Problema 1

(a) O código abaixo foi usado para gerar os resultados do problema 1:

Algoritmo 1: Divisão

```
import numpy as np

# Lista 2 - Problema 1
# Tipos de variaveis disponiveis no pacote numpy usados no c digo:
# https://docs.scipy.org/doc/numpy/user/basics.types.html

def divisao(n):
    cont = 0
    while(1+n != 1):
        n = n/2
        cont = cont+1
        print(n)
    print("Numero de Iteracoes", cont)
    return n

y = np.double(0.5) # 0.5 em precisao dupla

y = divisao(y)
print("Precisao dupla: ", y)
```

O valor obtido usando precisão dupla é:

$1.1102230246251565e^{-16}$

Esse número somado ao número 1 durante nosso loop gera um número muito próximo de 1, porém o loop é finalizado pois a máquina considera o número obtido na última iteração como um número igual a 1 devido sua precisão

Para precisão simples utilizamos o código abaixo em C++ para gerar o resultado:

Algoritmo 2: Divisão

```
 int main()
{
    float n = 0.5;
    int cont = 0;
    while(1+n != 1)
    {
        n = n/2;
        cont++;
    }
    cout <<"N = " << n << endl;
    cout <<"Numero de iteracoes"<< cont << endl;
```

```
    return 0;
}
```

Obtivemos o numero **5.96046e-008** após **23** iterações.

(b) Foram executadas 52 iterações do algoritmo para precisão dupla e 23 iterações para precisão simples, esses números podem ser relacionados com o número de bits na mantissa de um número com precisão dupla e simples, que são exatamente 52 e 23 bits, respectivamente.

Problema 2

(a) Para representar a fórmula de Bhaskara e uma equação quadrática em python, desenvolvemos o seguinte algoritmo:

Algoritmo 3: Fórmula de Bhaskara

```python
import math
import numpy as np

# Lista 2 - Problema 2
# Representacao de uma equacao quadratica: ax^2 + bx + c = 0

def bhaskara(a, b, c):
    delta = b**2 - (4*a*c)
    if(a == 0):
        x1 = -c/b
        print("x1: ",x1)
        return x1

    if(delta < 0):
        return print("O delta possui uma raiz imaginaria")

    x1 = (-b - math.sqrt(delta))/(2*a)
    x2 = (-b + math.sqrt(delta))/(2*a)

    print("x1: ", x1)
    print("x2: ", x2)
    raizes = [x1,x2]
    return raizes

print("Para a equacao 6x^2 + 5x - 4 foi obtido as raizes: ")
bhaskara(6,5,-4)

print("Para a equacao 6*10^30x^2 + 5*10^30x + 4*10^30 foi obtido as raizes: ")
bhaskara(6*(10^30),5*(10^30),-4*(10^30))

print("Para a equacao x + 1 foi obtido as raizes: ")
```

```
bhaskara(0,1,1)

print("Para a equacao x^2  - 10*^5x + 1 foi obtido as raizes: ")
bhaskara(1,-10**5,1)

print("Para a equacao x^2  - 4 + 3.999999 foi obtido as raizes: ")
bhaskara(1,-4,3.999999)

print("Para a equacao 10^-30x^2  - 10*^30x + 10^30 foi obtido as raizes: ")
bhaskara(10*(10^-30),-10*(10^-30),10*(10^-30))
```

(b)

1. Para a equação:
$$6x^2 + 5x - 4 = 0$$
Obtivemos as mesmas raizes, **x1 = -1,333333 e x2 = 0,5**
2. Para a equação:
$$6.10^{30}.x^2 + 5.10^{30}.x - 4.10^{30} = 0$$
Também obtivemos as mesmas raizes, **x1 = -1,33333 e x2 = 0,5**
3. Para a equação:
$$x - 1 = 0$$
Foi obtido a mesma raiz mostrada na tabela **x1 = -1**
4. Para a equação:
$$x^2 - 10^{-5} + 1$$
Foi obtido **x1 = 1.00000033853557559.10**-5 e x2 = 99999.99999**. Pode-se perceber que ocorreu um erro durante o cálculo de x1, o x1 encontrado se diferencia do x1 da tabela por alguns bits.
5. Para a equação:
$$x^2 - 4 + 3.999999$$
Obtivemos **x1 = 1.9989999999999302 e x2 = 2.00100000000007**. Ambos os números são muito próximos das raizes da tabela. Para o caso de x1 podemos arredondar o número encontrado e obter a raiz x1 (1.999) descrita na tabela. Para x2 podemos fazer um truncamento e também obter o mesmo x2 descrito na tabela (2.001).
6. Para a equação:
$$10^{30}.x^2 - 10^{30}.x + 10^{30} = 0$$
Não obtivemos nenhum resultado, pois encontramos um delta negativo ao calcular essa equação, ou seja, as raizes a serem encontrada seriam números complexos. Ao chegar em um delta negativo, a função dispara apenas um erro avisando ao usuário que o delta de sua equação possui uma raiz imaginária.

(c) Para chegar a resultados mais próximos dos encontrados na equação 5, podemos aplicar as técnicas de truncamento/arredondamento citadas no item anterior. No caso da equação 6, podemos utilizar funções disponíveis pelo python para obter a parte real e imaginária das raízes. As funções utilizadas nesse experimento podem ser encontradas em: Python Documentation. Para realizar a aritmética complexa utilizamos o pacote cmath, após algumas mudanças no código, obtivemos uma fórmula de Bhaskara com suporte a números complexos, que será mostrada abaixo:

Algoritmo 4: Fórmula de Bhaskara com suporte a números complexos

```
import math
import numpy as np
import cmath

# Lista 2 - Problema 2
# Representacao de uma equacao quadratica: ax^2 + bx + c = 0

def bhaskara(a, b, c):
    delta = b**2 - (4*a*c)
    if(a == 0):
        x1 = -c/b
        print("x1: ",x1)
        return x1

    x1 = (-b - (cmath.sqrt(delta) if delta < 0 else math.sqrt(delta)))/(2*a)
    x2 = (-b + (cmath.sqrt(delta) if delta < 0 else math.sqrt(delta)))/(2*a)

    print("x1: ", x1)
    print("x2: ", x2)
    raizes = [x1,x2]
    return raizes
```

Com isso, obtemos as raízes **x1 = 0.0 e x2 = 1e+60**. Pode-se, pois, perceber, que o x2 obtido é exatamente igual ao x2 descrito na tabela, porém o x1 obtido difere do x1 da tabela. Encontramos 0.0, enquanto o valor exato é 1.

Problema 3

(a) Algoritmo em Python da fórmula de diferenças finitas:

Algoritmo 5: Fórmula de diferenças finitas

```
import numpy as np
import math

# Lista 2 - Problema 3

def derivada(x, h):
```

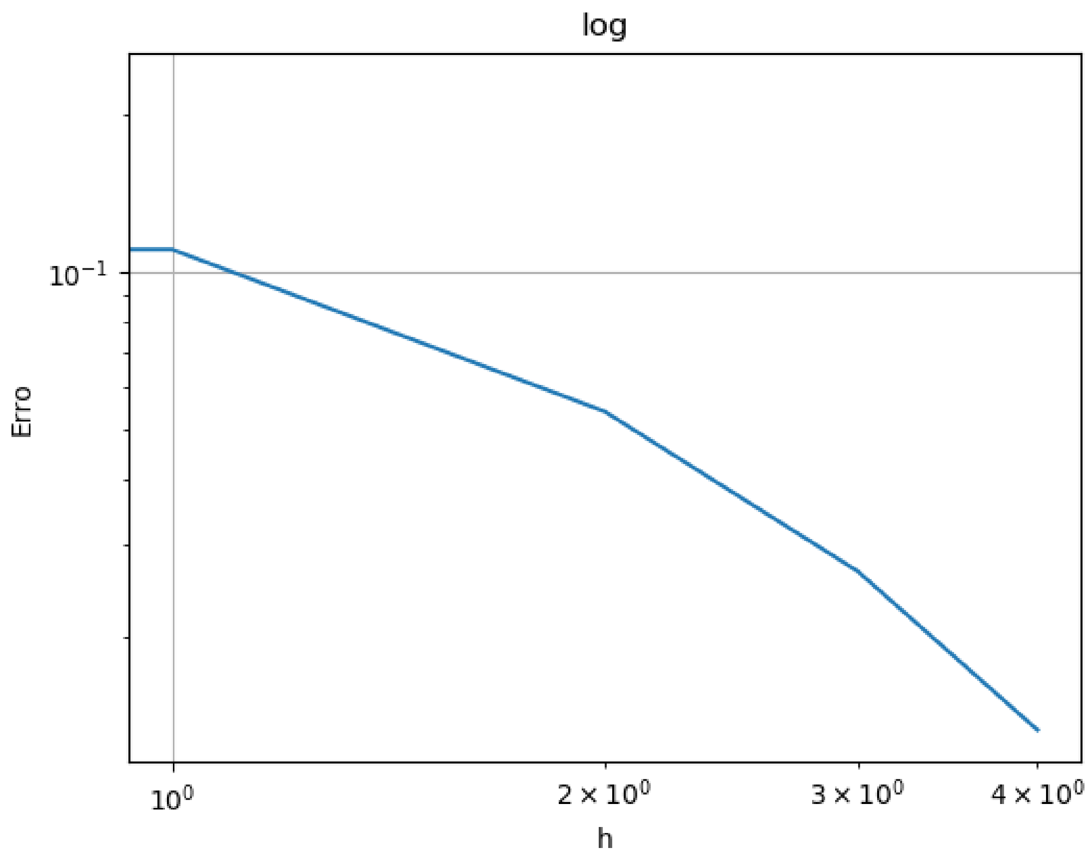
```
    return  (np.sin(x+h) - np.sin(x)) / h


derivada = derivada(1, 1)
print("Derivada encontrada pelo m todo: ", derivada)
print("Derivada exata: ", np.cos(1))
print("Erro: ", np.cos(1) - derivada)
```

Para o código acima, foi encontrado:

- Derivada pelo método para h = 1: 0.0678264420177852
- Derivada exata: 0.5403023058681398
- Erro: 0.47247586385035456

(b) Gráfico que demonstra a propagação do erro:



Demonstração do erro para 1/2, 1/4, 1/8... em escala logarítimica

(c) Sim, existe um valor mínimo para a magnitude do erro, que no caso, é o menor número possível a ser representado pela máquina através de h. Nesse caso usamos precisão dupla para as variáveis na função. Portanto o valor é $1.1102230246251565e^{-16}$ para a linguagem utilizada, Python.

(d) Algoritmo em Python da fórmula de aproximação por diferença central:

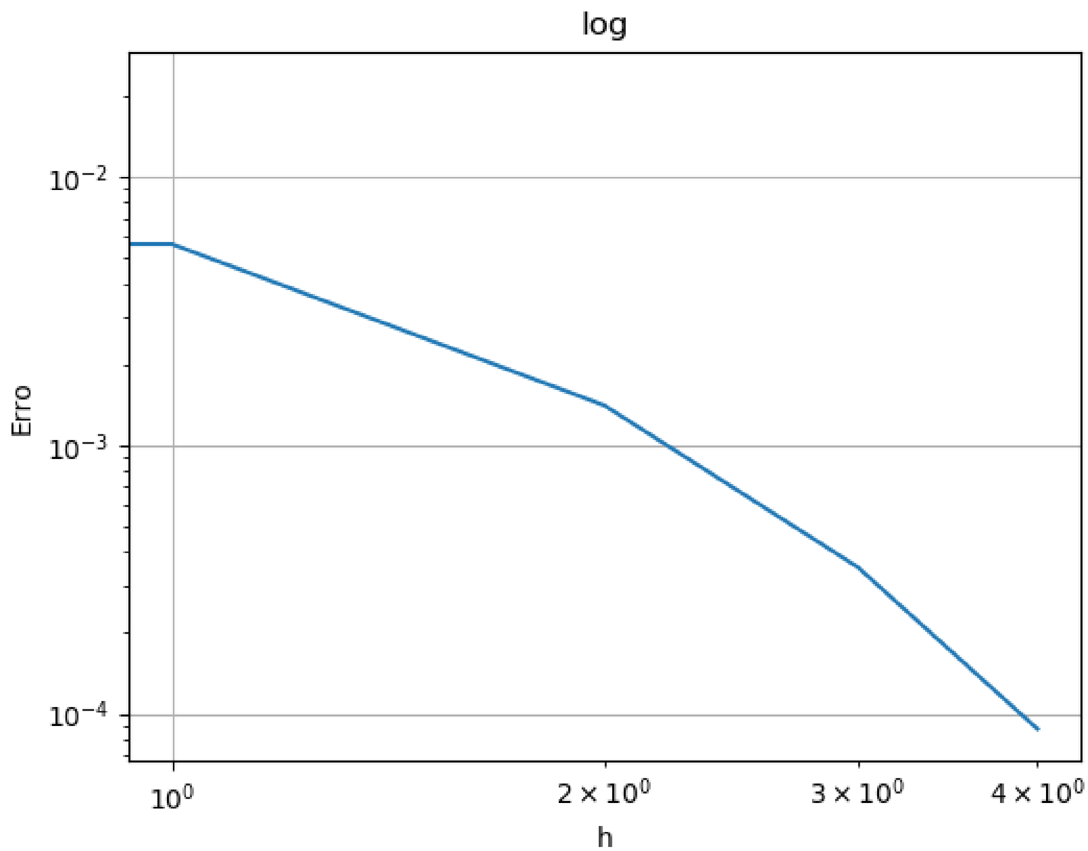
Algoritmo 6: Fórmula de aproximação por diferença central

```
import math
import matplotlib.pyplot as plt
    def diferencaCentral(x, h):
        return (math.sin(x+h) - math.sin(x - h))/(2*h)
```

Para o código acima, foi encontrado:

- Derivada pelo método para h = 1: 0.45464871341284085
- Derivada exata para h = 1: 0.5403023058681398
- Erro: 0.08565359245529891

Gráfico que demonstra a propagação do erro usando a aproximação por diferença central:



O valor mínimo para a magnitude do erro é duas vezes o valor encontrado na letra (c). Ou seja, $2.220446049e^{-16}$