Beleške sa vežbi

In [ ]:

```
from IPython.display import Image
```

Sve potrebne biblioteke:

In [ ]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import numpy.linalg as lin
import math
```

Funkcija np.arange() proizvodi numpy array (niz/vektor)

In [ ]:

```
x = np.arange(-10,10,0.01)
```

Kada primenimo funkciju kad celim vektorom vraca nam se novi vektor (upotrebljavati kao da radimo nad jednim elementom)

In [ ]:

```
y = np.sin(x) + 1/25*x**2
```

plt.plot() uzima dva obavezna parametra a to su vektori x i y, dole su upotrebljeni svi dodatni parametri koji su bitniji

In [ ]:

```
p1 = plt.plot(x, y, color='green', marker='o', linestyle='dashed', linewidth=2, markersi
ze=4, label = 'Label', markeredgewidth = 3)
plt.show()
```

p1 je linija koju nas grafik vraca

Neka je zadata funkcija:

$$f(\mathbf{x}) = -a * exp(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(cx_i)) + a + exp(1)$$

Kreiranje funkcije se vrsi na sledeci nacin:

In [ ]:

```
def ackley(x):
    sum1 = np.sum(x*x)
    sum2 = np.sum(np.cos(2*math.pi*x))
    total = -20.0*math.exp(-0.2*math.sqrt(sum(x**2))/len(x))-math.exp(sum2/len(x))+20.0+
math.e
    return total
```

Ovo je funkcija vise promenljivih i da bismo nju skicirali moramo napraviti sve kombinacije vrednosti unutar

intervala u kom se nalazimo, npr za interval [-6, 6] moramo imati (-6, 6), (-6, 5), (-6, 4)...(6, 6)

In [ ]:

```
x1v = np.arange(-6, 6, 0.01)
x2v = np.arange(-6, 6, 0.01)
x1, x2 = np.meshgrid(x1v, x2v)
f = np.zeros((len(x1),len(x2)))
```

np.meshgrid(x1v, x2v) nam sluzi da napravi matrice od nasih vektora kako bismo mogli da napravimo kombinacije elemenata, za vektor [1, 2, 3] ona vraca
[[1, 2, 3],
[1, 2, 3],
[1, 2, 3]]
Sa np.zeros(n, m) pravimo nula matricu dimenzija n*m
Petljom racunamo vrednosti svih kombinacija

In [ ]:

```
for i in range(0,len(x1),1):
    for j in range(0,len(x1),1):
        f[i,j]=ackley(np.array([x1[i,j], x2[i,j]]))
```

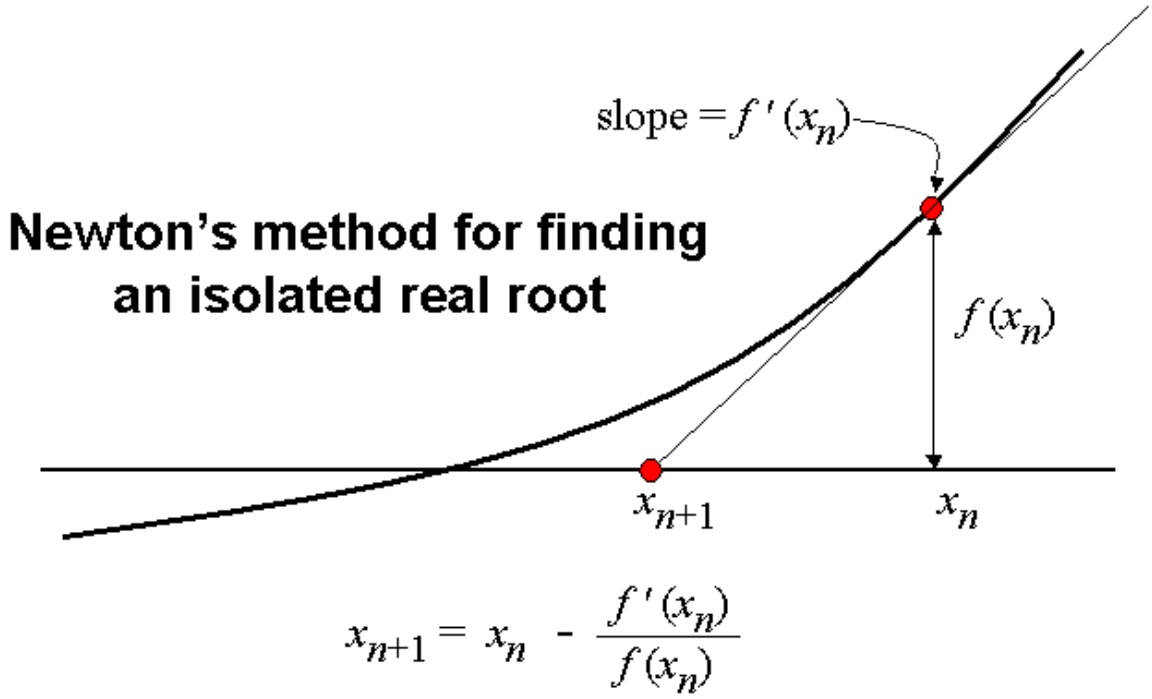
plt.figure() dobavlja figuru koju nas grafik koristi a fig.gca(projection = '3d') dobavlja nase ose u 3d formatu. GCA = get current axes. Na tim osama prikazi nasu funkciju (ax.plot_surface(x1, x2, f)) i konacno prikazi sve sa plt.show()

In [ ]:

```
fig = plt.figure()
ax = fig.gca(projection='3d')
p1 = ax.plot_surface(x1, x2, f)
plt.show()
```

Jednodimenziona numerika

Njutn-Rapsonova metoda:



$$x_{n+1} = x_n - \frac{f'(x_n)}{f(x_n)}$$

In [ ]:

```
def newtonRaphson(x0, tol):
```

```
    x_novo = x0
    x_pre = math.inf # sigurno upadamo u while petlju prvi put
    iteracije = 0

    while (abs(x_pre - x_novo) > tol):
        iteracije += 1
        x_pre = x_novo
        x_novo = x_pre - dfunc(x_pre)/ddfunc(x_pre)

    xopt = x_novo
    fopt = func(xopt)
    return xopt, fopt, iteracije
```

In [ ]:

```
# IZVODI FUNKCIJE
def func(x):
    f = -(x**4 - 5*x**3 - 2*x**2 + 24*x)
    return f

def dfunc(x):
    f = -(4*x**3 - 15*x**2 - 4*x + 24)
    return f

def ddfunc(x):
    f = -(12*x**2 - 30*x - 4)
    return f
```

In [ ]:

```
# TESTIRANJE ALGORITMA
tol = 0.0001
init_guess = 1
[xopt, fopt, iteracije] = newtonRaphson(init_guess, tol)
print(xopt, fopt, iteracije)

x = np.linspace(0, 4, 1000)
f = np.linspace(0, 0, len(x))
for i in range(0, len(x), 1):
    f[i] = func(x[i])

p = plt.plot(x, f, 'b--')
p = plt.plot(xopt, fopt, 'or', label = 'max[f(x)]', markersize = 15, markeredgewidth = 3
)
plt.show()
```

Metoda secice:

In [ ]:

```
def secica(x1, x0, tol):
    x_pre = x0
    x_ppre = math.inf
    x_novo = x1
    iteracije = 0
    # interval nad kojim radim aproksimaciju je [ x_ppre, x_pre] nalazim x_novo i onda po
stavljam novi
    # interval [x_pre, x_novo]
    while(abs(x_novo - x_pre) > tol):
        iteracije += 1
        x_ppre, x_pre = x_pre, x_novo
        x_novo = x_pre - dfunc(x_pre)*(x_pre - x_ppre)/(dfunc(x_pre) - dfunc(x_ppre))

    xopt = x_novo
    fopt = func(xopt)
    return xopt, fopt, iteracije
```

In [ ]:

```
# IZVODI
```

```
def func(x):
    f = -(x**4 - 5*x**3 - 2*x**2 + 24*x)
    return f

def dfunc(x):
    f = -(4*x**3 - 15*x**2 - 4*x + 24)
    return f
```

In [ ]:

```
# TESTIRANJE
tol = 0.0001
init_guess1 = 0
init_guess2 = 3

[xopt, fopt, iteracije] = secica(init_guess1, init_guess2, tol)
print(xopt, fopt, iteracije)

x = np.linspace(0, 4, 1000)
f = np.linspace(0, 0, len(x))
for i in range(0, len(x), 1):
    f[i] = func(x[i])

p = plt.plot(x, f, 'b--')
p = plt.plot(xopt, fopt, 'or', label = 'max[f(x)]', markersize = 15, markeredgewidth = 3
)
plt.show()
```

Metoda parabole:

In [ ]:

```
def parabola(x1, x3, tol):
    X = np.array([x1, (x1 + x3)/2, x3]).transpose()
    pom = np.array([1, 1, 1]).transpose()
    Y = np.array([pom, X, X*X]).transpose()
    # x = [x1 x2 x3]' pom = [1 1 1]'
    # Y = [1 1 1; x1 x2 x3; x1^2 x2^2 x3^2]
    F = np.linspace(0, 0, len(X))
    for i in range(0, len(X), 1):
        F[i] = func(X[i])

    abc = lin.solve(Y, F)

    x = -abc[1]/2/abc[2]
    fx = func(x)
    n = 0

    while np.abs(np.dot([1, x, x**2], abc) - fx) > tol:
        if (x > X[1]) and (x < X[2]):
            if (fx < F[1]) and (fx < F[2]):
                X = np.array([X[1], x, X[2]])
                F = np.array([F[1], fx, F[2]])
            elif (fx > F[1]) and (fx < F[2]):
                X = np.array([X[0], X[1], x])
                F = np.array([F[0], F[1], fx])
            else:
                print('Greska!')
        elif (x > X[0]) and (x < X[2]):
            if (fx < F[0]) and (fx < F[1]):
                X = np.array([X[0], x, X[2]])
                F = np.array([F[0], fx, F[2]])
            elif (fx > F[1]) and (fx < F[0]):
                X = np.array([x, X[1], X[2]])
                F = np.array([fx, F[1], F[2]])
            else:
                print('Greska!')
        else:
            print('x lezi van granica!')
```

```
        pom = np.array([1, 1, 1]).transpose()
        Y = np.array([pom, X, X*X]).transpose()
        F = np.linspace(0, 0, len(X))
        for i in range(0, len(X), 1):
            F[i] = func(X[i])

        abc = lin.solve(Y, F)

        x = -abc[1]/2/abc[2]
        fx = func(x)
        n = n + 1

    return x, fx, n
```

In [ ]:

```
def func(x):
    f = -(x**4 - 5*x**3 - 2*x**2 + 24*x)
    return f
```

In [ ]:

```
# TESTIRANJE
a = 0
b = 2
tol = 0.001

[xopt, fopt, n] = parabola(a, b, tol)
print(xopt, fopt, n)

x = np.linspace(0, 4, 1000)
f = np.linspace(0, 0, len(x))
for i in range(0, len(x), 1):
    f[i] = func(x[i])

p = plt.plot(x, f, 'b--')
p = plt.plot(xopt, fopt, '*r', label = 'max[f(x)]', markersize = 15, markeredgewidth = 3
)
plt.show()
```

Fibonacijev metod

In [ ]:

```
def fibonaci_metod(a, b, tol):
    # korak 1
    n = 1
    while ((b - a) / tol) > fibonaci_broj(n):
        n += 1

    # korak 2
    x1 = a + fibonaci_broj(n - 2) / fibonaci_broj(n) * (b - a)
    x2 = a + b - x1

    # korak 3 - iterativni postupak
    for i in range(2, n + 1):
        if func(x1) < func(x2):
            b = x2
            x2 = x1
            x1 = a + b - x2
        else:
            a = x1
            x1 = x2
            x2 = a + b - x1

    if func(x1) < func(x2):
        xopt = x1
        fopt = func(xopt)
    else:
```

```
        xopt = x2
        fopt = func(x2)

    return xopt, fopt, n
```

In [ ]:

```
def fibonaci_broj(n):
    #1 1 2 3 5 8
    if n < 3:
        f = 1
    else:
        fp = 1
        fpp = 1
        for i in range(3, n+1):
            f = fp + fpp
            fpp = fp
            fp = f
    return f

def func(x):
    f = -(x**4 - 5*x**3 - 2*x**2 + 24*x)
    return f
```

In [ ]:

```
# TESTIRANJE
a = 0
b = 3
tol = 0.0001

[xopt, fopt, n] = fibonaci_metod(a, b, tol)
print(xopt, fopt, n)

x = np.linspace(0, 4, 1000)
f = np.linspace(0, 0, len(x))
for i in range(0, len(x), 1):
    f[i] = func(x[i])

p = plt.plot(x, f, 'b--')
p = plt.plot(xopt, fopt, '*r', label = 'max[f(x)]', markersize = 15, markeredgewidth = 3
)
plt.show()
```

Zlatni presek

In [ ]:

```
def zlatniPresek(a, b, tol):

    c = (3 - math.sqrt(5)) / 2
    # korak 1 - x1 i x2
    x1 = a + c * (b - a)
    x2 = a + b - x1
    n = 1

    # korak 2 - iterativni postupak
    while (b - a) > tol:
        n += 1
        if func(x1) <= func(x2):
            b = x2
            x1 = a + c * (b - a)
            x2 = a + b - x1
        else:
            a = x1
            x1 = a + c * (b - a)
            x2 = a + b - x1

    if func(x1) < func(x2):
        xopt = x1
```

```
        fopt = func(x1)
    else:
        xopt = x2
        fopt = func(x2)

    return xopt, fopt, n
```

In [ ]:

```
def func(x):
    f = -(x**4 - 5*x**3 - 2*x**2 + 24*x)
    return f
```

In [ ]:

```
# TESTIRANJE
a = 0
b = 3
tol = 0.0001

[xopt, fopt, n] = zlatniPresek(a, b, tol)
print(xopt, fopt, n)

x = np.linspace(0, 4, 1000)
f = np.linspace(0, 0, len(x))
for i in range(0, len(x), 1):
    f[i] = func(x[i])

p = plt.plot(x, f, 'b--')
p = plt.plot(xopt, fopt, '*r', label = 'max[f(x)]', markersize = 15, markeredgewidth = 3
)
plt.show()
```

In [ ]:

```

```