Лабораторная работа по теме Итерационные методы решения слау

Вариант 4

Метод наискорейшего градиентного спуска (с 455)

описание метода

Метод наискорейшего градиентного спуска применяется для решения системы линейных алгебраических уравнений (СЛАУ) вида: $Ax = b$
где
A - симметричная положительно определенная матрица n×n
x - вектор неизвестных
b - вектор правых частей

In [98]:
```
import numpy as np
```

In [99]:
```
def solve(
    A: np.ndarray,
    b: np.ndarray,
    eps: float = 1e-15,
) -> tuple[np.ndarray]:
    """
    Решение системы методом наискорейшего градиентного спуска
    residual - невязка
    """

    """
    этап 1:
    готовим начальные x и невязку
    которая будет выполнять роль отслеживания насколько близко мы находимся
    """
    n = A.shape[0]
    x = np.zeros(n)
    x_new = np.ones(n)
    residual = A @ x - b
    residual_new = residual

    it_counter = 0

    while np.all(abs(x - x_new) > eps):
        it_counter += 1
        x = x_new
        residual = residual_new
        """
        этап 2:
```

```
        на основе вектора невязки вычисляем оптимальный шаг alpha
        alpha_k = (r_k, r_k) / (Ar_k, r_k)
        и обновляем решение
        """
        alpha = np.dot(residual, residual) / np.dot(A @ residual, residual)
        x_new = x - alpha * residual

        """
        этап 3:
        вычисляем новую невязку по новому решению
        r_k = F - AX_k
        """
        residual_new = A @ x_new - b

        # if it_counter % 100 == 0:
        #     print(f"{it_counter=}: невязка={np.linalg.norm(residual_new):.

    print("Итоги:")
    print(f"Количество итераций: {it_counter}")
    print(f"Невязка: {np.linalg.norm(residual_new):.6e}")

    return x
```

In [100…
```
def check_input_matrix(A: np.ndarray, b: np.ndarray) -> bool:
    eigvals = np.linalg.eigvals(A)
    if np.any(eigvals <= 0):
        return False

    return np.all(np.linalg.solve(A, b))
```

In [101…
```
def generate(size: int = 4, tryings: int = 10**7) -> tuple[np.ndarray, np.nd
    for _ in range(tryings):
        # A_random = np.random.randn(size, size)
        A_random = np.random.randint(-100, 100, size=(size, size))
        A = (A_random + A_random.T)

        b = np.random.randint(-100, 100, size=(size))
        if check_input_matrix(A, b):
            return A, b
```

In [102…
```
def test(A: np.ndarray, b: np.ndarray) -> None:
    print("Исходная матрица A")
    print(A)
    print("Исходный вектор b")
    print(b)
    # print(f"проверка на валидность входных данных {check_input_matrix(A, b

    my_x = solve(A, b)
    np_x = np.linalg.solve(A, b)
    error = abs(my_x - np_x)

    print(
        f"мой x: {my_x}",
        f"точный x: {np_x}",
```

```
        f"Ошибка: {error}",
        f"Максимальная ошибка: {np.max(error):.6e}",
        sep="\n"
    )
```

In [103…
```
# Тест: Простая симметричная матрица
A = np.array([
    [2, 1],
    [1, 2],
], dtype=float)
b = np.array([1,1])
test(A, b)
```

```
Исходная матрица A
[[2. 1.]
 [1. 2.]]
Исходный вектор b
[1 1]
Итоги:
Количество итераций: 3
Невязка: 0.000000e+00
мой x: [0.33333333 0.33333333]
точный x: [0.33333333 0.33333333]
Ошибка: [1.11022302e-16 5.55111512e-17]
Максимальная ошибка: 1.110223e-16
```

In [ ]:
```
A, b = generate(size=6)
test(A, b)
```

In [106…
```
A = np.array([
    [5, 7, 6, 5],
    [7, 10, 8, 7],
    [6, 8, 10, 9],
    [5, 7, 9, 10],
], dtype=float)
b = np.array([23,32,33,31])
test(A, b)
```

```
Исходная матрица A
[[ 5.  7.  6.  5.]
 [ 7. 10.  8.  7.]
 [ 6.  8. 10.  9.]
 [ 5.  7.  9. 10.]]
Исходный вектор b
[23 32 33 31]
Итоги:
Количество итераций: 21090
Невязка: 6.702494e-13
мой x: [1. 1. 1. 1.]
точный x: [1. 1. 1. 1.]
Ошибка: [3.91664479e-11 2.36435316e-11 9.82480763e-12 5.84088333e-12]
Максимальная ошибка: 3.916645e-11
```