Matematický software

Zápočtový dokument

Jméno: Michal Melicher

Kontaktní email: melicmic@gmail.com

Datum odevzdání: 12.07.2022

Odkaz na repozitář: https://github.com/mmelicher91/kmsw-seme

Úvod do lineární algebry

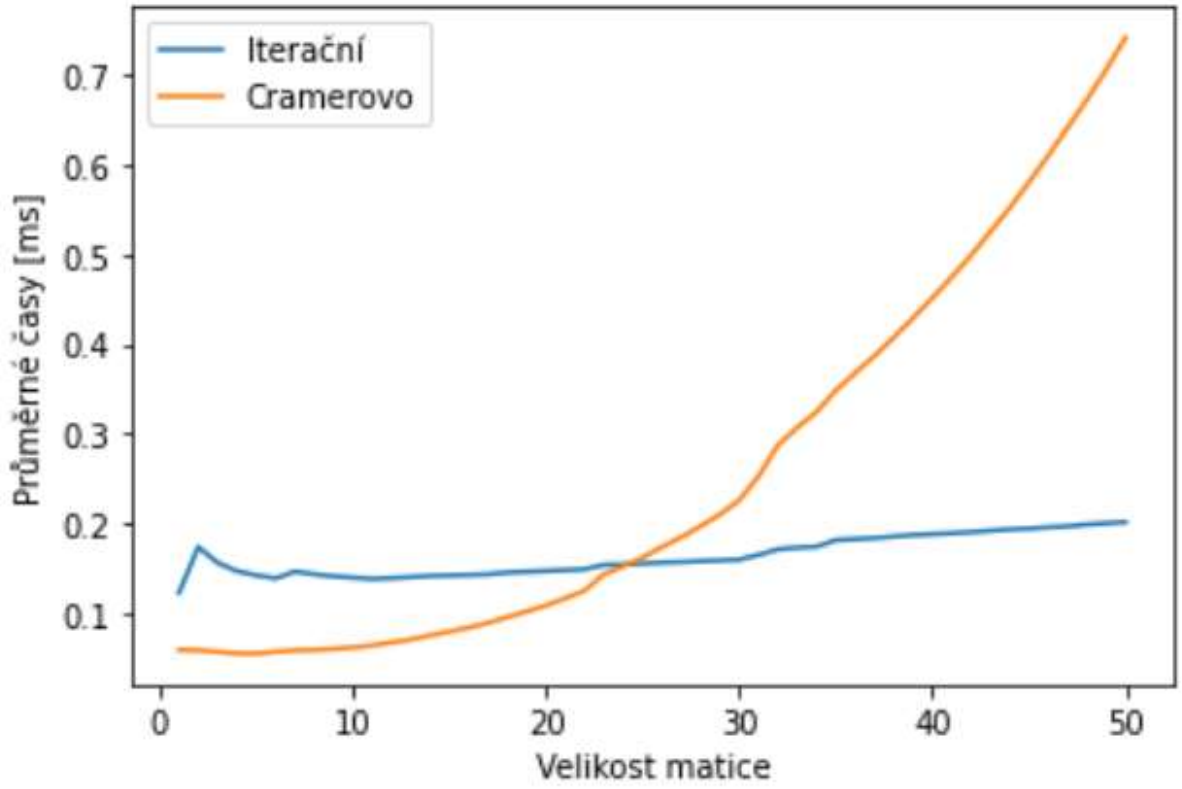
Zadání:

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:

Velikost matice	Iterační řešení [ms]	Cramerovo řešení [ms]
1	0.1219034194946289	0.058269500732421875
2	0.173187255859375	0.05766153335571289
3	0.15558401743570965	0.056099891662597656
4	0.1461803913116455	0.05443692207336426
5	0.1416158676147461	0.054149627685546875
6	0.13815959294637045	0.05613168080647787
7	0.1457691192626953	0.057618958609444756
8	0.1427382230758667	0.05810260772705078
9	0.14043384128146702	0.05924436781141493
10	0.139007568359375	0.06089925765991211
15	0.14146804809570312	0.07863521575927734
20	0.1463913917541504	0.10776400566101074
21	0.14750843956356957	0.11600653330485027
22	0.14841123060746628	0.1243081959811124
23	0.15324302341627036	0.14286974202031674
24	0.1535654067993164	0.1515557368596395
25	0.15475177764892578	0.1613311767578125
30	0.15910545984903973	0.22539536158243814
40	0.18761694431304932	0.4515331983566284
50	0.20107507705688477	0.7421488761901855

Cramerovo řešení je nejrychlejší pro nalezení výsledku matice do velikosti matice 20x20. Na větší matice je vhodné použít iterační metodu.



Iterační řešení

```
def jacobi(A, b, niteraci, x0=None):
    x = x0 if x0 else np.ones(len(A))
    D = np.diag(A)
    R = A - np.diagflat(D)
    for i in range(niteraci):
        x = (b - np.dot(R,x))/D
```

Cramerovo řešení

```
def cramer(A, b):
    N = A.shape[1]
    mask = np.broadcast_to(np.diag([1 for i in range(N)]), [N, N,
N]).swapaxes(0, 1)
    Ai = np.where(mask, np.repeat(b, N).reshape(N, N), A)
    x = np.linalg.det(Ai) / np.linalg.det(A)
    return x
```

Interpolace a aproximace funkce jedné proměnné

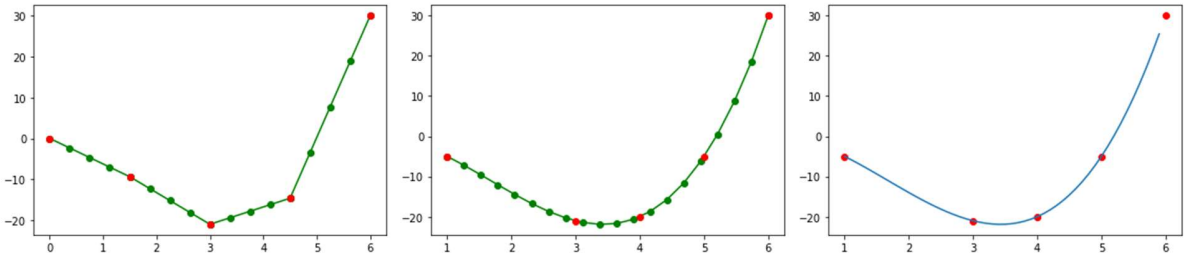
Zadání:

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.

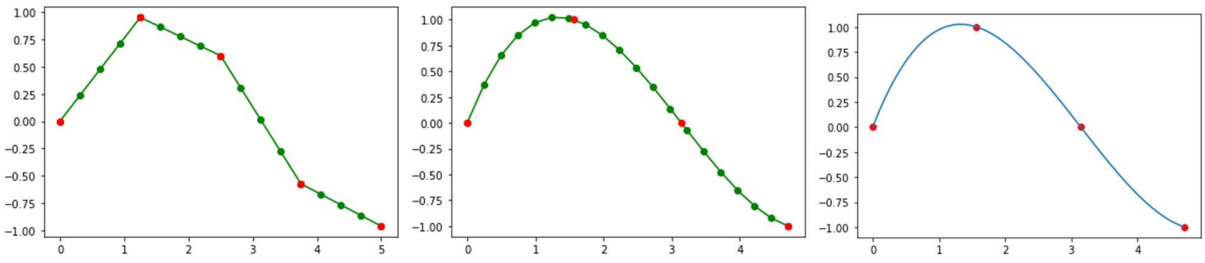
Řešení:

Polynomická funkce: $x^3 - 5 * x^2 - x$

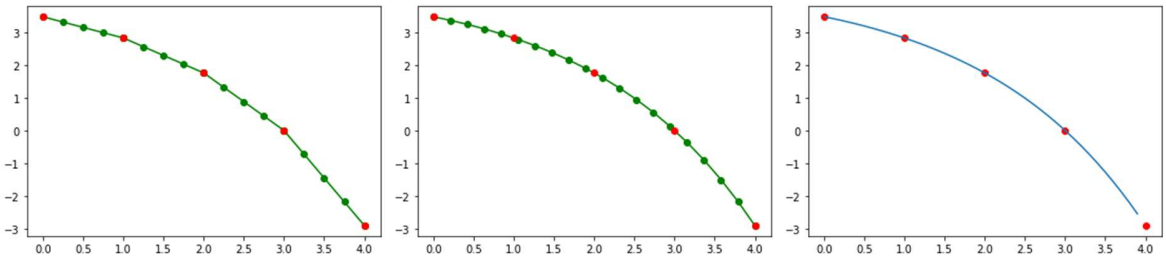
Lineární aproximace | Lagrange aproximace | Newtonova aproximace



Harmonická funkce: $\sin(x)$



Exponenciální funkce: $e^{\frac{3}{2}} - e^{\frac{x}{2}}$



Lineární aproximace

```
def linear(x, a, b):
    return f(a) + (f(b)-f(a))/(b-a)*(x-a)


for i in range(len(x)-1):
    xinterpol = np.linspace(x[i], x[i+1], ninterpol) #body, které mě
zajímají uvnitř interpolační funkce
    g = linear(xinterpol, x[i], x[i+1]) #hodnoty bodů interpolační funkce
    xphi.extend(xinterpol)
    phi.extend(g)
```

Lagrange aproximace

```
from scipy.interpolate import lagrange


phi = lagrange(x, fx)
print(phi)
print(phi.coef)

n = 20
xphi = np.linspace(x[0], x[-1], n)
```

Newtonova aproximace

```
def divided_diff(x, y):
    n = len(y)
    coef = np.zeros([n, n])
    coef[:,0] = y
    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = \
           (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])
    return coef

def newton_poly(coef, x_data, x):
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p
```

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:

- Polynomická funkce: $-\mathbf{10} + (\boldsymbol{x} - \mathbf{3})^{\mathbf{2}}$; $\boldsymbol{x} \in <\mathbf{0}, \mathbf{10}>$

Metoda	Časová náročnost [ms]	Průsečík s osou x
Bisekce pro odchylku	135.98942756652832	6.162277460098267
Regula falsi	0.7677078247070312	6.162271798827216
Newtonova metoda	1.3129711151123047	6.162277660168379

Bisekce pro odchylku potřebuje 23 mezivýpočtů k nalezení řešení rovnice.

Očekáváné řešení $3 + \sqrt{10} = 6{,}162277660168379 \ldots$. Nalezená řešení se liší až na 6 desetinném místě.

- Exponenciální funkce: $\frac{\mathbf{3}}{\mathbf{2}} * \mathbf{2}^{\boldsymbol{x}-\mathbf{2}} - \mathbf{6}$; $\boldsymbol{x} \in <\mathbf{0}, \mathbf{10}>$

Metoda	Časová náročnost [ms]	Průsečík s osou x
Bisekce pro odchylku	307.27696418762207	3.9999985694885254
Regula falsi	5.524873733520508	3.999863541701375
Newtonova metoda	2.550363540649414	4.0

Bisekce pro odchylku potřebuje 20 mezivýpočtů k nalezení řešení rovnice.

Očekáváné řešení 4. Nejpřesnější výsledek je z Newtonovy metody. U zbylých dvou metod je nepřesnost pravděpodobně způsobena ukládáním malých desetinných čísel do paměti. Po zaokrouhlení bychom získali požadovaný výsledek 4.

- Harmonická funkce: $\boldsymbol{sin(2 * x) + 3 * x - 9}$; $\boldsymbol{x} \in \boldsymbol{< 1, 10 >}$

Metoda	Časová náročnost [ms]	Průsečík s osou x
Bisekce pro odchylku	167.56653785705566	3.056471347808838
Regula falsi	2.3560523986816406	3.056472826279747
Newtonova metoda	2.329111099243164	3.0564728394177414

Bisekce pro odchylku potřebuje 22 mezivýpočtů k nalezení řešení rovnice.

Očekáváné řešení 3. Nepřesnost je pravděpodobně způsobena ukládáním malých desetinných čísel do paměti. Po zaokrouhlení bychom nezískali požadovaný výsledek 3, ale 3,1.

Bisekce pro odchylku

```
while b**2 - a**2 > 2*eps:
    x = (a + b)/2
    if f(a)*f(x) < 0:
        b = x
    else:
        a = x
```

Regula falsi

```
xnew = (a+b)/2
xold = a

while abs(xnew - xold) > delta:
    xold = xnew
    xnew = a - f(a)*(b-a)/(f(b)-f(a))
    if f(a)*f(xnew) < 0:
        b = xnew
    else:
        a = xnew
```

Newtonova metora

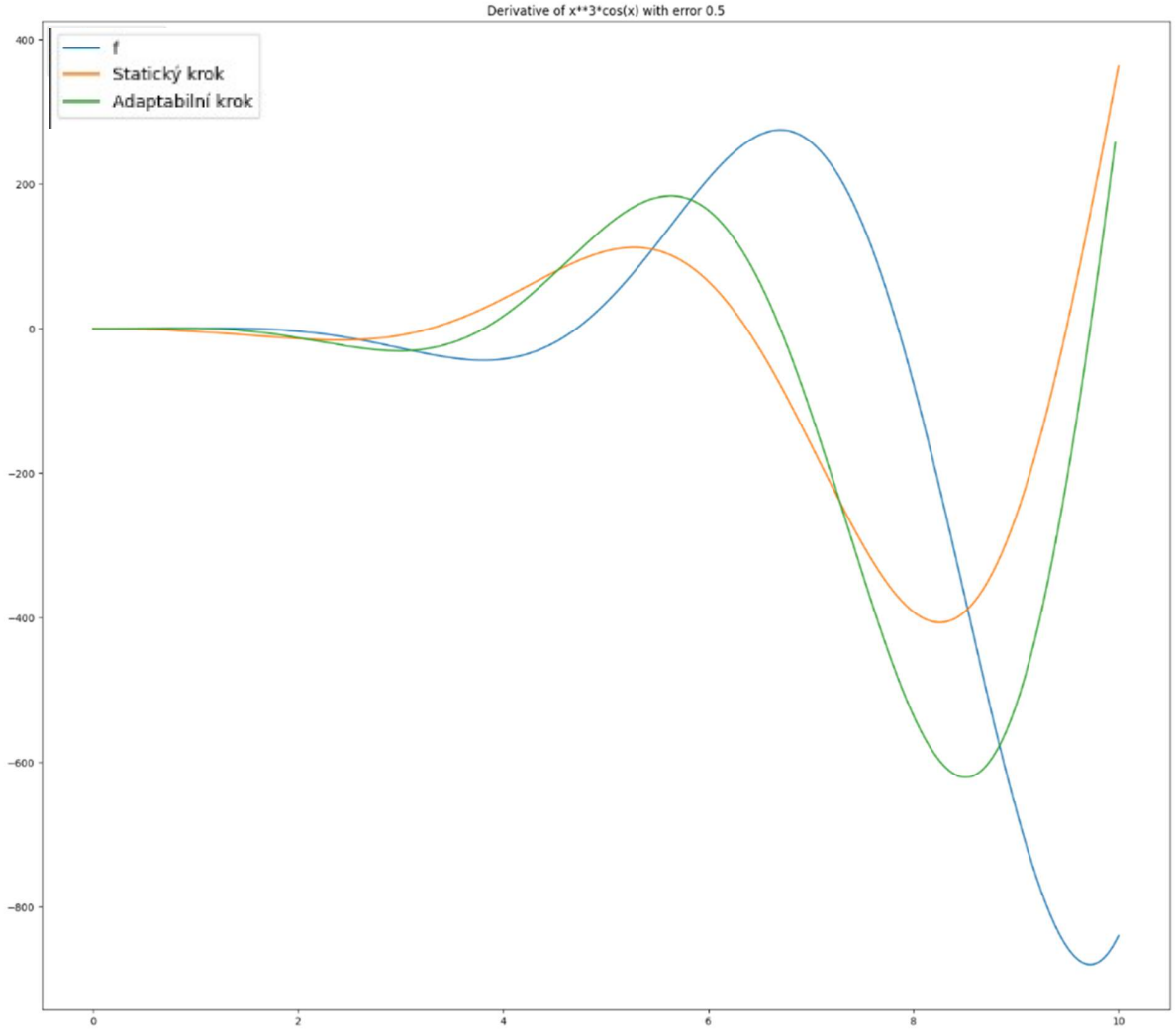
```
def df(x, h = 1E-5):
    return (f(x+h)-f(x-h))/(2*h)

xnew = (a+b)/2
xold = a

while abs(xnew - xold) > delta:
    xold = xnew
    xnew = xold - f(xold)/df(xold)
```

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:



```
def finite_diff(f, x, type="forward", error=0.01, static=True):

    if (type == "backward"):
        g = lambda x, h:    (f(x) - f(x-h))/h
    elif (type == "central"):
        g = lambda x, h:    (f(x+h) - f(x-h))/(2*h)
    elif (type == "forward"):
        g = lambda x, h:    (f(x+h) - f(x))/h

    D = []
    X = []
    x_ = x[0]
    if(not static):
        x_ = x[0]
        while x_ < x[-1]:

            h1 = error
            h2 = error/2

            dx1 = g(x_, h1)
            dx2 = g(x_, h2)

            while(abs(dx1 - dx2)>= error):
                h1 = h2
                h2 = h2/2

                dx1 = g(x_, h1)
                dx2 = g(x_, h2)

            D.append(dx2)
            X.append(x_)
            x_ = x_ + h2
    else:
        h = 2*np.sqrt(error)
        for x_ in x:
            dx = g(x_, h)
            D.append(dx)
            X.append(x_)

    return D, X
```

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:

Výsledky integrací:

Polynomická funkce: $\int_1^4 \frac{(x+1)^2}{x} dx = x + \frac{1}{x}$

Lichoběžníková metoda: 2.2483626815960536

Newton-Cotes vzorec: 2,2438626815960536

Rombergův vzorec: 2.2499999999988605

Analytické řešení: 2,25

Harmonická funkce: $\int_1^4 sin\,(x+1)dx = -cos\,(x+1)$

Lichoběžníková metoda: $-0.744690621947801 + 0.05 * \sin(2) = -0{,}69922575$

Newton-Cotes vzorec: $-0.744690621947801 + 0.05 * \sin(2) = -0{,}69922575$

Rombergův vzorec: $-0.698391520835868 + 0.0285631918311657 * \sin(5) + +0.0285631918311657 * \sin(2) = -0{,}696974019$

Analytické řešení: $-0{,}699809022$

Exponenciální funkce: $\int_1^4 e^2 - e^x dx = (x-1) * e^x$

Lichoběžníková metoda: $0.05 * e + 163.881446274428 = 164{,}0173603$

Newton-Cotes vzorec: $0.05 * e + 163.881446274428 = 164{,}0173603$

Rombergův vzorec: $0.0285631918311657 * e + 0.0285631918311657 * e^4 + 3 * e^2 + 50.2427279663545 = 74{,}04703633$

Analytické řešení: $3 * e^4 = 163.794450099$

Lichoběžníková metoda

```
x = a
i = 0

while x < b:
    integral += dx * (f(x) + f(x+dx))/2
    x += dx
```

Newton-Cotes vzorec

```
n = int((b-a)//dx)+1
integral = f(a) + f(b)

for i in range(1, n):
    integral += 2*f(a+i*dx)
integral *= dx/2
```

Rombergův vzorec

```
def df(x, h = 1E-5):
    return (f(x+h)-f(x-h))/(2*h)

xnew = (a+b)/2
xold = a

while abs(xnew - xold) > delta:
    xold = xnew
    xnew = xold - f(xold)/df(xold)
```