Tarea 1 **Calculo Computacional**

Victor Tortolero CI:24.569.609

Respuesta 1

Tenemos que $\frac{A+3}{13}$, como $A = 9$, tendríamos $\frac{9+3}{13} = \frac{12}{13}$. Ahora procedemos a convertir a binario.

$\frac{12}{13} \times 2 = \frac{24}{13}$, $b_0 = 1$
$\frac{11}{13} \times 2 = \frac{22}{13}$, $b_1 = 1$
$\frac{9}{13} \times 2 = \frac{18}{13}$, $b_2 = 1$
$\frac{5}{13} \times 2 = \frac{10}{13}$, $b_3 = 0$
$\frac{10}{13} \times 2 = \frac{20}{13}$, $b_4 = 1$
$\frac{7}{13} \times 2 = \frac{14}{13}$, $b_5 = 1$
$\frac{1}{13} \times 2 = \frac{2}{13}$, $b_6 = 0$

$\frac{2}{13} \times 2 = \frac{4}{13}$, $b_7 = 0$
$\frac{4}{13} \times 2 = \frac{8}{13}$, $b_8 = 0$
$\frac{8}{13} \times 2 = \frac{16}{13}$, $b_9 = 1$
$\frac{3}{13} \times 2 = \frac{6}{13}$, $b_{10} = 0$
$\frac{6}{13} \times 2 = \frac{12}{13}$, $b_{11} = 0$

Por lo tanto tenemos que:

$$0{,}1110110001001\overline{110110001001}\mathbf{1}\ldots$$

Observemos que el numero que vendria luego del bit 24 seria un 1. Entonces a la hora de redondear se suma 1. Por lo tanto, tenemos que $Fl(\frac{12}{13})_{Truncado} = 0{,}111011000100111011000100$, y que $Fl(\frac{12}{13})_{Redondeado} = 0{,}111011000100111011000101$.

Por Truncamiento tenemos que:

$$\begin{aligned} E_A = |x - Fl(x)_{Truncado}| &= 0,\underbrace{000\ldots000}_{\text{24 Ceros}}111011000100\ldots \\ &= 0,\underbrace{111011000100111011000100}_{\text{Esto es } \frac{12}{13}}\cdots \times 2^{-24} \\ &= \frac{12}{13} \times 2^{-24} \approx 5,50196 \times 10^{-8} \\ E_R = \frac{E_A}{|X|} &= \frac{\frac{12}{13} \times 2^{-24}}{\frac{12}{13}} \\ &= 2^{-24} \approx 5,96046 \times 10^{-8} \end{aligned}$$

Por Redondeo tenemos que:

$$\begin{aligned} E_A = |x - Fl(x)_{Redondeado}| &= |x - (Fl(x)_{Truncado} + 1 \times 2^{-24})| \\ &= |x - Fl(x)_{Truncado} - 1 \times 2^{-24}| \\ &= |\frac{12}{13} \times 2^{-24} - 1 \times 2^{-24}| \\ &= |\frac{12}{13} - 1| \times 2^{-24} \\ &= \frac{1}{13} \times 2^{-24} \approx 4,584 \times 10^{-9} \\ E_R = \frac{E_A}{|X|} &= \frac{\frac{1}{13} \times 2^{-24}}{\frac{12}{13}} \\ &= \frac{1}{12} \times 2^{-24} \approx 4,9670 \times 10^{-9} \end{aligned}$$

Respuesta 2

Respuesta 3

Respuesta 4

Respuesta 5

El mayor valor que llego a tomar la sumatoria fue **15.4036827008740234**. Fueron sumados **2097152 terminos**. La computadora no llega infinito al realizar la sumatoria debido a la precisión decimal, llega a un punto en que la computadora al sumar dos números, las magnitudes entre ellos son muy distintas y por lo tanto se queda con el numero mas grande y es como si no se le sumara nada.

Respuesta 6

- **Con precisión simple**: La mejor aproximación que se obtuvo para $x = 10$ fue $\boldsymbol{e^{10} = 22026{,}4667968750}$. Este resultado se obtuvo al sumar los términos desde un $n = 0$, y hasta que la suma dejara de "sumar"por las limitaciones de la precisión, y se llego a las **32 iteraciones**.

  Al cambiar el orden en que se suman los resultados, empezó desde $n = 32$ y se fue hasta $n = 0$, se obtuvo $\boldsymbol{e^{10} = 22026{,}464843750}$.

- **Con precisión doble**: La mejor aproximación que se obtuvo para $x = 10$ fue $\boldsymbol{e^{10} = 22026{,}4667968750}$. Este resultado se obtuvo al sumar los términos desde un $n = 0$, y hasta que la suma dejara de "sumar"por las limitaciones de la precisión, y se llego a las **32 iteraciones**.

  Al cambiar el orden en que se suman los resultados, empezó desde $n = 32$ y se fue hasta $n = 0$, se obtuvo $\boldsymbol{e^{10} = 22026{,}464843750}$.

Código Fuente

repuesta3.c

```c
#include <stdio.h>

void singlePrecision();
void doublePrecision();

int main(){
        singlePrecision();
        doublePrecision();
}

void singlePrecision(){
        float t=1.1, epsilon=1;
        int i=1;
        while(t > 1){
                t = 1 + (epsilon *= 0.5);
                printf("(%d)t=%.24lf, epsilon=%.24lf\n", i++, t, epsilon);
        }
}

void doublePrecision(){
        double t=1.1, epsilon=1;
        int i=1;
        while(t > 1){
                t = 1 + (epsilon *= 0.5);
                printf("(%d)t=%.24lf, epsilon=%.24lf\n", i++, t, epsilon);
        }
}
```

repuesta4.c

```c
#include <stdio.h>

float productoEscalarAscendenteF(float[], float[]);
float productoEscalarDescendenteF(float[], float[]);

double productoEscalarAscendenteD(double[], double[]);
double productoEscalarDescendenteD(double[], double[]);

int main(){
        float a_float[] = {2.718281828, -3.141592654, 1.414213562,
                                        0.5772156649, 0.3010299957};

        float b_float[] = {1485.2497, 878366.9879, -22.37492,
                                        4773714.647, 0.000185049};

        double a_double[] = {2.718281828, -3.141592654, 1.414213562,
                                          0.5772156649, 0.3010299957};

        double b_double[] = {1485.2497, 878366.9879, -22.37492,
                                          4773714.647, 0.000185049};

        printf("Ascendiente Precision Simple   = %.34f\n", productoEscalarAscendenteF(a_float, b_float));
        printf("Descendiente Precision Simple  = %.34f\n", productoEscalarDescendenteF(a_float, b_float));
        printf("----\n");
        printf("Ascendiente Precision Doble    = %.34lf\n", productoEscalarAscendenteD(a_double, b_double));
        printf("Descendiente Precision Doble   = %.34lf\n", productoEscalarDescendenteD(a_double, b_double));
}

// funciones float
float productoEscalarAscendenteF(float a[], float b[]){
        int i;
        float producto=0;
        for(i=0; i < 5; i++){
                //~ printf("%.30f x %.30f\n", a[i], b[i]);
                producto += a[i] * b[i];
        }
        return producto;
}

float productoEscalarDescendenteF(float a[], float b[]){
        int i;
        float producto=0;
        for(i=4; i >= 0; i--){
                //~ printf("%.30f x %.30f\n", a[i], b[i]);
                producto += a[i] * b[i];
        }
        return producto;
}


// funciones double
double productoEscalarAscendenteD(double a[], double b[]){
```

```
53          int i;
54          double producto=0;
55          for(i=0; i < 5; i++){
56                  producto += a[i] * b[i];
57          }
58          return producto;
59  }
60
61  double productoEscalarDescendenteD(double a[], double b[]){
62          int i;
63          double producto=0;
64          for(i=4; i >= 0; i--){
65                  producto += a[i] * b[i];
66          }
67          return producto;
68  }
```

repuesta5.c

```c
#include <stdio.h>
#include <math.h>

float serieArmonicaSingle();
double serieArmonicaDouble();

int main(){
        serieArmonicaSingle();
        printf("--------------\n");
        //~ serieArmonicaDouble();
}

float serieArmonicaSingle(){
        float serie=0, ant=1;
        float k=1;

        while(serie - ant != 0){
                ant = serie;
                serie += 1 / (k++);
                //~ printf("serie: %.10f\n", serie);
        }

        printf("%f terminos.\n", k);
        printf("Serie = %.24f\n", serie);

        return serie;
}

double serieArmonicaDouble(){
        double serie=0, ant=1;
        double k=1;

        while(serie - ant != 0){
                ant = serie;
                serie += 1 / (k++);
                //~ printf("serie: %.10f\n", serie);
        }

        printf("%lf terminos.\n", k);
        printf("Serie = %.10lf\n", serie);

        return serie;
}
```

repuesta6.c

```c
#include <stdio.h>
#include <math.h>

float exponencialAdelante(float);
float exponencialAtras(float);
float fact(float);
double exponencialAdelanteD(double);
double exponencialAtrasD(double);
double factD(double);

int main(){
        float x=10;
        printf("Precision Simple:\n");
        printf("\tHacia adelante = %.24f\n", exponencialAdelante(x));
        printf("\tHacia atras    = %.24f\n", exponencialAtras(x));
        printf("-------------------------------------------------------\n");
        printf("Precision Doble:\n");
        printf("\tHacia adelante = %.24lf\n", exponencialAdelanteD(x));
        printf("\tHacia atras    = %.24lf\n", exponencialAtrasD(x));


}

// Funciones float
float exponencialAdelante(float x){
        float e=0, factorial=0, potencia=0, ant=1, i=0;

        while(e != ant){
                factorial = factorial == 0 ? 1 : factorial * i;
                potencia = (i++) == 0 ? 1 : potencia * x;
                ant = e;
                e += potencia / factorial;
        }

        return e;
}

float exponencialAtras(float x){
        float e=0, factorial=0, potencia=0, n=32;

        while(n >= 0){
                factorial = fact(n);
                potencia = pow(x, (n--));
                e += potencia / factorial;
        }

        return e;
}

float fact(float n){
        int i;
        float f=1;
```

```
53          for(i=1; i <= n; i++){
54                  f *= i;
55          }
56          return f;
57  }
58
59
60  // Funciones double
61  double exponencialAdelanteD(double x){
62          double e=0, factorial=0, potencia=0, ant=1, i=0;
63
64          while(e != ant){
65                  factorial = (factorial == 0 ? 1 : factorial * i);
66                  potencia = ((i++) == 0 ? 1 : potencia * x);
67                  ant = e;
68                  e += potencia / factorial;
69          }
70
71          return e;
72  }
73
74  double exponencialAtrasD(double x){
75          double e=0, factorial=0, potencia=0, n=32;
76
77          while(n >= 0){
78                  factorial = fact(n);
79                  potencia = pow(x, (n--));
80                  e += potencia / factorial;
81          }
82
83          return e;
84  }
85
86  double factD(double n){
87          int i;
88          double f=1;
89          for(i=1; i <= n; i++){
90                  f *= i;
91          }
92          return f;
93  }
```