ОСОБЛИВОСТІ ЧИСЕЛЬНИХ РОЗРАХУНКІВ БУДІВЕЛЬНИХ КОНСТРУКЦІЙ ЗА ГРАНИЧНИМИ СТАНАМИ

Дагіль В.Г., Кучер Г.І
dahil_viktoriia@nuczu.edu.ua
м. Черкаси, Національний університет цивільного захисту України

Сучасний розвиток комп'ютерних технологій дозволяє перейти від експериментальних досліджень і аналітичних методів розрахунків будівельних конструкцій [3], до комп'ютерного моделювання та використання чисельних методів розрахунків. Не будемо зупинятися на всіх доступних програмних комплексах, а приділимо увагу засобу вирішення крайових задач при розрахунках будівельних конструкцій за граничними станами. Це мова програмування Python, яку використовують при написанні високоструктурованих програм і веб-застосунків, які інтерпретовані з платформою Microsoft.NET Framework. Python зручний для розв'язання математичних проблем, при рішенні крайових задач має обширну кількість бібліотек, але має недолік щодо швидкості при роботі в машинному навчанні з великою кількістю даних.

Рішення крайової задачі зводиться до наступних етапів:

1. Аналіз та постановка задачі: Визначення диференціального рівняння та крайових (граничних) умов.
2. Створення розрахункової схеми. Розрахункова схема призначена для визначення напружено-деформованого стану. Для реалізації цього етапу розв'язання задачі у процесорі комплексу (або в окремих CAD-системах) будують геометричну модель об'єкта.
3. Створення дискретної моделі: Заміна диференціального рівняння (метод скінченних різниць) або апроксимація функцій (метод скінченних елементів). Застосовуючи чисельний спосіб, складаємо модель, утворену з розрахункової схеми яка має кінцеве число ступенів свободи.
4. Розв'язання системи рівнянь: Обчислення факторів попередньо-напруженого стану у довільних точках кінцево-елементної моделі, на підставі відомих з теорії пружності та пластичності (внутрішніх зусиль, напруг, переміщень довільних точок) з побудовою їх епюр.
5. Аналіз результатів: Перевірка виконання умов та оцінка точності, зазначають матеріали. Оптимізація функцій.
6. Візуалізація та збереження отриманих результатів

Розглянемо саме питання аналізу результатів, їх оптимізацію та візуалізацію. В зв'язку з тим, що крайова задача – це задача визначення розв'язку диференціального рівняння, який задовольняє умови на границі області (крайові умови), то виникає необхідність оптимізації функцій у розрахунках будівельних конструкцій. Тобто пошук найкращих геометричних чи параметричних характеристик будівельних конструкції (мінімальна вага, вартість, максимальна жорсткість, вогнестійкість) за допомогою методів математичного програмування. Оптимізація базується на створені математичної моделі, де функція (поперечна сила, деформація, згинальний момент) залежить від керованих змінних.

Оберемо для тестування оптимізаційних функцій [1] в Python, одну з класичних функцій Хіммельблау [2] і виконаємо порівняння градієнтних методів оптимізації функцій кількох змінних в наступній послідовності:

In [1]: Визначення часткових похідних за допомогою символьних обчислень Sympy

```
import sympy as sp
from IPython.display import display

print('Обрана функція: (x**2 + y - 11)**2 + (x + y**2
- 7)**2', '\n') x, y = sp.symbols('x y')
print('Часткова похідна функції по x:', sp.diff((x**2 + y - 11)**2 + (x + y**2 - 7)**2,
x)) print('Часткова похідна функції по y:', sp.diff((x**2 + y - 11)**2 + (x + y**2 -
7)**2, y))
```

In [2]: Визначення градієнтних спусків

```
import
plotly.graph_objects as
go import
plotly.figure_factory
as ff import numpy as
np
import plotly.io as pio
pio.renderers.default='notebook'
def f(x:float, y:float):
    return (x**2 + y - 11)**2 + (x + y**2 - 7)**2
def df_dx(x:float, y:float):
    return 4*x*(x**2 + y - 11) + 2*x + 2*y**2 - 14
def df_dy(x:float, y:float):
    return 2*x**2 + 4*y*(x + y**2 - 7) + 2*y - 22
```

```
# plot_3d_surface() start
Contour=False
x_plt = np.arange(-4, 4, 0.05)
y_plt = np.arange(-4, 4, 0.05)
Z_plt = np.array([[f(x, y) for x in x_plt] for y in y_plt]) trace1 = go.Contour(x=x_plt, y=y_plt,

z=Z_plt)

#   створення даних для побудування фігури;   оптимальна палітра для конкретного графіку
trace2 = go.Surface(x=x_plt, y=y_plt, z=Z_plt, colorbar_x=-0.5, opacity= 0.83)

data1 = [trace1] data2 = [trace2]

fig1 = go.Figure(data = data1) fig2 = go.Figure(data = data2) # plot_3d_surface() end
```

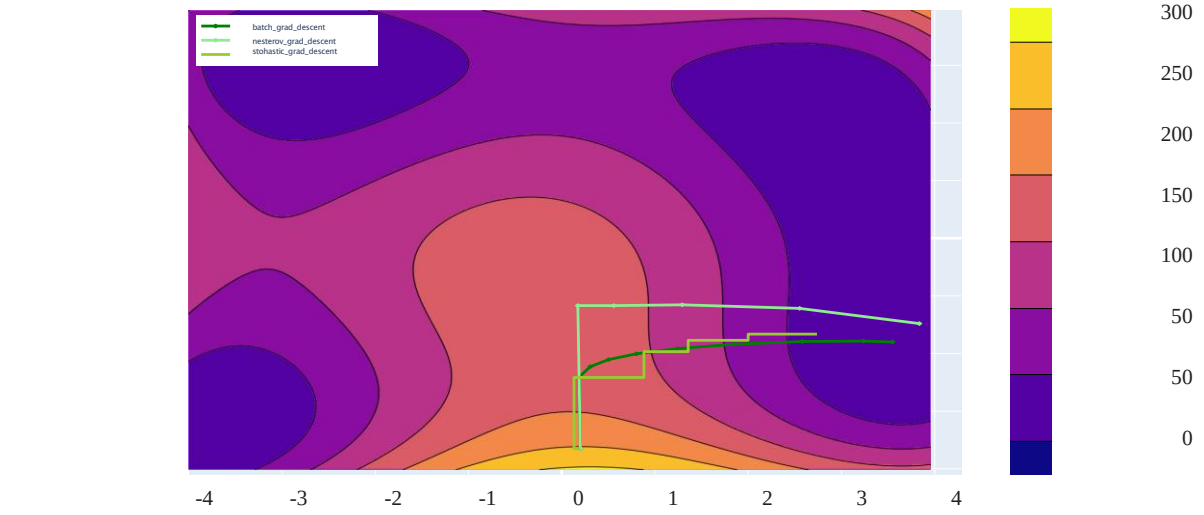
In [3]: Методи оптимізації

```
def batch_grad_descent(x, y, lmd_x, lmd_y, number_of_iterations):
    """Классичний градієнтний спуск з константними заданними кроками та наперед заданною кількістю
    ітерацій""" dots = np.array([x, y], dtype=float).reshape(-1, 2)
    for n in
        range(number_of_it
        erations): x = x -
        lmd_x * df_dx(x,
        y)
        y = y - lmd_y * df_dy(x, y)
        dots = np.append(dots, [[x, y]], axis=0)
    return dots
```

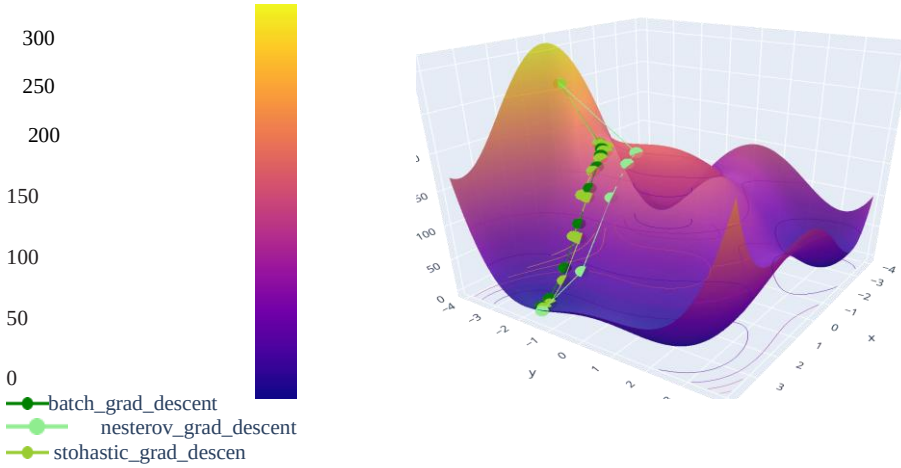
```
def nesterov_grad_descent(x, y, alpha, beta,
    number_of_iterations): """Accelerated gradient
    descent"""
    dots = np.array([x, y], dtype=float).reshape(-1, 2)
    x = x -
    alpha *
    df_dx(x, y)
    y = y -
    alpha *
    df_dy(x, y)
    dots = np.append(dots, [[x, y]], axis=0)
    for k in range(2, number_of_iterations):
        x = x - alpha * df_dx(x + beta*(x - dots[k-1, 0]), y) +
        beta*(x-dots[k-1, 0]) y = y - alpha * df_dy(x, y + beta*(y -
        dots[k-1, 1]))  + beta*(y-dots[k-1, 1])
        dots = np.append(dots, [[x, y]], axis=0)
```

```
def stohastic_grad_descent(x, y, lmd, number_of_iterations):
    """Стохастичний градієнтний спуск з константними заданними кроками та наперед заданною
    кількістю ітерацій""" dots = np.array([x, y], dtype=float).reshape(-1, 2)
    for n in range(number_of_iterations):
        if
            np.random.ra
            ndint(0,2)
            == 0: x = x
            - lmd *
            df_dx(x, y)
        else:
            y = y - lmd * df_dy(x, y)
        dots = np.append(dots, [[x, y]], axis=0)
    return dots
```

Level lines



Function



Висновок дослідив градієнтні методи оптимізації функцій кількох змінних, визначили, що результат роботи кожного з алгоритмів залежить від обраного початкового наближення та гіперпараметрів: стохастичному градієнтному спуску потрібно більше ітерацій, проте він має вдвічі менше обчислень, ніж у класичному; завдяки коефіцієнтам та і використанням попередніх результатів значно менше відхиляється від своєї траєкторії та швидше сходиться до мінімуму.

Використання таких методів дозволяє знайти найкращі рішення на етапі проектування, підвищуючи ефективність конструкцій.

Література

[1] ***https://en.wikipedia.org/wiki/Test_functions_for_optimization***

[2] ***https://en.wikipedia.org/wiki/Himmelblau%27s_function***

[3] Дагіль В.Г., Хаткова Л.В. Розробка методики розрахунку показників надійності будівель з використанням теорії імовірностей та математичної статистики. Збірник наукових праць Черкаського інституту пожежної безпеки імені Героїв Чорнобиля Національного університету цивільного захисту України «Надзвичайні ситуації: попередження та ліквідація» Том 7 №1 (2023) с.15-22 ***https://fire-journal.ck.ua/index.php/fire/article/view/161/138***