Question 1

In [10]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

# Function for user to enter two matrices

row1=int(input("Enter no. of rows"))
column1=int(input("Enter no. of columns"))
# Input the matrix A
A = [[float(input("Enter element A["+str(j)+"]["+str(i)+'] = ')) for i in range(column1)]
for j in range(row1)]
print(A)
print_matrix(A,row1,column1)
# Saving the data to CSV file
np.savetxt("matrixA.csv", A, delimiter = ",")


row2=int(input("Enter no. of rows"))
column2=int(input("Enter no. of columns"))
# Input the matrix B
B = [[float(input("Enter element B["+str(j)+"]["+str(i)+'] = ')) for i in range(column2)]
for j in range(row2)]
print_matrix(B,row2,column2)
# Saving the data to CSV file
np.savetxt("matrixB.csv", B, delimiter = ",")
```

```
[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]

The matrix is:
1.0    2.0    3.0

4.0    5.0    6.0



The matrix is:
1.0    2.0

3.0    4.0

5.0    6.0
```

In [11]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

# initializing array examples and saving it in CSV file

A = [[1,2,4],[0,1,2],[4,-2,3]]
print_matrix(A,3,3)
np.savetxt("matrixA.csv", A, delimiter = ",")

B = [[3,0,1],[-1,1,2],[0,2,1]]
print_matrix(B,3,3)
np.savetxt("matrixB.csv", B, delimiter = ",")

# C and D are column vectors

C = [[1],[4],[3]]
print_matrix(C,3,1)
np.savetxt("matrixC.csv", C, delimiter = ",")

D = [[2],[0],[-1]]
print_matrix(D,3,1)
np.savetxt("matrixD.csv", D, delimiter = ",")
```

```
The matrix is:
1    2    4

0    1    2

4    -2    3



The matrix is:
3    0    1

-1    1    2

0    2    1



The matrix is:
1

4

3



The matrix is:
2

0

-1
```

In [12]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

# Reading data from the CSV files
A=np.genfromtxt('matrixA.csv',delimiter=',')
B=np.genfromtxt('matrixB.csv',delimiter=',')
C=np.genfromtxt('matrixC.csv',delimiter=',')
D=np.genfromtxt('matrixD.csv',delimiter=',')

# matrix C and D become one dimensional arrays which need to be
# converted back to 2d for matrix multiplication
# So we check their dimension and reshape the matrices
# This could be done for A and B as well, but it is not required for the case
if C.ndim==1:
    C=np.reshape(C,(len(C),1))
if D.ndim==1:
    D=np.reshape(D,(len(D),1))

row1=len(A)
col1=len(A[0])
row2=len(B)
col2=len(B[0])
row3=len(C)
col3=len(C[0])
row4=len(D)
col4=len(D[0])

# matrix multiplication
# transposing C wherever required for compatibility

AB,R1,C1=matrix_multiply(A, row1, col1, B, row2, col2)
print_matrix(AB, R1, C1)
np.savetxt("matrixAB.csv", AB, delimiter = ",")

Ct,rt,ct=transpose_matrix(C, row3, col3)
CD,R2,C2=matrix_multiply(Ct, rt, ct, D, row4, col4)
print_matrix(CD, R2, C2)
np.savetxt("matrixCD.csv", CD, delimiter = ",")

Ct,rt,ct=transpose_matrix(C, row3, col3)
CA,R3,C3=matrix_multiply(Ct, rt, ct, A, row1, col1)
print_matrix(CA, R3, C3)
np.savetxt("matrixCA.csv", CA, delimiter = ",")

BD,R4,C4=matrix_multiply(B, row2, col2, D, row4, col4)
print_matrix(BD, R4, C4)
np.savetxt("matrixBD.csv", BD, delimiter = ",")
```

```
The matrix is:
1.0    10.0    9.0

-1.0    5.0    4.0

14.0    4.0    3.0



The matrix is:
2.0



The matrix is:
13.0    0.0    21.0



The matrix is:
6.0

-2.0

0.0
```

Question 2

In [13]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

a=3
b=2
print("The complex number is "+str(a)+" + ("+str(b)+" i)")
c=1
d=-2
print("The complex number is "+str(c)+" + ("+str(d)+" i)")

mc=myComplex(a,b,c,d) # calling the class

p,q=mc.sum_complex(a,b,c,d)
print("Sum of "+str(a)+" + ("+str(b)+" i) and "+str(c)+
      " + ("+str(d)+" i) is = "+str(p)+" + ("+str(q)+" i)")

e,f=mc.difference_complex(a,b,c,d)
print("Difference of "+str(a)+" + ("+str(b)+" i) and "+str(c)+
      " + ("+str(d)+" i) is = "+str(e)+" + ("+str(f)+" i)")

m,n=mc.product_complex(a,b,c,d)
print("Product of "+str(a)+" + ("+str(b)+" i) and "+str(c)+
      " + ("+str(d)+" i) is = "+str(m)+" + ("+str(n)+" i)")

w,x=mc.conjugate_complex(a,b)
print("Conjugate of "+str(a)+" + ("+str(b)+" i) is = "+str(w)+" + ("+str(x)+" i)")

y=mc.modulus_complex(a,b)
print("Modulus of "+str(a)+" + ("+str(b)+" i) is = "+str(y))

r,s=mc.divide_complex(a,b,c,d)
if r!=None and s!=None:
    print("Division of "+str(a)+" + ("+str(b)+" i) by "+str(c)+
          " + ("+str(d)+" i) gives = "+str(r)+" + ("+str(s)+" i)")

y=mc.phase_complex(a,b)
print("Phase angle of "+str(a)+" + ("+str(b)+"i) is = "+str(y)+" degrees")
```

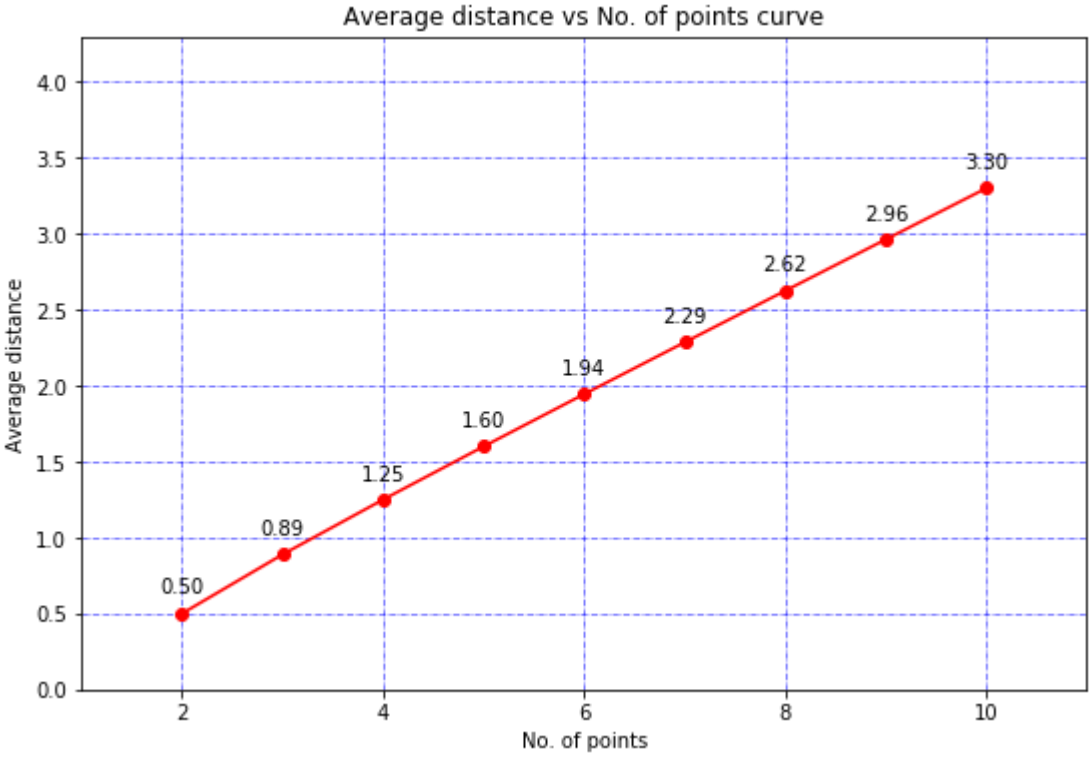
```
The complex number is 3 + (2 i)
The complex number is 1 + (-2 i)
Sum of 3 + (2 i) and 1 + (-2 i) is = 4 + (0 i)
Difference of 3 + (2 i) and 1 + (-2 i) is = 2 + (4 i)
Product of 3 + (2 i) and 1 + (-2 i) is = 7 + (-4 i)
Conjugate of 3 + (2 i) is = 3 + (-2 i)
Modulus of 3 + (2 i) is = 3.605551275463989
Division of 3 + (2 i) by 1 + (-2 i) gives = 1.3999999999999997 + (-0.79999999
99999998 i)
Phase angle of 3 + (2i) is = 33.690067525979785 degrees
```

Question 3

In [14]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

import matplotlib.pyplot as plt
plt.figure(figsize=(9,6))

n=10 # number of observations plotted
index=[]
avg_dist=[]
for k in range(2,n+1):
    index.append(k)
    avg_dist.append(calculate_avg_dist(k)) # function to calculate average distance

print("Number of points          Average distance") # prints number of points vs avg dista
nce
for i in range(len(index)):
    print("      "+str(index[i])+"             -     "+str(avg_dist[i]))

# to label data points
for x,y in zip(index,avg_dist):
    label = "{:.2f}".format(y)
    plt.annotate(label, # this is the text
                 (x,y), # these are the coordinates to position the label
                 textcoords="offset points", # how to position the text
                 xytext=(0,10), # distance from text to points (x,y)
                 ha='center') # horizontal alignment can be left, right or center

# plots number of points vs avg distance
plt.plot(index, avg_dist,'r-o')
plt.grid(color='b', ls = '-.', lw = 0.5)
plt.xlabel('No. of points')
plt.ylabel('Average distance')
plt.title('Average distance vs No. of points curve')
plt.xlim(1,n+1)
plt.ylim(0,avg_dist[-1]+1)
plt.show()
```

```
Number of points          Average distance
      2             -     0.5
      3             -     0.8888888888888888
      4             -     1.25
      5             -     1.6
      6             -     1.9444444444444444
      7             -     2.2857142857142856
      8             -     2.625
      9             -     2.962962962962963
      10             -     3.3
```



Question 4

In [15]:

```
%run CPL_Library.ipynb # Running the entire library in one line because
                       # importing does not work with JupyterLab

print("Please enter guess in small letters, the module is case sensitive.\n")
country, capital=generate_word()
print("Hint: capital of the country "+country)
print("\n\n")
word=capital
word2=list(word) # converting into list makes it easier to handle

chance=[]
for i in range(len(word2)): # initializing
    chance.append('_')
print(chance)

ch_left=math.ceil(len(word2)*0.4)
flag=0 # variable to declare win

i=0
while ch_left>0:
    count=0 # variable to decide number of chances left
    print("\nChances left : "+str(ch_left))
    print()
    guess=input("Enter your guess : ")

    for j in range(len(word2)):
        if chance[j]=='_': # loop runs only for blank spaces left
            if word2[j]==guess:
                chance[j]=guess
                count=1
                flag+=1
            else:
                chance[j]="_"
    print(chance)
    if count==0:
        ch_left-=1

    # checking the losing condition first so that the value of
    # ch_left is not altered by the winning condition ch_left=0
    if ch_left==0: # losing condition
        print("\nYou LOST the game.")
        print("_________") # Hangman picture
        print("|        |")
        print("|       _O_")
        print("|        |")
        print("|       /\\")
        print("|_____________")
        print("|_____________|")

    if flag==len(chance): # winning condition
        print("\nCongratulations, you WON the game.")
        print("\U0001f600 \U0001f600 \U0001f600 \U0001f600 \U0001f600") # happy emoji
        ch_left=0
    i+=1
```

```
Please enter guess in small letters, the module is case sensitive.

Hint: capital of the country Germany



['_', '_', '_', '_', '_', '_']

Chances left : 3

['b', '_', '_', '_', '_', '_']

Chances left : 3

['b', 'e', '_', '_', '_', '_']

Chances left : 3

['b', 'e', 'r', '_', '_', '_']

Chances left : 3

['b', 'e', 'r', 'l', '_', '_']

Chances left : 3

['b', 'e', 'r', 'l', 'i', '_']

Chances left : 3

['b', 'e', 'r', 'l', 'i', 'n']

Congratulations, you WON the game.
😀 😀 😀 😀 😀
```

In [ ]: