DNA解析における集団層別化のための主成分分析(PCA)の応用

Namir Garib

金沢大学　理工学域　電子情報通信学類

January 28th, 2025

はじめに

- PCA（主成分分析）は、大規模データセットの次元削減に広く使用される手法である。
- ゲノム研究において、PCAは遺伝的多様性や集団構造を明らかにするために用いられる。
- 個体間の遺伝的な類似性や差異を視覚化し、祖先集団の推定に寄与する。
- また、PCAは遺伝子変異のパターンを解析し、特定の遺伝子変異が疾病と関連しているかどうかを検出するために使用可能である。
- 特に、遺伝的疾患を引き起こす可能性のある突然変異を特定し、医学的応用における貢献が期待される。

主要用語

主成分分析 (PCA)

データセットの変数の数を減らしながら、なるべく多くの情報を保持する統計的手法の一つ

Reference genome (REF)

ある種の理想的な生物個体における遺伝子セットの代表例として科学者によって構築されたデジタル核酸配列データベース

Single Nucleotide Polymorphism (SNP)

日本語：一塩基多型。一塩基が参照配列（REF)と異なる場合に起こるDNAの変異の一種である

Variant calling (VC)

参照ゲノムと比較することによって個体ゲノムの差異を識別するプロセス

集団層別化

亜集団間の対立遺伝子頻度に系統的な差があることを指す

```
IND3    AACTGTCCCACGTAACGAAGCATCGATCGAA
IND2    AACTGTCCCACCTAACGATGCATCGATCGAA
IND1    AACTGTCCCACCTAACGATGCATCGATCGAA
REF     AACTGTCCCACGTAACGATGCATCGATCGAA
```

Figure 1. Example of Single Nucleotide Polymorphism (SNP)

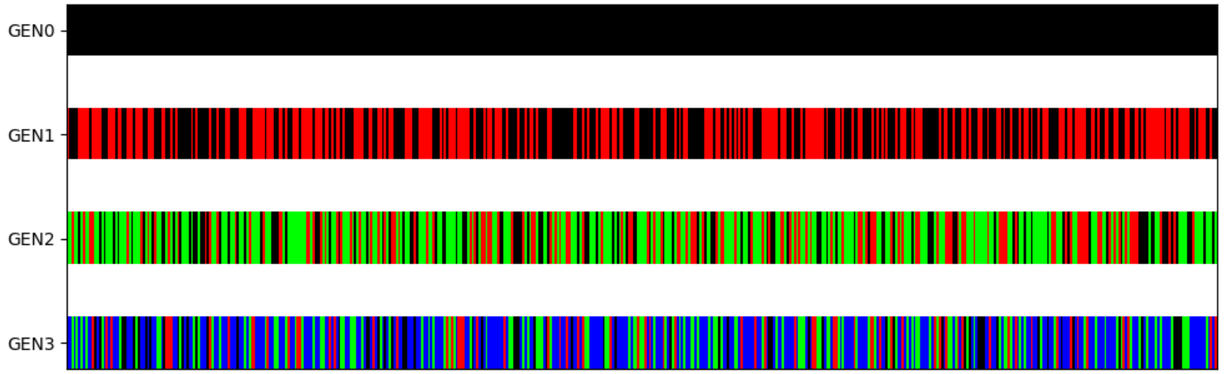


Figure 2. Random distribution of allele frequency over four generations

研究概要

目的

- CおよびRustで効果的なバリアントコーリングとPCAアルゴリズムをゼロから実装
- パフォーマンスの比較
- 既存ツール（GATK VCF, EIGENSTRAT）との性能比較

手法

- 自作アルゴリズムの設計と実装
- ベンチマークテストによる性能評価
- 実際のデータセットを使用した分析

簡略化のための前提

- 入力はゲノム配列のみで、メタデータ（ベースクオリティ、カバレッジ等）は含まない
- 挿入や闕失（indels）などの非SNP変異は対象外

Variant Calling

バリアント・パラメータ

```
.transition_weight   = 0.28 // 転移
.transversion_weight = 1.10 // 転換
.cpg_multiplier      = 1.80 // CpG部位
.cluster_factor      = 0.12 // クラスタリング補正
.logistic_scale      = 0.60 // ロジスティック補正
```

- **転移**の発生頻度が高いため、低い値を割り当てられる
- 転移に比べて**転換**の発生頻度が低く、より大きな影響を及ぼすため、高い値を割り当てられる
- **CpG部位**における変異率の上昇を補正する係数。
- SNPが局所的に発生する可能性を考慮した係数。ゲノム領域における変異の空間的な分布を調整するために、**クラスタリング補正係数**を設置する
- **ロジスティック係数**は確率分布を調整するスケーリング係数

```
IND3    ACGTAACGAAGCAT
IND2    ACCTAACGATGCAT
IND1    ACCCAACGATGCAT
REF     ACGTAACGATGCAT
```

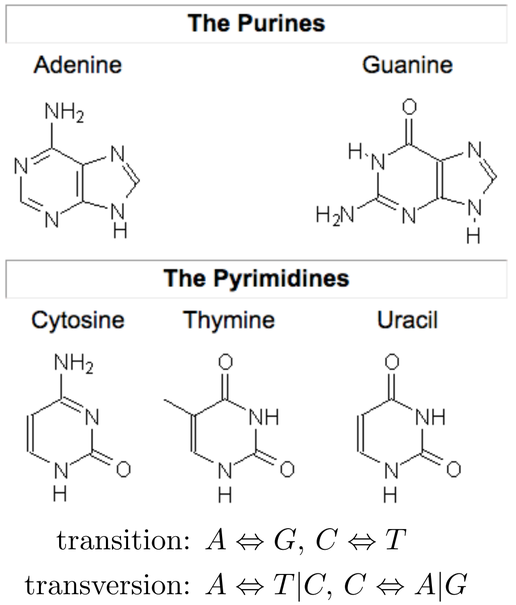
3個分 x 14ベース

```
0 0     0     0 0 0 0 0 0 1.10 0 0 0 0

0 0  1.10     0 0 0 0 0 0    0 0 0 0 0

0 0 0.238 0.132 0 0 0 0 0    0 0 0 0 0
```

3 x 14 行列を構成



PCAの数学的定式化

1. データの正規化

$$\mathbf{X_N} = \frac{\mathbf{X} - \mu}{\sigma}$$

2. データの中心化

$$\dot{\mathbf{X}}_\mathbf{N} = \mathbf{X_N} - \mu$$

3. 共分散行列の計算

$$\mathbf{C} = \frac{\dot{\mathbf{X}}_\mathbf{N}^\top \dot{\mathbf{X}}_\mathbf{N}}{n-1}$$

4. 固有値分解

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}$$

5. 主成分への射影

$$\mathbf{T} = \dot{\mathbf{X}}_\mathbf{N}\mathbf{W}$$

第一主成分の算出

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)}$$

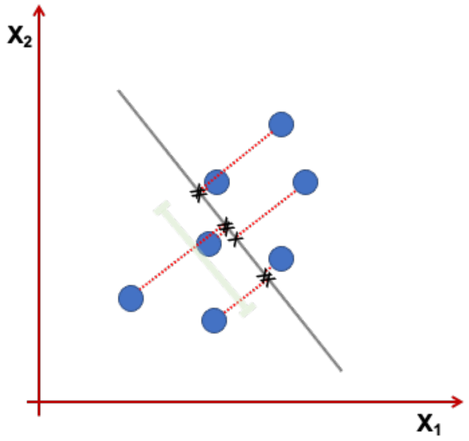
$$\text{for } i = 1, ..., n \text{ and } k = 1, ..., l$$

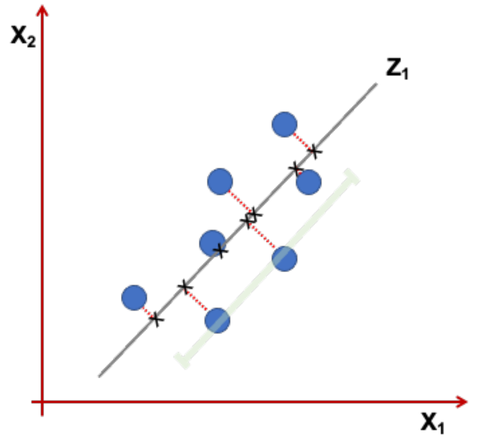
$$\mathbf{T} = \mathbf{X}\mathbf{W}$$

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(t_{1(i)}\right)^2 \right\} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w}\right)^2 \right\}.$$

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \|\mathbf{X}\mathbf{w}\|^2 \right\} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} \right\}.$$

$$\mathbf{w}_{(1)} = \arg\max \left\{ \frac{\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}}{\mathbf{w}^\top \mathbf{w}} \right\}$$

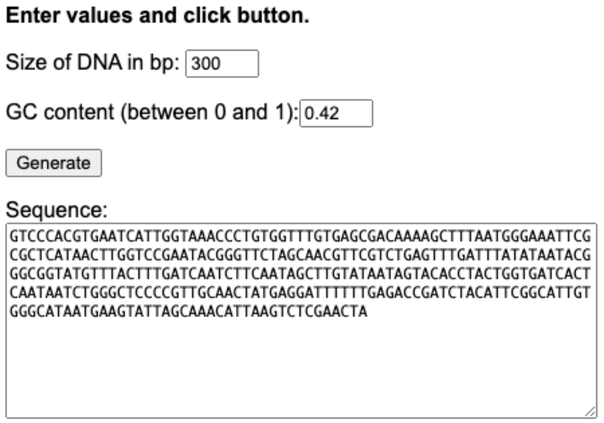




データセットの生成

カリフォルニア大学医学部の無料ソフトウェアを用いて擬似ヒトゲノムを生成した

RANDOM DNA SEQUENCE GENERATOR



ベースペアの数、GC含量を指定

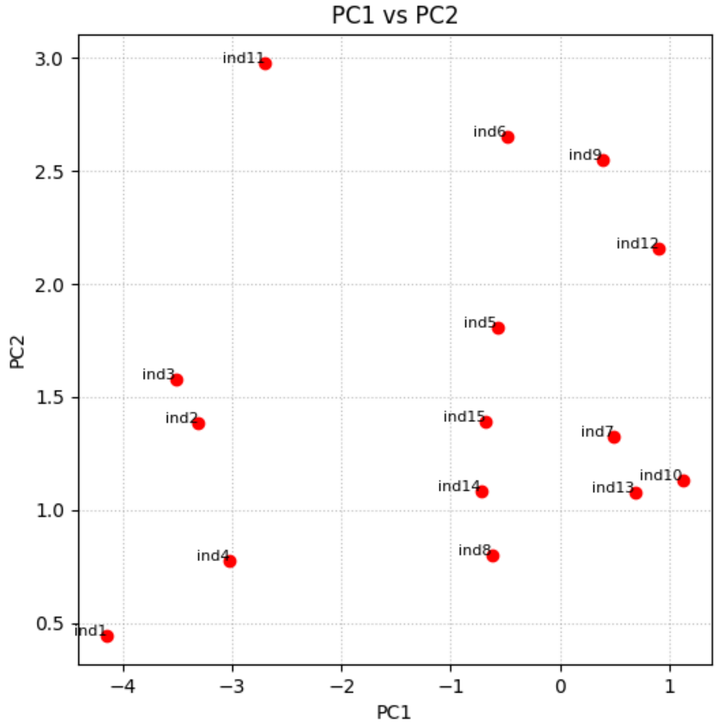
生成されたゲノムをリファレンスゲノム（ref.txt）とし、SNPをsnp.cで導入することで、10人分のデータセットを用意する。

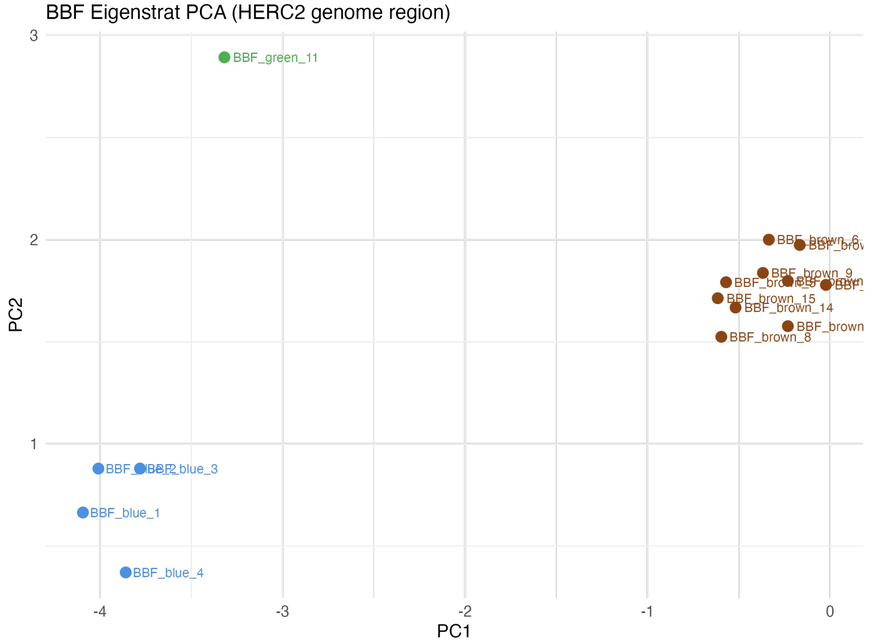
ベース数30、300、3000、30,000、…、30億のデータセットを生成。

実験１

第一実験：プログラム実装の検証

主成分分析（PCA）アルゴリズムの実装の妥当性を検証するために、ハーバード大学のDavid Reich研究室によって開発されたEIGENSTRATソフトウェアとの比較分析を行った。比較を簡略化するために、HERC2遺伝子領域に限定して解析を実施。HERC2遺伝子はメラニン生成に関与するタンパク質をコードしており、眼の色に影響する。データセットは1000 Genomes Projectからダウンロードした15の標本で構成。HERC2遺伝子領域のみを抽出するためにプルーニングを行いました。





実験②

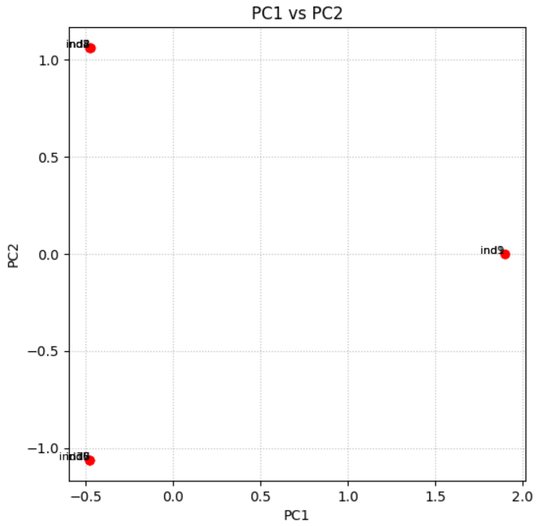
第二実験：大規模データ処理実験

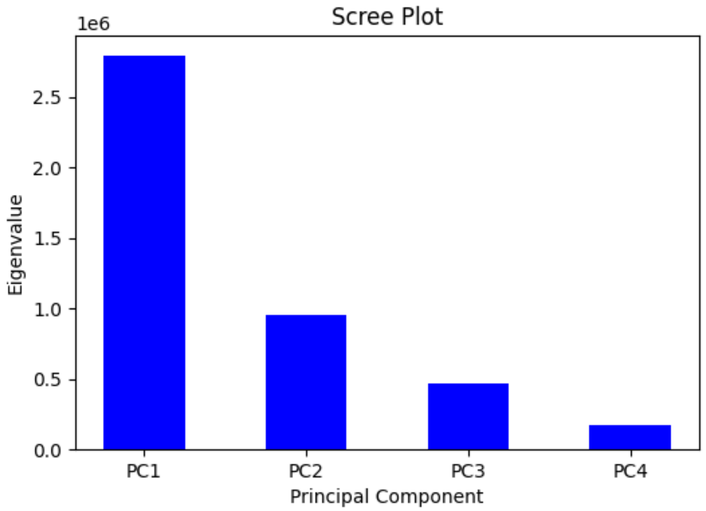
１）ベース数を3000万、3億、30億のデータセットを入力とし、プログラムを実行

２）`CHUNK_SIZE = 1000000` に設定して1MB単位で個体ごとにデータを読み込み、8MBを超えるとスタックオーバーフローが発生した。→ mallocを用いてチャンクをヒープに読み込む。

３）短い配列では問題なく動作したが、30M塩基対の長い配列を扱う際、30M × 30Mの共分散行列を生成しようとしたため、約7.8ペタバイトのメモリが必要になり、処理が不可能であった。→ 行列の冗長性を減らし、スパースモデリングに切り替え

４）計算した主成分の値はあまりにも高いことがあった→ 各主成分の次元ごとにZスコア正規化（平均を０、標準偏差を１）を実施した。





SNPの差異が極めて少ない場合、あるいは個体がほぼ同一である場合、PCAの結果にオーバーラップが生じることがある　（左の図は $\mathrm{n} = 30{,}000{,}000$）。

CとRustの実装比較

- C言語においてはstringというデータ型はない。`char*` あるいはread-onlyの場合、`const char*`でstringを表すのは一般的。そのため、文字列の最後にヌル終端文字をつける必要がある。

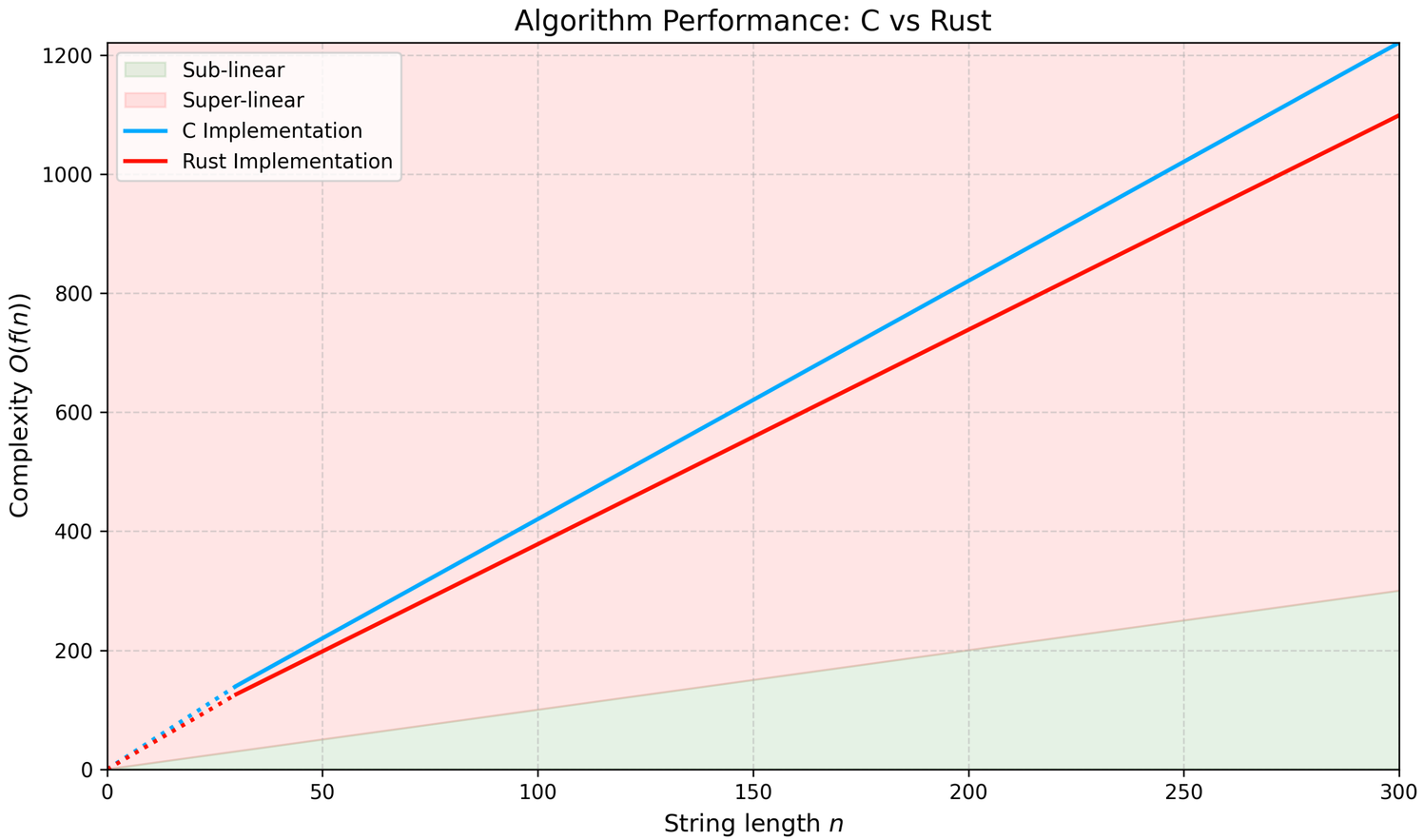
`strlen()` ヌル終端文字の確認を行うため、計算量は $O(\mathrm{n})$

- 一方、Rustの &str データ型は文字列の長さをメタデータとして格納するため、`.len()` の計算量は $O(1)$
- &strは文字列の読み取り専用のデータ型であるため、大量のデータ処理に最適。

Table 1. Differences between C and Rust

操作内容	C (`const char*`)	Rust (`&str`)
文字列の長さ	O(n) (\0を探すために線形探索)	O(1) (長さが事前に格納されている)
メモリの安全性	バファーオーバーフローのリスクあり	境界確認が行われるので安全
部分文字列の抽出	コピーが必要 (strncpyを使用)	コストゼロのスライス(`&s[start..end]`)

パフォーマンス評価



計算量の算出:

d: 個体数 (10)、n: 塩基数、v: 非ゼロのバリアント数

k: 主成分の個数 (4)、T: 主成分当たりの冪乗反復回数(20)

1) Variant calling

$$O(d \cdot n) + O(n) = O((d+1) \cdot n) = O(11n)$$

2) Partial PCA on Sparse Data

2.1 行列とベクトルの掛け算 (スパース)

$$O(v)$$

2.2 転置行列との掛け算, ベクトル正規化

$$O(k \cdot T \cdot v) + O(k \cdot T \cdot n)$$

上記を合わせて:

$$O(d \cdot n) + O(k \cdot T(v+n)) = O(11n) + O(80(v+n))$$

結論

- PCAはゲノム解析において強力な手法であるが、DNA全体の包括的な解析には制約が存在する。
- 高次元データセットにおける詳細な遺伝的相関を完全に捉えることは困難。
- 非線形な相関関係や微細な遺伝的差異はPCAで捉えきれない場合がある。
- C言語とRustの性能差は僅かであるが、以下の理由でRustが若干優れている：
  - Rustはメモリ管理が安全で効率的であり、スタックオーバーフローやメモリリークを防ぐ。
  - 同時に、計算コストを削減し、より高速な処理を可能にする。

今後の改良点：

- 非線形次元削減手法（例：t-SNEやUMAP）の導入。
- 分散コンピューティングを活用したスケーラビリティの向上。
- スパース表現のさらなる最適化によるメモリ使用量の削減。