Mini Project: Dynamic programming

Mini Project: Dynamic programming

小项目报告：动态规划在序列比对中的应用

小组11：王誉凯、刘正涛、高翔宇

一、引言

二、项目文件结构

三、算法与代码

1. 下载并解压后的 chr1 序列
2. 计算 chr1 中 A/C/G/T 频率
3. 按频率随机生成 DNA 序列
4. 全局比对（Needleman–Wunsch）
5. 绘制直方图
6. 串联全过程：生成→比对→绘图
7. 局部比对（Smith–Waterman）
8. 在随机序列上运行局部比对并回溯

四、运行与结果

五、结论

六、附录

1. **Needleman-Wunsch algorithm (global alignment)**
   - Generate 1000 random DNA sequences with the same A/C/G/T proportions as the chromosome 1, human reference genome (Note: You need to download the chr1 of the human genome, and count them).
   - Use your pre-built library to parse the DNA sequences stored in a FASTA file, and output the optimal pairwise global alignment score, and store them in a square matrix.
   - Draw a histogram of the alignment scores using the functions in Matplotlib package. Does it look like the bell-shape? That is, is it similar to Gaussian (normal) distribution?
   - You need to repeat the process with different length setting: $N=50, 100, 200, 500

2. **Smith-Waterman algorithm (local alignment)**
   - Write Smith-Waterman algorithm in Python using the fashion of object-oriented programming.
   - Run local alignment on the random sequences generated in the exercise above.

小项目报告：动态规划在序列比对中的应用

小组11：王誉凯、刘正涛、高翔宇

一、引言

本项目采用经典的动态规划算法——Needleman–Wunsch（全局比对）和 Smith–Waterman（局部比对）——对随机生成的 DNA 序列进行比对，验证其得分分布，并演示局部比对如何捕捉两条序列中最相似的片段。

二、项目文件结构

```
mini_project/
├── chr1.fa                        # 下载并解压后的 chr1 序列
├── count_freqs.py                 # 计算 chr1 中 A/C/G/T 频率
├── random_seqs.py                 # 按频率随机生成 DNA 序列
├── compute_scores.py              # 计算全局比对得分矩阵
├── plot_histograms.py             # 绘制得分直方图
├── run_all.py                     # 串联全过程：生成→比对→绘图
├── smith_waterman.py              # Smith-Waterman 局部比对 OOP 实现
├── local_alignment_demo.py        # 在随机序列上运行局部比对并回溯
├── requirements.txt               # numpy, matplotlib, biopython
└── README.md                      # 使用说明
```

三、算法与代码

1. 下载并解压后的 chr1 序列

```
curl -O
ftp://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/chr1.fa.gz
gunzip chr1.fa.gz
```

2. 计算 chr1 中 A/C/G/T 频率

```python
# count_freqs.py
from collections import Counter

def count_bases(fasta_path):
    freqs = Counter()
    total = 0
    with open(fasta_path) as f:
        for line in f:
            if line.startswith(">"):
                continue
            seq = line.strip().upper()
            freqs.update(seq)
            total += len(seq)
    # 归一化得到 A/C/G/T 的概率
    probs = {b: freqs[b]/total for b in "ACGT"}
    return probs

if __name__ == "__main__":
    p = count_bases("chr1.fa")
    print("A,C,G,T 频率:", p)
```

3. 按频率随机生成 DNA 序列

```python
# random_seqs.py
import random

def generate_random_seqs(probs, N, M=1000):
    """返回 M 条长度为 N、基于给定频率随机生成的序列列表"""
    bases, weights = zip(*probs.items())
    seqs = [
        "".join(random.choices(bases, weights, k=N))
        for _ in range(M)
    ]
    return seqs

if __name__ == "__main__":
    from count_freqs import count_bases
    probs = count_bases("chr1.fa")
    # 示例：N=50 的随机序列
    seqs50 = generate_random_seqs(probs, 50)
```

4. 全局比对（Needleman-Wunsch）

```
# compute_scores.py
import numpy as np
from Bio import pairwise2

def score_matrix(seqs, match=1, mismatch=-1, gap_open=-2,
gap_extend=-0.5):
    M = len(seqs)
    mat = np.zeros((M, M))
    for i in range(M):
        for j in range(i, M):
            # globalms 使用 match/mismatch/gap 分数
            aln = pairwise2.align.globalms(
                seqs[i], seqs[j],
                match, mismatch,
                gap_open, gap_extend,
                score_only=True
            )
            mat[i,j] = mat[j,i] = aln
    return mat

if __name__ == "__main__":
    from random_seqs import generate_random_seqs
    from count_freqs import count_bases

    probs = count_bases("chr1.fa")
    for N in [50, 100, 200, 500]:
        seqs = generate_random_seqs(probs, N)
        print(f"正在计算 N={N} 的比对得分 …")
        mat = score_matrix(seqs)
        np.save(f"scores_N{N}.npy", mat)
```

5. 绘制直方图

```
# plot_histograms.py
import numpy as np
import matplotlib.pyplot as plt

plt.rcParams['font.family'] = 'Heiti TC'

def plot_hist(mat, N):
    idx = np.triu_indices_from(mat, k=1)
    scores = mat[idx]
    plt.figure()
    plt.hist(scores, bins=50)
    plt.title(f"全局比对得分分布 (N={N})")
    plt.xlabel("得分")
    plt.ylabel("频数")
    plt.grid(True)
    plt.savefig(f"hist_N{N}.png")
    plt.close()

if __name__ == "__main__":
    for N in [50, 100, 200, 500]:
        mat = np.load(f"scores_N{N}.npy")
        plot_hist(mat, N)
        print(f"已保存 hist_N{N}.png")
```

6. 串联全过程：生成→比对→绘图

```python
# run_all.py
import os
from count_freqs import count_bases
from random_seqs import generate_random_seqs
from compute_scores import score_matrix
from plot_histograms import plot_hist
import numpy as np

def main():
    probs = count_bases("chr1.fa")
    for N in [50, 100, 200, 500]:
        print(f"\n=== N = {N} ===")
        seqs = generate_random_seqs(probs, N)
        mat = score_matrix(seqs)
        npy = f"scores_N{N}.npy"
        png = f"hist_N{N}.png"
        np.save(npy, mat)
        print(f"已保存 {npy}")
        plot_hist(mat, N)
        print(f"已保存 {png}")

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

7. 局部比对（Smith-Waterman）

```python
# smith_waterman.py

import numpy as np

class SmithWaterman:
    """
    Smith-Waterman 本地比对实现。
    参数:
        match      : 匹配得分 (默认为 +2)
        mismatch   : 错配惩罚 (默认为 -1)
        gap        : 缺口惩罚 (默认为 -1)
    方法:
        score(seq1, seq2) -> (max_score, H, max_pos)
            计算得分矩阵 H 并返回最大得分及其位置。
        traceback(seq1, seq2, H, max_pos) -> (aligned1, aligned2)
            从 H 和 max_pos 回溯出一条最优局部比对。
    """

    def __init__(self, match=2, mismatch=-1, gap=-1):
        self.match = match
        self.mismatch = mismatch
        self.gap = gap

    def score(self, seq1: str, seq2: str):
        m, n = len(seq1), len(seq2)
        H = np.zeros((m+1, n+1), dtype=int)
        max_score = 0
        max_pos = (0, 0)

        # 填表
        for i in range(1, m+1):
            for j in range(1, n+1):
                if seq1[i-1] == seq2[j-1]:
                    diag = H[i-1, j-1] + self.match
                else:
                    diag = H[i-1, j-1] + self.mismatch
                up   = H[i-1, j]   + self.gap
                left = H[i,   j-1] + self.gap
                H[i, j] = max(0, diag, up, left)
                if H[i, j] > max_score:
                    max_score = H[i, j]
                    max_pos   = (i, j)

        return max_score, H, max_pos
```

```python
    def traceback(self, seq1: str, seq2: str, H: np.ndarray, max_pos):
        """
        从 max_pos 开始回溯，直到遇到 0 为止，重构一条最优局部比对串。
        返回 aligned1, aligned2（带 '-' 的对齐序列）。
        """
        aligned1, aligned2 = [], []
        i, j = max_pos
        while i > 0 and j > 0 and H[i, j] > 0:
            score_cur = H[i, j]
            if seq1[i-1] == seq2[j-1]:
                score_diag = H[i-1, j-1] + self.match
            else:
                score_diag = H[i-1, j-1] + self.mismatch

            if score_cur == score_diag:
                aligned1.append(seq1[i-1])
                aligned2.append(seq2[j-1])
                i -= 1; j -= 1
            elif score_cur == H[i-1, j] + self.gap:
                aligned1.append(seq1[i-1]); aligned2.append('-')
                i -= 1
            else:
                aligned1.append('-'); aligned2.append(seq2[j-1])
                j -= 1

        return ''.join(reversed(aligned1)),
''.join(reversed(aligned2))
```

8. 在随机序列上运行局部比对并回溯

```python
# local_alignment_demo.py

import numpy as np
from count_freqs    import count_bases
from random_seqs    import generate_random_seqs
from smith_waterman import SmithWaterman

def compute_local_matrix(seqs, aligner):
    M = len(seqs)
    mat = np.zeros((M, M), dtype=int)
    for i in range(M):
        for j in range(i, M):
            score, _, _ = aligner.score(seqs[i], seqs[j])
            mat[i, j] = mat[j, i] = score
    return mat

if __name__ == "__main__":
    # 示例: N=50, 序列数 M=10 (为了快速演示)
    N, M = 50, 10
    probs = count_bases("chr1.fa")
    seqs  = generate_random_seqs(probs, N, M)
    aligner = SmithWaterman(match=2, mismatch=-1, gap=-1)

    # 1) 计算局部比对得分矩阵
    local_mat = compute_local_matrix(seqs, aligner)
    print("Local score matrix shape:", local_mat.shape)
    print(local_mat)

    # 2) 对第 0 条和第 1 条序列回溯一次
    s1, s2 = seqs[0], seqs[1]
    max_score, H, max_pos = aligner.score(s1, s2)
    aln1, aln2 = aligner.traceback(s1, s2, H, max_pos)
    print("\nSeq[0]:", s1)
    print("Seq[1]:", s2)
    print("Local alignment score:", max_score)
    print("Alignment result:")
    print(aln1)
    print(aln2)

    # 3) 保存矩阵到文件
```

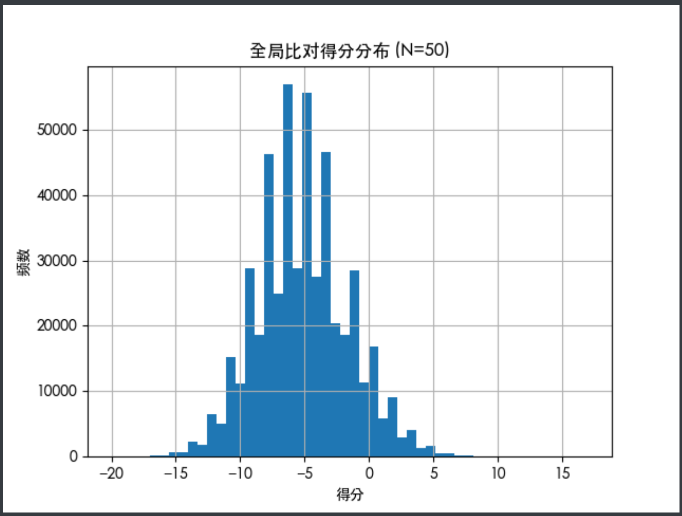
```
        np.save(f"local_scores_N{N}.npy", local_mat)
        print(f"已保存 local_scores_N{N}.npy")
```

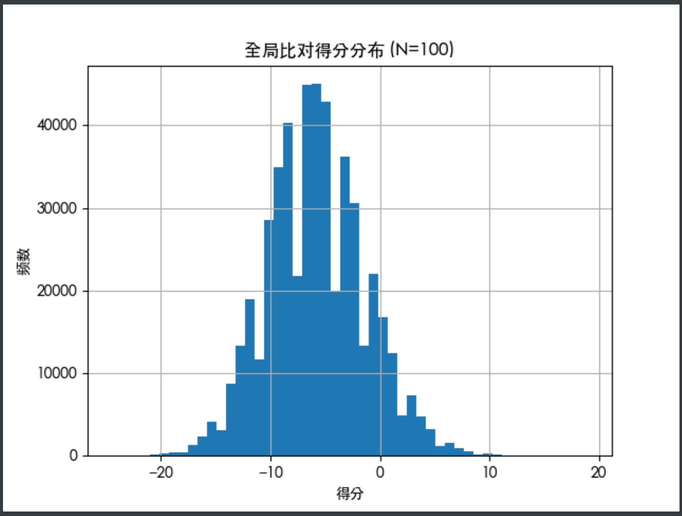
四、运行与结果

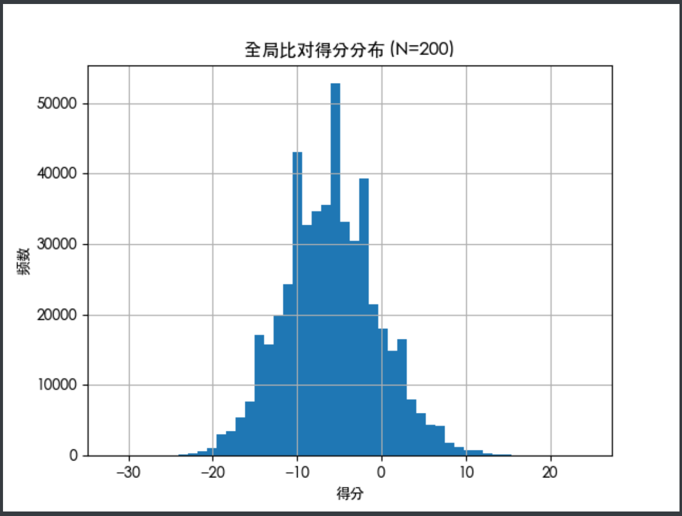
1. **全局比对**
   - 对每个 N∈{50,100,200,500}，生成 1000 条随机序列，计算 1000×1000 全局得分矩阵，保存为 `scores_N{N}.npy` 。
   - 绘制直方图 `hist_N{N}.png` 。

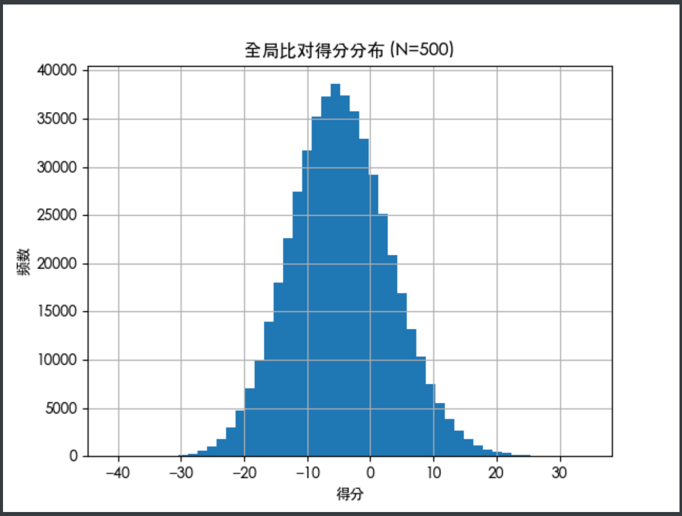
观察：随着 N 增大，得分分布更趋近于钟形，标准差相对集中，符合中心极限定理。

结果：图 1–4 全局比对得分直方图（N=50,100,200,500）









2. **局部比对**
   - 在 M=10、N=50 的小样本上演示：

```
Local score matrix shape: (10, 10)
[[100  34  34  34  38  39  35  29  28  41]
 [ 34 100  33  34  39  39  37  41  41  37]
 [ 34  33 100  35  35  33  38  38  29  38]
 [ 34  34  35 100  45  40  32  34  33  36]
 [ 38  39  35  45 100  45  34  40  31  39]
 [ 39  39  33  40  45 100  32  39  32  34]
 [ 35  37  38  32  34  32 100  32  34  37]
 [ 29  41  38  34  40  39  32 100  36  32]
 [ 28  41  29  33  31  32  34  36 100  32]
 [ 41  37  38  36  39  34  37  32  32 100]]
```

   - 对序列 0 和 1 回溯：

```
Seq[0]: TTCGCAAGAGCGTGTCTTGGCCATCGGAAAGTTGCTAGCGTGCATTATCA
Seq[1]: AGATGCTCCCTCATATAGTACAGCACGTGATGCCTAATTTAAAACACAGA
Local alignment score: 34
Alignment result:
AGAGCGTGTCTTGGCCATCGGA-A-AGT--TGCTAGCGTGCAT---T-AT
AGA---TG-C-T--CCCTC--ATATAGTACAGC-A-CGTG-ATGCCTAAT
```

五、结论

- **全局比对**：随机序列之间的全局得分分布近似正态，且随序列长度增加，分布更加集中。
- **局部比对**：能够精准捕捉两条序列中连续相似的片段，对功能域或保守区分析尤为有效。

六、附录

```
# requirements.txt

numpy>=1.24.0
matplotlib>=3.7.0
biopython>=1.81
```

~~~
# README.md

# Mini Project: Dynamic Programming in Sequence Alignment

## 项目简介
本项目演示了两种经典的生物序列比对算法：
- **Needleman–Wunsch** （全局比对）
- **Smith–Waterman** （局部比对）

通过随机生成与人类 Chr1 相同碱基频率的 DNA 序列，并实施全局和局部比对，分析比对得分分布与示例对齐结果。

## 文件结构
```text
mini_project/
~~~

```
├── __pycache__/                  # Python 缓存目录
├── chr1.fa                       # 下载并解压后的 Chr1 序列文件
├── count_freqs.py                # 计算碱基频率脚本
├── random_seqs.py                # 按频率随机生成 DNA 序列脚本
├── compute_scores.py             # 全局比对得分矩阵计算脚本
├── plot_histograms.py            # 绘制直方图脚本
├── run_all.py                    # 串联全过程的主脚本
├── smith_waterman.py             # 面向对象的 Smith-Waterman 实现
├── local_alignment_demo.py       # 局部比对示例脚本（带回溯）
├── miniproject-dp.md             # 项目报告 Markdown 文件
├── requirements.txt              # Python 依赖列表
├── README.md                     # 本说明文件
├── hist_N50.png                  # N=50 全局比对得分直方图
├── hist_N100.png                 # N=100 全局比对得分直方图
├── hist_N200.png                 # N=200 全局比对得分直方图
├── hist_N500.png                 # N=500 全局比对得分直方图
├── scores_N50.npy                # N=50 全局比对得分矩阵
├── scores_N100.npy               # N=100 全局比对得分矩阵
├── scores_N200.npy               # N=200 全局比对得分矩阵
├── scores_N500.npy               # N=500 全局比对得分矩阵
├── local_scores_N50.npy          # N=50 局部比对得分矩阵示例
```