Matematický software

Zápočtový dokument

Jméno: Jaroslav Radimský

Kontaktní email: pokejara@seznam.cz

Datum odevzdání: 15. 05.2024

Odkaz na repozitář: https://github.com/jardik14/UJEP/tree/main/MSW/zápočtové%20úlohy

Formální požadavky

Cíl předmětu:
Cílem předmětu je ovládnout vybrané moduly a jejich metody pro jazyk Python, které vám mohou být užitečné jak v dalších semestrech vašeho studia, závěrečné práci (semestrální, bakalářské) nebo technické a výzkumné praxi.

Získání zápočtu:

Pro získání zápočtu je nutné částečně ovládnout více než polovinu z probraných témat. To prokážete vyřešením vybraných úkolů. V tomto dokumentu naleznete celkem 10 zadání, která odpovídají probíraným tématům. Vyberte si 6 zadání, vypracujte je a odevzdejte. Pokud bude všech 6 prací korektně vypracováno, pak získáváte zápočet. Pokud si nejste jisti korektností vypracování konkrétního zadání, pak je doporučeno vypracovat více zadání a budou se započítávat také, pokud budou korektně vypracované.

Korektnost vypracovaného zadání:
Konkrétní zadání je považováno za korektně zpracované, pokud splňuje tato kritéria:

1. Použili jste numerický modul pro vypracování zadání místo obyčejného pythonu
2. Kód neobsahuje syntaktické chyby a je interpretovatelný (spustitelný)
3. Kód je čistý (vygooglete termín clean code) s tím, že je akceptovatelné mít ho rozdělen do Jupyter notebook buněk (s tímhle clean code nepočítá)

Forma odevzdání:
Výsledný produkt odevzdáte ve dvou podobách:

1. Zápočtový dokument
2. Repozitář s kódem

Zápočtový dokument (vyplněný tento dokument, který čtete) bude v PDF formátu. V řešení úloh uveďte důležité fragmenty kódu a grafy/obrázky/textový výpis pro ověření funkčnosti. Stačí tedy uvést jen ty fragmenty kódu, které přispívají k jádru řešení zadání. Kód nahrajte na veřejně přístupný repozitář (github, gitlab) a uveďte v práci na něj odkaz v titulní straně dokumentu. Strukturujte repozitář tak, aby bylo intuitivní se vyznat v souborech (doporučuji každou úlohu dát zvlášť do adresáře).

Podezření na plagiátorství:
Při podezření na plagiátorství (významná podoba myšlenek a kódu, která je za hranicí pravděpodobnosti shody dvou lidí) budete vyzváni k fyzickému dostavení se na zápočet do prostor univerzity, kde dojde k vysvětlení podezřelých partií, nebo vykonání zápočtového testu na místě z matematického softwaru v jazyce Python.

Kontakt:
Při nejasnostech ohledně zadání nebo formě odevzdání se obraťte na vyučujícího.

1. Knihovny a moduly pro matematické výpočty

Zadání:
V tomto kurzu jste se učili s některými vybranými knihovnami. Některé sloužily pro rychlé vektorové operace, jako numpy, některé mají naprogramovány symbolické manipulace, které lze převést na numerické reprezentace (sympy), některé mají v sobě funkce pro numerickou integraci (scipy). Některé slouží i pro rychlé základní operace s čísly (numba).

Vaším úkolem je změřit potřebný čas pro vyřešení nějakého problému (např.: provést skalární součin, vypočítat určitý integrál) pomocí standardního pythonu a pomocí specializované knihovny. Toto měření proveďte alespoň pro 5 různých úloh (ne pouze jiná čísla, ale úplně jiné téma) a minimálně porovnejte rychlost jednoho modulu se standardním pythonem. Ideálně proveďte porovnání ještě s dalším modulem a snažte se, ať je kód ve standardním pythonu napsán efektivně.

Řešení:

```
from functools import wraps
import time
import numpy as np
import pandas as pd


def timer(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        print(f"Elapsed time: {end - start}")
        return result
    return wrapper

# Dot product

@timer
def dot_product_python(a, b):
    return sum(x * y for x, y in zip(a, b))

@timer
def dot_product_numpy(a, b):
    return np.dot(a, b)

@timer
def dot_product_pandas(a, b):
    return pd.Series(a).dot(pd.Series(b))

# Matrix multiplication
```

```python
@timer
def matrix_multiplication_python(a, b):
    return [[sum(x * y for x, y in zip(row, col)) for col in zip(*b)] for row in a]

@timer
def matrix_multiplication_numpy(a, b):
    return np.dot(a, b)

@timer
def matrix_multiplication_pandas(a, b):
    return pd.DataFrame(a).dot(pd.DataFrame(b))

# Scaler addition

@timer
def scaler_addition_python(a, b):
    return [x + b for x in a]

@timer
def scaler_addition_numpy(a, b):
    return a + b

@timer
def scaler_addition_pandas(a, b):
    return pd.Series(a) + b

# Matrix addition

@timer
def matrix_addition_python(a, b):
    return [[x + y for x, y in zip(row_a, row_b)] for row_a, row_b in zip(a, b)]

@timer
def matrix_addition_numpy(a, b):
    return np.add(a, b)

@timer
def matrix_addition_pandas(a, b):
    return pd.DataFrame(a) + pd.DataFrame(b)

# Creating zero matrix

@timer
def zeroing_matrix_python(a):
    return [[0 for _ in row] for row in a]

@timer
def zeroing_matrix_numpy(a):
    return np.zeros_like(a)
```

```
@timer
def zeroing_matrix_pandas(a):
    return pd.DataFrame(a) * 0


def main():
    a = np.random.rand(10000000)
    b = np.random.rand(10000000)
    print("Dot product:")
    print(" Python:")
    dot_product_python(a, b)
    print(" Numpy:")
    dot_product_numpy(a, b)
    print(" Pandas:")
    dot_product_pandas(a, b)

    a = np.random.rand(300, 300)
    b = np.random.rand(300, 300)
    print("\nMatrix multiplication:")
    print(" Python:")
    matrix_multiplication_python(a, b)
    print(" Numpy:")
    matrix_multiplication_numpy(a, b)
    print(" Pandas:")
    matrix_multiplication_pandas(a, b)

    a = np.random.rand(10000000)
    b = 1
    print("\nScaler addition:")
    print(" Python:")
    scaler_addition_python(a, b)
    print(" Numpy:")
    scaler_addition_numpy(a, b)
    print(" Pandas:")
    scaler_addition_pandas(a, b)

    a = np.random.rand(1000, 1000)
    b = np.random.rand(1000, 1000)
    print("\nMatrix addition:")
    print(" Python:")
    matrix_addition_python(a, b)
    print(" Numpy:")
    matrix_addition_numpy(a, b)
    print(" Pandas:")
    matrix_addition_pandas(a, b)

    a = np.random.rand(1000, 1000)
    print("\nZeroing matrix:")
    print(" Python:")
    zeroing_matrix_python(a)
```

```
    print(" Numpy:")
    zeroing_matrix_numpy(a)
    print(" Pandas:")
    zeroing_matrix_pandas(a)


if __name__ == "__main__":
    main()
```

Výsledek:
Dot product:
 Python:
Elapsed time: 1.8051738739013672
 Numpy:
Elapsed time: 0.012998580932617188
 Pandas:
Elapsed time: 0.00794672966003418

Matrix multiplication:
 Python:
Elapsed time: 5.25997257232666
 Numpy:
Elapsed time: 0.004954338073730469
 Pandas:
Elapsed time: 0.002991914749145508

Scaler addition:
 Python:
Elapsed time: 1.2037415504455566
 Numpy:
Elapsed time: 0.029920578002929688
 Pandas:
Elapsed time: 0.021941661834716797

Matrix addition:
 Python:
Elapsed time: 0.15957307815551758
 Numpy:
Elapsed time: 0.002992391586303711
 Pandas:
Elapsed time: 0.002991914749145508

Zeroing matrix:
 Python:
Elapsed time: 0.055819034576416016
 Numpy:
Elapsed time: 0.0029931068420410156
 Pandas:

Elapsed time: 0.0019617080688476562

2. Vizualizace dat

Zadání:
V jednom ze cvičení jste probírali práci s moduly pro vizualizaci dat. Mezi nejznámější moduly patří matplotlib (a jeho nadstavby jako seaborn), pillow, opencv, aj. Vyberte si nějakou zajímavou datovou sadu na webovém portále Kaggle a proveďte datovou analýzu datové sady. Využijte k tomu různé typy grafů a interpretujte je (minimálně alespoň 5 zajímavých grafů). Příklad interpretace: z datové sady pro počasí vyplynulo z liniového grafu, že v létě je vyšší rozptyl mezi minimální a maximální hodnotou teploty. Z jiného grafu vyplývá, že v létě je vyšší průměrná vlhkost vzduchu. Důvodem vyššího rozptylu může být absorpce záření vzduchem, který má v létě vyšší tepelnou kapacitu.

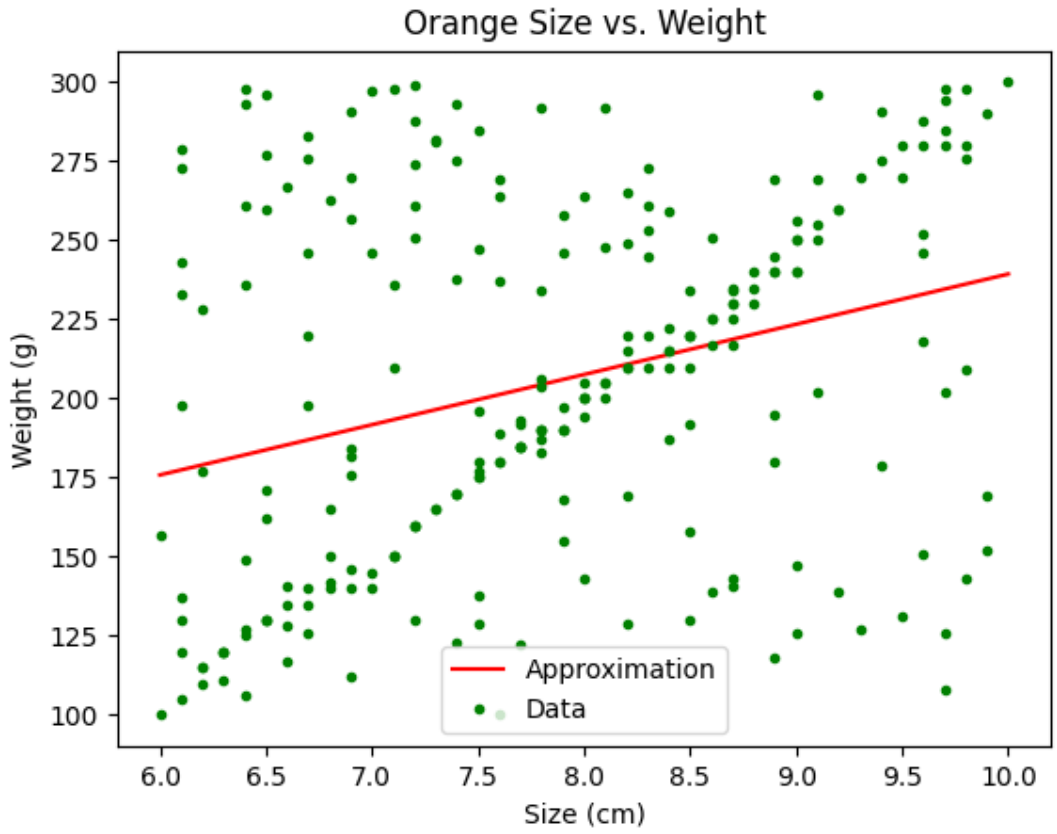
Řešení:

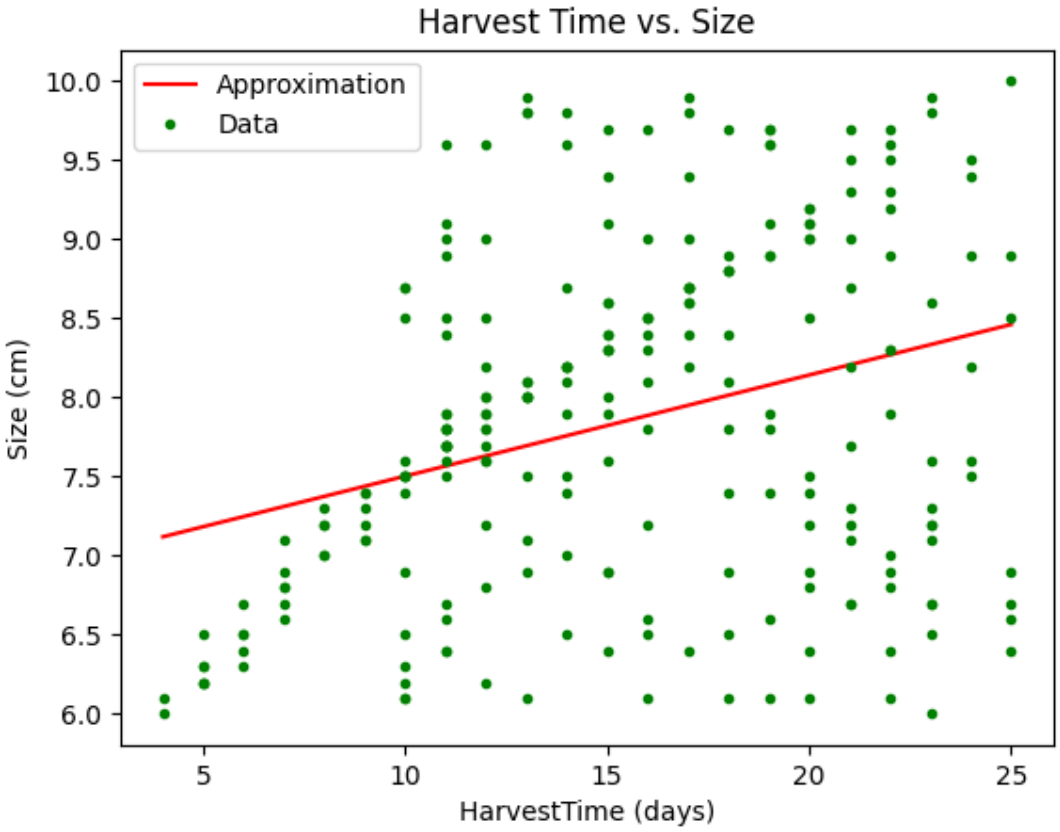
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

def linear_approx(x: list, y: list):
    xnadruhou = []
    for e in x:
        e = e**2
        xnadruhou.append(e)
    xkraty = []
    for e in range(len(x)):
        i = x[e] * y[e]
        xkraty.append(i)

    suma_x = sum(x)
    suma_y = sum(y)
    suma_xnadruhou = sum(xnadruhou)
    suma_xkraty = sum(xkraty)

    n = len(x)

    D = np.array([[suma_x,suma_xnadruhou], [n,suma_x]])

    D_a = np.array([[suma_x,suma_xkraty], [n,suma_y]])
    D_b = np.array([[suma_xkraty,suma_xnadruhou], [suma_y,suma_x]])

    a = np.linalg.det(D_a) / np.linalg.det(D)
    b = np.linalg.det(D_b) / np.linalg.det(D)

    res_x = np.linspace(min(x),max(x),50)
    res_y = [a*xi + b for xi in res_x]

    return res_x, res_y
```

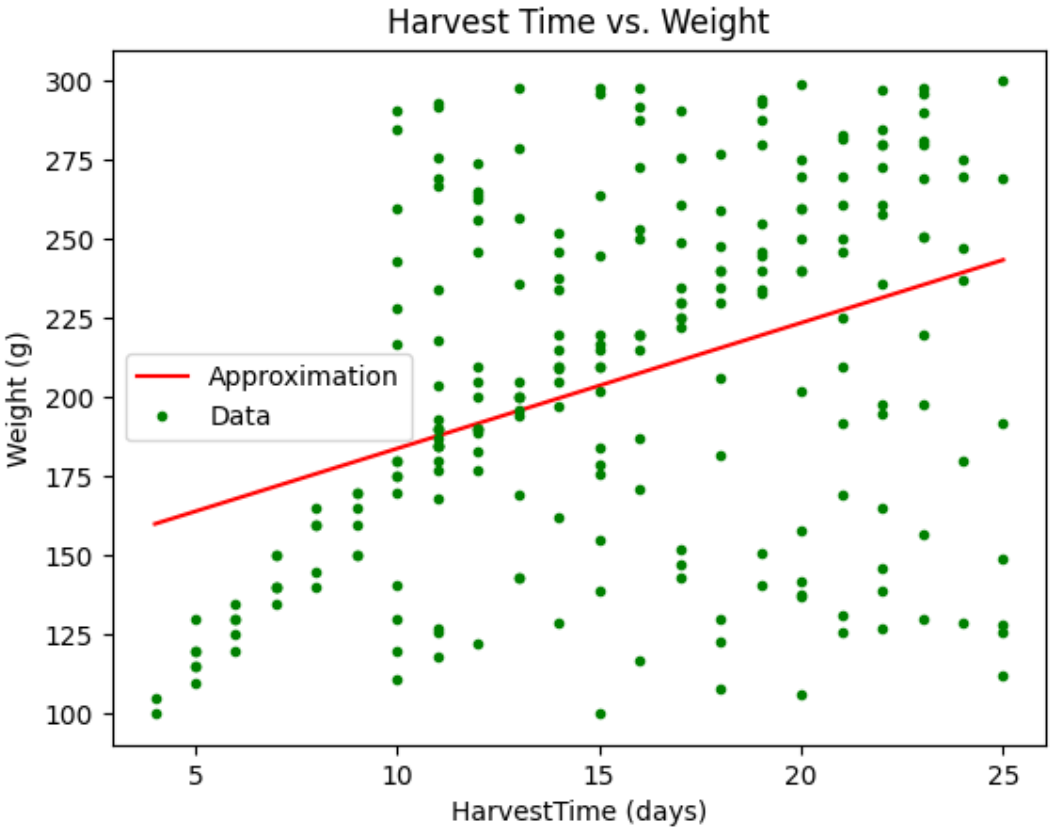
```
data = pd.read_csv("Orange Quality Data.csv")



x,y = linear_approx(data["Size (cm)"], data["Weight (g)"])
plt.plot(x, y, "r-")
plt.plot(data["Size (cm)"], data["Weight (g)"], "g.")
plt.xlabel("Size (cm)")
plt.ylabel("Weight (g)")
plt.legend(["Approximation", "Data"])
plt.title("Orange Size vs. Weight")
plt.show()


x,y = linear_approx(data["HarvestTime (days)"], data["Size (cm)"])
plt.plot(x, y, "r-")
plt.plot(data["HarvestTime (days)"], data["Size (cm)"], "g.")
plt.xlabel("HarvestTime (days)")
plt.ylabel("Size (cm)")
plt.legend(["Approximation", "Data"])
plt.title("Harvest Time vs. Size")
plt.show()



x,y = linear_approx(data["HarvestTime (days)"], data["Weight (g)"])
plt.plot(x, y, "r-")
plt.plot(data["HarvestTime (days)"], data["Weight (g)"], "g.")
plt.xlabel("HarvestTime (days)")
plt.ylabel("Weight (g)")
plt.legend(["Approximation", "Data"])
plt.title("Harvest Time vs. Weight")
plt.show()


x,y = linear_approx(data["Weight (g)"], data["Brix (Sweetness)"])
plt.plot(x, y, "r-")
plt.plot(data["Weight (g)"], data["Brix (Sweetness)"], "g.")
plt.xlabel("Weight (g)")
plt.ylabel("Brix (Sweetness)")
plt.legend(["Approximation", "Data"])
plt.title("Weight vs. Sweetness")
plt.show()

x,y = linear_approx(data["Size (cm)"], data["Quality (1-5)"])
plt.plot(x, y, "r-")
plt.plot(data["Size (cm)"], data["Quality (1-5)"], "g.")
plt.xlabel("Size (cm)")
plt.ylabel("Quality (1-5)")
```

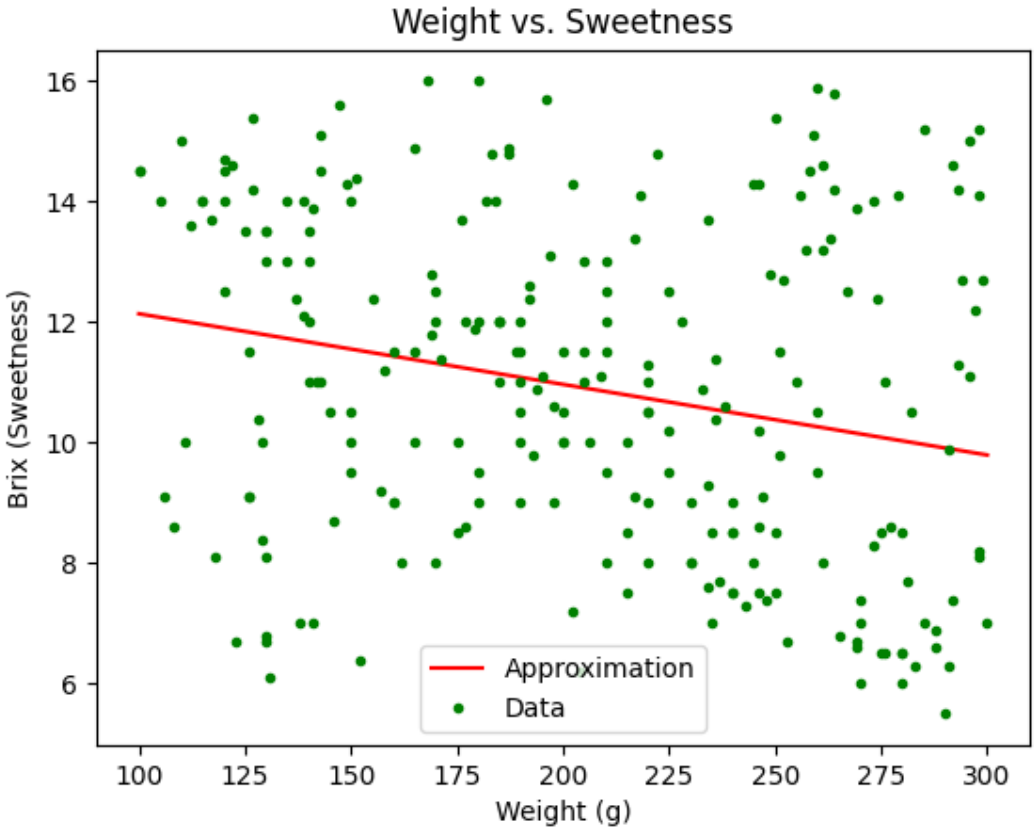
```
plt.legend(["Approximation", "Data"])
plt.title("Size vs. Quality")
plt.show()
```

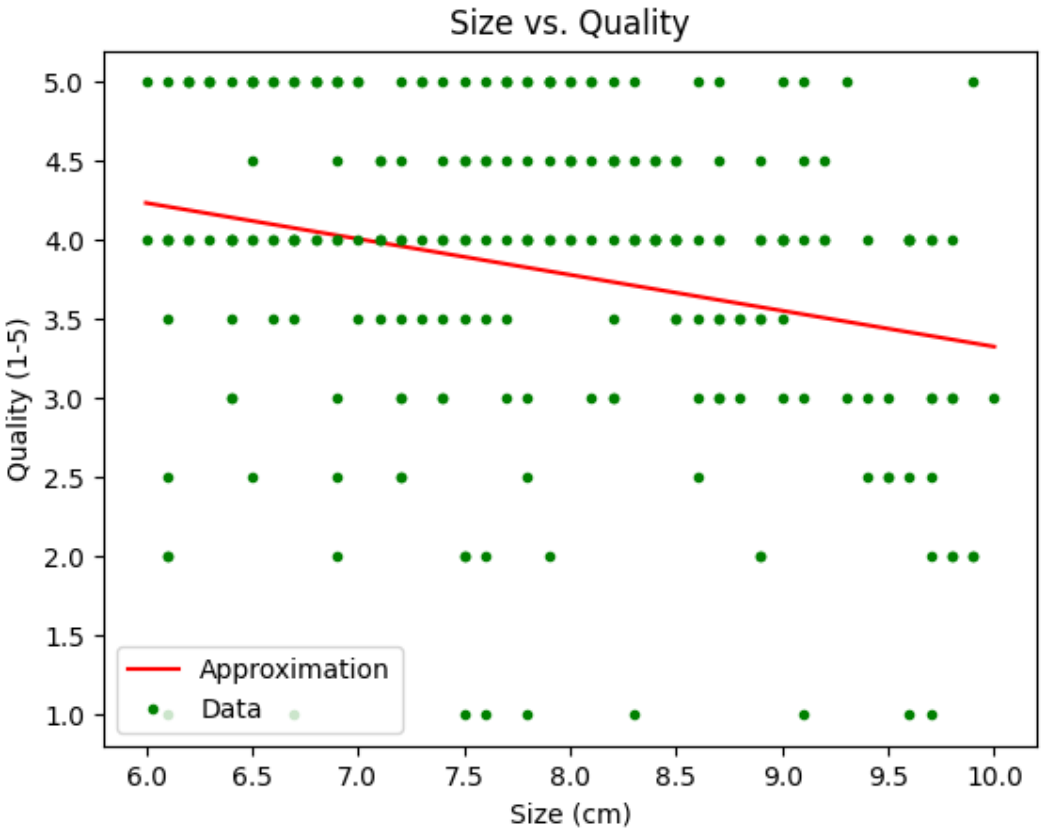
Výsledky:






Harvest Time vs. Weight
Weight (g)
HarvestTime (days)
Approximation
Data
300
275
250
225
200
175
150
125
100
5
10
15
20
25


Weight vs. Sweetness
Brix (Sweetness)
Weight (g)
Approximation
Data
16
14
12
10
8
6
100
125
150
175
200
225
250
275
300


Size vs. Quality
Quality (1-5)
Size (cm)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
Approximation
Data

3. Úvod do lineární algebry

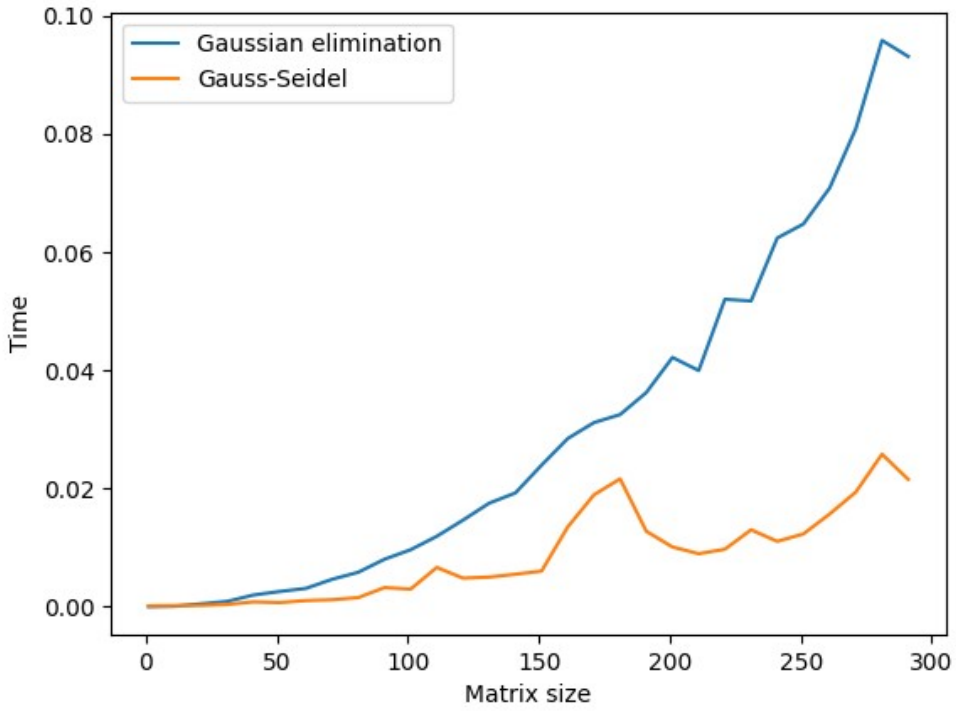
Zadání:
Důležitou částí studia na přírodovědecké fakultě je podobor matematiky zvaný lineární algebra. Poznatky tohoto oboru jsou základem pro oblasti jako zpracování obrazu, strojové učení nebo návrh mechanických soustav s definovanou stabilitou. Základní úlohou v lineární algebře je nalezení neznámých v soustavě lineárních rovnic. Na hodinách jste byli obeznámeni s přímou a iterační metodou pro řešení soustav lineárních rovnic. Vaším úkolem je vytvořit graf, kde na ose x bude velikost čtvercové matice a na ose y průměrný čas potřebný k nalezení uspokojivého řešení. Cílem je nalézt takovou velikost matice, od které je výhodnější využít iterační metodu.

Řešení:

```
@timer
def gaussian_elimination(A, b):
    n = len(A)

    Ab = np.column_stack((A, b)).astype(np.float64)

    for i in range(n):
        if Ab[i, i] == 0.0:
            return None
        for j in range(i + 1, n):
            ratio = Ab[j, i] / Ab[i, i]
            Ab[j] -= ratio * Ab[i]

    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (Ab[i, -1] - np.dot(Ab[i, :-1], x)) / Ab[i, i]

    return x

@timer
def gauss_seidel(A, b, niteraci, x0):
    try:
        x = x0
        U = np.triu(A, k = 1)
        Lstar = np.tril(A, k = 0)
        T = np.matmul(-np.linalg.inv(Lstar), U)
        C = np.matmul(np.linalg.inv(Lstar), b)
        for i in range(niteraci):
            x = np.matmul(T, x) + C
        return x
    except:
        return None
```

```
gauss_times = []
gauss_seidel_times = []

max_size = 300
increment = 10

for i in range(1, max_size, increment):
    gauss_times_partial = []
    gauss_seidel_times_partial = []
    for _ in range(50):
        A = generate_matrix(i)
        b = generate_vector(i)
        gauss_times_partial.append(gaussian_elimination(A, b))
        gauss_seidel_times_partial.append(gauss_seidel(A, b, 20, np.ones(len(A))))
    while None in gauss_times_partial:
        gauss_times_partial.remove(None)
    gauss_times.append(np.mean(gauss_times_partial))
    while None in gauss_seidel_times_partial:
        gauss_seidel_times_partial.remove(None)
    gauss_seidel_times.append(np.mean(gauss_seidel_times_partial))

plt.plot(range(1, max_size, increment), gauss_times, label="Gaussian elimination")
plt.plot(range(1, max_size, increment), gauss_seidel_times, label="Gauss-Seidel")
plt.xlabel("Matrix size")
plt.ylabel("Time")
plt.legend()
plt.show()
```

Výsledek:



4. Interpolace a aproximace funkce jedné proměnné

Zadání:
Během měření v laboratoři získáte diskrétní sadu dat. Často potřebujete data i mezi těmito diskrétními hodnotami a to takové, které by nejpřesněji odpovídaly reálnému naměření. Proto je důležité využít vhodnou interpolační metodu. Cílem tohoto zadání je vybrat si 3 rozdílné funkce (např. polynom, harmonická funkce, logaritmus), přidat do nich šum (trošku je v každém z bodů rozkmitejte), a vyberte náhodně některé body. Poté proveďte interpolaci nebo aproximaci funkce pomocí alespoň 3 rozdílných metod a porovnejte, jak jsou přesné. Přesnost porovnáte s daty, které měly původně vyjít. Vhodnou metrikou pro porovnání přesnosti je součet čtverců (rozptylů), které vzniknou ze směrodatné odchylky mezi odhadnutou hodnotou a skutečnou hodnotou.

Řešení:

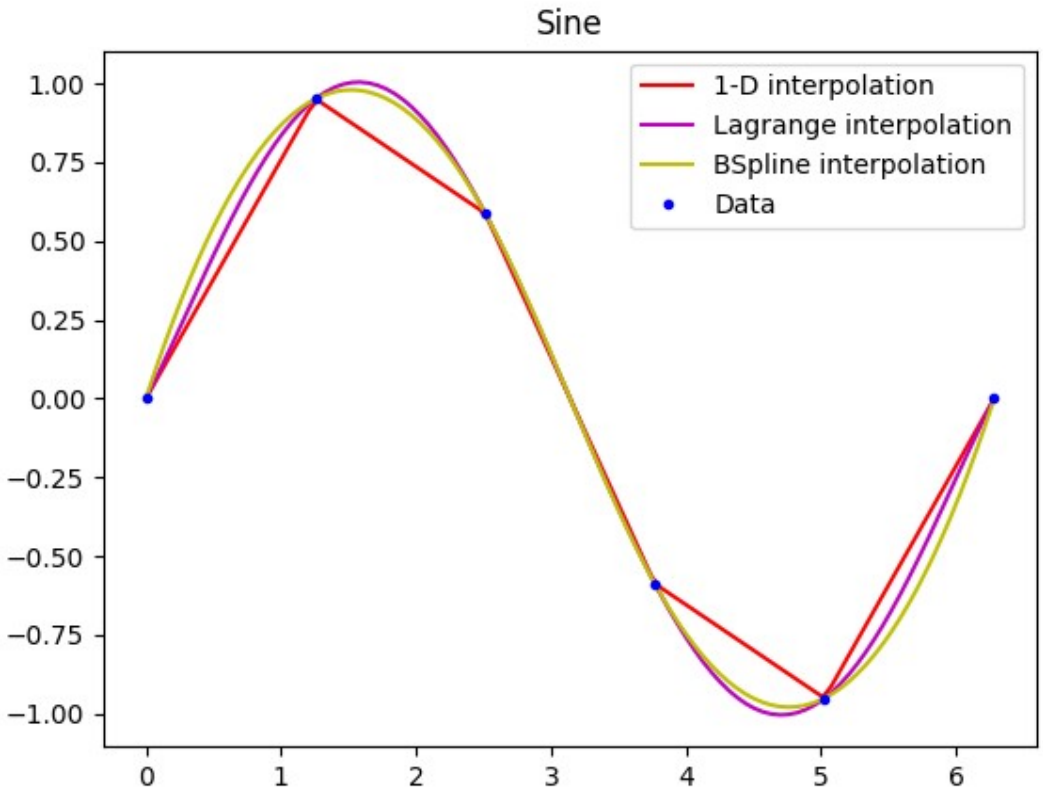
```
# sine function
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

x_interpol = np.linspace(0, 2 * np.pi, 6)
y_interpol = np.sin(x_interpol)
y_noised = y_interpol.copy()

# add noise
for i in range(len(y_interpol)):
    y_noised[i] += random.uniform(-0.1, 0.1)

# 1-D interpolation
y_1d = interpolate.interp1d(x_interpol, y_interpol)
xnew = np.linspace(0, 2 * np.pi, 100)
ynew = y_1d(xnew)

# Lagrange interpolation
y_lagrange = interpolate.lagrange(x_interpol, y_interpol)
xphi = np.linspace(0, 2 * np.pi, 100)
phi = y_lagrange(xphi)

# BSpline interpolation
tck = interpolate.splrep(x_interpol, y_interpol)
x_bspline = np.linspace(0, 2 * np.pi, 100)
y_bspline = interpolate.splev(x_bspline, tck)



plt.plot(xnew, ynew, "r-")
plt.plot(xphi, phi, "m-")
plt.plot(x_bspline, y_bspline, "y-")
plt.plot(x_interpol, y_interpol, "b.")
plt.legend(["1-D interpolation", "Lagrange interpolation", "BSpline interpolation" , "Data"])
plt.title("Sine")
```

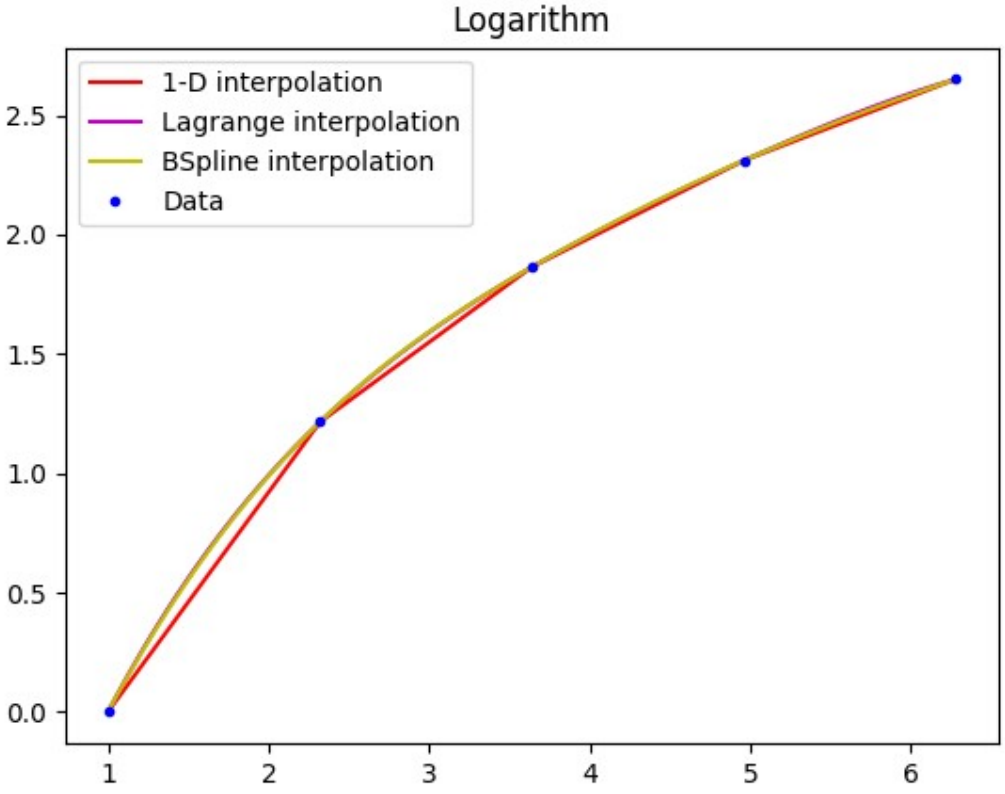
```
plt.show()

# porovnání přesnosti interpolace pomocí součtu čtverců (rozptyylů)
ctverce_1D = 0
for i in range(len(y)):
    ctverce_1D += (y[i] - y_1d(x)[i])**2

ctverce_lagrange = 0
for i in range(len(y)):
    ctverce_lagrange += (y[i] - y_lagrange(x)[i])**2

ctverce_bspline = 0
for i in range(len(y)):
    ctverce_bspline += (y[i] - y_bspline[i])**2

print("1-D interpolation: ", ctverce_1D)
print("Lagrange interpolation: ", ctverce_lagrange)
print("BSpline interpolation: ", ctverce_bspline)

# logarithm function
x = np.linspace(1, 2 * np.pi, 100)
y = np.log2(x)

x_interpol = np.linspace(1, 2 * np.pi, 5)
y_interpol = np.log2(x_interpol)
y_noised = y_interpol.copy()

# add noise
for i in range(len(y_noised)):
    y_noised[i] += random.uniform(-0.1, 0.1)

# 1-D interpolation
y_1d = interpolate.interp1d(x_interpol, y_interpol)
xnew = np.linspace(1, 2 * np.pi, 100)
ynew = y_1d(xnew)

# Lagrange interpolation
y_lagrange = interpolate.lagrange(x_interpol, y_interpol)
xphi = np.linspace(1, 2 * np.pi, 100)
phi = y_lagrange(xphi)

# BSpline interpolation
tck = interpolate.splrep(x_interpol, y_interpol)
x_bspline = np.linspace(1, 2 * np.pi, 100)
y_bspline = interpolate.splev(x_bspline, tck)

plt.plot(xnew, ynew, "r-")
plt.plot(xphi, phi, "m-")
plt.plot(x_bspline, y_bspline, "y-")
plt.plot(x_interpol, y_interpol, "b.")
```

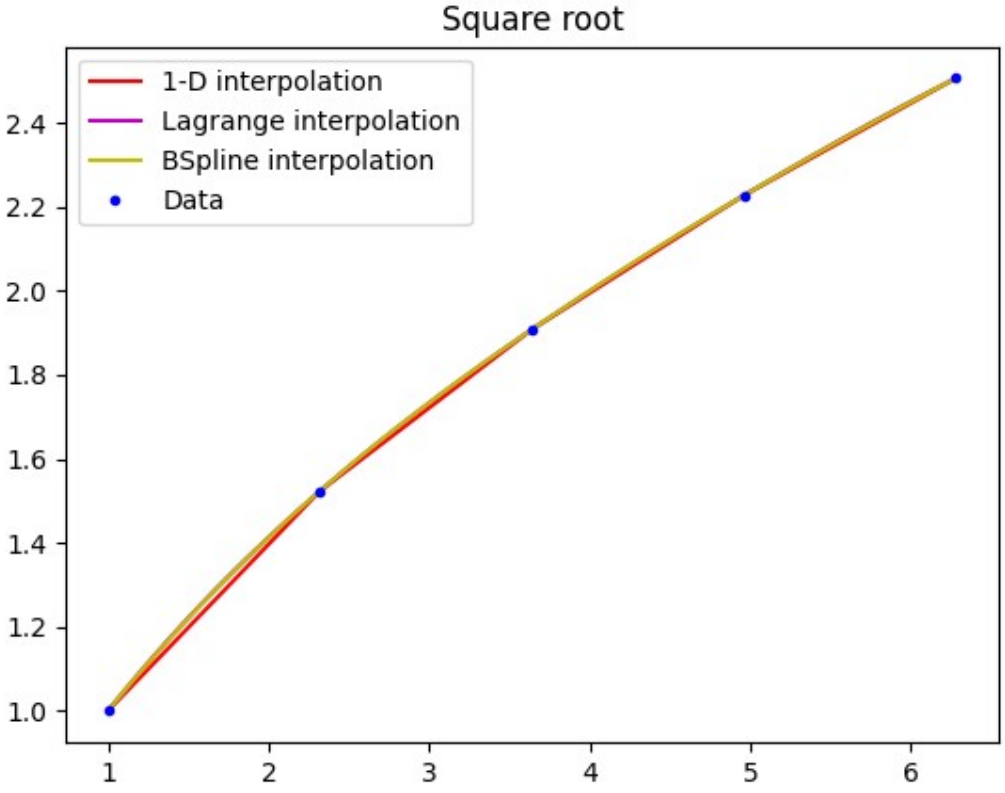
```
plt.legend(["1-D interpolation", "Lagrange interpolation", "BSpline interpolation" , "Data"])
plt.title("Logarithm")
plt.show()

ctverce_1D = 0
for i in range(len(y)):
    ctverce_1D += (y[i] - y_1d(x)[i])**2

ctverce_lagrange = 0
for i in range(len(y)):
    ctverce_lagrange += (y[i] - y_lagrange(x)[i])**2

ctverce_bspline = 0
for i in range(len(y)):
    ctverce_bspline += (y[i] - y_bspline[i])**2

print("1-D interpolation: ", ctverce_1D)
print("Lagrange interpolation: ", ctverce_lagrange)
print("BSpline interpolation: ", ctverce_bspline)

# square root function
x = np.linspace(1, 2 * np.pi, 100)
y = np.sqrt(x)

x_interpol = np.linspace(1, 2 * np.pi, 5)
y_interpol = np.sqrt(x_interpol)
y_noised = y_interpol.copy()

# add noise
for i in range(len(y_noised)):
    y_noised[i] += random.uniform(-0.1, 0.1)

# 1-D interpolation
y_1d = interpolate.interp1d(x_interpol, y_interpol)
xnew = np.linspace(1, 2 * np.pi, 100)
ynew = y_1d(xnew)

# Lagrange interpolation
y_lagrange = interpolate.lagrange(x_interpol, y_interpol)
xphi = np.linspace(1, 2 * np.pi, 100)
phi = y_lagrange(xphi)

# BSpline interpolation
tck = interpolate.splrep(x_interpol, y_interpol)
x_bspline = np.linspace(1, 2 * np.pi, 100)
y_bspline = interpolate.splev(x_bspline, tck)

plt.plot(xnew, ynew, "r-")
plt.plot(xphi, phi, "m-")
plt.plot(x_bspline, y_bspline, "y-")
```

```
plt.plot(x_interpol, y_interpol, "b.")
plt.legend(["1-D interpolation", "Lagrange interpolation", "BSpline interpolation" , "Data"])
plt.title("Square root")
plt.show()

ctverce_1D = 0
for i in range(len(y)):
    ctverce_1D += (y[i] - y_1d(x)[i])**2

ctverce_lagrange = 0
for i in range(len(y)):
    ctverce_lagrange += (y[i] - y_lagrange(x)[i])**2

ctverce_bspline = 0
for i in range(len(y)):
    ctverce_bspline += (y[i] - y_bspline[i])**2

print("1-D interpolation: ", ctverce_1D)
print("Lagrange interpolation: ", ctverce_lagrange)
print("BSpline interpolation: ", ctverce_bspline)
```

Výsledky:




Logarithm
1-D interpolation
Lagrange interpolation
BSpline interpolation
Data
2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6


Square root
1-D interpolation
Lagrange interpolation
BSpline interpolation
Data
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
1
2
3
4
5
6

5. Hledání kořenů rovnice

Zadání:
Vyhledávání hodnot, při kterých dosáhne zkoumaný signál vybrané hodnoty je důležitou součástí analýzy časových řad. Pro tento účel existuje spousta zajímavých metod. Jeden typ metod se nazývá ohraničené (například metoda půlení intervalu), při kterých je zaručeno nalezení kořenu, avšak metody typicky konvergují pomalu. Druhý typ metod se nazývá neohraničené, které konvergují rychle, avšak svojí povahou nemusí nalézt řešení (metody využívající derivace). Vaším úkolem je vybrat tři různorodé funkce (například polynomiální, exponenciální/logaritmickou, harmonickou se směrnicí, aj.), které mají alespoň jeden kořen a nalézt ho jednou uzavřenou a jednou otevřenou metodou. Porovnejte časovou náročnost nalezení kořene a přesnost nalezení.

Řešení:

```
@timer
def bisect(f, a, b):
    return optimize.bisect(f, a, b)

@timer
def newton(f, x0):
    return optimize.newton(f, x0)

# funkce 1
print("Funkce 1: x^2 - 4")
def f1(x):
    return x**2 - 4

a0 = 0
b0 = 4

res = 4**(1/2)

# bisekce
cas, x = bisect(f1, a0, b0)
print(f"Bisekce: {x}, čas: {cas} ms, chyba: {abs(res - x)}")


# newton
cas, x = newton(f1, x0=(a0+b0)/2)
print(f"Newton: {x}, čas: {cas} ms, chyba: {abs(res - x)}")


# funkce 2
print("\nFunkce 2: 1/(5^(2x-4)) - 125")
def f2(x):
    return 1/(5**(2*x-4)) - 125

a0 = 0
```

```
b0 = 4


res = 1/2

# bisekce
cas, x = bisect(f2, a0, b0)
print(f"Bisekce: {x}, čas: {cas} ms, chyba: {abs(res - x)}")


# newton
cas, x = newton(f2, x0=(a0+b0)/2)
print(f"Newton: {x}, čas: {cas} ms, chyba: {abs(res - x)}")


# funkce 3
print("\nFunkce 3: 2 * sin(x) - 0.5")
def f3(x):
    return 2 * np.sin(x) - 0.5

a0 = -2
b0 = 2

res = np.arcsin(0.25)

# bisekce
cas, x = bisect(f3, a0, b0)
print(f"Bisekce: {x}, čas: {cas} ms, chyba: {abs(res - x)}")


# newton
cas, x = newton(f3, x0=(a0+b0)/2)
print(f"Newton: {x}, čas: {cas} ms, chyba: {abs(res - x)}")
```

Funkce 1: x^2 - 4
Bisekce: 2.0, čas: 0.0 ms, chyba: 0.0
Newton: 2.0, čas: 0.0 ms, chyba: 0.0

Funkce 2: 1/(5^(2x-4)) - 125
Bisekce: 0.5, čas: 0.0 ms, chyba: 0.0
Newton: 2.0003, čas: 0.0 ms, chyba: 1.5003000000000002

Funkce 3: 2 * sin(x) - 0.5
Bisekce: 0.2526802551419678, čas: 0.9973049163818359 ms, chyba: 1.1085576900882188e-13
Newton: 0.25268025514207865, čas: 0.0 ms, chyba: 0.0

6. Generování náhodných čísel a testování generátorů

Zadání:
Tento úkol bude poněkud kreativnější charakteru. Vaším úkolem je vytvořit vlastní generátor semínka do pseudonáhodných algoritmů. Jazyk Python umí sbírat přes ovladače hardwarových zařízení různá fyzická a fyzikální data. Můžete i sbírat data z historie prohlížeče, snímání pohybu myší, vyzvání uživatele zadat náhodné úhozy do klávesnice a jiná unikátní data uživatelů.

Řešení:

```
import psutil
import pyautogui
import random

# get CPU cores usage to use as a seed for random number generator
cores_prctg = psutil.cpu_percent(interval=1, percpu=True)

# get mouse position
x,y = pyautogui.position()

# calculate seed
my_seed_x = x
my_seed_y = y
for prctg in cores_prctg:
    my_seed_x += prctg
    my_seed_y -= prctg

my_seed = abs(my_seed_x - my_seed_y)

# set seed for random number generator
random.seed(my_seed)
```

7. Metoda Monte Carlo

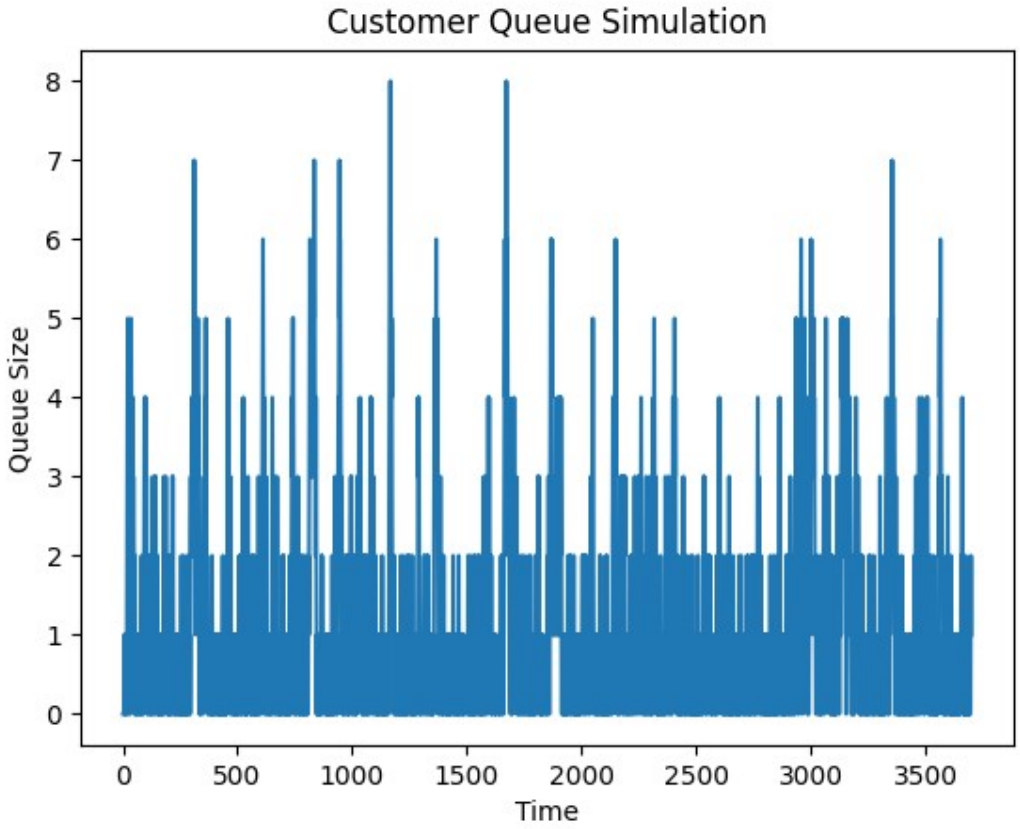
Zadání:
Metoda Monte Carlo představuje rodinu metod a filozofický přístup k modelování jevů, který využívá vzorkování prostoru (například prostor čísel na herní kostce, které mohou padnout) pomocí pseudonáhodného generátoru čísel. Jelikož se jedná spíše o filozofii řešení problému, tak využití je téměř neomezené. Na hodinách jste viděli několik aplikací (optimalizace portfolia aktiv, řešení Monty Hall problému, integrace funkce, aj.). Nalezněte nějaký zajímavý problém, který nebyl na hodině řešen, a získejte o jeho řešení informace pomocí metody Monte Carlo. Můžete využít kódy ze sešitu z hodin, ale kontext úlohy se musí lišit.

Řešení:

```
class CustomerQueueSimulation:
    def __init__(self, arrival_rate, service_rate, num_iterations, max_time):
        self.arrival_rate = arrival_rate
        self.service_rate = service_rate
        self.num_iterations = num_iterations
        self.max_time = max_time

    def simulate(self):
        queue_sizes = []

        for _ in range(self.num_iterations):
            time = 0
            queue_size = 0

            while time < self.max_time:
                inter_arrival_time = random.expovariate(self.arrival_rate)
                service_time = random.expovariate(self.service_rate)

                if inter_arrival_time < service_time:
                    time += inter_arrival_time
                    queue_size += 1
                else:
                    time += service_time
                    queue_size = max(0, queue_size - 1)

                queue_sizes.append(queue_size)


        return queue_sizes

# Parameters
arrival_rate = 0.5  # Mean inter-arrival time (customers arrivals per minute)
service_rate = 1  # Mean service time (services done per minute)
num_iterations = 1 # Number of simulations
max_time = 2500  # Maximum time to simulate (minutes in opening hours)
```

```
# Create a simulation instance
simulation = CustomerQueueSimulation(arrival_rate, service_rate, num_iterations, max_time)

# Run the simulation
queue_sizes = simulation.simulate()

# Plot the queue size over time
plt.plot(queue_sizes)
plt.xlabel('Time')
plt.ylabel('Queue Size')
plt.title('Customer Queue Simulation')
plt.show()
```

Výsledek:



8. Derivace funkce jedné proměnné

Zadání:
Numerická derivace je velice krátké téma. V hodinách jste se dozvěděli o nejvyužívanějších typech numerické derivace (dopředná, zpětná, centrální). Jedno z neřešených témat na hodinách byl problém volby kroku. V praxi je vhodné mít krok dynamicky nastavitelný. Algoritmům tohoto typu se říká derivace s adaptabilním krokem. Cílem tohoto zadání je napsat program, který provede numerickou derivaci s adaptabilním krokem pro vámi vybranou funkci. Proveďte srovnání se statickým krokem a analytickým řešením.

Řešení:
doplňte

9. Integrace funkce jedné proměnné

Zadání:
V oblasti přírodních a sociálních věd je velice důležitým pojmem integrál, který představuje funkci součtů malých změn (počet nakažených covidem za čas, hustota monomerů daného typu při posouvání se v řetízku polymeru, aj.). Integraci lze provádět pro velmi jednoduché funkce prostou Riemannovým součtem, avšak pro složitější funkce je nutné využít pokročilé metody. Vaším úkolem je vybrat si 3 různorodé funkce (polynom, harmonická funkce, logaritmus/exponenciála) a vypočíst určitý integrál na dané funkci od nějakého počátku do nějakého konečného bodu. Porovnejte, jak si každá z metod poradila s vámi vybranou funkcí na základě přesnosti vůči analytickému řešení.

Řešení:
doplňte

10. Řešení obyčejných diferenciálních rovnic

Zadání:
Diferenciální rovnice představují jeden z nejdůležitějších nástrojů každého přírodovědně vzdělaného člověka pro modelování jevů kolem nás. Vaším úkolem je vybrat si nějakou zajímavou soustavu diferenciálních rovnic, která nebyla zmíněna v sešitech z hodin a pomocí vhodné numerické metody je vyřešit. Řešením se rozumí vizualizace jejich průběhu a jiných zajímavých informací, které lze z rovnic odvodit. Proveďte také slovní okomentování toho, co lze z grafu o modelovaném procesu vyčíst.

Řešení:
doplňte