/3\ r^A_{S_2}) \\ \vdots \\ r_{S_n} &=& \dfrac{n-1}{n}\ r_{S_{n-1}} + \dfrac{1}{n} r_a + r^A_{S_n} &=& r_a + r^A_{S_n} + \dfrac{n-1}{n}\left(r^A_{S_{n-1}} + ... + \dfrac{4}{5}\left(r^A_{S_4} + \dfrac{3}{4}\left(r^A_{S_3} + \dfrac{2}{3} r^A_{S_2}\right)\right)...\right) \end{cases}$$

$$\Rightarrow r_{S_n} = \underbrace{r^I_a}_{\text{inherent error}} \underbrace{+ r^A_a + r^A_{S_n} + \frac{n-1}{n}\left(r^A_{S_{n-1}} + ... + \frac{4}{5}\left(r^A_{S_4} + \frac{3}{4}\left(r^A_{S_3} + \frac{2}{3} r^A_{S_2}\right)\right)...\right)}_{\text{storing errors}}$$

We see that the inherent error of the data is propagated as it is.

**Sol. 12.b)**

Bounding the error:

$$\begin{aligned}
|r_{S_n}| &\leq |r_a^I| + \left[2 + \frac{n-1}{n}\left(1 + ... + \frac{4}{5}\left(1 + \frac{3}{4}\left(1 + \frac{2}{3}\right)\right) ...\right)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{n}\left((n-1) + ... + 4 + 3 + 2\right)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{n}\,\frac{2 + (n-1)}{2}\,(n-2)\right] r_M \\
&= |r_a^I| + \left[2 + \frac{1}{2n}\,(n+1)(n-2)\right] r_M \\
&= |r_a^I| + \frac{1}{2n}\left[4n + (n+1)(n-2)\right] r_M \\
&= |r_a^I| + \frac{1}{2n}[4n + n^2 - 2n + n - 2]\, r_M \\
&= |r_a^I| + \frac{1}{2n}[n^2 + 3n - 2]\, r_M \\
&= |r_a^I| + [n/2 + 3/2 - 1/n]\, r_M
\end{aligned}$$

Then

$$\boxed{|r_{S_n}|^{max} \sim |r_a^I| + \frac{n}{2}\, r_M} \qquad \text{(When $n$ is large)}$$

The error grows linearly with $n$.

This maximum error is extremely unlikely, because for it to be reached, all storage errors must be equal to their maximum (the machine error) and of the same sign. The final result will presumably be smaller. In practice the storage errors will have varying values between $(-r_M)$ and $(r_M)$. Therefore, to quantify this effect we could perform a statistical calculation and obtain the probable error (the mathematical expectation of $r_{S_n}$).

**Sol. 12.c)**

If we operate as $S_n = n \cdot a$ then:

$$r_{S_n} = r_n + r_a + r_{S_n}^a$$

Generally, unless $n$ is too large, $r_n = 0$, then

$$\boxed{r_{S_n} = r_a^I + r_a^A + r_{S_n}^A}$$

As in the previous case, the error inherent in the data is propagated as is.

$$\boxed{|r_{S_n}| \leq \left|r_a^I\right| + 2\, r_M}$$

In the calculation by repeated additions, the error increases as more additions are made, while in the calculation by multiplication the error remains at the same level.

**Sol. 12.d)**

The effect of the accumulated errors will be noticed when

$$(n \cdot a)\,|r_{S_n}| \approx \frac{1}{2}\,0.001, \qquad \text{with} \qquad a = 0.1$$

since the error could start to change the third decimal.

This effect could start when:

$$(n \cdot a)\,|r_{S_n}|^{max} \approx \frac{1}{2}\,0.001, \qquad \text{with} \qquad a = 0.1$$

being

$$|r_{S_n}|^{max} \sim \left|r_a^I\right| + \frac{n}{2}\,r_M$$

with $r_M = \dfrac{1}{2}\,2^{-m+2}$ and for large $n$, $\left|r_a^I\right| << \dfrac{n}{2}\,r_M$:

$$(n\ 0.1)\frac{n}{2}\,2^{-m+2} \approx \frac{1}{2}\,0.001 \Longleftrightarrow n^2 \approx 10^{-2}\,2^{m-1} \Leftrightarrow \boxed{n \approx 10^{-1}\,2^{\frac{m-1}{2}}}$$

This way, for variables of type:

$$\texttt{REAL*4} \rightarrow n \approx 290$$

$$\texttt{REAL*8} \rightarrow n \approx 6\,710\,886$$

The numerical examples and the proposed programs (SumaIterada_R4 and SumaIterada_R8) cast the following results:

$$\texttt{REAL}*4 \quad \rightarrow \quad n = 703 \qquad S_n = 70.299$$

$$\texttt{REAL}*8 \quad \rightarrow \quad n = 17\,704\,899 \quad S_n = 1\,770\,489.901$$

Then the phenomenon takes longer to appears than we had anticipated. This is because the error bound $r_{S_n}$ is exaggerated in relation to the error that is actually produced in practice.

---

**13.—** An engineer usually programs in FORTRAN and uses a certain digital computer. The engineer wants to know what is the machine error $r_M$ with which the calculations are performed, both for variables of type `REAL*4` (single precision) and for variables of type `REAL*8` (double precision).

The engineer does not have access to manuals or any documentation explaining how the data is stored and how many bits are allocated to mantissa storage in each case.

**a)** Relate the machine error $r_M$ with the value of the Machine Epsilon $\varepsilon > 0$, corresponding to the smallest positive real number such that $\mathcal{A}(1+\varepsilon) \neq \mathcal{A}(1)$, with $\mathcal{A}(x)$ being the stored value corresponding to $x$.

**b)** To realize a FORTRAN program that allows to obtain experimentally the value of the Machine Epsilon $\varepsilon$ for the two types of precision (single and double) on the computer on which the program is run.

**c)** Test the program on a computer and compare the results with theoretical predictions. with the theoretical predictions. Explain the discrepancies, if any.

---

**Sol. 13.a)**

$$\mathcal{A}(1) = (0.100\ldots000)_B\ B^1$$

$$\mathcal{A}(1+\epsilon) = (0.\underbrace{100\ldots001}_{(m-1)\ digits})_B\ B^1$$

Then:

$$\varepsilon = \frac{1}{2}(0.\underbrace{000\ldots001}_{(m-1)\ digits})_B\ B^1$$

The term $\frac{1}{2}$ appears due to approximation by rounding.

$$\Rightarrow \varepsilon = \frac{1}{2}\ B^{-(m-1)}\ B^1 = \frac{1}{2}\ B^{-m+2}$$

Therefore:

$$\boxed{\varepsilon = r_M = \frac{1}{2}\ B^{-m+2}}$$

In a digital computer $B = 2 \Rightarrow$.

$$\epsilon = r_M = 2^{-m+1} \rightarrow \begin{cases} \texttt{REAL}*4 \ \rightarrow\ m = 24 \ \Rightarrow\ \varepsilon = 2^{-23} \\ \texttt{REAL}*8 \ \rightarrow\ m = 53 \ \Rightarrow\ \varepsilon = 2^{-52} \end{cases}$$

But in practice it is normal that the first bit is not stored (since it is always 1 except for the number 0). In this way one more bit is available for the mantissa, then:

$$\epsilon = r_M = 2^{-m} \rightarrow \begin{cases} \texttt{REAL}*4 \ \rightarrow\ m = 24 \ \Rightarrow\ \varepsilon = 2^{-24} \\ \texttt{REAL}*8 \ \rightarrow\ m = 53 \ \Rightarrow\ \varepsilon = 2^{-53} \end{cases}$$

**Sol. 13.b)**

```
c====================================Program MachineEpsilon
c========================================================
      real*4 eps
      real*8 deps
c.....Single Precision (REAL*4)
      call MEps_Single(le2,eps)
      write(6,100) le2,eps
  100 format(' REAL*4 -> Machine Epsilon = 2**',i3,' = ',e15.6)

c.....Double Precision (REAL*8)
      call MEps_Double(lde2,deps)
      write(6,200) lde2,deps
 200 format(' REAL*8 -> Machine Epsilon = 2**',i3,' = ',d15.6)
      end

c-----------------------------Subroutine MEps_Single-----
c       Machine epsilon in single precision
c----------------------------------------------------------
      subroutine MEps_Single(le2,eps)
      implicit real*4 (a-h,o-z)
      uno=1.e+00
      dos=2.e+00
      eps=1.e+00
      le2=0
      q =uno+eps/dos
      do while (uno.ne.q)
        eps=eps/dos
        le2=le2-1
        q =uno+eps/dos
      enddo

      return
      end
c-------------------------Subroutine MEps_Double----------
c        Machine epsilon in double precision
c---------------------------------------------------------
      subroutine MEps_Double(le2,eps)
      implicit real*8 (a-h,o-z)
      uno=1.d+00
      dos=2.d+00
      eps=1.d+00
      le2=0
      q =uno+eps/dos
      do while (uno.ne.q)
        eps=eps/dos
        le2=le2-1
        q=uno+eps/dos
```

```
enddo

return
end
```