

# Функции расстояния (метрики) на множестве аминокислот (« 20 × 20 »)

Киселев Никита Сергеевич, M05-402в

20 мая 2025 г.

## 1 Введение

Выравнивание биологических последовательностей является фундаментальной задачей в биоинформатике, играющей центральную роль в понимании структурных, функциональных и эволюционных взаимоотношений между белками. Точность выравнивания напрямую зависит от используемой функции расстояния между элементами последовательностей, в частности, аминокислотами. Традиционные подходы, основанные на матрицах PAM (Point Accepted Mutation) или BLOSUM (BLOcks SUBstitution Matrix), используют эмпирические данные о частоте замен аминокислот в процессе эволюции, однако они не учитывают прямую физико-химические свойства аминокислот и часто не обладают важными математическими свойствами метрических пространств.

В настоящей работе мы представляем SNACK (Sequence Normalized Alignment Comparison Kit) — теоретически обоснованную параметрическую функцию расстояния на множестве аминокислот  $\mathcal{A} = \{a_1, a_2, \dots, a_{20}\}$ , которая:

- 1) Учитывает структурные, химические и биофизические свойства аминокислот через их представление в многомерном признаковом пространстве;
- 2) Обеспечивает согласованность с критериями оптимальности выравнивания;
- 3) Оптимизирована для высокопроизводительных вычислений с использованием GPU/MPS (Metal Performance Shaders) акселерации;

4) Демонстрирует превосходство над традиционными эмпирическими подходами.

Математическая формулировка предлагаемой метрики основана на параметрической квадратичной форме в пространстве признаков аминокислот. Каждая аминокислота  $a_i \in \mathcal{A}$  представляется вектором признаков  $\phi(a_i) \in \mathbb{R}^k$ , где компоненты соответствуют таким свойствам как гидрофобность, молекулярный вес, полярность, заряд и объем. Расстояние между аминокислотами определяется как:

$$d(a_i, a_j) = (\phi(a_i) - \phi(a_j))^T M (\phi(a_i) - \phi(a_j)), \quad (1)$$

где  $M \in \mathbb{R}^{k \times k}$  — положительно полуопределенная симметричная матрица, определяющая метрику в пространстве признаков. Параметры матрицы  $M$  оптимизируются с использованием градиентных методов на основе набора эталонных выравниваний из базы данных BALIBASE.

Программная реализация SNACK включает оптимизированный алгоритм Needleman-Wunsch для глобального выравнивания с использованием полученной метрики, а также оптимизации для параллельных вычислений с применением JIT-компиляции и векторизации для CPU (Numba) и вычислений на GPU (PyTorch). Результаты экспериментов показывают улучшение точности выравнивания на 2.5% по сравнению с традиционными матрицами BLOSUM и PAM, а также значительное ускорение при использовании аппаратного ускорения.

## 2 Теоретические основы

В данном разделе мы формализуем задачу построения метрики на множестве аминокислот и связываем её с задачей оптимального выравнивания последовательностей.

### 2.1 Формализация признакового пространства аминокислот

Пусть  $\mathcal{A} = \{a_1, a_2, \dots, a_{20}\}$  — конечное множество стандартных аминокислот, дополненное символом пробела "—" для выравниваний. В нашем подходе каждая аминокислота представляется набором биофизических и химических характеристик, образующих признаковое пространство размерности  $k = 5$ .

Отображение  $\phi : \mathcal{A} \rightarrow \mathbb{R}^k$  сопоставляет каждой аминокислоте  $a_i \in \mathcal{A}$  вектор признаков  $\phi(a_i) = \mathbf{x}_i \in \mathbb{R}^k$ , где компоненты представляют следующие свойства:

$$\mathbf{x}_i = [h_i, w_i, p_i, c_i, v_i]^T, \quad (2)$$

где  $h_i$  — индекс гидрофобности по шкале Kyte-Doolittle,  $w_i$  — нормализованная молекулярная масса,  $p_i$  — полярность,  $c_i$  — заряд при pH=7,  $v_i$  — нормализованный объем.

Выбор именно этих характеристик обусловлен их ролью в определении структурных и функциональных свойств белков. Гидрофобность и полярность влияют на пространственную организацию белка, заряд определяет электростатические взаимодействия, а молекулярная масса и объем характеризуют стерические эффекты.

## 2.2 Параметрическая метрика в пространстве аминокислот

Для любых  $a_i, a_j \in \mathcal{A}$  функция расстояния  $d : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$  определяется как:

$$d(a_i, a_j; M) = (\phi(a_i) - \phi(a_j))^T M (\phi(a_i) - \phi(a_j)) \quad (3)$$

$$= \Delta_{ij}^T M \Delta_{ij}, \quad (4)$$

где  $\Delta_{ij} = \mathbf{x}_i - \mathbf{x}_j$  — разность признаков векторов, а  $M \in \mathbb{R}^{k \times k}$  — симметричная положительно полуопределенная матрица параметров метрики.

Использование квадратичной формы с матрицей  $M$  позволяет учесть взаимозависимости между различными характеристиками аминокислот. Так, связь между полярностью и гидрофобностью отражается в соответствующих недиагональных элементах матрицы  $M$ .

Для функции  $d$  можно выделить следующие важные свойства:

При  $M \succeq 0$  (положительная полуопределённость) функция  $d$  обладает следующими свойствами:

1.  $d(a_i, a_j) \geq 0$  для всех  $a_i, a_j \in \mathcal{A}$  (неотрицательность)
2.  $d(a_i, a_i) = 0$  для всех  $a_i \in \mathcal{A}$  (рефлексивность)
3.  $d(a_i, a_j) = d(a_j, a_i)$  для всех  $a_i, a_j \in \mathcal{A}$  (симметричность)

Для обеспечения всех свойств метрического пространства, включая неравенство треугольника, необходимы дополнительные ограничения на матрицу  $M$ .

## 2.3 Целевая функция для оптимизации параметров метрики

Основной задачей является определение оптимальной матрицы параметров  $M$ , обеспечивающей наилучшее соответствие выравниваний, полученных с использованием метрики  $d$ , с эталонными выравниваниями.

Определим составную целевую функцию как:

$$\mathcal{L}(M) = \underbrace{\mathcal{L}_{\text{align}}(M)}_{\text{Ошибка выравнивания}} + \underbrace{\lambda_1 \cdot \mathcal{L}_{\text{asym}}(M)}_{\text{Асимметрия}} + \underbrace{\lambda_2 \cdot \mathcal{L}_{\text{triangle}}(M)}_{\text{Нарушение неравенства треугольника}}, \quad (5)$$

где  $\lambda_1, \lambda_2 \geq 0$  — коэффициенты регуляризации, балансирующие влияние различных компонентов.

### 2.3.1 Компонент ошибки выравнивания $\mathcal{L}_{\text{align}}(M)$

Пусть имеется набор эталонных данных  $\mathcal{D} = \{(S_1^{(n)}, S_2^{(n)}, A_{\text{ref}}^{(n)})\}_{n=1}^N$ , где  $S_1^{(n)}, S_2^{(n)}$  — пары последовательностей, а  $A_{\text{ref}}^{(n)}$  — их эталонные выравнивания.

Для заданной матрицы  $M$  и соответствующей функции расстояния  $d_M$ , алгоритм выравнивания (в нашем случае — модифицированный Needleman–Wunsch) порождает выравнивание  $A_M^{(n)}$  для каждой пары последовательностей. Функция потерь определяется как:

$$\mathcal{L}_{\text{align}}(M) = \frac{1}{N} \sum_{n=1}^N \left( 1 - \frac{|A_M^{(n)} \cap A_{\text{ref}}^{(n)}|}{|A_{\text{ref}}^{(n)}|} \right), \quad (6)$$

где  $|A_M^{(n)} \cap A_{\text{ref}}^{(n)}|$  — количество совпадающих пар в выравниваниях.

### 2.3.2 Регуляризация симметричности $\mathcal{L}_{\text{asym}}(M)$

Хотя по построению матрица  $M$  симметрична, для численной устойчивости и явной регуляризации вводим дополнительный штраф за асимметрию:

$$\mathcal{L}_{\text{asym}}(M) = \sum_{i=1}^{20} \sum_{j=i+1}^{20} |d(a_i, a_j; M) - d(a_j, a_i; M)|. \quad (7)$$

Данный компонент должен обращаться в ноль для корректно параметризованной метрики, но помогает стабилизировать обучение.

### 2.3.3 Регуляризация неравенства треугольника $\mathcal{L}_{\text{triangle}}(M)$

Для обеспечения свойства метрического пространства, функция расстояния должна удовлетворять неравенству треугольника:

$$d(a_i, a_k; M) \leq d(a_i, a_j; M) + d(a_j, a_k; M), \quad \forall a_i, a_j, a_k \in \mathcal{A} \quad (8)$$

Мы вводим штраф за нарушение этого неравенства:

$$\mathcal{L}_{\text{triangle}}(M) = \sum_{i=1}^{20} \sum_{j=1}^{20} \sum_{k=1}^{20} \max(0, d(a_i, a_j; M) + d(a_j, a_k; M) - d(a_i, a_k; M))^2. \quad (9)$$

## 2.4 Задача оптимизации

Итоговая задача оптимизации формулируется как:

$$M^* = \arg \min_{M \succeq 0} \mathcal{L}(M) = \arg \min_{M \succeq 0} \{\mathcal{L}_{\text{align}}(M) + \lambda_1 \mathcal{L}_{\text{asym}}(M) + \lambda_2 \mathcal{L}_{\text{triangle}}(M)\}, \quad (10)$$

где условие  $M \succeq 0$  означает, что матрица  $M$  должна быть положительно полуопределенной.

Эта задача решается с использованием стохастических градиентных методов оптимизации с соответствующими проекциями на множество положительно полуопределенных матриц.

## 3 Алгоритм выравнивания и его оптимизация

### 3.1 Алгоритм Needleman-Wunsch с параметрической метрикой

Для выравнивания последовательностей с использованием разработанной метрики мы применяем модифицированный алгоритм Needleman-Wunsch, который является классическим подходом к глобальному выравниванию последовательностей. Основное отличие нашей реализации состоит в использовании параметрической метрики  $d(a_i, a_j; M)$  в качестве штрафа за замену.

### 3.2 Оптимизация производительности алгоритма

Для повышения производительности выравнивания, особенно при работе с большими наборами последовательностей или при выполнении процессов обучения, мы применяем несколько ключевых оптимизаций.

#### 3.2.1 Кэширование метрических значений

Вычисление значений метрики  $d(a_i, a_j)$  может быть вычислительно затратным, так как включает вычисление признаков векторов и матричные операции. Для ускорения мы используем технику кэширования, которая сохраняет уже вычисленные значения метрики:

$$d_{cached}(a_i, a_j) = \begin{cases} cache[(a_i, a_j)], & \text{если } (a_i, a_j) \in cache \\ compute\_and\_store(a_i, a_j), & \text{иначе} \end{cases} \quad (11)$$

Для реализации кэширования в Python мы используем декоратор '@lru\_cache' из стандартной библиотеки 'functools'.

#### 3.2.2 JIT-компиляция с Numba

Для дальнейшего ускорения алгоритма выравнивания мы применяем JIT-компиляцию с использованием библиотеки Numba. Это позволяет

преобразовать интерпретируемый код Python в оптимизированный машинный код, что особенно эффективно для циклов и численных вычислений:

```
1 @njit
2 def _nw_core(len1, len2, dp, match_matrix, delete_matrix,
3             insert_matrix):
4     """JIT-compiled core of Needleman-Wunsch algorithm"""
5     for i in range(1, len1 + 1):
6         for j in range(1, len2 + 1):
7             match = dp[i - 1][j - 1] + match_matrix[i -
8             1][j - 1]
9             delete = dp[i - 1][j] + delete_matrix[i - 1]
10            insert = dp[i][j - 1] + insert_matrix[j - 1]
11            dp[i][j] = min(match, delete, insert)
12    return dp
```

### 3.2.3 Векторизация с NumPy

В случаях, когда Numba недоступна, мы используем векторизованные операции NumPy для эффективной обработки матрицы динамического программирования. Это позволяет заменить явные циклы более эффективными векторными операциями, которые выполняются на низком уровне с использованием оптимизированных библиотек линейной алгебры:

```
1 def _nw_numpy(seq1, seq2, metric_values):
2     """NumPy implementation of Needleman-Wunsch"""
3     # Matrix initialization and filling with vectorized
4     # NumPy operations
5     # ...
```

### 3.2.4 Аппаратное ускорение для GPU/MPS

Для максимальной производительности на современном оборудовании мы реализовали поддержку GPU-вычислений через PyTorch. Система автоматически определяет доступные аппаратные ускорители (CUDA для NVIDIA GPU или MPS для Apple Silicon) и адаптирует вычисления:

```

1 def get_device():
2     """Determine the best available device for PyTorch"""
3     if torch.cuda.is_available():
4         return torch.device("cuda")
5     elif hasattr(torch.backends, "mps") and torch.
6         backends.mps.is_available():
7         return torch.device("mps") # Apple Silicon GPU
8     else:
9         return torch.device("cpu")

```

### 3.2.5 Предварительное вычисление признаков векторов

Для ускорения доступа к признаковым векторам аминокислот мы предварительно вычисляем их для всех стандартных аминокислот при инициализации:

```

1 def _precompute_features(self):
2     """Pre-compute feature vectors for all standard amino
3     acids"""
4     for aa in self.amino_acids:
5         self._compute_features(aa)

```

### 3.2.6 Батч-обработка для эффективного обучения

При обучении модели мы группируем последовательности сходной длины в мини-батчи для более эффективного использования параллельных вычислений и минимизации накладных расходов на передачу данных между CPU и GPU:

```

1 # Sort by sequence length for more efficient batch
2   processing
3 alignment_data.sort(key=lambda pair: (len(pair[0]), len(
4   pair[1])))
5
6 # Process data in batches
7 for i in range(0, len(alignment_data), batch_size):
8     batch = alignment_data[i:i+batch_size]
9     # ... batch processing

```



## 4 Архитектура библиотеки SNACK

SNACK (Sequence Normalized Alignment Comparison Kit) представляет собой модульную библиотеку на языке Python, оптимизированную для высокопроизводительных вычислений и обеспечивающую гибкую работу с метриками выравнивания последовательностей. Основная структура библиотеки представлена на следующих компонентах:

### 4.1 Модульная организация

Библиотека организована в несколько взаимосвязанных модулей, каждый из которых отвечает за определенный функциональный аспект:

- **features.py** — модуль для работы с признаковым пространством аминокислот, включая нормализацию и кэширование признаков;
- **metric.py** — реализация параметрической метрики и функций потерь для оптимизации;
- **alignment.py** — оптимизированные алгоритмы выравнивания с поддержкой Numba и NumPy;
- **data.py** — функции для загрузки и предобработки данных из форматов MSF (BALIBASE);
- **train.py** — процессы обучения с поддержкой аппаратного ускорения.

### 4.2 Интеграция с экосистемой научного Python

SNACK интегрируется с основными библиотеками научного Python:

- **PyTorch** используется для дифференцируемых вычислений, оптимизации матричных параметров и работы с GPU;
- **NumPy** обеспечивает векторизованные операции для эффективной работы с матрицами;
- **Numba** применяется для JIT-компиляции критичных к производительности участков кода;

- **Matplotlib** и **Seaborn** используются для визуализации результатов в интерактивных блокнотах.

## 4.3 Основные классы и интерфейсы

Ключевыми компонентами программной архитектуры являются:

1. **FeatureSpace** — класс для работы с признаковыми представлениями аминокислот:

```

1 class FeatureSpace:
2     def __init__(self, device=None):
3         self.feature_dim = 5
4         self.device = device
5         self._feature_cache = {}
6         self.amino_acids = "ACDEFGHIKLMNPQRSTVWY-"
7         # Bio-characteristics initialization...
8
9     def get_features(self, amino_acid):
10        # Feature vector with caching

```

2. **Snack** — основной класс для параметрической метрики, наследуемый от `torch.nn.Module`:

```

1 class Snack(nn.Module):
2     def __init__(self, feature_space, lambda1=1.0,
3         lambda2=1.0):
4         super().__init__()
5         self.feature_space = feature_space
6         self.lambda1 = lambda1 # asymmetry weight
7         self.lambda2 = lambda2 # triangle inequality
8         weight
9         self.M = nn.Parameter(torch.eye(self.
10            feature_space.feature_dim))
11         self._features_cache = {}
12
13     def __call__(self, i, j):
14         # calculate distance between acids
15
16     def total_loss(self, alignment_data):
17         # total loss for optimization

```

3. **needleman\_wunsch** — оптимизированная функция для выравнивания последовательностей:

```
1 def needleman_wunsch(seq1, seq2, metric_func):  
2     # Optimized alignment with cache
```

## 4.4 Оптимизация производительности

Для обеспечения высокой производительности в SNACK реализованы:

1. Многоуровневое кэширование признаков векторов и метрических значений;
2. Автоматическое определение и использование GPU/MPS акселерации;
3. Адаптивная стратегия объединения последовательностей в пакеты по длине;
4. Оптимизированные пути исполнения в зависимости от доступных библиотек.

## 5 Экспериментальное исследование

### 5.1 Сравнение методов выравнивания

Мы провели сравнительное исследование параметрической метрики SNACK с традиционными матричными подходами (BLOSUM62 и PAM250) на эталонных данных BALIBASE.

#### 5.1.1 Данные и протокол эксперимента

В эксперименте использовались 218 эталонных выравниваний из набора BALIBASE 3.0, разделенных на пять категорий по сложности. Для обучения модели использовалось 80% данных, а оставшиеся 20% составили тестовую выборку.

Для каждого метода мы выполняли глобальное выравнивание всех пар последовательностей и оценивали точность выравнивания по сравнению с эталонным, используя метрику SPS (Sum-of-Pairs Score):

$$\text{SPS} = \frac{\text{количество корректно выровненных пар}}{\text{общее количество пар в эталонном выравнивании}} \quad (12)$$

### 5.1.2 Результаты выравнивания

Результаты сравнения точности выравнивания представлены в таблице:

| Категория | BLOSUM62 | PAM250 | SNACK        |
|-----------|----------|--------|--------------|
| Ref1 (V1) | 0.814    | 0.805  | <b>0.842</b> |
| Ref1 (V2) | 0.795    | 0.783  | <b>0.821</b> |
| Ref2      | 0.752    | 0.741  | <b>0.773</b> |
| Ref3      | 0.687    | 0.679  | <b>0.703</b> |
| Ref4      | 0.731    | 0.723  | <b>0.751</b> |
| Ref5      | 0.697    | 0.685  | <b>0.729</b> |
| Среднее   | 0.746    | 0.736  | <b>0.770</b> |

Таблица 1: Сравнение точности выравнивания (SPS) для различных методов

Предложенная параметрическая метрика SNACK демонстрирует повышение точности выравнивания в среднем на 2.4-3.4% по сравнению с традиционными подходами. Наибольшее преимущество наблюдается для сложных выравниваний с низкой идентичностью последовательностей и значительными структурными особенностями.

## 5.2 Анализ производительности

Мы провели исследование производительности различных реализаций алгоритма выравнивания на последовательностях разной длины. В эксперименте сравнивались:

1. Базовая реализация алгоритма Needleman-Wunsch;
2. Оптимизированная NumPy-реализация;
3. JIT-компилированная реализация с Numba;
4. GPU-ускоренная реализация с использованием CUDA или MPS.

### 5.2.1 Результаты сравнения производительности

| Длина послед. | Базовый | NumPy  | Numba  | CUDA   | MPS    |
|---------------|---------|--------|--------|--------|--------|
| 100           | 42 мс   | 12 мс  | 5 мс   | 3 мс   | 4 мс   |
| 500           | 980 мс  | 243 мс | 63 мс  | 18 мс  | 24 мс  |
| 1000          | 4.2 с   | 0.96 с | 0.25 с | 0.06 с | 0.08 с |
| 2000          | 17.8 с  | 3.8 с  | 0.97 с | 0.22 с | 0.29 с |
| 5000          | 112 с   | 23.7 с | 6.1 с  | 1.4 с  | 1.7 с  |

Таблица 2: Время выполнения алгоритма выравнивания для последовательностей различной длины

Результаты показывают, что использование JIT-компиляции с Numba позволяет ускорить выравнивание в 16-18 раз по сравнению с базовой реализацией. Применение GPU-акселерации с CUDA обеспечивает дополнительное ускорение в 4-5 раз по сравнению с Numba и в 70-80 раз по сравнению с базовой версией.

Использование Metal Performance Shaders (MPS) на устройствах с Apple Silicon показывает немного более низкую производительность по сравнению с CUDA, но все равно обеспечивает значительное ускорение — в 60-65 раз по сравнению с базовой версией алгоритма.

## 6 Заключение

В данной работе мы представили SNACK — библиотеку для выравнивания последовательностей, основанную на параметрической метрике, обучаемой из данных. Основные результаты работы:

1. Разработана математически обоснованная параметрическая метрика на множестве аминокислот, использующая их биохимические и физические свойства. Метрика обеспечивает более точное выравнивание по сравнению с традиционными подходами.
2. Предложен подход к оптимизации параметров метрики с учетом требований выравнивания, симметричности и неравенства треугольника, что обеспечивает хорошие теоретические свойства и практическую применимость.

3. Реализованы высокопроизводительные алгоритмы для вычисления метрики и выравнивания последовательностей, с использованием многоуровневого кэширования, JIT-компиляции и аппаратного ускорения.
4. Экспериментально показано превосходство предложенного подхода над традиционными матрицами замен как по точности выравнивания (на 2.4-3.4%), так и по вычислительной эффективности.

SNACK предоставляет гибкую основу для дальнейших исследований в области выравнивания последовательностей и может быть расширен для решения других задач биоинформатики, таких как множественное выравнивание, филогенетический анализ и структурное предсказание.