- **Desarrollo Pregunta 1**:

(a) **[15 puntos]**

Lo primero que se realizará es re-escribir la aproximación de forma explícita en función de m considerando que $n = 2^m$:

$$\left(1+\frac{1}{n}\right)^n = \left(1+2^{-m}\right)^{2^m}.$$

En particular en esta pregunta nos enfocaremos en la computación previo al cálculo de la potencia, es decir en $c_m = 1 + 2^{-m}$.

Para responder lo que ocurre con cada m solicitado, se analizará caso a caso:

$m = 51$: En este caso el coeficiente c_{51} es $1 + 2^{-51}$, lo cual es representable en *double precision*, y luego al elevarlo a 2^{51}, se obtiene, por lo mostrado en la Tabla 1, una muy buena aproximación de $\exp(1)$.

$m = 52$: En este caso el coeficiente c_{52} es $1 + 2^{-52}$, lo cual también es representable en *double precision*. Más aún, c_{52} es el sucesor de 1.0 en *double precision*. Y luego al elevarlo a 2^{52}, según la Tabla 1, corresponde de forma exacta al valor de $\exp(1)$ utilizado como referencia en el gráfico. Por esta razón la diferencia es exactamente 0.

$m = 53$: Por otro lado, en esta caso el coeficiente c_{53} corresponde a $1 + 2^{-53}$, y, según la regla de redondeo, el número almacenado será 1.0. Posteriormente al elevarlo a 2^{53}, el resultado no cambia. Por esta razón, y debido a pérdida de importancia, el error obtenido corresponde a $\log_{10}(|\exp(1) - 1.0|) \approx 0.23509439727547035$, que es exactamente el valor que entrega la Tabla 1.

(b) **[15 puntos]**

Primero, según lo presentado en la Figura 4.(b), podemos determinar que el mejor valor de n en el rango de valores presentados para $x = \left[10^{-2}, 10^{-1}\right]$ es $n = 2^{23}$. El error $\mathrm{Error}(x,n)$ es aproximadamente 10^{-9}. La Figura 4.(a) confirma que un buen candidato de valor n está en el vecindario de 2^{23}.

Segundo, la obtención del valor de $\log(y)$ a partir de la función $\exp(x)$ se puede lograr por medio de una búsqueda de raíces. En particular se puede buscar la raíz de la siguiente función $f(x) = y - \exp(x)$. Por ejemplo, si despejamos x se obtiene,

$$\begin{aligned} y - \exp(r) &= 0, \\ y &= \exp(r), \\ \log(y) &= r. \end{aligned}$$

Por lo tanto, la raíz r de $f(x)$ corresponde al valor de $\log(y)$. Ahora, en nuestro caso no tenemos acceso a la función exponencial $\exp(x)$, sino a una aproximación de esta, por lo tanto la función a la cual debemos buscar la raíz corresponde a,

$$f_n(x) = y - \left(1+\frac{x}{n}\right)^n.$$

A esta función le debemos calcular la derivada para aplicar el método de Newton, entonces,

$$\begin{aligned} f_n'(x) &= \frac{\mathrm{d}}{\mathrm{d}x}\left(y - \left(1+\frac{x}{n}\right)^n\right) \\ &= -n\left(1+\frac{x}{n}\right)^{n-1}\frac{1}{n} \\ &= -\left(1+\frac{x}{n}\right)^{n-1}. \end{aligned}$$

Por último, la iteración de Newton quedaría expresada de la siguiente forma:

$$\begin{aligned} x_0 &= 10^{-1.5}, \text{ valor en el intervalo de estudio.} \\ x_{i+1} &= x_i + \frac{y - \left(1+\frac{x}{n}\right)^n}{\left(1+\frac{x}{n}\right)^{n-1}}. \end{aligned}$$

(c) **[20 puntos]**

```
'''
input:
y       : (float) Input value for the approximation of log(y).
n       : (int) Integer value for the exponential approximation.
x0      : (float) Initial guess for Newton's method
k       : (int) Number of iterations to be used in Newton's method.

output:
ly      : (float) The approximation of log(y).
'''
def compute_log_y(y,n,x0,k):

    f = lambda x: y-np.power(1.+x/n,n)
    fp = lambda x: -np.power(1.+x/n,n-1)

    for k in np.arange(k):
        x1 = x0-f(x0)/fp(x0)
        x0 = x1
    ly = x1

    return ly
```

- **Desarrollo Pregunta 2**:

(a) **[15 puntos]** Es conveniente considerar que para todos los casos el condicional de la línea `13` es falso, por lo tanto concluye que no retornará `None` en ningún caso.

(I) Si $r_2 < c$, entonces

1) el condicional de la línea `18` es falso ya que el punto medio no es una raíz ($r_2 < c < r_3$) y
2) el condicional de la línea `20` es falso, ya que como el punto medio $c = (a+b)/2$ está a la derecha de r_2 entonces no hay cambio de signo. Por lo tanto, el algoritmo entra al `else` de la línea `23` buscando la raíz en el intervalo $[c, b]$ y por consecuencia encuentra la raíz r_3.

(II) Si $r_2 = c$, entonces

1) el condicional de la línea `18` es verdadero ya que el punto medio es una raíz ($r_2 = c$). Por lo tanto, el algoritmo entra al `if` de la línea `18` retornando la raíz r_2. Notar que en la línea `26` se obtiene c nuevamente, pero sigue siendo el punto medio.

(III) Si $r_2 > c$, entonces

1) el condicional de la línea `18` es falso ya que el punto medio no es una raíz y
2) el condicional de la línea `20` es verdadero, ya que como el punto medio $c = (a + b)/2$ está a la izquierda de r_2 entonces hay cambio de signo. Por lo tanto, el algoritmo entra al `elif` de la línea `20` buscando la raíz en el intervalo $[a, c]$ y por consecuencia encuentra la raíz r_1.

```
 1: def bisection(f, a, b, tol=1e-12):
 2:   """
 3:   input:
 4:   f   : (callable) function to evaluate.
 5:   a   : (double)   left value of interval.
 6:   b   : (double)   right value of interval.
 7:   tol : (double)   tolerance.
 8:
 9:   output:
10:   r   : (double)   root approximation of f.
11:   """
12:   fa,fb = f(a),f(b)
13:   if np.sign(fa*fb) > 0:
14:       return None
15:   while((b-a)/2 > tol):
16:     c = (a+b)/2
17:     fc = f(c)
18:     if fc == 0:
19:        break
20:     elif np.sign(fa*fc) < 0:
21:         b = c
22:         fb = fc
23:     else:
24:         a = c
25:         fa = fc
26: r = (a + b)/2
27: return r
```

(b) **[15 puntos]**

- Según (III), del contexto de esta pregunta, sabemos que $f''(x)$ tiene una raíz de multiplicidad 1, denominada x_3. Es decir $f''(x_3) = 0$. Esto implica:
  - No tiene otra raíz
  - $f'(x)$ es creciente/decreciente o decreciente/creciente en $[a, x_3]$ y $[x_3, b]$, respectivamente.
- Lo que implica que $f'(x)$ tiene una raíz en $[a, x_3]$, denominada x_1, y la otra en $[x_3, b]$, denominada x_2.
- Considerando ahora (I) y (II), podemos notar que:
  - $f(x)$ es creciente o decreciente en $[a, x_1]$ y tiene una raíz en el mismo intervalo denominada r_1. La cual se puede obtener ejecutando el método de la Bisección en el intervalo $[a, x_1]$.
  - $f(x)$ es decreciente o creciente (notar que alterna respecto a lo que ocurra en el intervalo $[x_1, x_2]$) y tiene una raíz en el mismo intervalo denominada r_2. La cual se puede obtener ejecutando el método de la Bisección en el intervalo $[x_1, x_2]$.
  - $f(x)$ es creciente o decreciente en $[x_2, b]$ y tiene una raíz en el mismo intervalo denominada r_3. La cual se puede obtener ejecutando el método de la Bisección en el intervalo $[x_2, b]$.
- Por lo tanto, se debe obtener primero x_3, luego x_1 y x_2, para finalmente obtener r_1, r_2, y r_3.

(c)
```
'''
input:
f   : (callable) triple-root-one function f.
fp  : (callable) derivative of the triple-root-one function f.
fpp : (callable) second derivative of the triple-root-one function f.
a   : (double)   left value of interval.
b   : (double)   right value of interval.

output:
r1   : (float) The approximation of the first root of the triple-root-one function f.
r2   : (float) The approximation of the second root of the triple-root-one function f.
r3   : (float) The approximation of the third root of the triple-root-one function f.
'''
def triple_root_one(f,fp,fpp,a,b):
```

- **[5 puntos]** Obtener la raíz x_3 de $f''(x)$ en $[a, b]$.
  ```
  x3 = bisection(fpp,a,b)
  ```
- **[5 puntos]** Obtener las raíces x_1 y x_2 de $f'(x)$ en $[a, b]$.
  ```
  x1 = bisection(f,a,x3)
  x2 = bisection(f,x3,b)
  ```
- **[10 puntos]** Obtener las raíces r_1, r_2 y r_3 de $f(x)$ en $[a, b]$.
  ```
  r1 = bisection(f,a,x1)
  r2 = bisection(f,x1,x2)
  r3 = bisection(f,x2,b)

  return r1,r2,r3
  ```