TUGAS METODE NUMERIK – 02

Dosen : Dr. Taufik Sutanto, M.Sc.Tech., Ph.D

Nama Anggota Kelompok

1. **Abdul Haffizh - 11160940000056**
2. **Sharfinna Zaldy - 11190940000011**
3. **Ruly Erhandi - 11190940000012**
4. **Solihatun Khasanah - 11190940000015**
5. **Salsabila Farah Harissaputri - 11190940000029**
6. **Shabrina Eriyanti - 11190940000031**

SOAL NO 1

Perhatikan persamaan berikut $2x^2 - 5x = 0$

a. Dengan menggunakan metode **bisection**, dan tebakan awal a=1 serta b=9, tentukan Error relatifnya setelah tiga iterasi [C yang ke-tiga, terhadap X*].
b. Dengan menggunakan metode **Regula Falsi**, dan tebakan awal a=1 serta b=9, tentukan Error relatifnya setelah tiga iterasi [C yang ke-tiga, terhadap X*].

JAWABAN :

Diketahui : Persamaan $2x^2 - 5x = 0$

a = 1

b = 9

Ditanya : Error relative setelah tigas iterasi [C yang ke-tiga terhadap X*] ?

Penyelesaian :

1a.) Menggunakan metode bisection

Iterasi 1

a = 1 , f(a) = -3
b = 9 , f(b) = 117
$c_1 = \frac{1+9}{2} = 5$, f(c) = 25

$f(a) \cdot f(c) = -75 < 0$

Iterasi 2

$a_1 = 1$, $f(a_1) = -3$
$b_1 = 5$, $f(b_1) = -3$

$$c_2 = \frac{1+5}{2} = 3, \quad f(c_1) = 3$$

$$f(a)\cdot f(c) = -9 < 0$$

Iterasi 3

$a_2 = 1$, $f(a_2) = -3$

$b_2 = 3$, $f(b_2) = 3$

$$c_3 = \frac{1+3}{2} = 2, \quad f(c_2) = -2$$

$$f(a)\cdot f(c) = 6 > 0$$

❖ **Error relatif dari perasamaan $2x^2 - 5x = 0$ setlah tiga iterasi adalah**

$$= \frac{(c_3 - c_2)}{c_3} = \frac{(2-3)}{2} = 0.5$$

❖ **Menggunakan Python**

In [3]:
```
# No 1A Aplikasi Bisection di Python

import math

def f(x):
    return 2*x**2-5*x

a = 1
b = 9
n = 3 # iterasi

if f(a)*f(b)<0:
    for i in range(n):
        c = (a+b)/2 # THE BiSection
        print(a,b,c,f(c))
        if f(a)*f(c)<0:
            b = c
        else:
            a = c
else:
    print('Error, a dan b tidak mengapit akar')
```

```
1 9 5.0 25.0
1 5.0 3.0 3.0
1 3.0 2.0 -2.0
```

dari hasil diatas kita mengambil nilai iterasi 2 dan 3 kemudian kita simbolkan dengan c2 dan c3 untuk mencari nilai relatif errornya.

```
In [2]: # No 1A Error Relatif
        c2 = 3
        c3 = 2
        error_relatif = abs(c3 - c2)/c3
        print ("Error relatif setelah tiga iterasi adalah", error_relatif)

        Error relatif setelah tiga iterasi adalah 0.5
```

1b) Menggunakan metode Regula Falsi

Iterasi 1

a = 1 , f(a) = -3
b = 9 , f(b) = 117

$$c_1 = b - \frac{f(b)[b-a]}{f(b)-f(a)}$$

C_1= 1.2000 , f(c) = -3.12

Iterasi 2

a_2 = 1.2 , $f(a_2)$ = -3.12
b_2 = 9 , $f(b_2)$ = 117

$$c_2 = b - \frac{f(b_2)[b_2-a_2]}{f(b_2)-f(a_2)}$$

C_2= 1.4025 , $f(c_2)$ = -3.0784

Iterasi 3

a_3 = 1.4025 , $f(a_3)$ = 3.0784
b_3 = 9 , $f(b_3)$ = 117

$$c_3 = b - \frac{f(b_3)[b_3-a_3]}{f(b_3)-f(a_3)}$$

C_3= 1.5973 , $f(c_3)$ = -2.8836

❖ **Error relatif dari perasamaan $2x^2 - 5x = 0$ setlah tiga iterasi adalah**

$$= \frac{(c_3-c_2)}{c_3} = \frac{(1.5973-1.4025)}{1.5973} = 0.1219$$

❖ **Menggunakan Python**

In [3]:
```
# NO 1B Aplikasi Regula Falsi di Python

def f(x):
    return 2*x**2-5*x #x*math.sin(x) - 1

a = 1
b = 9
n = 3 # iterasi

if f(a)*f(b)<0:
    for i in range(n):
        c = b - (f(b)*(b-a)) / (f(b)-f(a)) # THE RegulaFalsi
        print(a,b,c,f(c))
        if f(a)*f(c)<0:
            b = c
        else:
            a = c
else:
    print('Error, a dan b tidak mengapit akar')
```

```
1 9 1.2000000000000002 -3.12
1.2000000000000002 9 1.4025974025974026 -3.0784280654410523
1.4025974025974026 9 1.5973705834018075 -2.8836673555741754
```

Kesimpulan :

Error relative dapat dicari dengan membagi selisih antara nilai sebenarnya (x) dan nilai observasi atau nilai pendekatan ($x^{'}$) dengan nilai sebenarnya (x). Berdasarkan hasil diatas dapat diketahui jika kita memakai metode Bisection mendapatkan error relative 0.5 ,sedangkan metode regula falsi mendapatkan 0.1219 yang artinya lebih baik menggunakan metode falsi.

SOAL NO 2

Perhatikan persamaan berikut $x^3+x^2-3x=3,$

a. Dengan menggunakan metode Newton, dan tebakan awal $x_0=1$, tentukan Error relatifnya setelah tiga iterasi [X_3 yang ke-tiga, terhadap X*].
b. Dengan menggunakan metode Secant, dan tebakan awal $x_0=1$ dan h=0.01, tentukan Error relatifnya setelah tiga iterasi [X_3 yang ke-tiga, terhadap X*].

JAWABAN :

Metode Newton-Rapshon

Diketahuhi:

$f(x) = x^3 + x^2 - 3x = 3$

$f(x) = x^3 + x^2 - 3x - 3$

$f'(x) = 3x^2 + 2x - 3$

Iterasi ke 1

$x_0 = 1$

$$f(x_0) = x^3 + x^2 - 3x - 3$$
$$f(1) = 1^3 + 1^2 - 1x - 3 = \quad -4$$
$$f'(x_0) = 3x^2 + 2x - 3$$
$$f'(x_0) = 3.1^2 + 2.1 - 3 = 2$$
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{(-4)}{2} = 3$$

Iterasi ke 2

$$x_1 = 3$$
$$f(x_1) = 24$$
$$f'(x_1) = 30$$
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 3 - \frac{24}{30} = 2,2$$

Iterasi ke 3

$$x_2 = 2,2$$
$$f(x_2) = 5,888$$
$$f'(x_2) = 15,92$$
$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 2,2 - \frac{5,888}{15,92} = 1,83015075376844$$

In [30]:
```
# No 2A Aplikasi Newton-Rhapson di Python

def f(x):
    return x**3 + x**2 -3*x -3 # x**3-3*x+2

def df(x): # Turunan pertama f
    return 3*x**2 + 2*x - 3 #3*x**2-3

x = 1 # -2.4 # Initial point/guess
n = 3 # Jumlah iterasi

print(x, end = ', ')
for i in range(n):
    x = x - f(x)/(df(x))
    print(x, end = ', ')

# Nilai Error Relatif
error_relatif = abs(2.2-1.732)/1.732
print ("     ")
print ("     ")
print ("Error relatif setelah tiga iterasi adalah", error_relatif)
```

```
1, 3.0, 2.2, 1.8301507537688444,

Error relatif setelah tiga iterasi adalah 0.2702078521939955
```

Metode Secant

Diketahui :

$$f(x) = x^3 + x^2 - 3x - 3$$

$x_0 = 1$

$h = 0,01$

Iterasi ke 1

$x_0 = 1$

$f(x_0) = x^3 + x^2 - 3x - 3$

$f(1) = 1^3 + 1^2 - 1x - 3 = \ -4$

$f(x_0 + h) = x^3 + x^2 - 3x - 3$

$f(1,01) = (1,01)^3 + (1,01)^2 - 3(1,01) - 3 = -3,979599$

$f'(x_0) = \frac{f(x_0+h)-f(x_0)}{h} = \frac{-3,979599-(-4)}{0,01} = 2,0401$

$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{(-4)}{2,0401} = 2,960688$

Iterasi ke 2

$x_1 = 2,960688$

$f(x_1) = 22,8334$

$f(x_1 + h) = x^3 + x^2 - 3x - 3$

$f(2,970688) = (2,970688)^3 + (2,970688)^2 - 3(2,970688) - 3 = 23,1266$

$f'(x_1) = \frac{f(x_1+h)-f(x_1)}{h} = \frac{23,1266-22,8334}{0,01} = 29,32$

$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2,960688 - \frac{22,8334}{29,32} = 2,1818$

Iterasi ke 3

$x_2 = 2,1818$

$f(x_2) = 5,6007$

$f(x_2 + h) = x^3 + x^2 - 3x - 3$

$f(2,1918) = (2,1918)^3 + (2,1918)^2 - 3(2,1918) - 3 = 5,7579$

$f'(x_2) = \frac{f(x_2+h)-f(x_2)}{h} = \frac{5,7579-5,6007}{0,01} = 15,72$

$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 2,1818 - \frac{5,6007}{15,72} = 1,8255$

```
In [5]: # No 2B Aplikasi Metode Secant di Python

def dfs(x, h=10**-2):
    return (f(x+h)-f(x))/h # Aproksimasi Turunan pertama f

x = 1 # -2.4 # Initial point/guess
n = 3 # Jumlah iterasi

print(x, end = ', ')
for i in range(n):
    x = x - f(x)/(dfs(x))
    print(x, end = ', ')

# Nilai Error Relatif
error_relatif = abs(2.9606-1.8255)/2.9606
error_relatif

1, 2.9606882015587788, 2.1817616576450876, 1.8255016711899639,

Out[5]: 0.38340201310545163
```

Kesimpulan :

Berdasarkan hasil diatas setelah dibuat tiga iterasi dengan menggunakan 2 metode mendapatkan hasil yang hamper sama. Hal itu karna metode secant memiliki kemiripan dengan metode newton-rapson, akan tetapi metode secant lebih efesien untuk digunakan karna kalau metode newton-repson memerlukan lebih banyak langkat iterasi untuk mengukur tingkat akurasinya dan metode secant juga dapat menyelesaikan masalah yang terdapat pada metode newton-rapson yang terkadang sulit ditemukan.

SOAL NO 3

Jika fungsi **y=f(x)=3sin(2x)** didefinisikan pada interval [0, 1], dan sebuah fungsi aproksimasi akan dibuat menggunakan panjang interval yang seragam, maka tentukanlah:

a. $P_1(x)$ dan $P_2(x)$ menggunakan Polynomial Newton.
b. Tentukan batas atas error dari pendekatan di soal sebelumnya.

JAWABAN :

a. Diketahui

$y = 3\,sin\,(2x)$ pada interval $[0,\ 1]$

maka, $x_0 = 0,\ x_1 = \frac{1}{2},\ x_3 = 1$

Rumus Umum Polynomial Newton

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \ldots + a_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

Ditanya

$P_1(x)$ dan $P_2(x)$

Jawaban

x_i	$f(x_i)$	$f(x_i, x_{i-1})$	$f(x_i, x_{i-1}, x_{i-2})$
0	0		
$\frac{1}{2}$	2,5244129544	5,0488259088	
1	2,7278922805	0,4069586521	-4,6418672567

$$a_0 = f(x_0) = 0$$

$$a_1 = f(x_1, x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{3\ sin\ (1) - 0}{\frac{1}{2} - 0} = \frac{2,5244}{\frac{1}{2}} = 5,0488$$

$$a_2 = f(x_2, x_1, x_0) = \frac{f(x_2, x_1) - f(x_1, x_0)}{x_2 - x_0} = = \frac{\frac{3\ sin\ (2)}{1 - \frac{1}{2}} - 5,0488}{1-0} = -4,6419$$

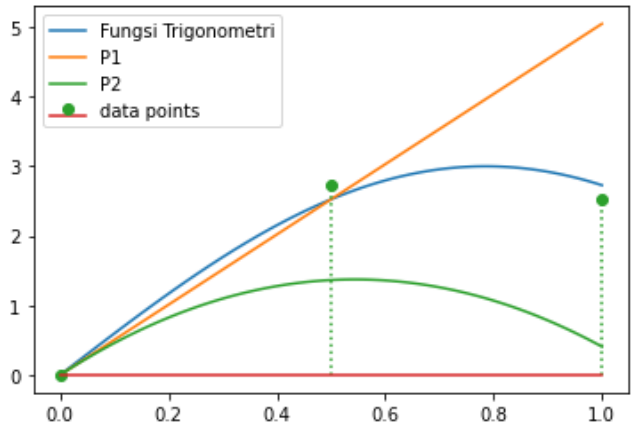
$P_1(x)$ dan $P_2(x)$

$$P_1(x) = a_0 + a_1(x - x_0)$$
$$= 0 + 5,0488(x - 0)$$
$$= 5,0488x$$

$$P_2(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1)$$
$$= 0 + 5,0488(x - 0) + (-4,6419)(x - 0)(x - \frac{1}{2})$$
$$= 5,0488x - 4,6419x^2 + 2,3209$$

```
In [1]: import warnings; warnings.simplefilter('ignore')
        import numpy as np, matplotlib.pyplot as plt, math

        def f(x): # Fungsi Trigonometri
            return 3*math.sin(2*x)

        def pol_2(x): # Fungsi Pendekatan Polinomial Derajat 2
            return 5.0488*x

        def che_2(x): # Fungsi Pendekatan Chebisev Derajat 2
            return 5.0488*x - 4.6419*x**2
```

```
In [2]: X = np.linspace(0, 1, 100)
        y1 = [f(x) for x in X]
        y2 = [pol_2(x) for x in X]
        y3 = [che_2(x) for x in X]

        x_data = [0, 0.5, 1]
        y_data = [0, 2.727 , 2.524]

        plt.plot(X, y1, label = "Fungsi Trigonometri")
        plt.plot(X, y2, label = "P1")
        plt.plot(X, y3, label = "P2")
        plt.stem(x_data, y_data, label = "data points", linefmt='C2:', markerfmt = 'C2o')
        plt.legend()
        plt.show()
```




```
In [3]: error_poly = [abs(y_f-y_pol) for y_f,y_pol in zip(y1,y2)]
        error_che = [abs(y_f-y_che) for y_f,y_che in zip(y1,y3)]

        print('Maksimum Error pendekatan P1 = ', max(error_poly))
        print('Maksimum Error pendekatan P2= ', max(error_che))
```

```
Maksimum Error pendekatan P1 =  2.320907719522955
Maksimum Error pendekatan P2=  2.3209922804770446
```

Kesimpulan :

Metode polynomial ini merupakan yang paling mudah dibentuk dan paling efisien untuk dihitung serta polynomial newton ini lebih akurat hasilnya.

SOAL NO 4

Diketahui sebuah matriks

$$A = (1.5\ 0.5\ 0.5\ 1.5)$$

Dengan metode metode Power Method dan Rayleigh Quotient serta vector awal $X_0=[0\ 1]^T$ tentukan ke- 2 error relative dari kedua metode tersebut setelah 5 iterasi.

JAWABAN :

a. Mencari Power Method

$A = [1.5\ 0.5\ 0.5\ 1.5]$ $\qquad$ $X_0 = [0,\ 1]^T$

$X_{n+1} = AX_n$

$X_1 = AX_0 = [1.5\ 0.5\ 0.5\ 1.5]\ [0\ 1] = [0.5\ 1.5]$

$X_2 = AX_1 = [1.5\ 0.5\ 0.5\ 1.5]\ [0.5\ 1.5] = [1.5\ 2.5]$

$X_3 = AX_2 = [1.5\ 0.5\ 0.5\ 1.5]\ [1.5\ 2.5] = [3.5\ 4.5]$

$X_4 = AX_3 = [1.5\ 0.5\ 0.5\ 1.5]\ [3.5\ 4.5] = [7.5\ 8.5]$

$X_5 = AX_4 = [1.5\ 0.5\ 0.5\ 1.5]\ [7.5\ 8.5] = [15.5\ 16.5]$

$\lambda_1^{(1)} = \frac{Maks\ X_2}{Maks\ X_1} = \frac{2.5}{1.5} = 1.667$

$\lambda_1^{(2)} = \frac{Maks\ X_3}{Maks\ X_2} = \frac{4.5}{2.5} = 1.8$

$\lambda_1^{(3)} = \frac{Maks\ X_4}{Maks\ X_3} = \frac{8.5}{4.5} = 1.889$

$\lambda_1^{(4)} = \frac{Maks\ X_5}{Maks\ X_4} = 1.941176471$

$$E_{relatif} = \frac{\left|\lambda_1^{(4)} - Nilai\ Eksak\right|}{|Nilai\ Eksak|}$$

$$= \frac{|1.941176471-2|}{|2|}$$

$$= \frac{0.058823529}{2}$$

$$= 0.029411765$$

In [21]:
```
# No 4A Metode power setelah5 iterasi

A = np.array([[ 1.5, 0.5 ],[ 0.5, 1.5 ]])
x = np.array([0,1]).transpose()
N = 5
for i in range(N):
    xo = x
    x = A.dot(x)
    x1 = x
    eigen = max(abs(x1))/max(abs(xo))
print('eigenvalue = ', eigen)
print('eigenvector = ', x)
```

```
eigenvalue =  1.9411764705882353
eigenvector =  [15.5 16.5]
```

b. Rayleigh

$$X_5 = [15.5\ 16.5\ ]$$

$$\lambda = \frac{X^T AX}{X^T X}$$

$$\lambda = \frac{[15.5\ 16.5\ ][1.5\ 0.5\ 0.5\ 1.5\ ][15.5\ 16.5\ ]}{[15.5\ 16.5\ ]\ [15.5\ 16.5\ ]}$$

$$= \frac{[31.5\ 32.5\ ]\ [15.5\ 16.5\ ]}{[15.5\ 16.5\ ]\ [15.5\ 16.5\ ]}$$

$$= \frac{1024.5}{512.5}$$

$$= 1.99902439$$

$$E_{relatif} = \frac{|\lambda - Nilai\ Eksak|}{|Nilai\ Eksak|}$$

$$= \frac{|1.99902439-2|}{|2|}$$

$$= \frac{0.0009756}{2}$$

$$= 0.000487805$$

In [39]:
```
# No 4B Rayleigh setelah iterasi

import numpy as np

A = np.array([[ 1.5, 0.5 ],[ 0.5, 1.5 ]])
x = np.array([0,1]).transpose()
N = 5
eksak = 2
for i in range(N):
    xo = x
    x = A.dot(x)
    x1 = x
    eigen = x.transpose().dot(A).dot(x)/x.transpose().dot(x)
print('eigen value = ', eigen)
print('error Relatif = ', abs((eigen-eksak)/eksak)
```

```
  File "<ipython-input-39-ba03a9600075>", line 15
    print('error Relatif = ', abs((eigen-eksak)/eksak)
                                                       ^
SyntaxError: unexpected EOF while parsing
```

In [36]:
```
eVa, eVe = np.linalg.eig(A)
print('Eigenvalues \n',eVa)
print('Eigenvectors \n',eVe)
```

```
Eigenvalues
 [2. 1.]
Eigenvectors
 [[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```

In [40]:
```
#Rayleigh dan Power Method 3
import numpy as np
A = np.array([[ 1.5, 0.5 ],[ 0.5, 1.5 ]])
x = np.array([0,1]).transpose()
N = 5
eksak = 2
for i in range(N):
    xo = x
    x = A.dot(x)
    x1 = x
    eigenPowerMethod = max(abs(x1))/max(abs(xo))
    eigenRayleigh = x.transpose().dot(A).dot(x)/x.transpose().dot(x)

print('EigenValue Menurut metode Rayleigh 5 iterasi =', eigenRayleigh)
print('Eigenvalue Menurut metode Powermethod 5 iterasi = ', eigenPowerMethod)

print('Error Relatif (Power Method)= ', abs(eigenPowerMethod-eksak)/eksak)
print('Error Relatif (Rayleigh)= ', abs(eigenRayleigh-eksak)/eksak)
print('Eigen vector = ', x/x.max())
```

```
EigenValue Menurut metode Rayleigh 5 iterasi = 1.9990243902439024
Eigenvalue Menurut metode Powermethod 5 iterasi =  1.9411764705882353
Error Relatif (Power Method)=  0.02941176470588236
Error Relatif (Rayleigh)=  0.0004878048780487809
Eigen vector =  [0.93939394 1.        ]
```

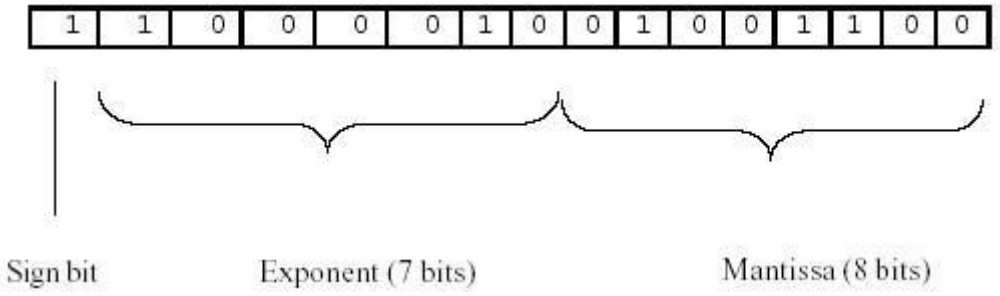
Kesimpulan :

Dari hasil percobaan kedua metode diatas dari nilai eigen dapat kita lihat bahwa metode power lebih efisien dari dapa metode Rayleigh dan untuk error relative nya justru metode rayligh lebih baik daripada metode power.

SOAL NO 5

Jika diketahui sebuah system bilangan binari menggunakan 2 bytes (16 binary digits) seperti pada gambar dibawah. Digit awal digunakan untuk menyatakan sign (0 negatif dan 1 positif). Lalu 7 digit berikutnya adalah **exponent** (angka di depan koma) dan sisanya adalah **mantissa** (angka dibelakang koma). Tentukan:

1. Bilangan terbesar pada system ini (overflow level)
2. Bilangan positif terkecil system ini (undeflow value)
3. Epsilon computer/system ini.
4. Banyaknya bilangan floating point yang dapat direpresentasikan system ini.



JAWABAN :

1. Untuk mencari bilangan terbesar pada system (overflow level), kita hanya perlu memasukkan angka 1 pada setiap bits exponent dan mantissa. Lalu didapatkan

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

kemudian konversi bilangan binari tersebut ke bilangan desimal.

a) Untuk angka didepan titik (Exponent) :

$= \left(1 \times 2^{6}\right) + \left(1 \times 2^{5}\right) + \left(1 \times 2^{4}\right) + \left(1 \times 2^{3}\right) + \left(1 \times 2^{2}\right) + \left(1 \times 2^{1}\right) + \left(1 \times 2^{0}\right)$

$= 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= 127$

b) Untuk angka dibelakang titik (Mantissa) :

$= (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-5}) + (1 \times 2^{-6})$

$+ (1 \times 2^{-7}) + (1 \times 2^{-8})$

$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256}$

$= \frac{255}{256}$

$= 0,99609357$

a + b = $127,99609357$

Sehingga, bilangan terbesar pada system (overflow level) adalah $127,99609357$.

❖ Menggunakan Python

```
In [1]: #a. Bilangan terbesar pada system ini (overflow level)
        sum_ = -1
        for i in range(9):
            sum_ += (1/(2**i))

        sum_a = 0
        for i in range(7):
            sum_a += (2**i)

        b = (sum_ + sum_a)

        print(b)
```

```
127.99609375
```

2. Untuk mencari bilangan positif terkecil system (undeflow value), kita hanya perlu memasukkan angka 0 pada setiap bits exponent dan mantissa kecuali bits mantissa terakhir diberi angka 1. Lalu didapatkan

0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

kemudian konversi bilangan binari tersebut ke bilangan desimal.

a) Untuk angka didepan titik (Exponent) :

$$= \left(0 \times 2^{6}\right) + \left(0 \times 2^{5}\right) + \left(0 \times 2^{4}\right) + \left(0 \times 2^{3}\right) + \left(0 \times 2^{2}\right) + \left(0 \times 2^{1}\right) + \left(0 \times 2^{0}\right)$$

$$= 0$$

b) Untuk angka dibelakang titik (Mantissa) :

$$= (0 \times 2^{-1}) + (0 \times 2^{-2}) + (0 \times 2^{-3}) + (0 \times 2^{-4}) + (0 \times 2^{-5}) + (0 \times 2^{-6})$$

$$+ (0 \times 2^{-7}) + (1 \times 2^{-8})$$

$$= \frac{1}{256}$$

$$= 0,00390625$$

a + b = $0,00390625$

Sehingga, bilangan positif terkecil system (undeflow value) adalah $0,00390625$

3. Untuk mencari Epsilon computer/system, kita dapat menggunakan Python.

```
In [2]: #c. Epsilon computer/system ini
        import numpy as np
        print(np.finfo(np.float16).eps)

        0.000977
```

Pada python, kita gunakan modul numpy untuk mencari epsilon dari data type float 16, yaitu Half precision float dikarenakan pada soal, diberikan system bilangan binari 16 digit dan diperoleh hasil epsilon adalah $0,000977$.

4. Untuk mencari banyaknya bilangan floating point yang dapat direpresentasikan system, kita hanya perlu memasukkan angka 1 pada setiap bits exponent dan mantissa. Lalu didapatkan

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

kemudian konversi bilangan binari tersebut ke bilangan desimal.

a) Untuk angka didepan titik (Exponent) :

$$= \left(1 \times 2^{6}\right) + \left(1 \times 2^{5}\right) + \left(1 \times 2^{4}\right) + \left(1 \times 2^{3}\right) + \left(1 \times 2^{2}\right) + \left(1 \times 2^{1}\right) + \left(1 \times 2^{0}\right)$$

$$= 64 + 32 + 16 + 8 + 4 + 2 + 1$$

$$= 127$$

b) Untuk angka dibelakang titik (Mantissa) :

$$= (1 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) + (1 \times 2^{-4}) + (1 \times 2^{-5}) + (1 \times 2^{-6}) + (1 \times 2^{-7}) + (1 \times 2^{-8})$$

$$= \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256}$$

$$= \frac{255}{256}$$

$$= 0,99609357$$

Sehingga, banyaknya bilangan floating point yang dapat direpresentasikan system didepan titik adalah sebanyak 127 dan dibelakang titik adalah sebanyak $0,99609357$.

❖ Menggunakan Python

```
In [3]: #d. Banyaknya bilangan floating point yang dapat direpresentasikan system ini.

        #dibelakang titik
        sum_ = -1
        for i in range(9):
            sum_ += (1/(2**i))

        'Max exponent part dibelakang titik = ', sum_

Out[3]: ('Max exponent part dibelakang titik = ', 0.99609375)

In [4]: #didepan titik
        sum_a = 0
        for i in range(7):
            sum_a += (2**i)

        'Max exponent part didepan titik = ', sum_a

Out[4]: ('Max exponent part didepan titik = ', 127)
```