CALCUL NUMERIC – LABORATOR #2

3. GRAFICE DE FUNCŢII. MODULUL MATPLOTLIB

3.1. Construcția graficelor folosind instanțele plt.figure() împreună cu plt.axes().

Ex. #1 *Să se construiască în Python graficul funcţiei* $y = sin(x)$, *pe intervalul* $[-\pi, \pi]$ *şi să se formateze. Pentru rezolvarea acestui exerciţiu se va folosi sintaxa: fig = plt.figure(), ax = plt.axes()*

```
import numpy as np
import matplotlib.pyplot as plt

# Se creează instanţa fig care are rolul de a pune la dispoziţie un spaţiu pentru zona grafică.

fig = plt.figure(1)

# Se creează zona grafică împreună cu un sistem de axe în plan.

ax = plt.axes()

# Se discretizează intervalul [-pi,pi] cu 100 de elemente.

x = np.linspace(-np.pi,np.pi,100)

# Se calculează valorile ordonatelor punctelor situate pe graficul funcţiei

y = np.sin(x)

# np.sin() este o funcţie vectorizată, adică aceasta acţionează element cu element.

# Se construieşte graficul funcţiei y = sin(x) în baza setului de date (x,y).

ax.plot(x,y,linestyle = ':', lw = 3, color = 'r',label = 'y = sin(x)')

# Parametrii utilizaţi în formatarea liniei sunt:
# linestyle - tipul liniei;
# lw sau linewidth - grosimea liniei; color - culoarea liniei
# label - text care va fi afisat în legenda graficului folosind metoda legend.

# Optiuni pentru tipul liniei:
# '-' = linie continuă;
# '--' = linie întreruptă;
# ':' = linie punctată

# Opţiuni pentru culoarea liniei:
# 'b' = albastru; 'g' = verde; 'r' = rosu;  'c' = azuriu
# 'm' = purpuriu; 'y' = galben; 'k' = negru; 'w' = alb.


```

```
    # Afişarea grilei sistemului de axe.

    ax.grid(True,color = 'b')

    # Alegerea unei spaţieri proprii pentru grilă, specificandu-se capetele
    intervalului şi distanţa dintre marcaje pe cele două axe.

    plt.xticks(np.arange(-4, 4, 1))
    plt.yticks(np.arange(-1.5,1.6,0.5))

    # Afişează legenda în poziţia cea mai favorabilă.

    ax.legend(loc='best')

    # Alte opţiuni de localizare a legendei: 'upper left', 'upper right',
    # 'lower left', 'lower right' ,'upper center', 'lower center', 'center left',
    # 'center right' , 'center', 'best'

    # Etichetarea axei x

    plt.xlabel('Axa x')

    # Etichetarea axei y

    plt.ylabel('Axa y')

    # Afişarea titlului figurii

    plt.title('Graficul funcţiei y = sin(x) pe intervalul [-pi,pi]')

    # Trasarea axei Ox

    ax.axhline(y=0, color='k')

    # Trasarea axei Oy
    ax.axvline(x=0, color='k')

    plt.show()


```

3.2. Construcția graficelor folosind sintaxa plt.subplots()

Ex. #2 Să se îndeplinească următoarele sarcini:

1. Fiind date funcțiile $y = sin(x), x \in [0, 10]$ şi $y = cos(3x), x \in [0, 5]$ să se construiască graficele funcțiilor în aceeași figura, dar în celule diferite.
2. Să se construiască grafic cercul centrat în origine şi de rază 2 dat de ecuațiile parametrice: $x = 2cos(t); y = 2sin(t), t \in [0, 2\pi]$. Graficul va fi construit în aceeași figura dar în celulă diferită. Figura va avea forma unei coloane cu trei celule, fiecare celulă având sistemul propriu de coordonate. Se va folosi funcția plt.subplots(). Se vor formata graficele.

```
    # Construim instanţele fig şi ax în baza funcţiei subplots(). Instanţa ax în
    cazul nostru va fi un vector (coloană) cu trei elemente, axcesul la elemente f
    ăcându-se cu un singur indice. Fiecare element din ax va construi un sistem de
     axe în celula corespunzătoare.

```

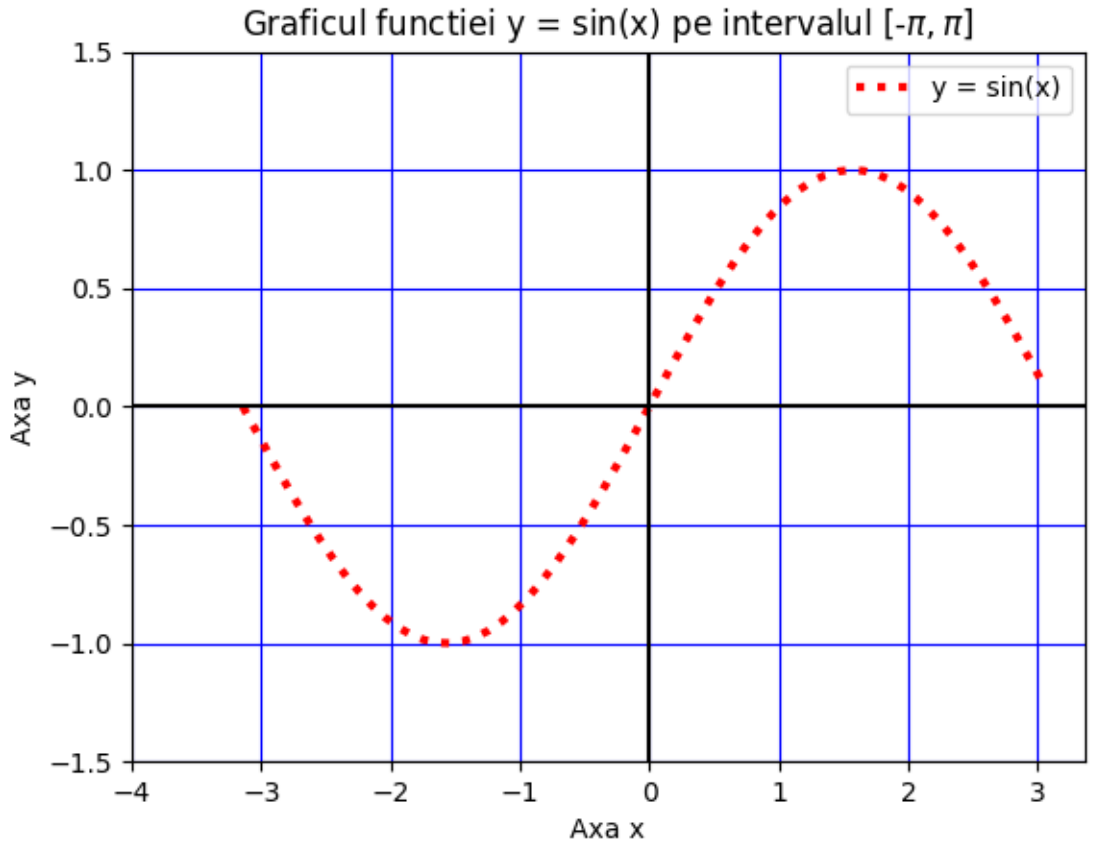

Figure 1:

```
    fig, ax = plt.subplots(3,1)
    # Discretizarea intervalului [0,10] şi [0,5] cu 100 de noduri
    x1 = np.linspace(0,10,100)
    x2 = np.linspace(0,5,100)

    #Se calculează cotele în baza celor două funcţii
    y1 = np.sin(x1)
    y2 = np.cos(3*x2)

    #Se construiesc cele două grafice în prima şi respectiv a doua celulă.
    ax[0].plot(x1, y1,linestyle = ':', lw = 3, color = 'r',label = 'y = sin(x)')
   #pentru inserarea unei legende se va trece parametrul label = şir de caractere
    în lista parametrilor
    ax[0].legend(loc = 'best') #Plasează legenda în cea mai bună poziţie
    ax[0].set_xlim([0,10]) #Setează limitele pentru axa x
    ax[0].set_ylim([-1,1]) #Setează limitele pentru axa y
    ax[0].grid()


    ax[1].plot(x2, y2,linestyle = '-', lw = 3, color = 'b',label = 'y = cos(3x)')
    ax[1].legend(loc = 'best')
    ax[1].set_xlim([0,5])
    ax[1].set_ylim([-1,1])
    ax[1].grid()

```

```
26
27      # Discretizăm intervalul de variaţie al parametrului t
28      t = np.linspace(0,2 * np.pi,100)
29      xc = 2 * np.cos(t)
30      yc = 2 * np.sin(t)
31
32      ax[2].plot(xc, yc,linestyle = '-', lw = 3, color = 'g',label = 'x =  2 cos(t)
        , y = 2sin(t)')
33      ax[2].legend(loc = 'best')
34      ax[2].set_xlim([-2,2])
35      ax[2].set_ylim([-2,2])
36      ax[2].axis('equal')
37      ax[2].grid()
38
```

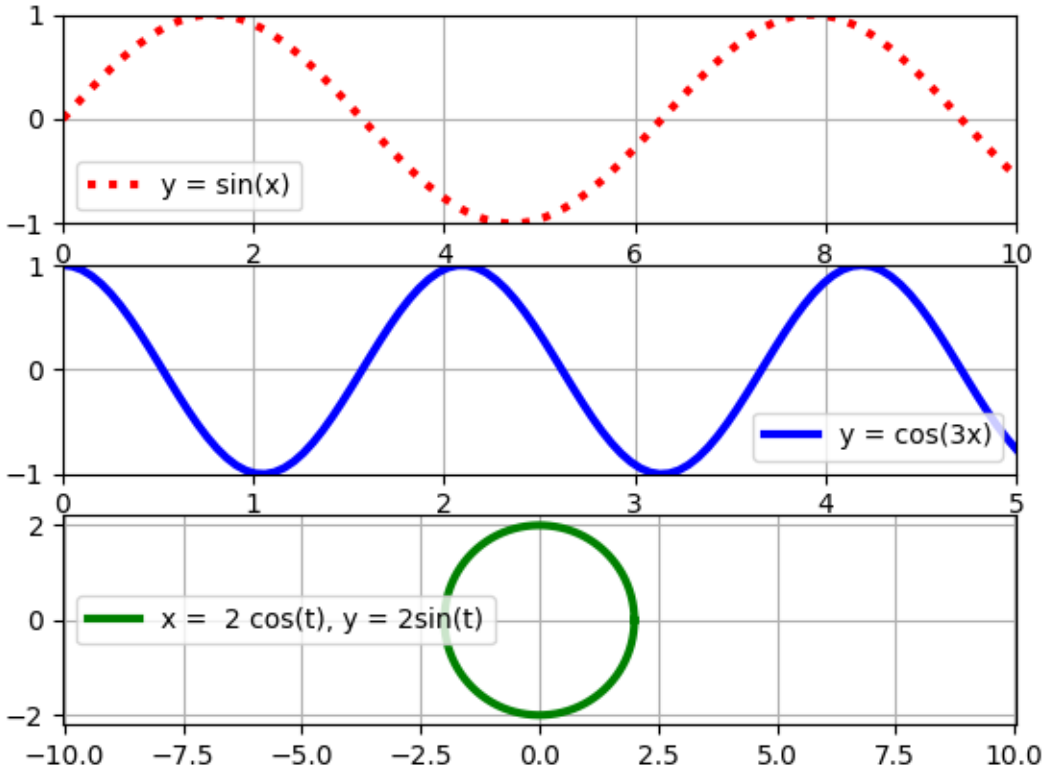

Figure 2:

4. ELEMENTE DE PROGRAMARE

4.1. Construcția *if*

if conditie :

blocul de instrucțiuni 1

blocul de instrucțiuni 2

execută *blocul de instrucțiuni 1* (care trebuie indentat) urmat de *blocul de instrucțiuni 2* dacă *conditie* întoarce *True*. Dacă *conditie* întoarce *False*, se execută doar *blocul de instrucțiuni*

2 și restul programului. În cazul în care *blocul de instrucțiuni 2* lipsește se continuă restul programului. Construcția if poate fi urmată de una sau mai multe construcții $elif$ (prescurtarea de la "$elseif$")

$elif$ conditie :
  blocul de instrucțiuni 1

care se comportă în aceeași manieră ca și construcția if. Clauza $else$

$else$:
  blocul de instrucțiuni 1

se folosește pentru a executa *blocul de instrucțiuni 1* în cazul în care nici una din clauzele $if-elif$ nu sunt adevărate.

Forma generală a construcției $if - elif - else$

if conditie 1 :
  blocul de instrucțiuni 1
$elif$ conditie 2 :
  blocul de instrucțiuni 2
$elif$ conditie 3 :
  pass
$else$:
  blocul de instrucțiuni 3
blocul de instrucțiuni 4

În cazul în care nu se dorește a se executa un anumit bloc de intrucțiuni acesta se va înlocui cu comanda *pass*.

4.2. Construcția for

for contor in element iterabil :
  blocul de instrucțiuni 1

Variabila *contor* ia valori rând pe rând din *elementul iterabil*. *elementul iterabil* poate fi orice mărime ordonată, cum ar fi o listă, un tuplu, un dicționar, un șir de numere generat de funcția *range*, etc.

4.3. Instrucțiunea *continue*

for contor in element iterabil :
  blocul de instructiuni 1
  if conditie 1 :
    continue
  blocul de instructiuni 2
blocul de instructiuni 3

La fiecare iterație a buclei *for* se execută *blocul de instrucțiuni 1*. Dacă *conditie* 1 este *True* se trece la următoarea iterație. În cazul în care *conditie* 1 devine *False* se execută *blocul de instrucțiuni 2* . După terminarea buclei *for* se execută *blocul de instrucțiuni 3*.

Următoarea secvență va selecta toate numerele de la 1 la 99 divizibile la 7.

Exemplul #1

```
x = [] #Creează o listă goală
for i in range(1,100):
  if i%7 != 0: # Testează dacă restul împărţirii la 7 este diferit de zero
    continue # Dacă nu este divizibil cu 7 se trece la următoarea iteraţie fără a
    se executa restul corpului buclei for
  x.append(i) # Adaugă numărul divizibil cu 7 la lista
print(x)
```

4.4. Instrucțiunea *break* apelată în bucla *for*

for contor *in* element iterabil :
  blocul de instrucțiuni 1
  if conditie 1 :
    break
  blocul de instrucțiuni 2
else :
  blocul de instrucțiuni 3
blocul de instrucțiuni 4

Clauza *else* poate fi folosită imediat după bucla *for* (la același nivel) dacă bucla *for* este însoțită de instrucțiunea *break*. *blocul de instrucțiuni 3* este executat dacă și numai dacă *conditie 1* nu a luat valoare *True* la nici o iterație a buclei *for* astfel că nu s-a ajuns la execuția instrucțiunii *break*. Dacă *conditie 1* returnează *True* se iese din bucla *for* și se trece la execuția *blocului de instrucțiuni 4*.

Exemplul #2 Următoarea secvență de instrucțiuni verifică dacă un număr este prim sau nu.

```
y=37
for x in range(2,y):
  if y%x == 0:
    print( y, " nu este prim")
    break
else:
  print( y, " este prim.")
```

4.5. Construcția *while*

while conditie:
  blocul de instrucțiuni 1
blocul de instructiuni 2

execută *blocul de instrucțiuni 1* dacă *conditie* returnează *True*. După ce corpul de instrucțiuni a fost executat *conditie* este verificată din nou. Procesul continuă până când *conditie* devine *False*. Se continuă cu execuția *blocului de intrucțiuni 2*.

Exemplul #3 Mai jos avem un exemplu care va rula la infinit dacă nu se intervine din exterior. Execuția se oprește prin combinația de taste $ctrl + C$.

```
while True :
  print("Apasa Ctrl - C pentru a opri bucla infinita" )
```

Obs.: Ca și în cazul buclei *for*, rămân valabile clauzele *else*, *continue* și *break* având aceeași sintaxă.

5. FUNCȚII

O funcție Python poate fi definită oriunde în program înainte de a fi efectiv folosită. Sintaxa unei funcții este:

def nume(argumente) :
  corpul functiei
  return variabile

Exemplul #4 Următoarea secvență calculează suma a două numere și returnează suma lor.

```
def suma(a,b):
  c = a**2 + b
  return c
print(suma(1,2))
```

Dacă apelul funcției se face prin valorile parametrilor, atunci rezultatul depinde de ordinea scrierii acestora. Dacă însă funcția este apelată atât cu numele parametrilor cat și cu valorile acestora, atunci ordinea numai contează, programul știe să le atribuie corect.

```
print(suma(b = 3, a = 4))
```

În cazul în care vrem să atribuim unui parametru o valoare implicită, în lista de parametri ai funcției vom atribui parametrului valoarea corespunzatoare, folosind operatorul de atribuire. La apel parametrul cu valoare imlicită poate fi omis.

```
def aria_cerc(R, pi = 3.14):
  aria = pi * R**2
  print(f'Aria cercului de raza {R} este:',aria)
aria_cerc(3)
```

Dacă dorim să redefinim parametrul *pi* cu o valoare mai exactă, la apelul funcției în lista cu argumente putem defini noua valoare.

```
aria_cerc(3,pi = 3.14159)# În acest caz Python ignoră valoarea veche a lui pi
```

În cazul în care nu stim apriori câte argumente vor fi transmise funcției, putem adauga $*$ înainte de parametru. Astfel, funcția va primi un tuplu de argumente și va avea acces la valorile acestuia.

```
def suma_n(* a):
  n = len(a)
  s = 0
  for i in range(6):
    s = s + a[i]
  print(s)
  print(a)
suma_n(1,2,3,4,5,6)
```

Dacă se dorește a se folosi la un loc parametrii obligatorii și cu cei arbitrari, atunci aceștia din urmă se scriu la final.

```
def Parametri_oblig_arbit(a,*b):
  print(b) #Tuplu de argumente
  s = 0
  for i in range(len(b)):
    s = s + b[i]
  s = s + a
  print(s)
Parametri_oblig_arbit(7,1,2,3)  #Primul argument corespunde parametrului a, iar
    restul valorilor sunt atribuite.

```

Primul argument corespunde parametrului a, iar restul valorilor sunt atribuite tuplului de argumente b.

Dacă expresia unei funcții este simplă o vom defini folosind sintaxa *lambda*.

func nume = *lambda* param1, param2,... : expresia

```
c = lambda x,y : x**2 + y**2
print(c(3,4))
```

Ex. #3 Fie ecuaţia $x^3 - 7x^2 + 14x - 6 = 0$

a. Să se construiască în Python procedura cu sintaxa **Bisectie**(f, a, b, ε) (ε fiind un parmetru impus cu valoarea implicit 10^{-3}), care implementează algoritmul metodei bisecţiei. Procedura **Bisectie** va returna, atât soluţia aproximativă x_{aprox} cât şi numărul de iteraţii N necesar pentru obţinerea soluţiei aproximative cu eroarea ε.

b. Să se construiască în Python graficul funcţiei $f(x) = x^3 - 7x^2 + 14x - 6$ pe intervalul $[0, 4]$. Să se calculeze toate rădăcinile de pe intervalul dat şi să se construiască rădăcinile pe grafic. Se va folosi criteriul de oprire $\dfrac{|x_k - x_{k-1}|}{|x_k|} < \varepsilon$.

Ex.4 Fie ecuaţia $x^3 - 7x^2 + 14x - 6 = 0$.

a. Să se construiască în Python o procedură **NR**$(f, df, x_0, \varepsilon)$ (ε fiind un parmetru impus cu valoarea implicita 10^{-3}) conform algoritmului metodei Newton-Raphson. Procedura **NR** va returna, atât soluţia aproximativă x_{aprox} cât şi numărul de iteraţii N necesar pentru obţinerea soluţiei aproximative cu eroarea ε

b. Să se construiască graficul funcţiei $f(x) = x^3 - 7x^2 + 14x - 6$ pe intervalul $[0, 4]$. Alegeţi din grafic trei valori iniţiale x_0 corespunzătoare fiecărei rădăcini, astfel încât metoda Newton-Raphson să fie convergentă. Aflaţi cele trei soluţii apelând procedura **NR** cu eroarea de aproximare $\varepsilon = 10^{-3}$. Se va folosi criteriul de oprire $\dfrac{|x_k - x_{k-1}|}{|x_k|} < \varepsilon$.