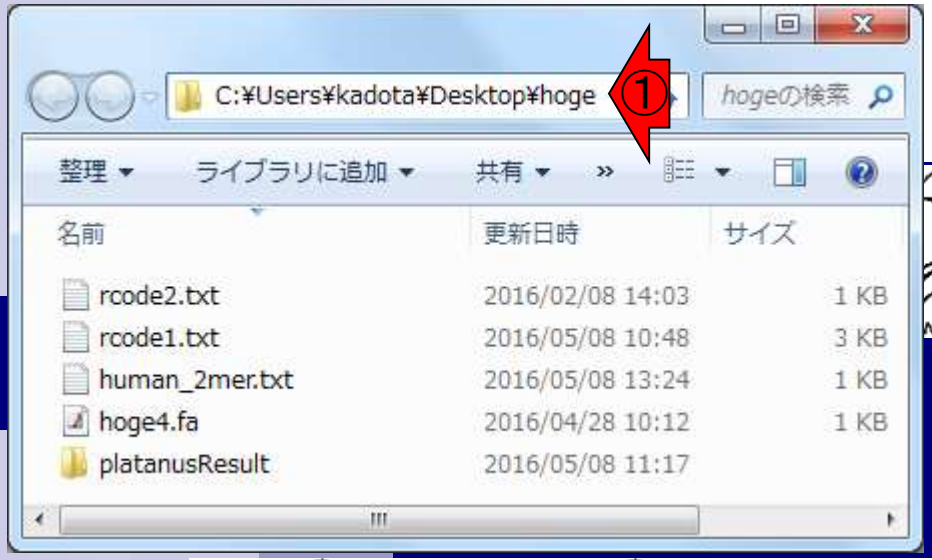
2016.07.12版

①hogeフォルダがあり、そこに解析用入力ファイルが存在するという前提で行います。
印刷は2016.06.23版です。若干見栄えや文言を微修正しました

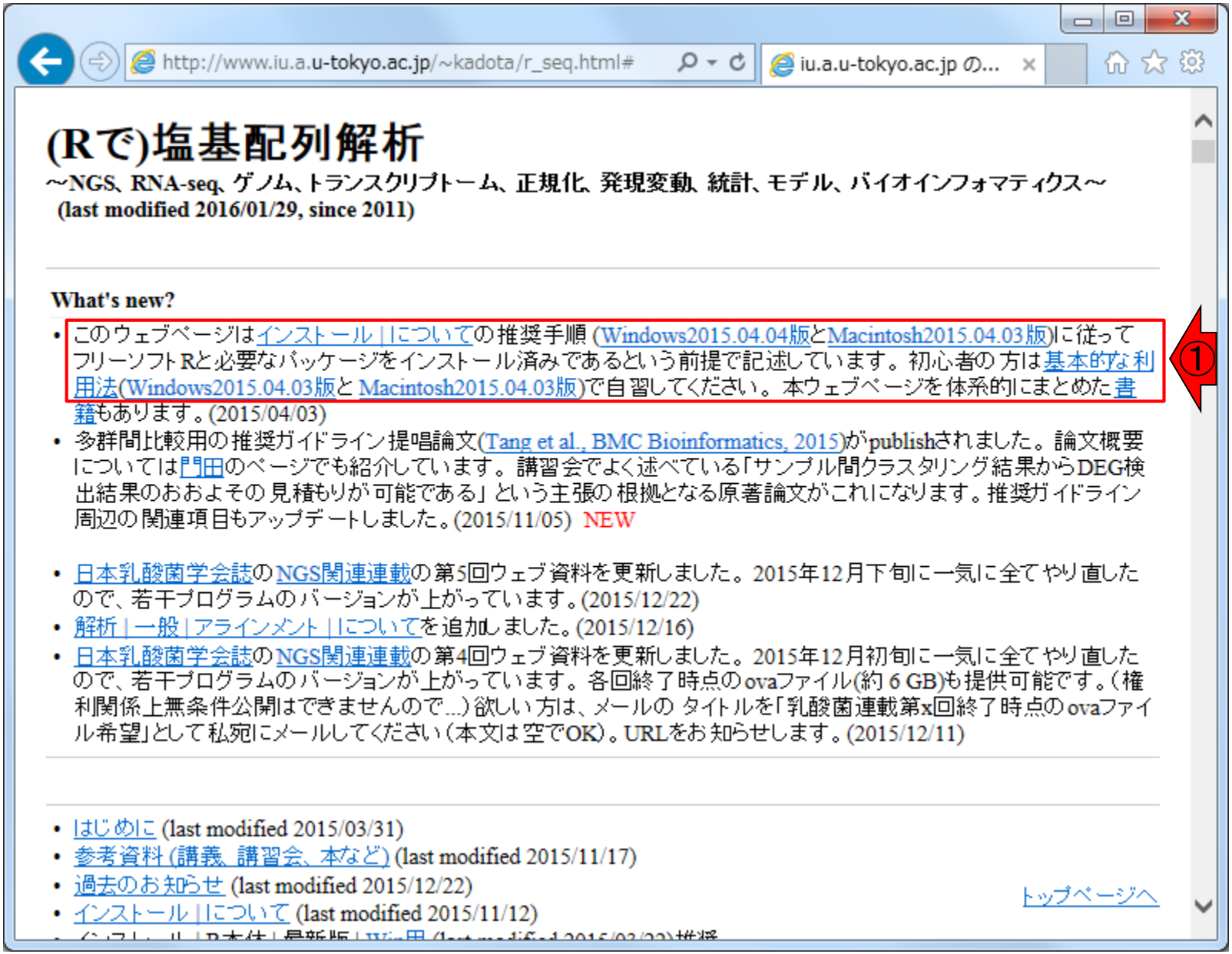


第1部：統計解析
～ゲノム解析、塩基配列解析～

東京大学・大学院農学生命科学研究科
アグリバイオインフォマティクス教育研究プログラム
門田幸二（かどた　こうじ）
kadota@iu.a.u-tokyo.ac.jp
http://www.iu.a.u-tokyo.ac.jp/~kadota/

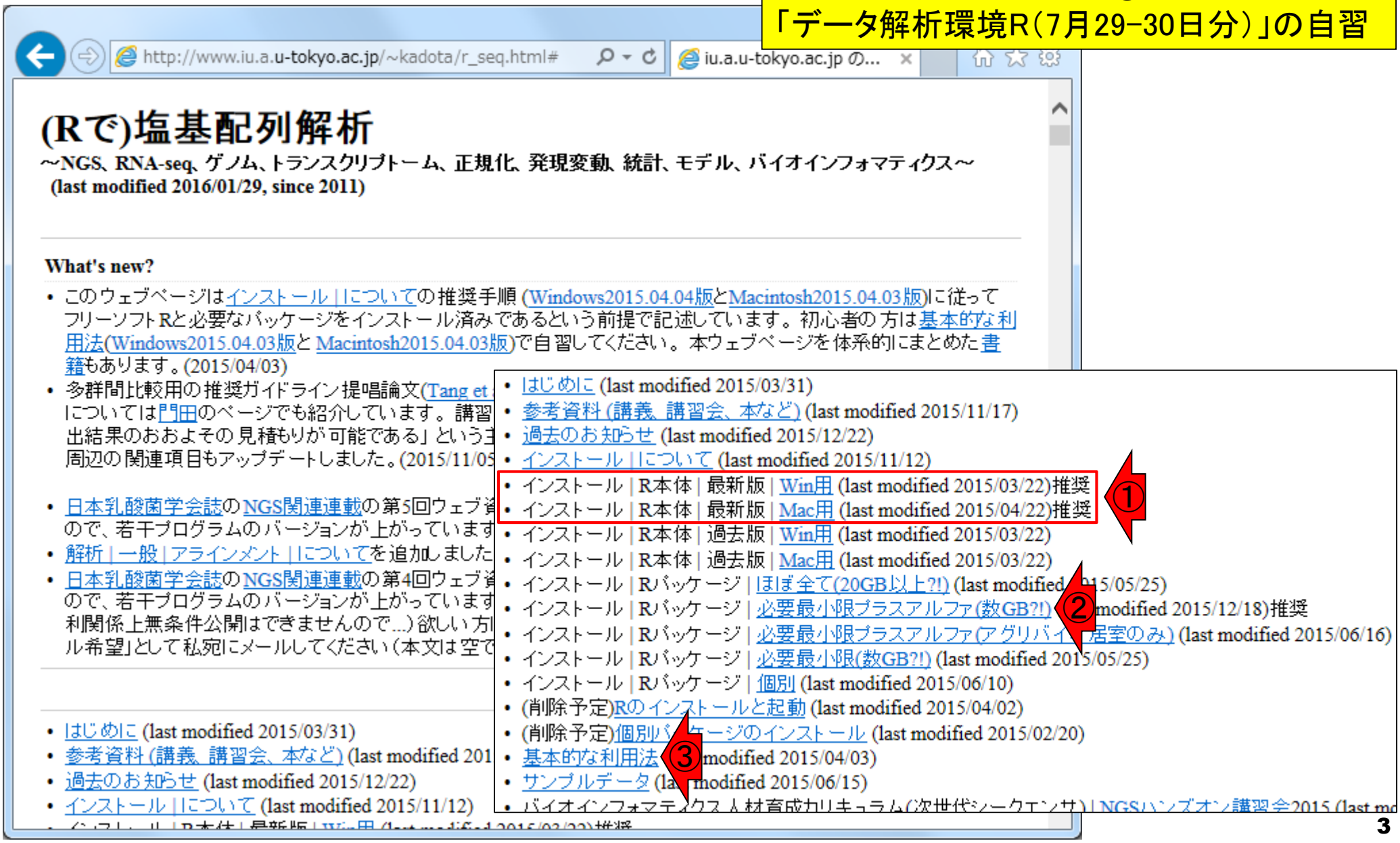
講習会のPC環境

実習用PC環境は、①の手順に従って「Rおよび必要なパッケージ」のインストールが完了している状態



予習事項の再確認

具体的な順番は、①R本体のインストール、②各種Rパッケージのインストールです。③基本的な利用法、および④H27年度講習会「データ解析環境R（7月29-30日分）」の自習



基本スタンスは、**優先順位とエフォート**。基本独裁、一匹狼。
受益者が金をかけずに効率的に学べる教材整備が最優先。

私の主な活動

- 東大アグリバイオの大学院講義（バイオインフォ全般）
  - Rを中心としたハンズオン講義（平成16年度～）
    - 受講人数が多い（最大130名）ので、クラウド（ウェブツール）系実習は実質的に不可能
    - 講義補助員(TA)が数名のみなので、Linux系実習も困難
- NBDC/東大アグリバイオの**NGSハンズオン講義**（NGSに特化）
  - Linuxを中心としたハンズオン講義（平成26年度～）
    - 受講人数は多い（H27年度は最大71名；おそらくアグリバイオ本体に次ぐ規模）が、受講生の意識レベルが高く（きっちり予習をやるヒトが多数派）、環境構築済みノートPC数、TA数が充実しているため、本格的なLinux実習が成立しうる。
- 日本乳酸菌学会誌のNGS連載
  - Linuxを中心とした自習用教材（平成26年度～）
    - バクテリア（乳酸菌）データを、主にBio-Linux上で解析するノウハウを提供。
    - 第6回（2016年3月）分以降は、DDBJ Pipeline（ウェブツール）の利用法も紹介。
    - データ取得・インストール・実行に時間がかかるものも、自習なので時間を気にせずにできる。ハンズオン講義よりも心穏やか。
- その他
  - 研究（発現変動解析精度向上のためのアルゴリズム開発や評価）
  - …

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

NGSデータ解析戦略

自分の置かれている環境・予算・ポリシーによっても異なる。どの選択肢でも基本正解。
Rは、主に統計解析部分で使われる

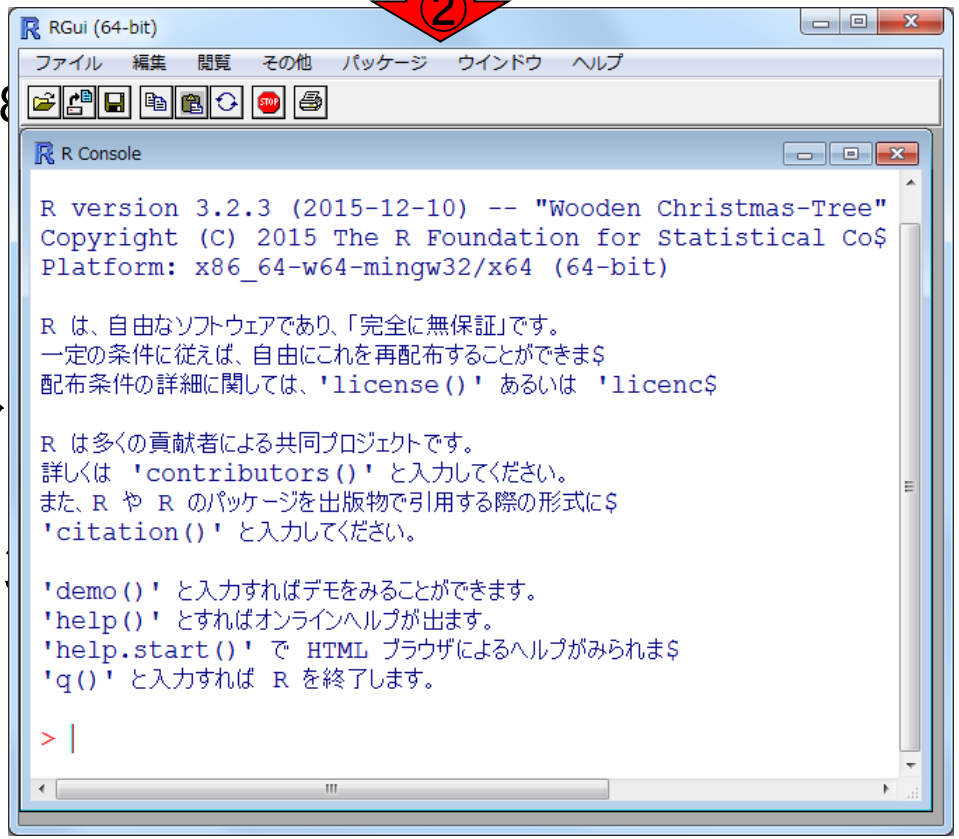
- 解析受託企業に外注：Linuxコマンドを知らなくてもよい
  - 

- クラウド（ウェブツール）：Linuxコマンドを知らなくてもよい
  - DDBJ Pipeline (Nagasaki et al., *DNA Res.*, **20**: 383–390, 2013)
  - Illumina BaseSpace
  - Galaxy (Goecks et al., *Genome Biol.*, **11**: R86, 2010)
  - …
- Linuxコマンドを駆使（旧来型）
  - なるべく自力で解析
  - LinuxコマンドやNGS解析用プログラムのインストールなどを練習し、スパコンを使いこなす
  - NBDC/東大アグリバイオの「NGSハンズオン講習会」の方向性

DDBJ PipelineとR

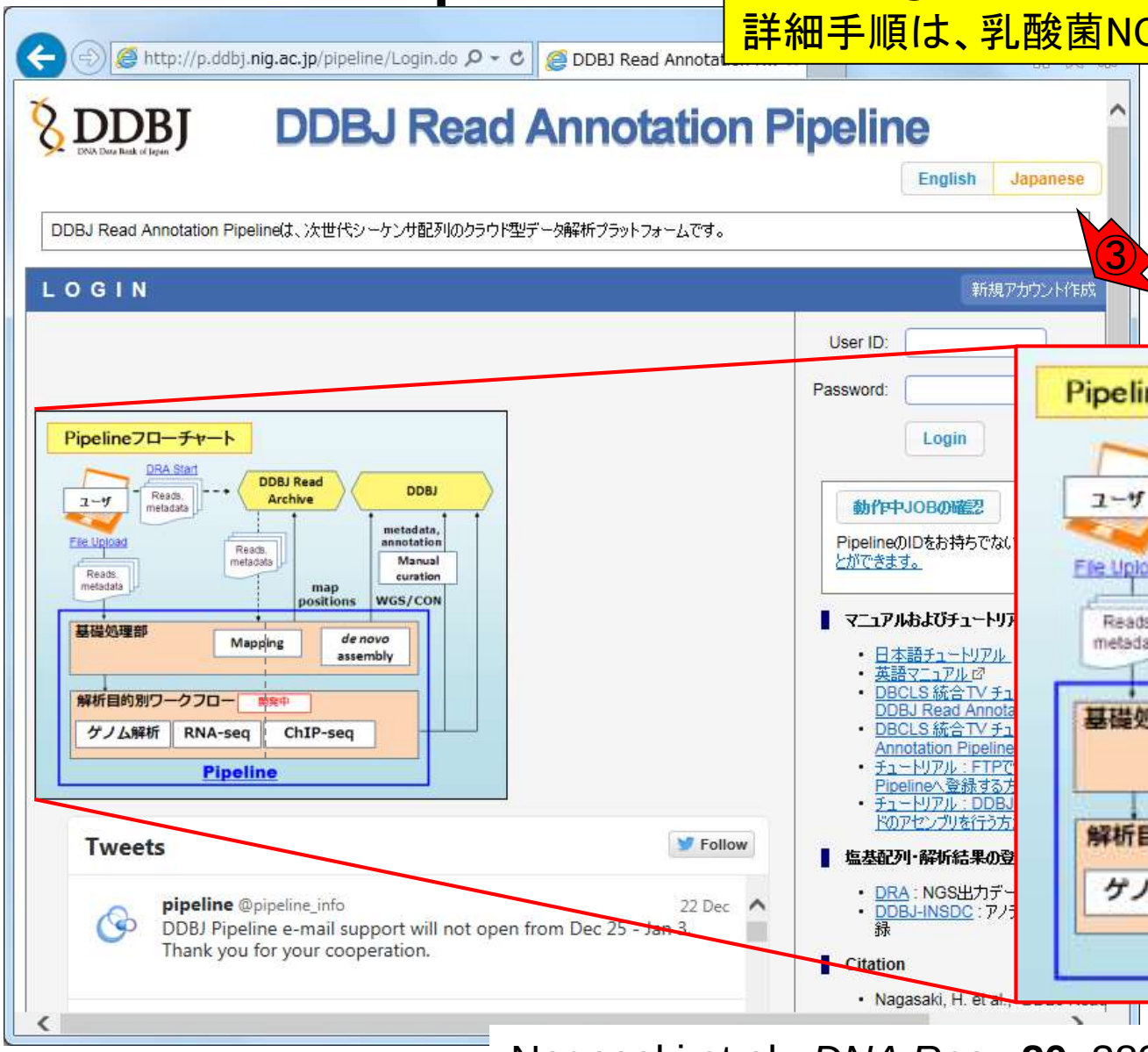
①DDBJ Pipelineだけで全てのNGS解析ができるわけではない。②Rもまた然り。特にRでは、(門田の知る限り) *de novo*アセンブリは不可能。現実を知り、うまく使い分けるべし。DDBJ Pipeline上で*de novo*アセンブリを行った結果の解釈や確認をRで行うところからスタート

- 解析受託企業に外注：Linuxコマ
  - Google ngs 受託解析
- クラウド(ウェブツール)：Linuxコマンドを知らなくてもよい
  - ① DDBJ Pipeline (Nagasaki et al., *DNA Res.*, **20**: 383–390, 2013)
  - Illumina BaseSpace
  - Galaxy (Goecks et al., *Genome Biol.*, **11**: R8
  - …
- Linuxコマンドを駆使(旧来型)
  - なるべく自力で解析
  - LinuxコマンドやNGS解析用プログラムのイ
    スパコンを使いこなす
  - NBDC/東大アグリバイオの「NGSハンズオ

② 

DDBJ Pipeline

DDBJ Pipeline（の基礎処理部）では、主に①マッピングや②*de novo*アセンブリができる。特に後者ができるのは非常に有難い。③新規アカウント作成から*de novo*アセンブリまでの詳細手順は、乳酸菌NGS連載第6回ウェブ資料を参照



Nagasaki et al., *DNA Res.*, **20**: 383-390, 2013

NGSデータ

乳酸菌(*Lactobacillus hokkaidonensis* LOOC260^T) ゲノム解読論文(PMID: 25879859)。Illumina MiSeqデータ(DRR24501)とPacBioデータ(DRR024500*)を併用することでcomplete genomeを得ることができた、という内容。講習会では、MiSeqデータをDDBJ Pipelineで*de novo*アセンブリした結果をRで解析し、アセンブリ過程の理解を深める。これは*k*-mer解析のイントロでもある

http://www.ncbi.nlm.nih.gov/pubmed/25879859

NCBI Resources How To

PubMed.gov
US National Library of Medicine
National Institutes of Health

PubMed

Advanced Help

Abstract Send to:

BMC Genomics. 2015 Mar 25;16:240. doi: 10.1186/s12864-015-1435-2.

Complete genome sequence and analysis of Lactobacillus hokkaidonensis LOOC260(T), a psychrotrophic lactic acid bacterium isolated from silage.

Tanizawa Y[1,2], Tohno M[3], Kaminuma E[4], Nakamura Y[5], Arita M[6,7].

Author information

Abstract

BACKGROUND: Lactobacillus hokkaidonensis is an obligate heterofermentative lactic acid bacterium, which is isolated from Timothy grass silage in Hokkaido, a subarctic region of Japan. This bacterium is expected to be useful as a silage starter culture in cold regions because of its remarkable psychrotolerance; it can grow at temperatures as low as 4°C. To elucidate its genetic background, particularly in relation to the source of psychrotolerance, we constructed the complete genome sequence of L. hokkaidonensis LOOC260(T) using PacBio single-molecule real-time sequencing technology.

RESULTS: The genome of LOOC260(T) comprises one circular chromosome (2.28 Mbp) and two circular plasmids: pLOOC260-1 (81.6 kbp) and pLOOC260-2 (41.0 kbp). We identified diverse mobile genetic elements, such as prophages, integrated and conjugative elements, and conjugative plasmids, which may reflect adaptation to plant-associated niches. Comparative genome analysis also detected unique genomic features, such as genes involved in pentose assimilation and NADPH generation.

CONCLUSIONS: This is the first complete genome in the L. vaccinostercus group, which is poorly characterized, so the genomic information obtained in this study provides insight into the genetics and evolution of this group. We also found several factors that may contribute to the ability of L. hokkaidonensis to grow at cold temperatures. The results of this study will facilitate further investigation for the cold-tolerance mechanism of L. hokkaidonensis.

PMID: 25879859 [PubMed - in process] PMCID: PMC4377027 Free PMC Article

Full text links

Read free full text at BioMed Central

PMC FREE Full text

Save items

Add to Favorites

Similar articles

Lactobacillus hokkaidonensis sp. no [Int J Syst Evol Microbiol. 2013]

Insights into the completely annotated ger [J Biotechnol. 2012]

Lactobacillus wasatchensis sp. no [Int J Syst Evol Microbiol. 2015]

Review Genomic organization of [Antonie Van Leeuwenhoek. 1996]

Review Low-redundancy [Antonie Van Leeuwenhoek. 1999]

See reviews...

See all...

Related information

Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

参考

NGSデータ

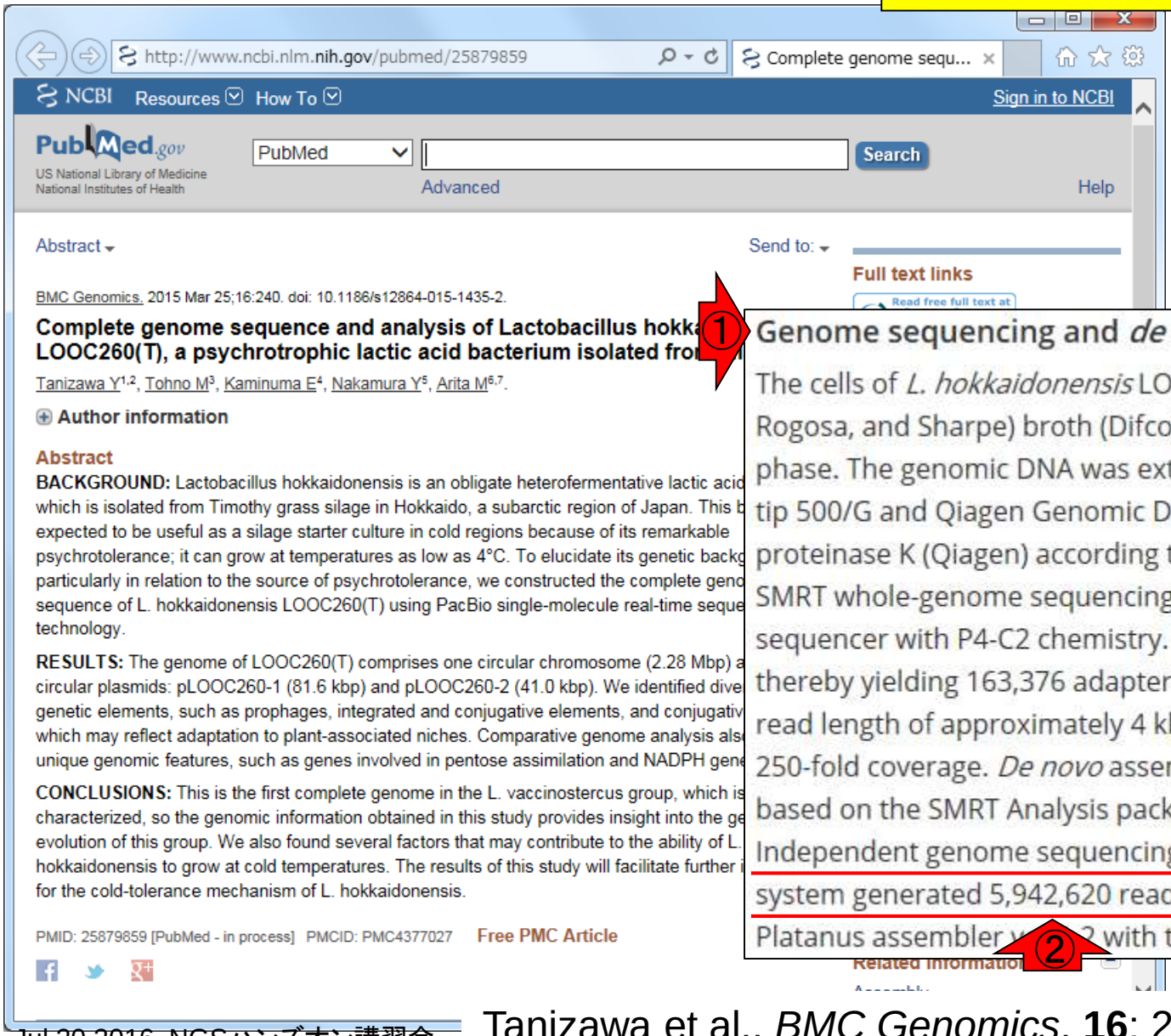
①Full textリンク先で全文を見られる。②Availability of supporting dataという項目をよく眺めると、生データがDDBJ Sequence Read Archive (DDBJ SRA; 略してDRA)にDRR024500とDRR24501というIDで登録されていることがわかる。尚、DRR024500は登録内容の問題が判明し、2016年1月末に削除されDRR054113-054116に差し替えられている

Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

参考

NGSデータ

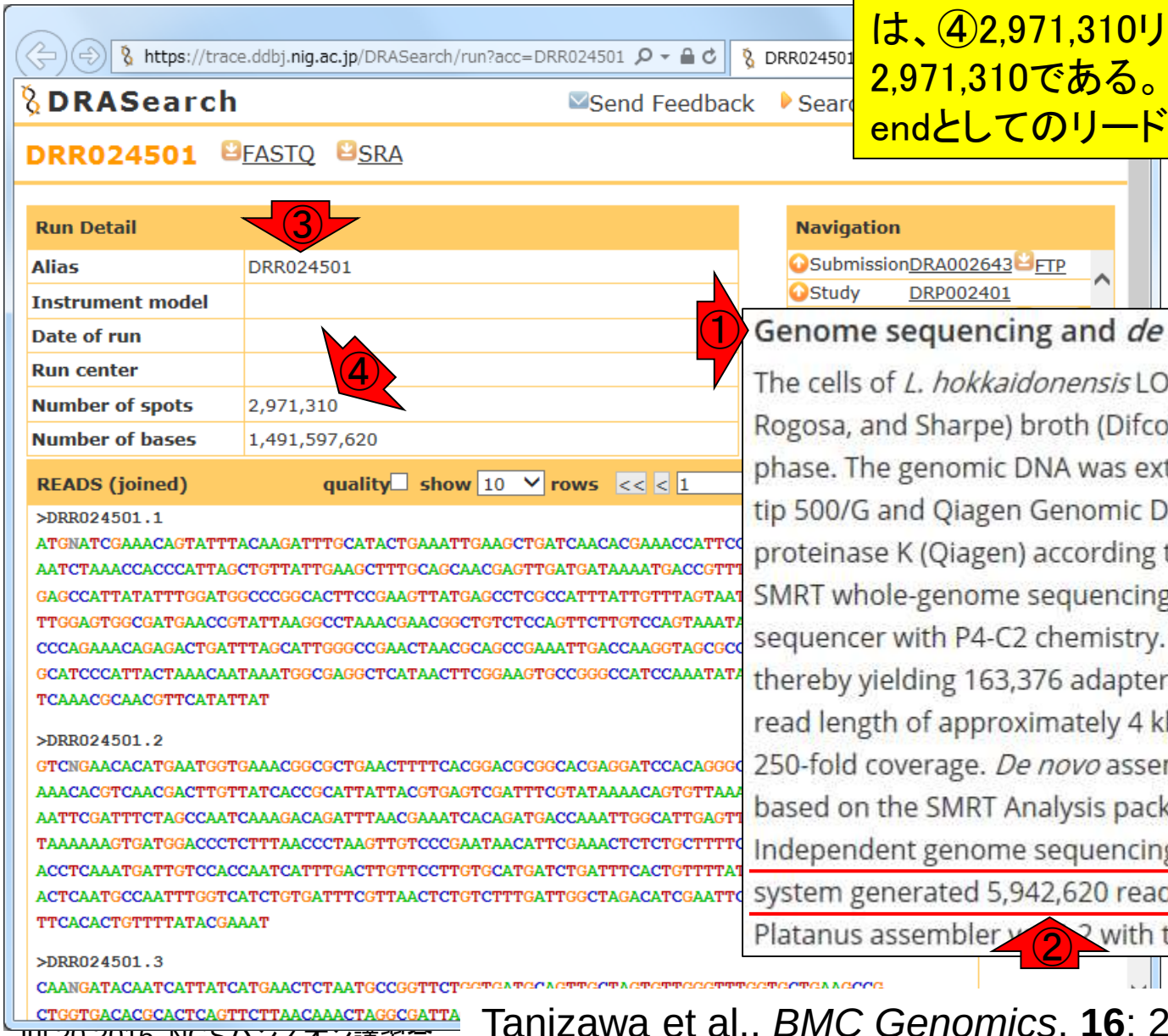
①Genome sequencing and *de novo* assemblyという項目を見ると、②paired-endで5,942,620リードと書いてある。一応公共DB（DRA）上で確認する



Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

NGSデータ

①Genome sequencing and *de novo* assemblyという項目を見ると、②paired-endで5,942,620リードと書いてある。③MiSeqデータに相当するDRR024501は、④2,971,310リードと書いてある。5,942,620/2 = 2,971,310である。ウェブサイト上の数値は、single-endとしてのリード数と考えれば妥当



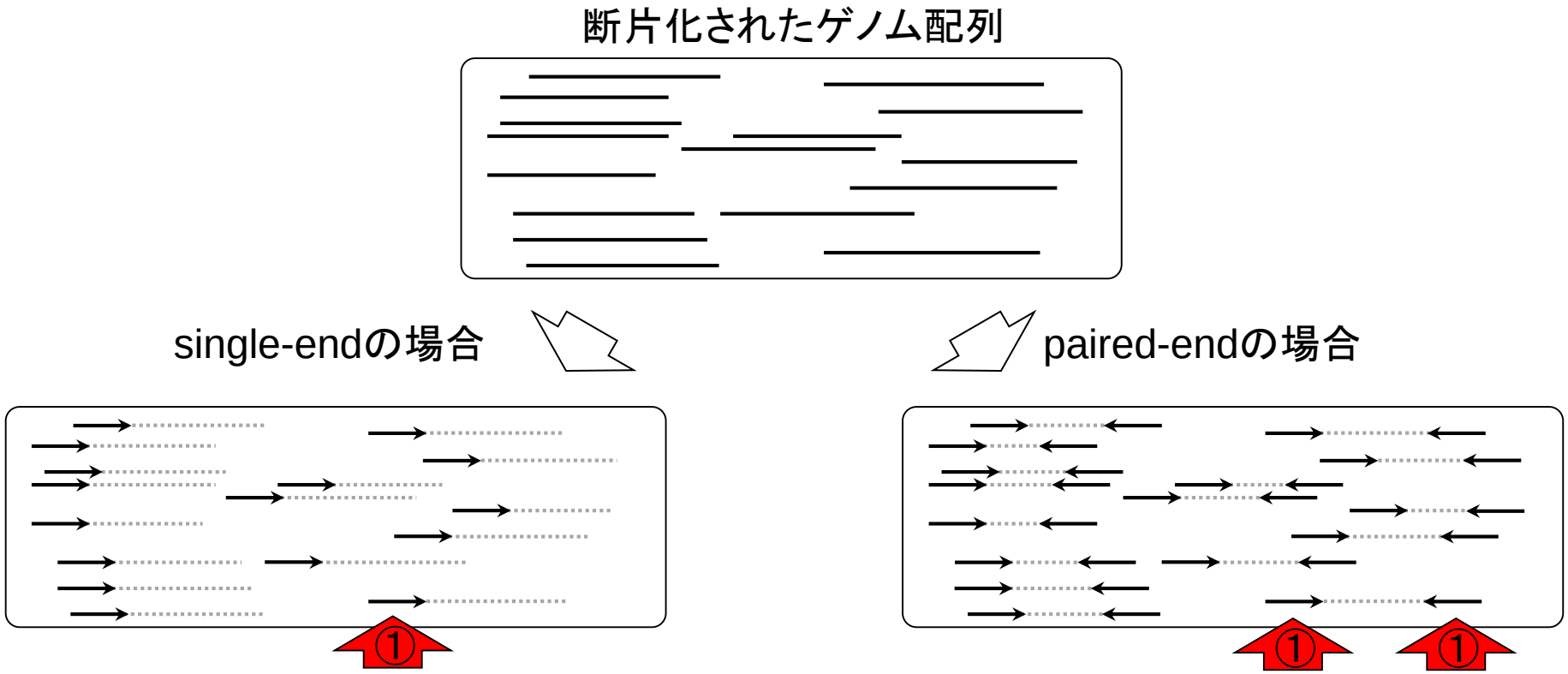
Genome sequencing and *de novo* assembly

The cells of *L. hokkaidonensis* LOOC260^T were cultured in MRS (de Man, Rogosa, and Sharpe) broth (Difco) and were harvested in the mid-logarithmic phase. The genomic DNA was extracted and purified using Qiagen Genomic-tip 500/G and Qiagen Genomic DNA Buffer Set with lysozyme (Sigma) and proteinase K (Qiagen) according to the manufacturer's instruction. PacBio SMRT whole-genome sequencing was performed using a PacBio RSII sequencer with P4-C2 chemistry. Four SMRT cells were used for sequencing, thereby yielding 163,376 adapter-trimmed reads (subreads) with an average read length of approximately 4 kbp, which corresponded to approximately 250-fold coverage. *De novo* assembly was conducted using the HGAP method based on the SMRT Analysis package 2.0, which yielded seven contigs. Independent genome sequencing using the 250-bp paired-end Illumina MiSeq system generated 5,942,620 reads, which were assembled into contigs using Platanus assembler v. 2 with the default settings [40]. The initial contigs

Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

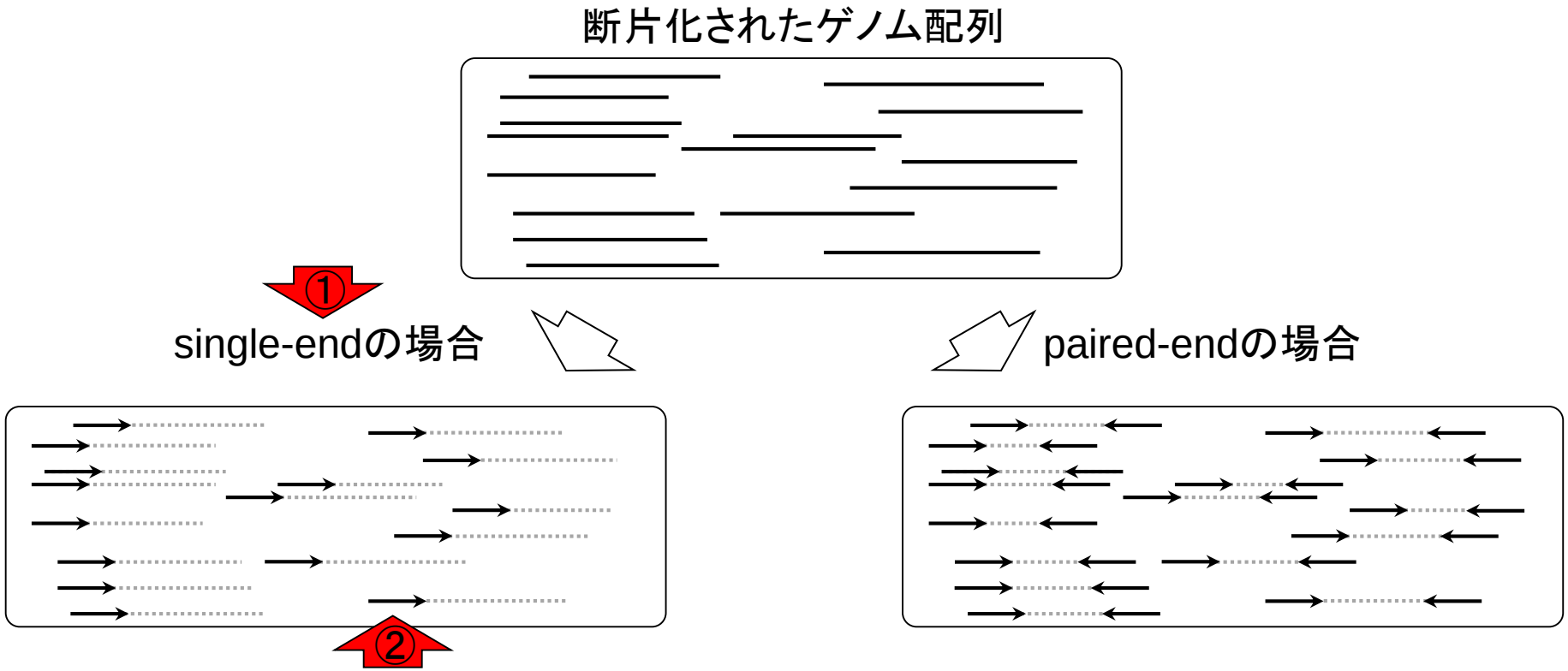
用語：リード

リードとは、Sequencerで読んだ塩基配列のこと。①黒矢印の一本一本がリードに相当する



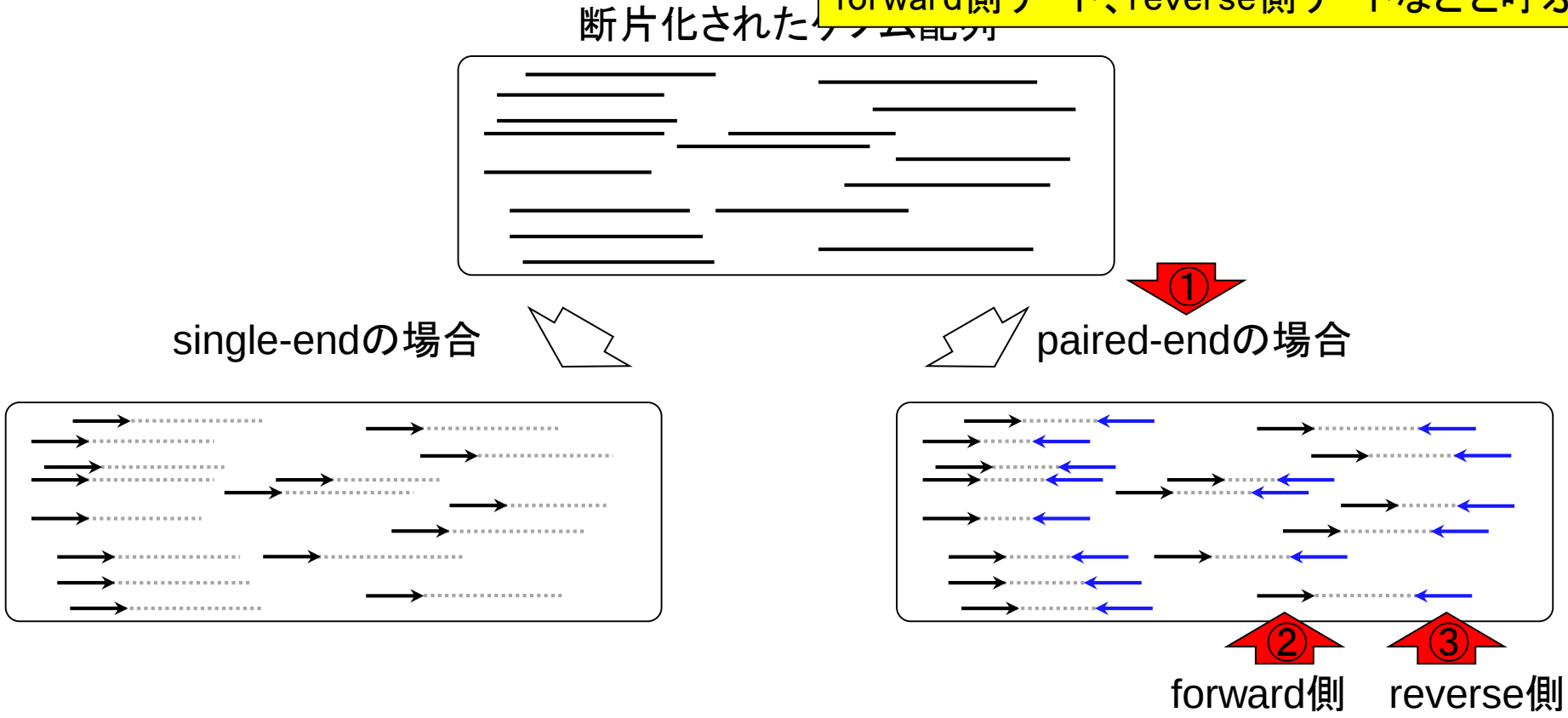
用語：single-end

①断片化された配列の片側のみを読む場合をsingle-endという。この場合は右向き矢印のみ



用語：paired-end

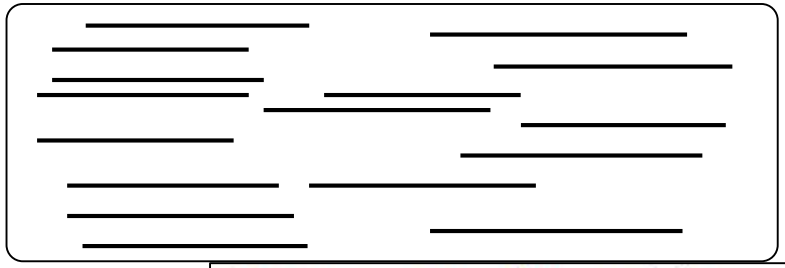
①断片化された配列の両側から読む場合をpaired-endという。②右向き矢印と③左向き矢印のリードが読まれることになる。それぞれをforward側リード、reverse側リードなどと呼ぶ



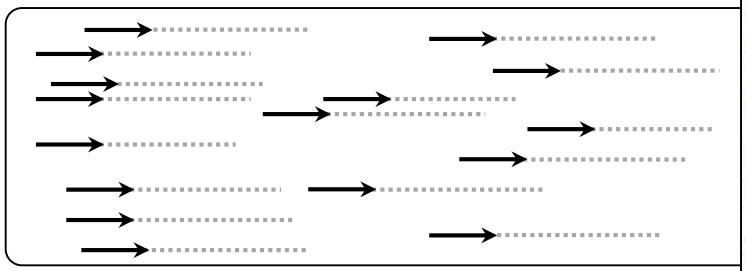
用語：paired-end

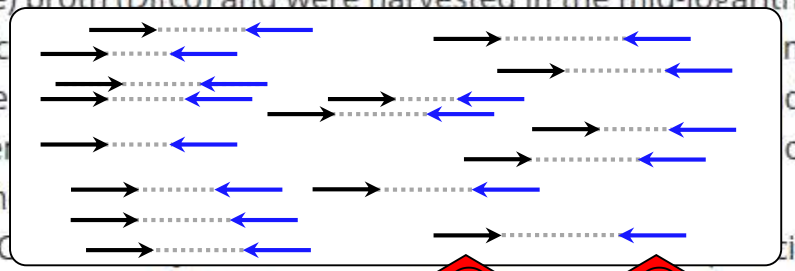
Illumina MiSeqデータ(DRR24501)の場合、①forward側、②reverse側ともに、矢印の長さが250 bp、矢印の本数（リード数）が計5,942,620個（約594万；片側のみで約297万）に相当

断片化されたゲノム配列



single-endの場合





Genome sequencing and *de novo* assembly

The cells of *L. hokkaidonensis* LOOC260^T were cultured in MRS (de Man, Rogosa, and Sharpe) broth (Difco) and were harvested in the mid-logarithmic phase. The genomic mic-
tip 500/G and Qiage d
proteinase K (Qiage o
SMRT whole-genom
sequencer with P4-C ting,
thereby yielding 163,376 adapter-trimmed reads reads) with an average read length of approximately 4 kbp, which corresponded to approximately 250-fold coverage. *De novo* assembly was conducted using the HGAP method based on the SMRT Analysis package 2.0, which yielded seven contigs. Independent genome sequencing using the 250-bp paired-end Illumina MiSeq system generated 5,942,620 reads, which were assembled into contigs using Platanus assembler ver 1.2 with the default settings [40]. The initial contigs

参考

DDBJ SRA (DRA)

DRAでIllumina MiSeqデータ(DRR024501)を概観。①Paired-endのFASTQファイルをダウンロードする場合は、②forward側と③reverse側の2つに分割されます。①をクリック

https://trace.ddbj.nig.ac.jp/DRASearch/run?acc=DRR024501

DRR024501 - DRA Search

DRASearch Send Feedback | Search Home | DRA Home

DRR024501 FASTQ SRA

Run Detail	
Alias	DRR024501
Instrument model	
Date of run	
Run center	
Number of spots	2,971,310
Number of bases	1,491,597,620

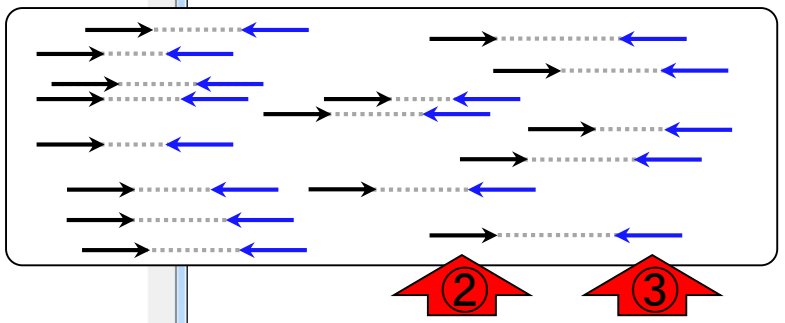
Navigation

Submission DRA002643 FTP
Study DRP002401
Experiment DRX022186 FASTQ SI

①

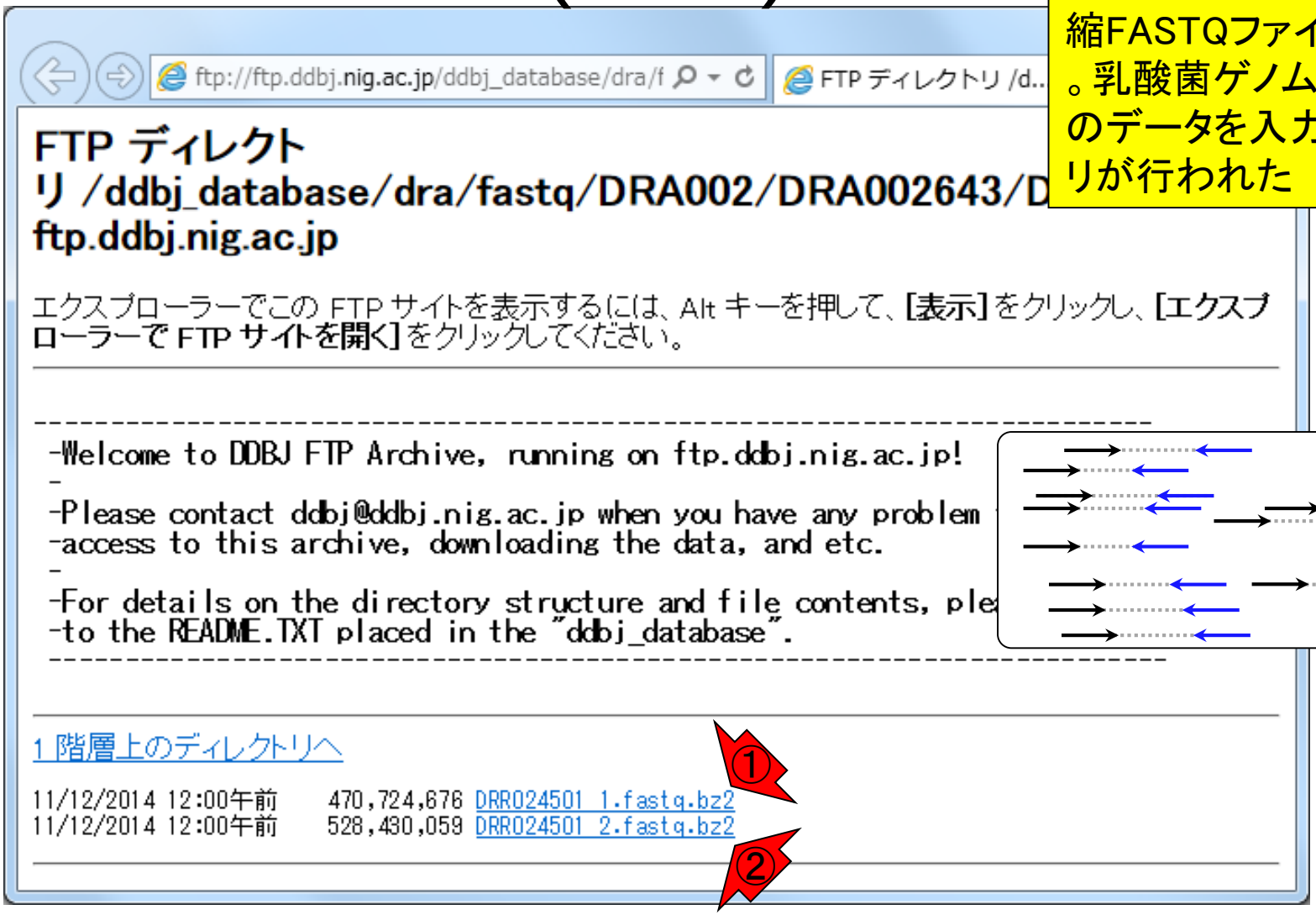
READS (joined) quality show 10 rows << < 1 / 297131 Page > >>

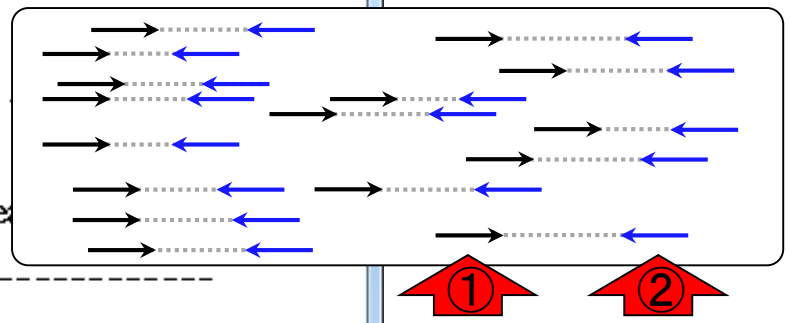
```
>DRR024501.1
ATGNATCGAAACAGTATTTACAAGATTTGCATACTGAAATTGAAGCTGATCAACACGAAACCATTCCAGCCGGCAAGGGT
AATCTAAACCACCCATTAGCTGTTATTGAAGCTTTGCAGCAACGAGTTGATGATAAAATGACCGTTTCGGTTGATGTGGG
GAGCCATTATATTTGGATGGCCCGGCACTTCCGAAGTTATGAGCCTCGCCATTTATTGTTTAGTAATGGGATGCAGACGC
TTGGAGTGGCGATGAACCGTATTAAGGCCTAAACGAACGGCTGTCTCCAGTTCTTGTCCAGTAAATAAGAATCCGGCATC
CCCAGAAACAGAGACTGATTTAGCATTGGGCCGAACTAACGCAGCCGAAATTGACCAAGGTAGCGCCACTCCAAGCGTCT
GCATCCCATTACTAAACAATAAATGGCGAGGCTCATAACTTCGGAAGTGCCGGGCCATCCAAATATAATGGCTCCCCACA
TCAAACGCAACGTTCATATTAT

>DRR024501.2
GTCNGAACACATGAATGGTGAAACGGCGCTGAACTTTTCACGGACGCGGCACGAGGATCCACAGGGCGATTATGGCCGGC
AAACACGTCAACGACTTGTTATCACCGCATTATTACGTGAGTCGATTTCGTATAAAACAGTGTTAAACACTAAGTTTTTG
AATTCGATTTCTAGCCAATCAAAGACAGATTTAACGAAATCACAGATGACCAAATTGGCATTGAGTTACCGCAATGCTGA
TAAAAAAGTGATGGACCCTCTTTAACCCTAAGTTGTCCCGAATAACATTCGAAACTCTCTGCTTTTCAGTACTTGATACC
ACCTCAAATGATTGTCCACCAATCATTTGACTTGTTCCTTGTGCATGATCTGATTTCACTGTTTTATCAGCATTTCGGTA
ACTCAATGCCAATTTGGTCATCTGTGATTTCGTTAACTCTGTCTTTGATTGGCTAGACATCGAATTCAAAAACTTAGTGT
TTCACACTGTTTTATACGAAAT

>DRR024501.3
CAANGATACAATCATTATCATGAACTCTAATGCCGGTTCTGGTGATGCAGTTGCTAGTGTTGGGTTTGGTGCTGAAGCCG
CTGGTGACACGCACTCAGTTCTTAACAAACTAGGCGATTACTTCAAACCGGAATTCTTAAACCGATTTGATGCAATTGTA
```



DDBJ SRA (DRA)

①forward側：DRR024501_1.fastq.bz2、
②reverse側：DRR024501_2.fastq.bz2、
のような感じ。DRAの場合は、bzip2圧縮FASTQファイルをダウンロード可能。乳酸菌ゲノム配列決定論文では、このデータを入力として*de novo*アセンブリが行われた





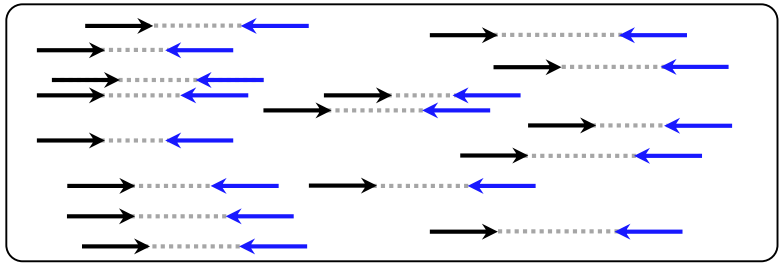
Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

参考

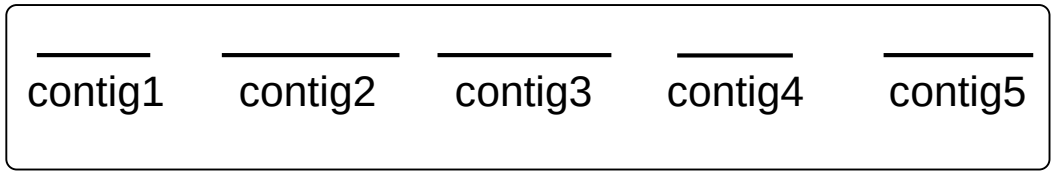
用語：コンティグ

（通常は）paired-endのリードファイルを入力として、*de novo*アセンブリプログラムを実行した結果として得られる、異なる複数のリードが（ACGTの切れ目なく）つなげられたもの。contiguous sequence（連続的な配列）という意味。通常、元のリード長よりも長くなる

入力：paired-end FASTQファイル



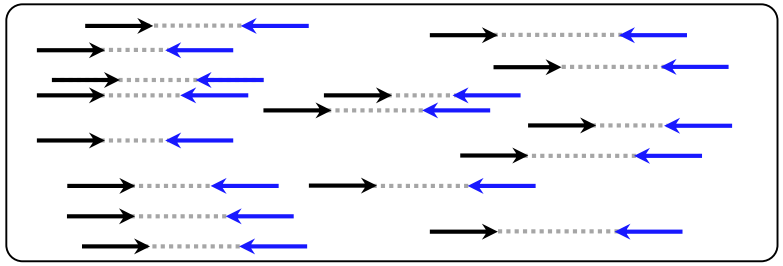
Assembly（コンティグの作成）



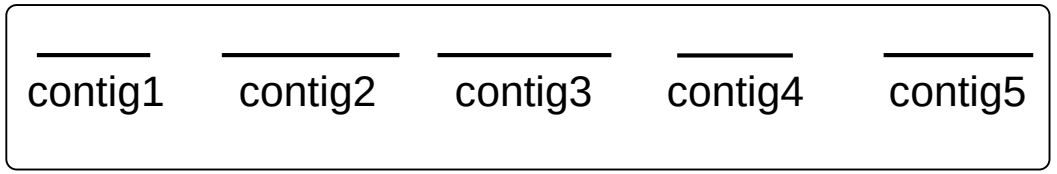
得られたコンティグにリードをマップし…

用語：scaffold

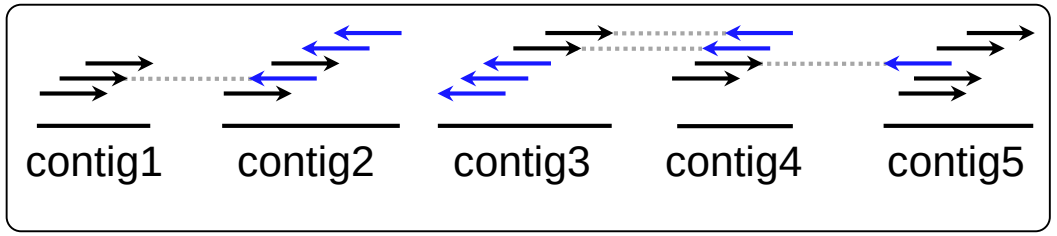
入力：paired-end FASTQファイル



Assembly（コンティグの作成）

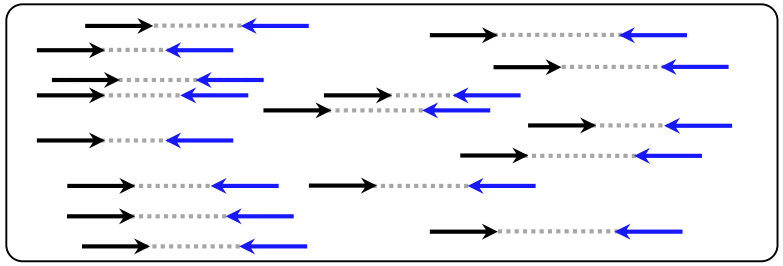


Scaffold

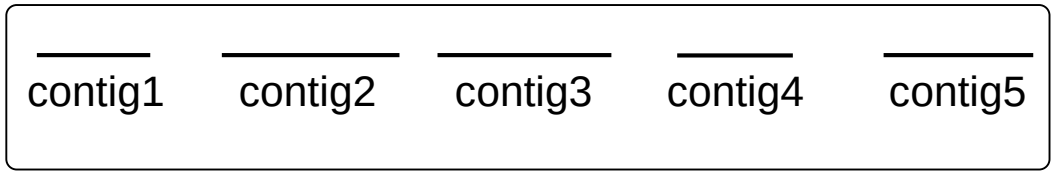


用語：scaffold

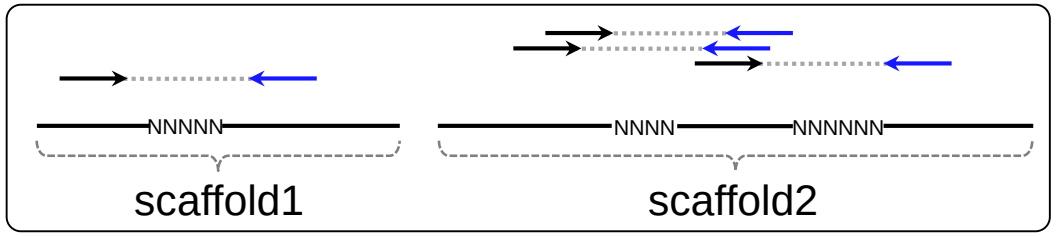
入力：paired-end FASTQファイル



Assembly（コンティグの作成）



Scaffold

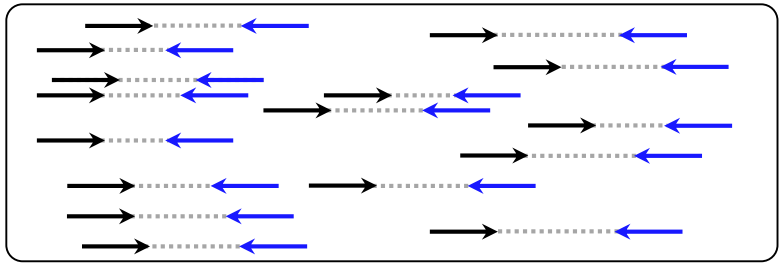


得られたコンティグにリードをマップし…ペアの情報を頼りにコンティグ間にNを入れて連結したもの。supercontigともいう。scaffoldの数はcontigの数よりも少なくなる。尚、Nを入れた部分をgapという

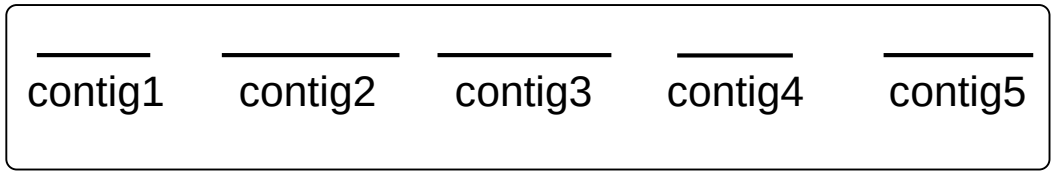
得られたscaffoldsにリードをマップし…

用語：gap close

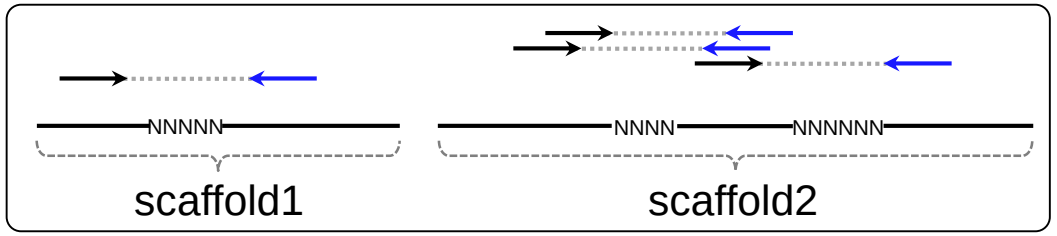
入力：paired-end FASTQファイル



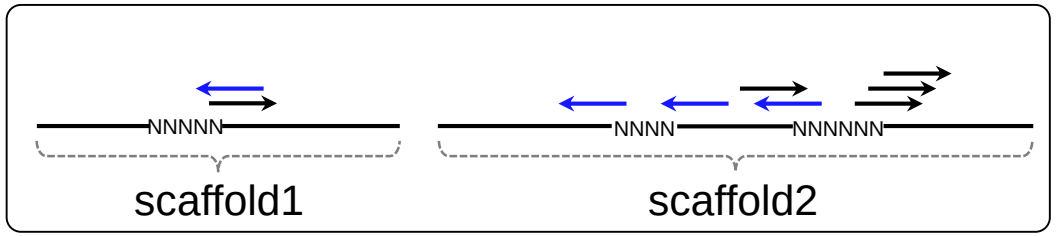
Assembly（コンティグの作成）



Scaffold



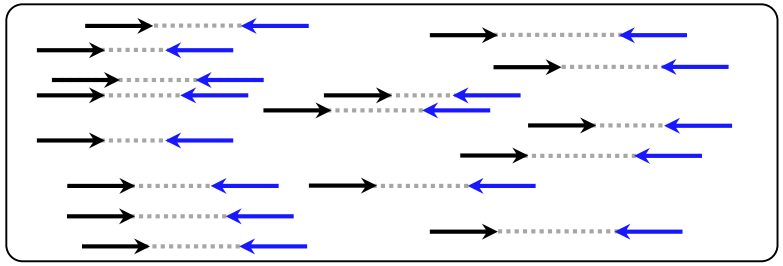
Gap close



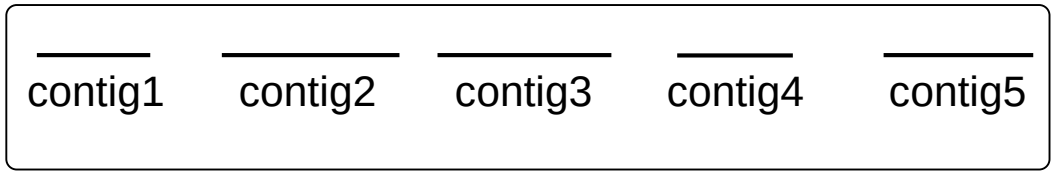
用語：gap close

得られたscaffoldsにリードをマップし…gap周辺にマップされたリードの塩基でNを置換。gapのNがなくなり、閉じていく(close)のでgap closeという（おそらく）

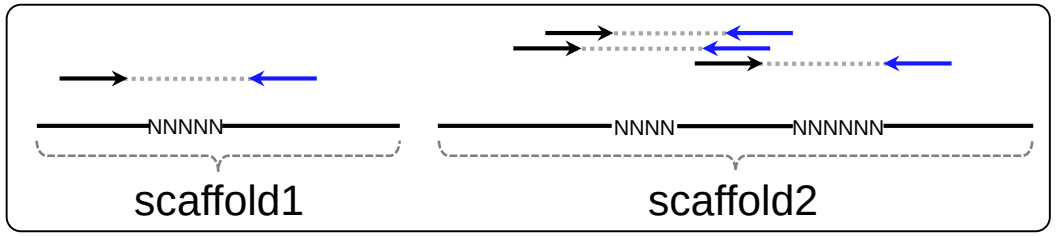
入力：paired-end FASTQファイル



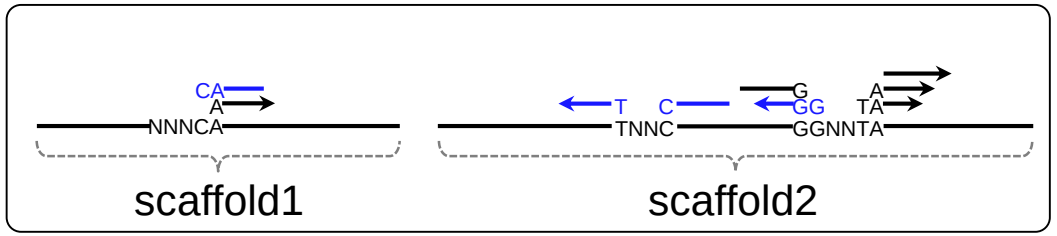
Assembly（コンティグの作成）



Scaffold



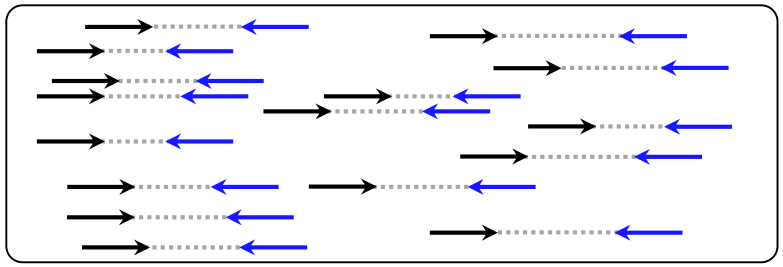
Gap close



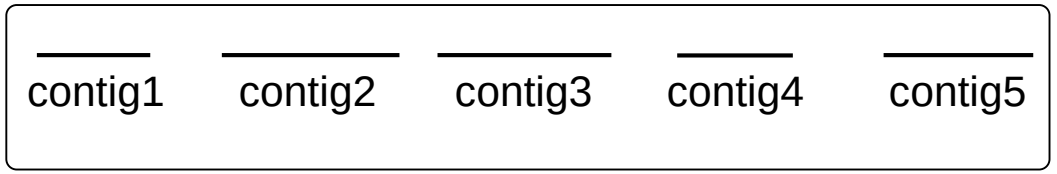
*de novo*アセンブリ

①最も有名なNGSデータ用*de novo*ゲノムアセンブリプログラムであるVelvet (Zerbino and Birney, *Genome Res.*, 2008)は、Step2までを実行。②比較的最近開発されたPlatanus (Kajitani et al., *Genome Res.*, 2014)は、Step3までを実行してくれる

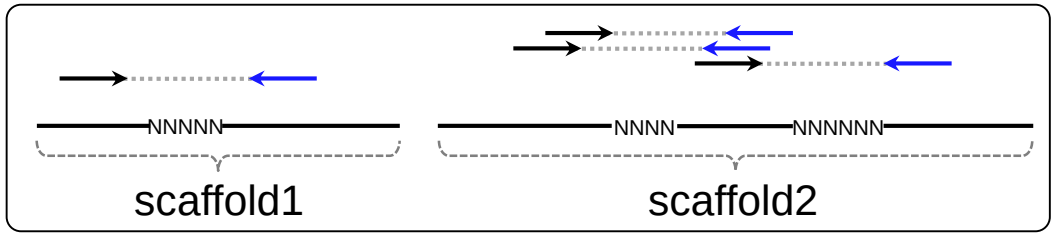
入力：paired-end FASTQファイル



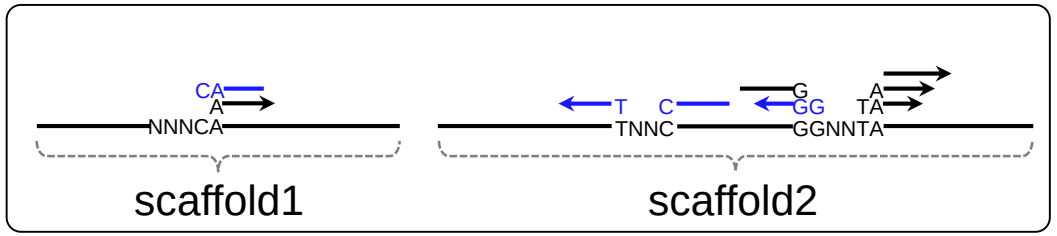
Step1: Assembly



Step2: Scaffold



Step3: Gap close

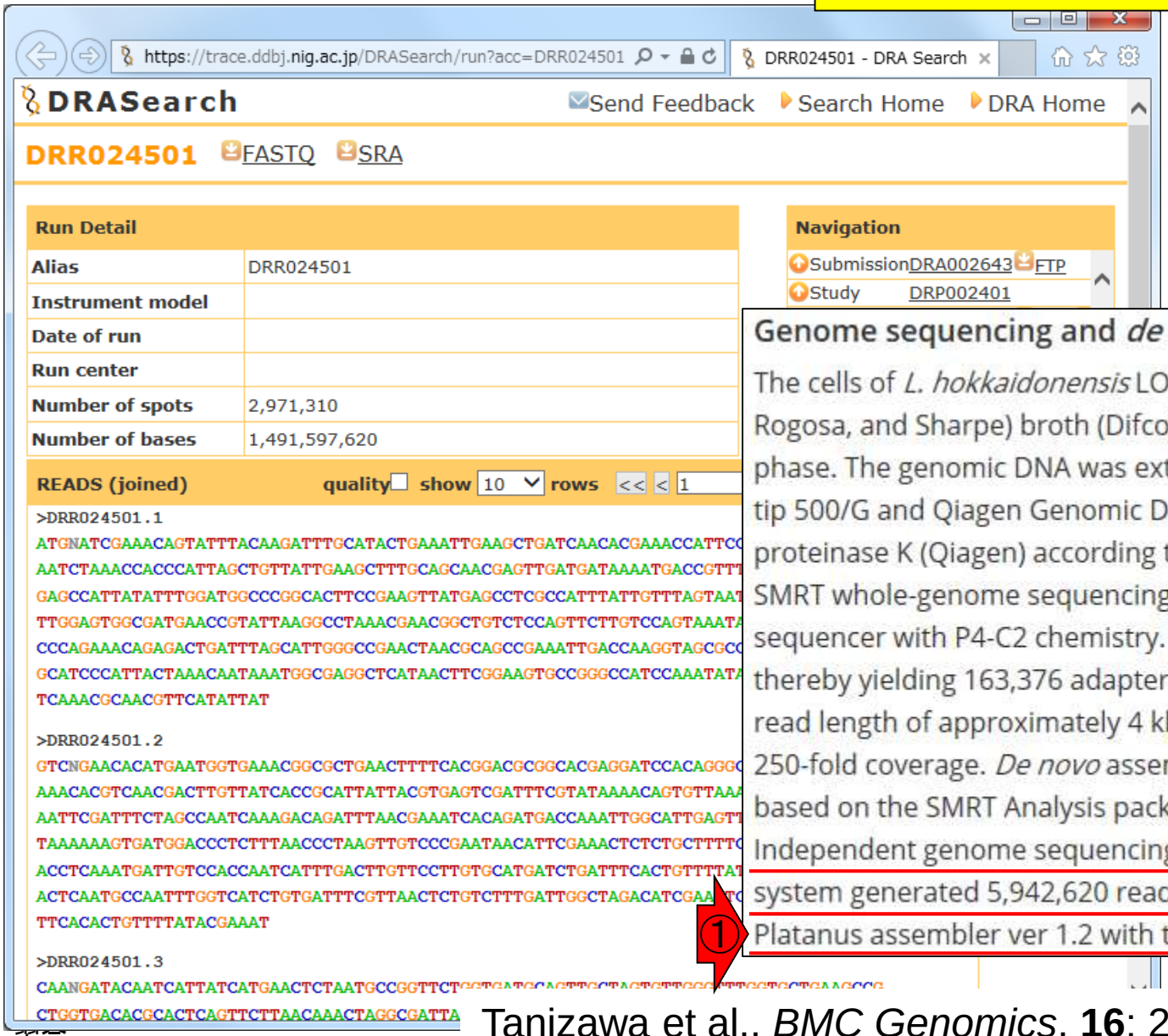


①Velvet

②Platanus

乳酸菌論文は…

乳酸菌(*Lactobacillus hokkaidonensis* LOOC260^T) ゲノム解読論文では、Illumina MiSeqデータ(DRR24501)の*de novo*アセンブリに①Platanus (ver. 1.2)を利用



Genome sequencing and *de novo* assembly

The cells of *L. hokkaidonensis* LOOC260^T were cultured in MRS (de Man, Rogosa, and Sharpe) broth (Difco) and were harvested in the mid-logarithmic phase. The genomic DNA was extracted and purified using Qiagen Genomic-tip 500/G and Qiagen Genomic DNA Buffer Set with lysozyme (Sigma) and proteinase K (Qiagen) according to the manufacturer's instruction. PacBio SMRT whole-genome sequencing was performed using a PacBio RSII sequencer with P4-C2 chemistry. Four SMRT cells were used for sequencing, thereby yielding 163,376 adapter-trimmed reads (subreads) with an average read length of approximately 4 kbp, which corresponded to approximately 250-fold coverage. *De novo* assembly was conducted using the HGAP method based on the SMRT Analysis package 2.0, which yielded seven contigs. Independent genome sequencing using the 250-bp paired-end Illumina MiSeq system generated 5,942,620 reads, which were assembled into contigs using Platanus assembler ver 1.2 with the default settings [40]. The initial contigs

①

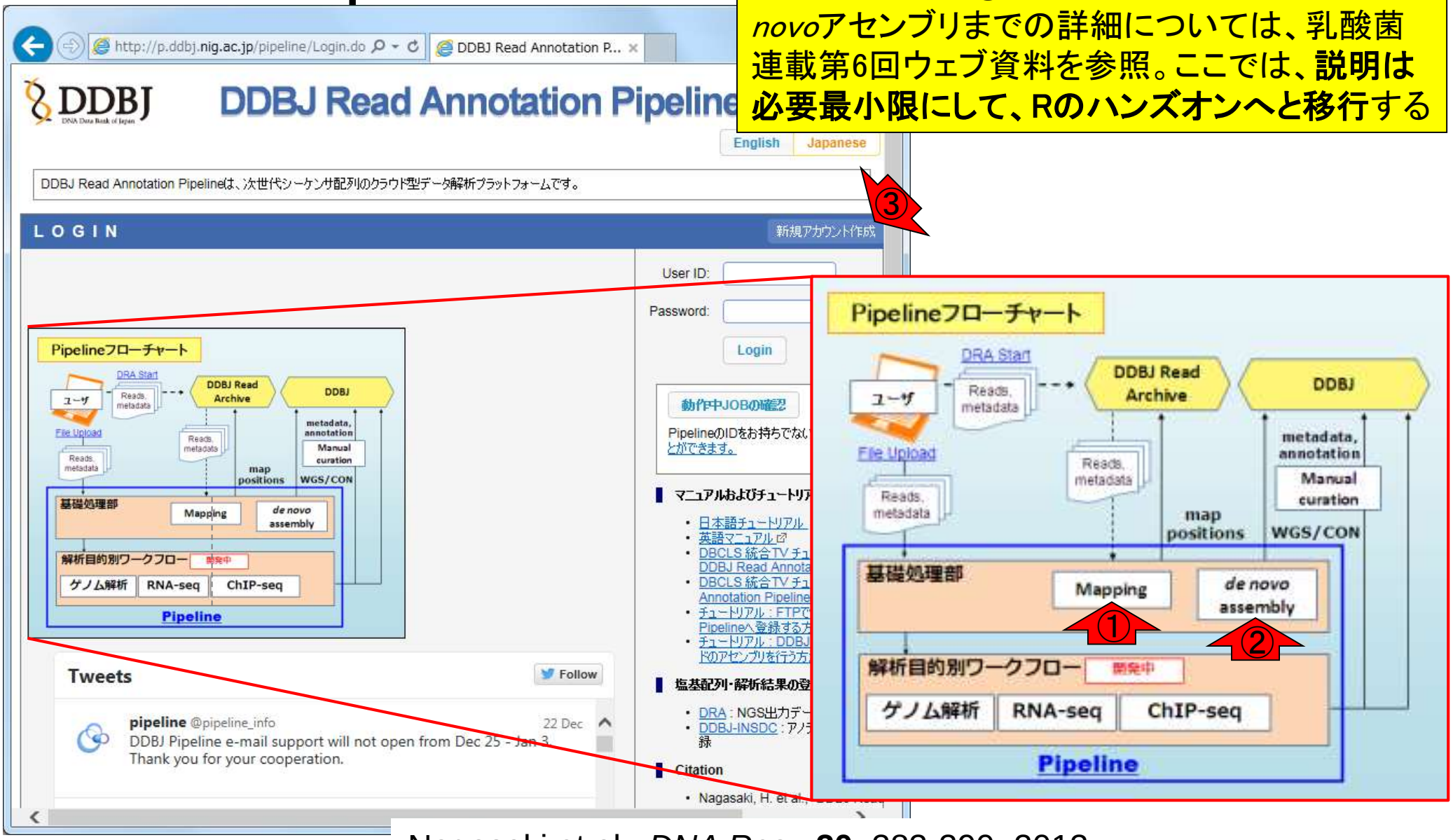
Tanizawa et al., *BMC Genomics*, **16**: 240, 2015

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

DDBJ Pipeline

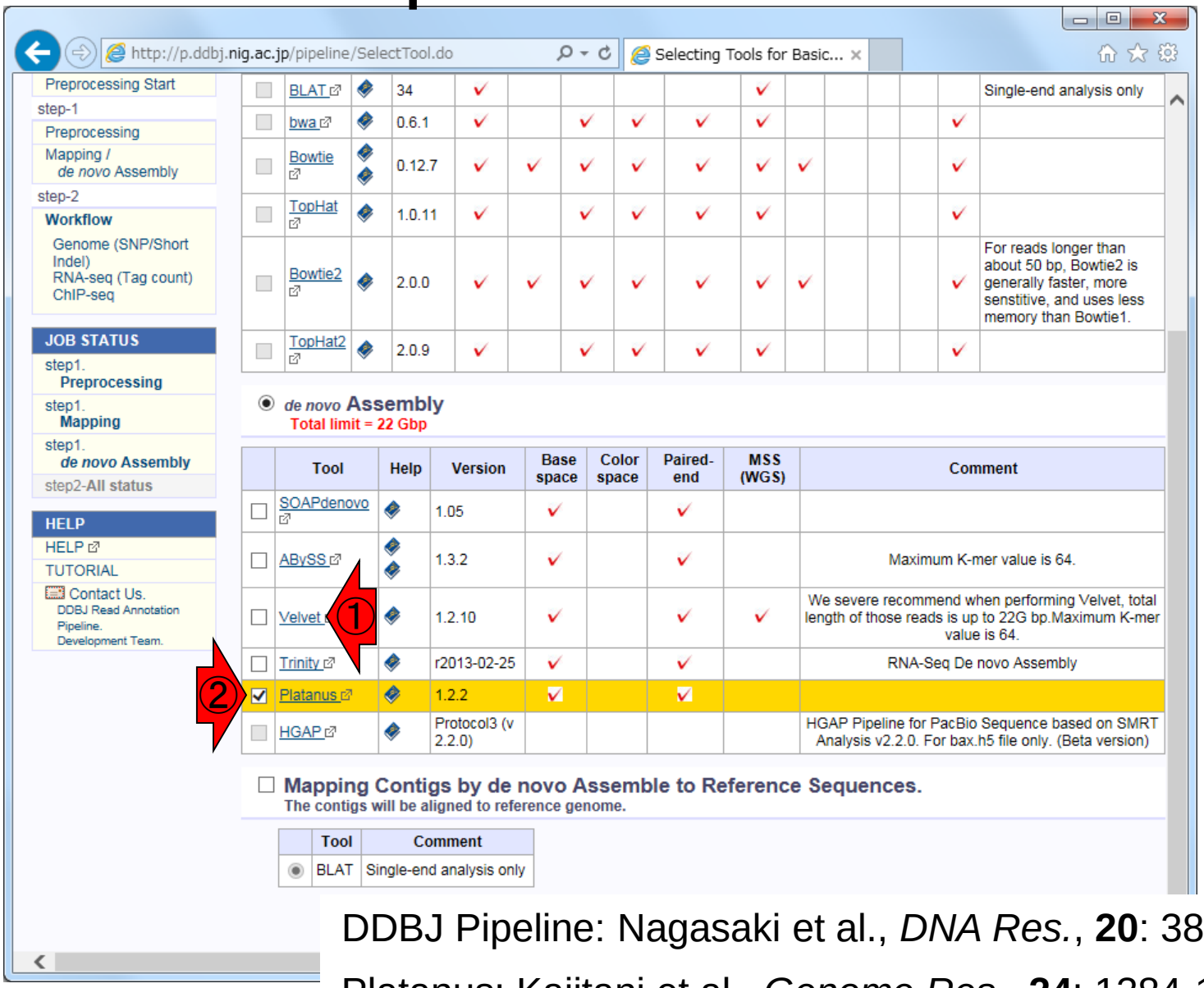
DDBJ Pipelineでは、主に①マッピングや②*de novo*アセンブリができる。特に後者ができるのは非常に有難い。③新規アカウント作成から*de novo*アセンブリまでの詳細については、乳酸菌連載第6回ウェブ資料を参照。ここでは、**説明は必要最小限にして、Rのハンズオンへと移行**する



Nagasaki et al., *DNA Res.*, **20**: 383-390, 2013

DDBJ PipelineでPlatanus

DDBJ Pipelineのプログラム選択画面。①Velvetや②Platanusを選択可能

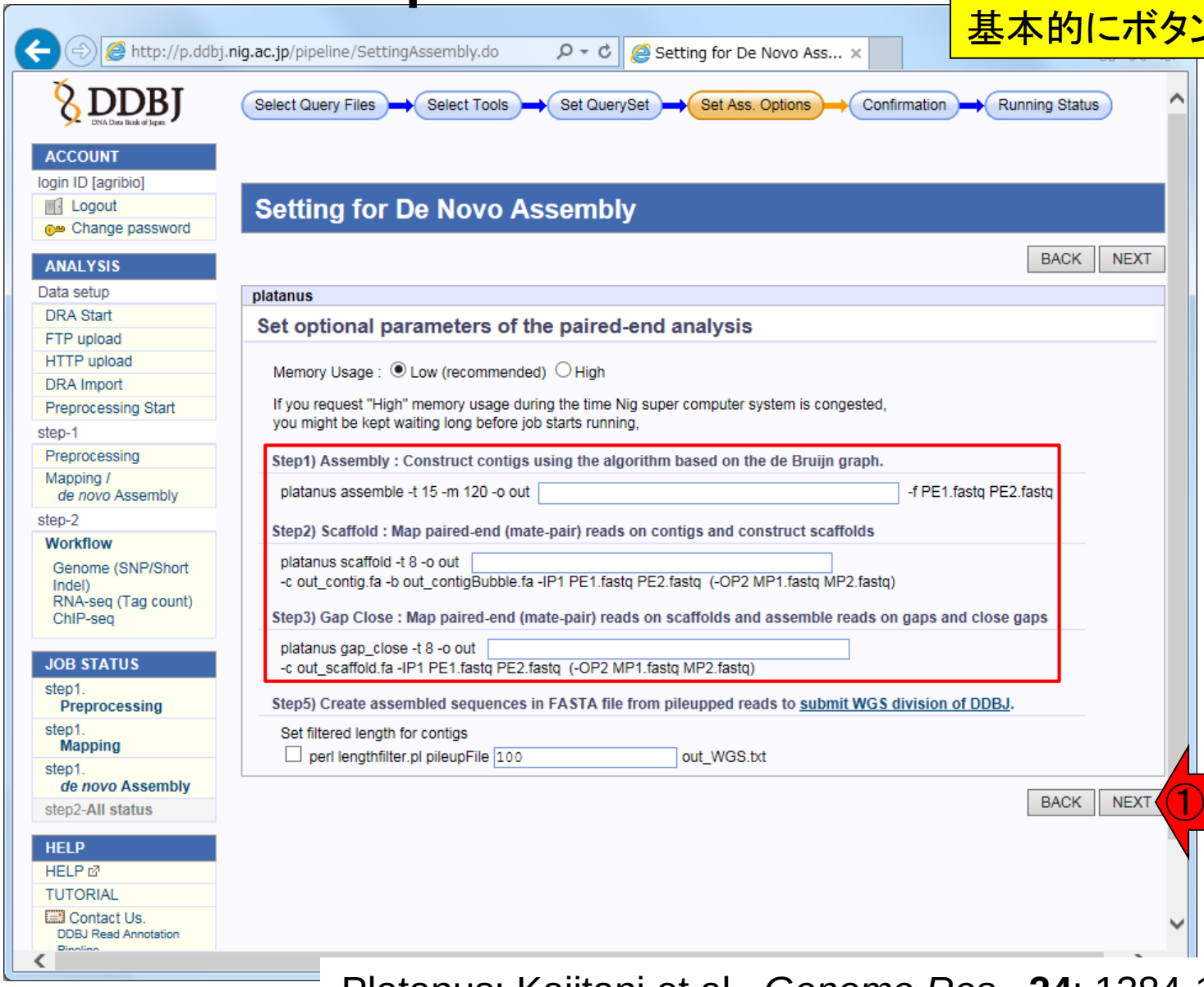


DDBJ Pipeline: Nagasaki et al., *DNA Res.*, **20**: 383-390, 2013

Platanus: Kajitani et al., *Genome Res.*, **24**: 1384-1395, 2014

DDBJ PipelineでPlatan

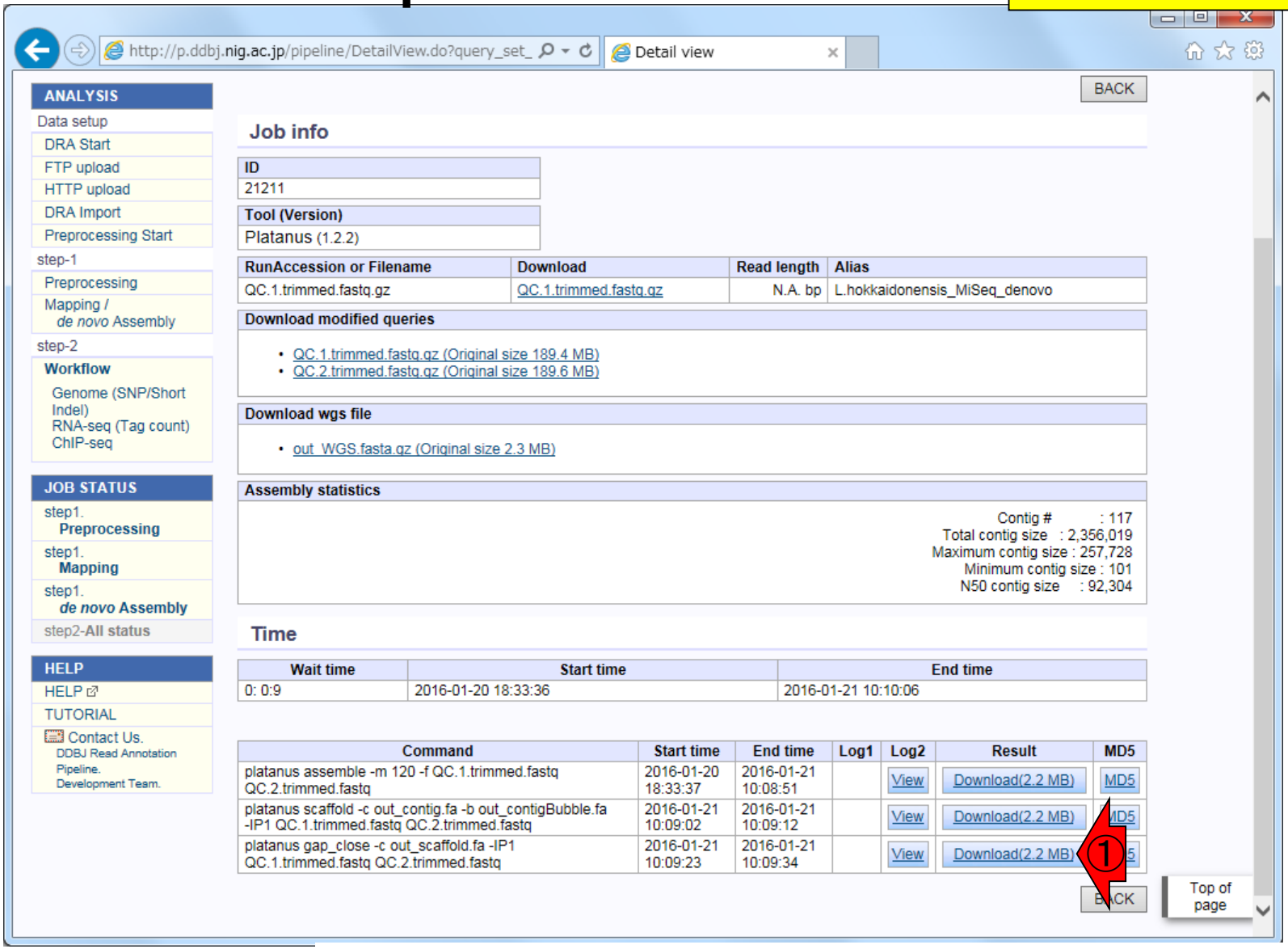
*de novo*アセンブリの一般的な手順がわかっていれば、赤枠内のStep1–3の説明の意味がなんとなくわかる。①DDBJ Pipelineは基本的にボタンをポチポチ押していくだけ



Platanus: Kajitani et al., *Genome Res.*, **24**: 1384-1395, 2014

DDBJ PipelineでPlatanus

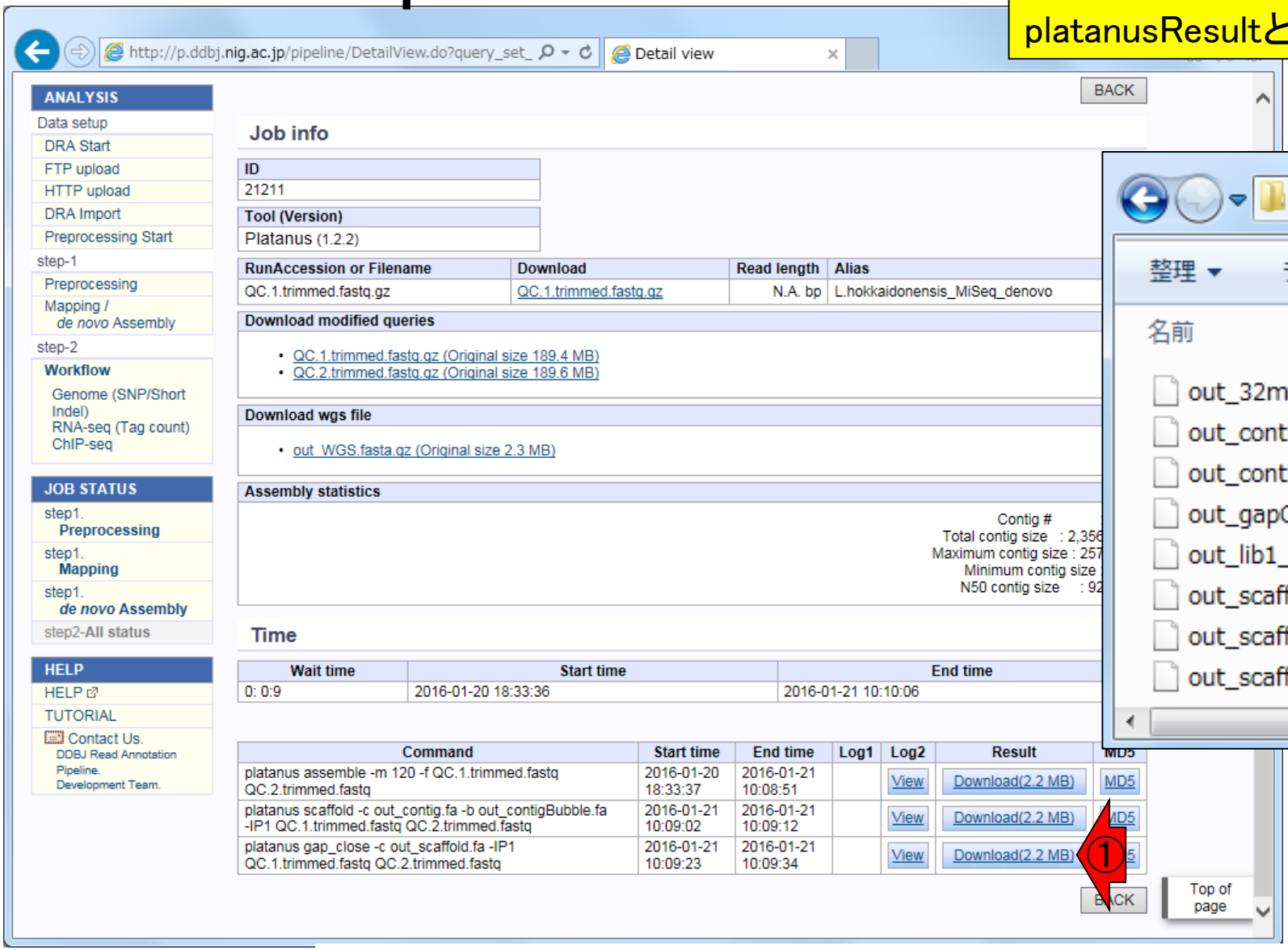
アセンブリ終了後の画面。①Platanus実行結果ファイル(platanusResult.zip)をダウンロードして解凍したのが…



Platanus: Kajitani et al., *Genome Res.*, **24**: 1384-1395, 2014

DDBJ PipelineでPlatanus

アセンブリ終了後の画面。①Platanus実行結果ファイル(platanusResult.zip)をダウンロードして解凍したのが…②platanusResultというフォルダ

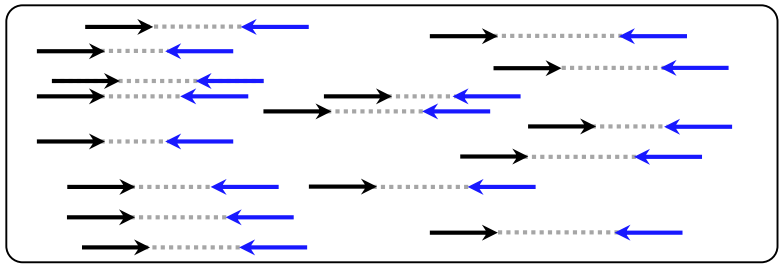


Platanus: Kajitani et al., *Genome Res.*, **24**: 1384-1395, 2014

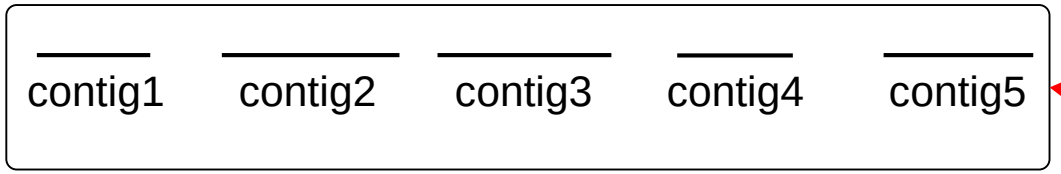
DDBJ PipelineでPlatanu

一般的な*de novo*アセンブリの手順を知っておけば、ファイル名から最終的な結果が③out_gapClosed.faだと認識できる

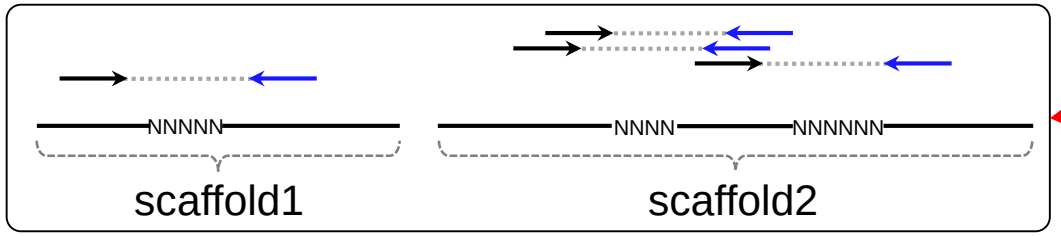
入力：paired-end FASTQファイル



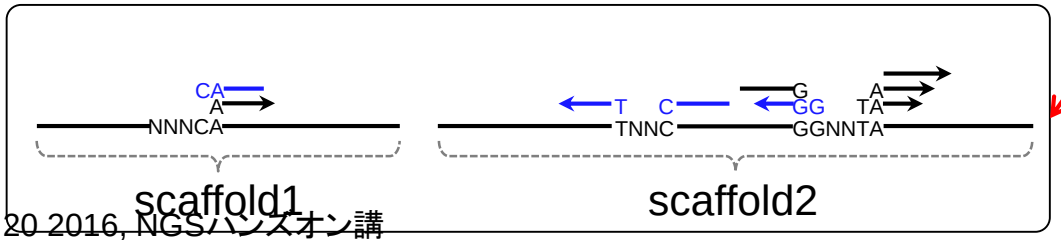
Step1: Assembly

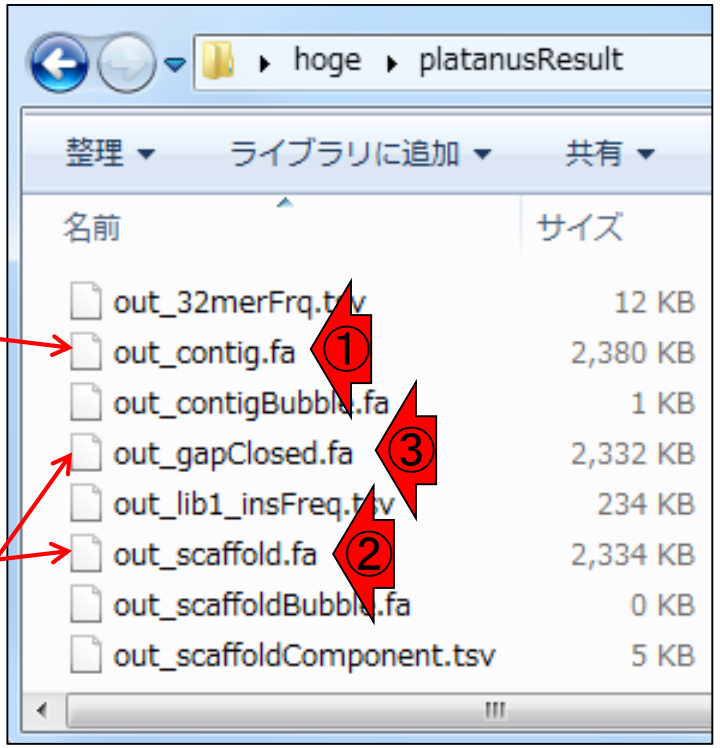


Step2: Scaffold



Step3: Gap close



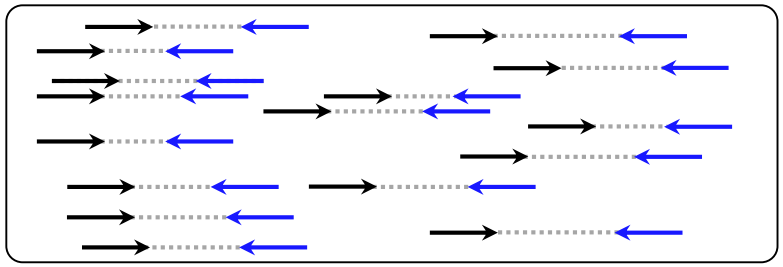


Contents

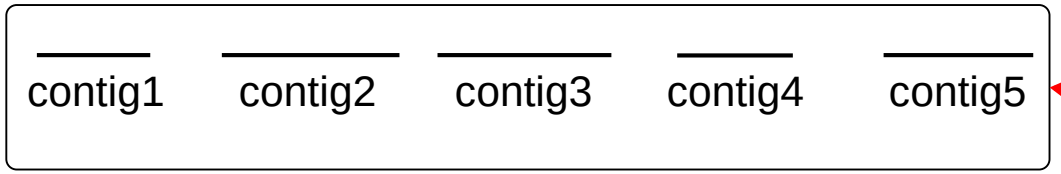
- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

k-mer解析(k=1)

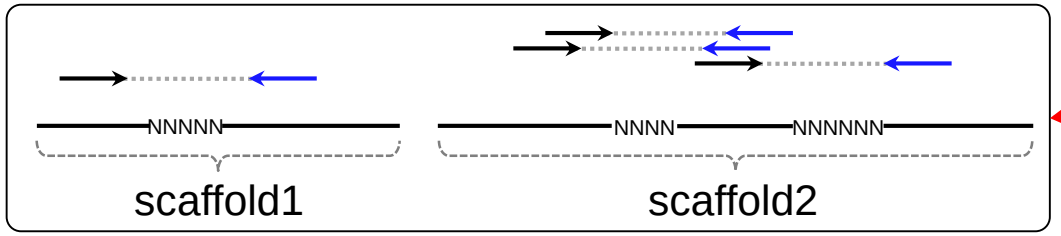
入力：paired-end FASTQファイル



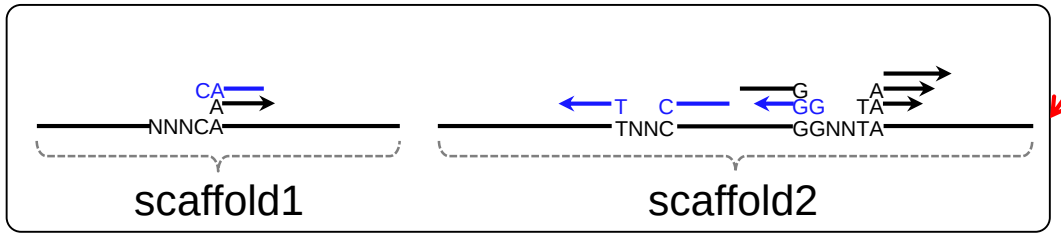
Step1: Assembly



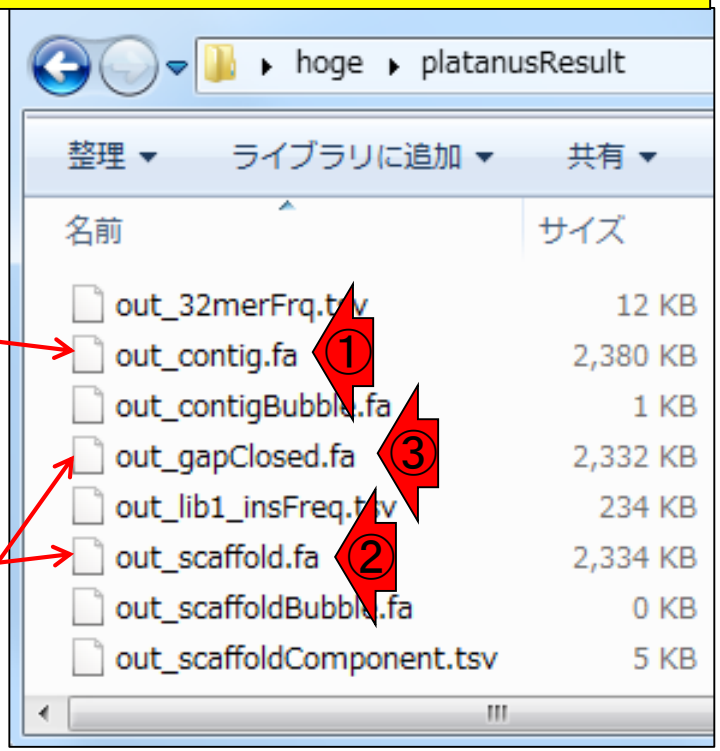
Step2: Scaffold



Step3: Gap close



（アセンブリ実行結果の）multi-FASTAファイルを読み込んで、**塩基ごとの出現頻度解析**(k=1のk-mer解析に相当)ができる。①Step1実行後（out_contig.fa）はNがなく、②Step2実行後（out_scaffold.fa）にNができて、③Step3実行後（out_gapClosed.fa）にNが減るのだろうと妄想できる。それを自力で確認することで、アルゴリズムの理解を深められる



k-mer解析(k=1)

①(アセンブリ実行結果の)multi-FASTAファイルを読み込んで、塩基ごとの出現頻度解析を行う項目

(Rで)塩基配列解析

～NGS、RNA-seq、ゲノム、トランスクリプトーム、正規化、発現変動、統計、モデル、バイオインフォマティクス～
(last modified 2016/02/03, since 2011)

What's new?

- このウェブページは
  フリーソフトRと必要
  利用法(Windows20
  書籍もあります。(20
- 多群間比較用の推
  については門田の
  検出結果のおおよそ
  イン周辺の関連項目
- Erratum。2014.06.
  には、kの値を小さ
  ばいいです。(見て
  たm(_ _)m(2016/02

イントロ | 一般 | k-mer解析 | k=1(塩基ごとの出現頻度解析) | Biostrings NEW

Biostringsパッケージを用いて、multi-FASTA形式ファイルを読み込んで、"A", "C", "G", "T", ..., "N", ...など塩基ごとの出現頻度を調べるやり方を示します。k-mer解析のk=1の場合に相当します。
「ファイル」-「ディレクトリの変更」で解析したいファイルを置いてあるディレクトリに移動し以下をコピペ。

1. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の場合:

配列ごとに出現頻度をカウントした結果を返すやり方です。

```
in_f <- "hoge4.fa"                          #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.txt"                        #出力ファイル名を指定してout_fに格納

#必要なパッケージをロード
library(Biostrings)                         #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み

#本番
out <- alphabetFrequency(fasta)             #A,C,G,T,..の数を各配列ごとにカウントした結果をoutl
out                                         #outの中身を表示
```

k-mer解析(k=1)

①例題7が、PlatanusのStep3実行後のファイル（②out_gapClosed.fa）を入力とするものなので、そのままコピペできて便利。これを実行します

イントロ | 一般 | k-mer解析 | k=1(塩基ごとの出現頻度解析) | Biostrings NEW

Biostringsパッケージを用いて、multi-FASTA形式ファイルを読み込んで、"A", "C", "G", "T", ..., "N", ...など塩基ごとの出現頻度を調べるやり方を示します。k-mer解析のk=1の場合に相当します。
「ファイル」−「ディレクトリの変更」で解析したいファイルを置いてあるディレクトリに移動し以下をコピペ。

1. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の場合:

配列ごとに出現頻度をカウント

```
in_f <- "hoge4.fa"
out_f <- "hoge1.txt"

#必要なパッケージをロード
library(Biostrings)

#入力ファイルの読み込み
fasta <- readDNAStringSet

#本番
out <- alphabetFrequency(
out

#ファイルに保存
tmp <- cbind(names(fasta)
write.table(tmp, out_f, s
```

①

7. FASTA形式ファイル(out_gapClosed.fa)の場合:

DDBJ Pipeline (Nagasaki et al., DNA Res., 2013)上で de novoゲノムアセンブリプログラムPlatanus (Kajitani et al., Genome Res., 2014)を実行して得られたmulti-FASTA形式ファイル(out_gapClosed.fa; 約2.4MB)です。

②

```
in_f <- "out_gapClosed.fa"                #入力ファイル名を指定してin_fに格納
out_f <- "hoge7.txt"                      #出力ファイル名を指定してout_fに格納
param_base <- c("A", "C", "G", "T", "N")#出力させたい塩基を指定

#必要なパッケージをロード
library(Biostrings)                       #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み

#本番
hoge <- alphabetFrequency(fasta)          #A,C,G,T,..の数を各配列ごとにカウントした結果
obj <- is.element(colnames(hoge), param_base)#条件を満たすかどうかを判定した結果をob
#out <- colSums(hoge[, obj])              #列ごとの総和をoutに格納
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ごとの総和をoutに格納

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=T, col.names=F)#tm
```

k-mer解析(k=1)

つまり、Platanus実行結果ファイル(platanusResult.zip)をダウンロードし、解凍して得られた①platanusResultフォルダ中の②out_gapClosed.faを入力として、塩基ごとの出現頻度解析を行う

イントロ | 一般 | k-mer解析 | k=1(塩基ごとの出現頻度解析) | Biostrings NE

Biostringsパッケージを用いて、multi-FASTA形式ファイルを読み込んで、"A", "C", "G", "T", ..., "N", ...など塩基
の出現頻度を調べるやり方を示します。k-mer解析のk=1の場合に相当します。
「ファイル」-「ディレクトリの変更」で解析したいファイルを置いてあるディレクトリに移動し以下をコピペ。

1. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の場合:

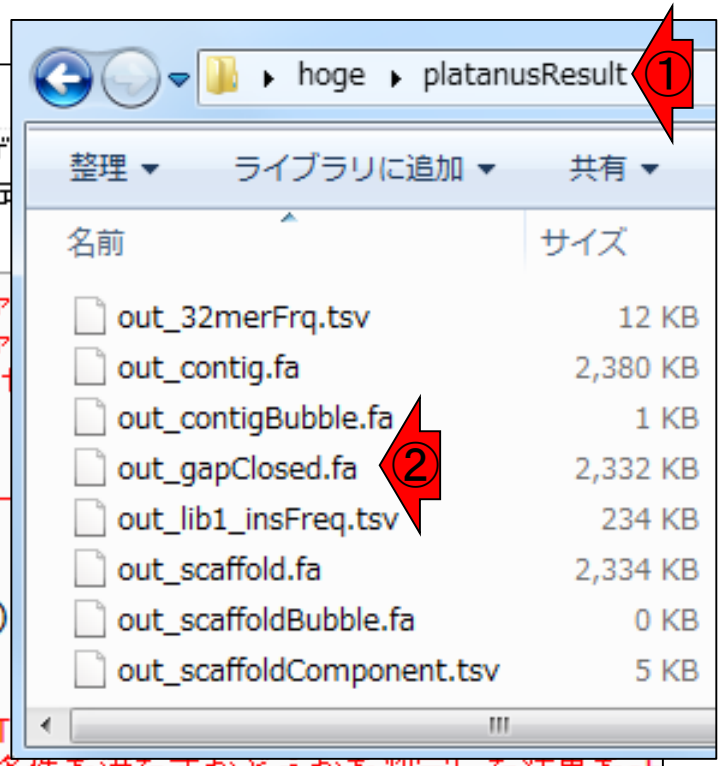
配列ごとに出現頻度をカウントした

```
in_f <- "hoge4.fa"
out_f <- "hoge1.txt"

#必要なパッケージをロード
library(Biostrings)

#入力ファイルの読み込み
fasta <- readDNAStringSet

#本番
out <- alphabetFrequency(
out

#ファイルに保存
tmp <- cbind(names(fasta)
write.table(tmp, out_f, s
```

7. FASTA形式ファイル(out_gapClosed.fa)の場合:

DDBJ Pipeline (Nagasaki et al., DNA Res., 2013)上で de novoゲ
et al., Genome Res., 2014)を実行して得られたmulti-FASTA形式
す。

```
in_f <- "out_gapClosed.fa"                    #入力ファ
out_f <- "hoge7.txt"                          #出力ファ
param_base <- c("A", "C", "G", "T", "N")#出力さ

#必要なパッケージをロード
library(Biostrings)                           #パッケー

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")

#本番
hoge <- alphabetFrequency(fasta)              #A,C,G,T
obj <- is.element(colnames(hoge), param_base)#条件を満たすかどうかを判定した結果をob
#out <- colSums(hoge[, obj])                  #列ごとの総和をoutに格納
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ごとの総和をoutに格納

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=T, col.names=F)#tm
```

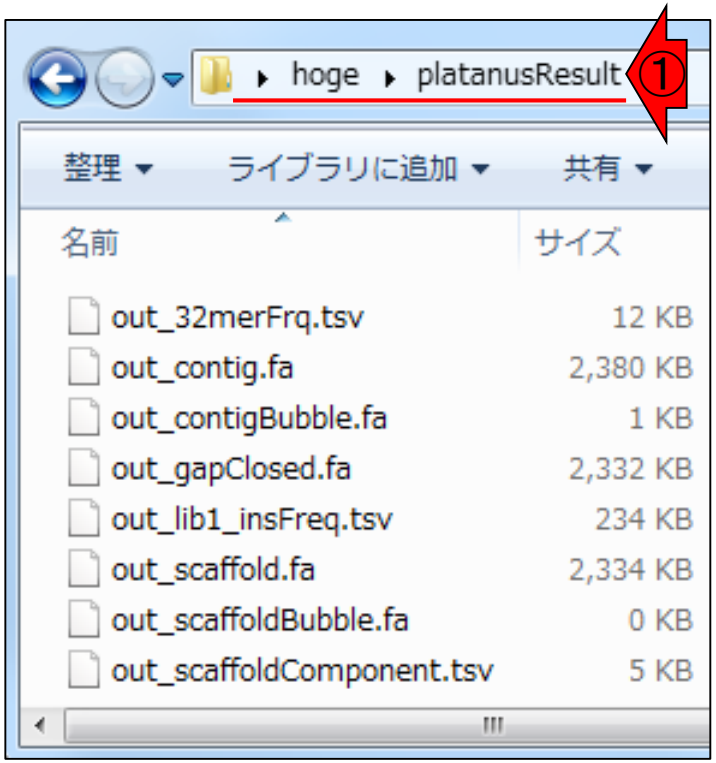


これくらい当然ついてこれますよねw

getwd()とlist.files()

R Console

```
'demo()' と入力すればデモをみることができます。
'help()' とすればオンラインヘルプが出ます。
'help.start()' で HTML ブラウザによるヘルプがみら$
'q()' と入力すれば R を終了します。

> getwd()
[1] "C:/Users/kadota/Desktop/hoge/platanusResult"
> list.files()
[1] "out_32merFrq.tsv"
[2] "out_contig.fa"
[3] "out_contigBubble.fa"
[4] "out_gapClosed.fa"
[5] "out_lib1_insFreq.tsv"
[6] "out_scaffold.fa"
[7] "out_scaffoldBubble.fa"
[8] "out_scaffoldComponent.tsv"
>
```



コピペ実行結果

①解析結果(塩基ごとの出現頻度情報)はhoge7.txtに保存される。②list.files()で、自分が出力として指定した③hoge7.txtが存在することを確認。④EXCELで眺めるとこんな感じ

7. FASTA形式ファイル(out_gapClosed.fa)の場合:

DDBJ Pipeline (Nagasaki et al., DNA Res., 2013)上で de novoゲノムアセンブリプログラムPlatanus (Kajitani et al., Genome Res., 2014)を実行して得られたmulti-FASTA形式ファイル(out_gapClosed.fa; 約2.4MB)です。

```
in_f <- "out_gapClose     fa"              #入力ファイル名を指定してin_fに格納
out_f <- "hoge7.txt"                       #出力ファイル名を指定してout_fに格納
param_base <- c("A",  ,  "G", "T", "N")#出力

#必要なパッケージをロード
library(Biostrings)                        #パック

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta

#本番
hoge <- alphabetFrequency(fasta)           #A,C,
obj <- is.element(colnames(hoge), param_bas
#out <- colSums(hoge[, obj])               #列
out <- apply(as.matrix(hoge[, obj]), 2, sum)

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, q
```

①

R Console

```
> hoge <- alphabetFrequency(fast
> obj <- is.element(colnames(hog
> #out <- colSums(hoge[, obj])
> out <- apply(as.matrix(hoge[,
>
> #ファイルに保存
> write.table(out, out_f, sep="\
> list.files()
 [1] "hoge7.txt"
 [2] "out_32merFr    .tsv"
 [3] "out_contig.fa"
 [4] "out_contigBubble.fa"
 [5] "out_gapClosed.fa"
 [6] "out_lib1_insFreq.tsv"
 [7] "out_scaffold.fa"
 [8] "out_scaffoldBubble.fa"
 [9] "out_scaffoldComponent.tsv"
>
```

②

③

④

hoge7.txt - Excel

D7

	A	B
1	A	729772
2	C	458211
3	G	440585
4	T	727451
5	N	0

総塩基数

①outオブジェクトは数値ベクトル。②和を計算するsum関数実行結果(2,356,019)は、入力ファイル(out_gapClosed.fa)中の総塩基数に相当

7. FASTA形式ファイル(out_gapClosed.fa)の場合:

DDBJ Pipeline (Nagasaki et al., DNA Res., 2013)上で de novoゲノムアセンブリプログラムPlatanus (Kajitani et al., Genome Res., 2014)を実行して得られたmulti-FASTA形式ファイル(out_gapClosed.fa; 約2.4MB)です。

```
in_f <- "out_gapClosed.fa"                #入力ファイル名を指定してin_fに格納
out_f <- "hoge7.txt"                      #出力ファイル名を指定してout_fに格納
param_base <- c("A", "C", "G", "T", "N")#出力

#必要なパッケージをロード
library(Biostrings)                       #パック

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta

#本番
hoge <- alphabetFrequency(fasta)          #A,C,G
obj <- is.element(colnames(hoge), param_base)
#out <- colSums(hoge[, obj])              #列ごと
out <- apply(as.matrix(hoge[, obj]), 2, sum)#

#ファイルに保存
write.table(out, out_f, sep="\t", append=F,
```

①

R Console

```
[4] "out_contigBubble.fa"
[5] "out_gapClosed.fa"
[6] "out_lib1_insFreq.tsv"
[7] "out_scaffold.fa"
[8] "out_scaffoldBubble.fa"
[9] "out_scaffoldComponent.tsv"
> getwd()
[1] "C:/Users/kadota/Desktop/hog
> date()
[1] "Thu Feb 04 14:27:38 2016"
> out_f
[1] "hoge7.txt"
> out
     A      C      G      T      N
729772 458211 440585 727451      0
> sum(out)
[1] 2356019
>
```

①

②

hoge7.txt - Excel

D7

	A	B
1	A	729772
2	C	458211
3	G	440585
4	T	727451
5	N	0

総塩基数

①DDBJ Pipeline実行結果画面上の数値と同じ。
②入力ファイル(out_gapClosed.fa)は、DDBJ Pipeline上でPlatanusというde novoアセンブリプログラムを実行した結果だったことを思い出そう

ID
21211

Tool (Version)
Platanus (1.2.2)

RunAccession or Filename	Download	Read length	Alias
QC.1.trimmed.fastq.gz	QC.1.trimmed.fastq.gz	N.A. bp	L.hokkaidonensis_MiSeq_denovo

Download modified queries

- QC.1.trimmed.fastq.gz (Original size 189.4 MB)
- QC.2.trimmed.fastq.gz (Original size 189.6 MB)

Download wgs file

- out_WGS.fasta.gz (Original size 2.3 MB)

Assembly statistics

Contig # : 117
Total contig size : 2,356,019 ①
Maximum contig size : 257,728
Minimum contig size : 101
N50 contig size : 92,304

Time

Wait time	Start time	End time
0: 0:9	2016-01-20 18:33:36	2016-01-21 10:10:06

Command	Start time	End time	Log1	Log2	Result	MD5
platanus assemble -m 120 -f QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-20 18:33:37	2016-01-21 10:08:51		View	Download(2.2 MB)	MD5
platanus scaffold -c out_contig.fa -b out_contigBubble.fa -IP1 QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-21 10:09:02	2016-01-21 10:09:12		View	Download(2.2 MB)	MD5
platanus gap_close -c out_scaffold.fa -IP1 QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-21 10:09:23	2016-01-21 10:09:34		View	Download(2.2 MB) ②	D5

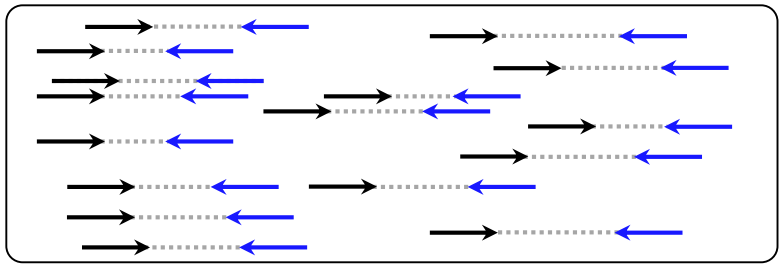
Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

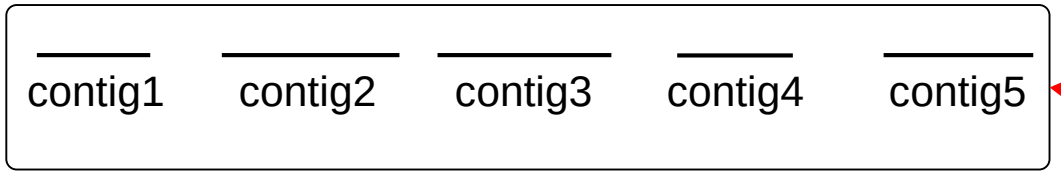
目的をおさらい

（アセンブリ実行結果の）multi-FASTAファイルを読み込んで、**塩基ごとの出現頻度解析**ができる。①Step1実行後（out_contig.fa）はNがなく、②Step2実行後（out_scaffold.fa）にNができて、③Step3実行後（out_gapClosed.fa）にNが減るのだろうと妄想できる。それを自力で確認することで、アルゴリズムの理解を深めることができる

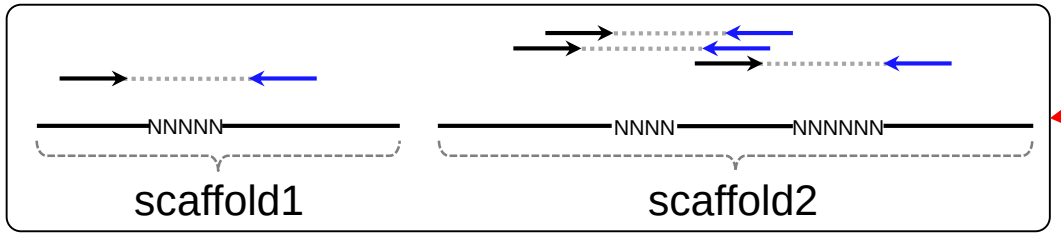
入力：paired-end FASTQファイル



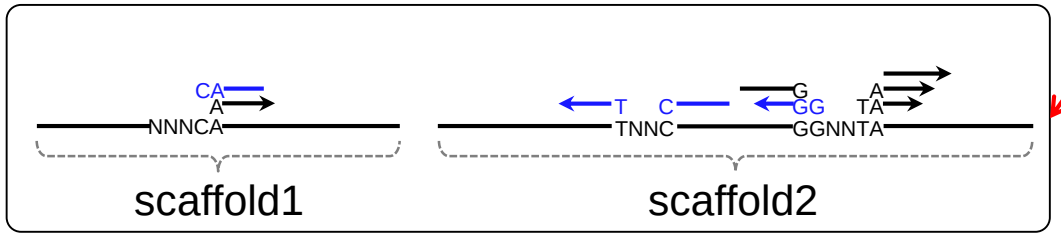
Step1: Assembly

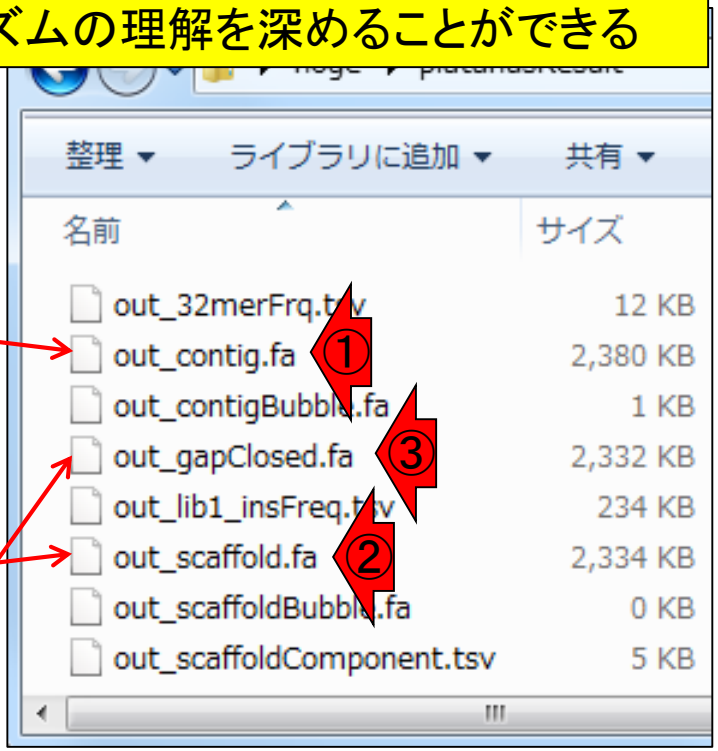


Step2: Scaffold



Step3: Gap close

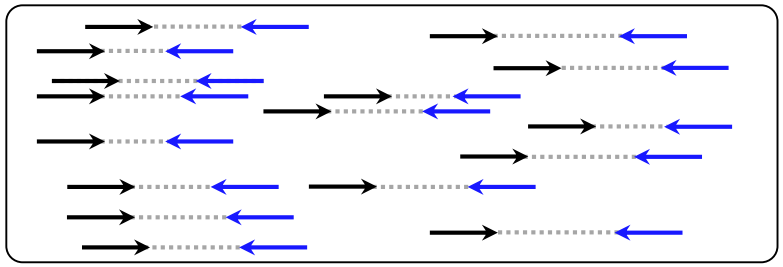




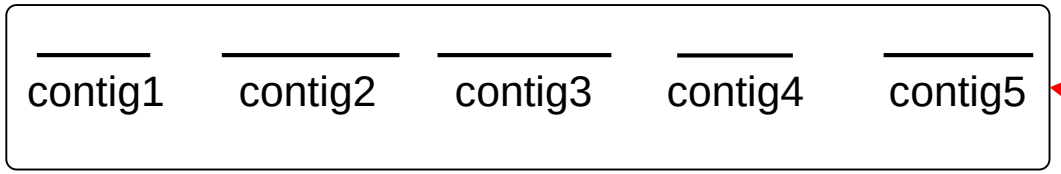
一気に結果を得る

①Step1実行後（out_contig.fa）はNがなく、
②Step2実行後（out_scaffold.fa）にNができて、
③Step3実行後（out_gapClosed.fa）にNが減る
のだろうという妄想を一気に確認したい!

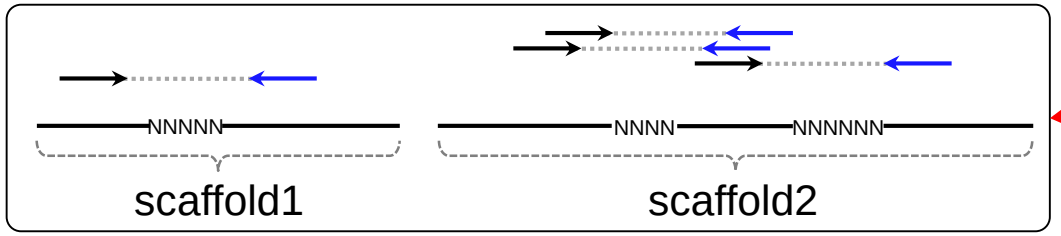
入力：paired-end FASTQファイル



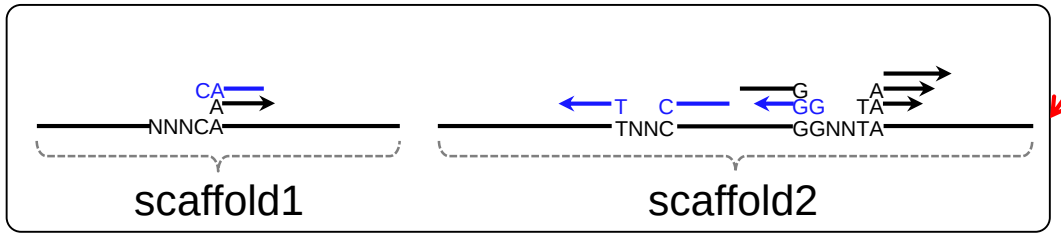
Step1: Assembly

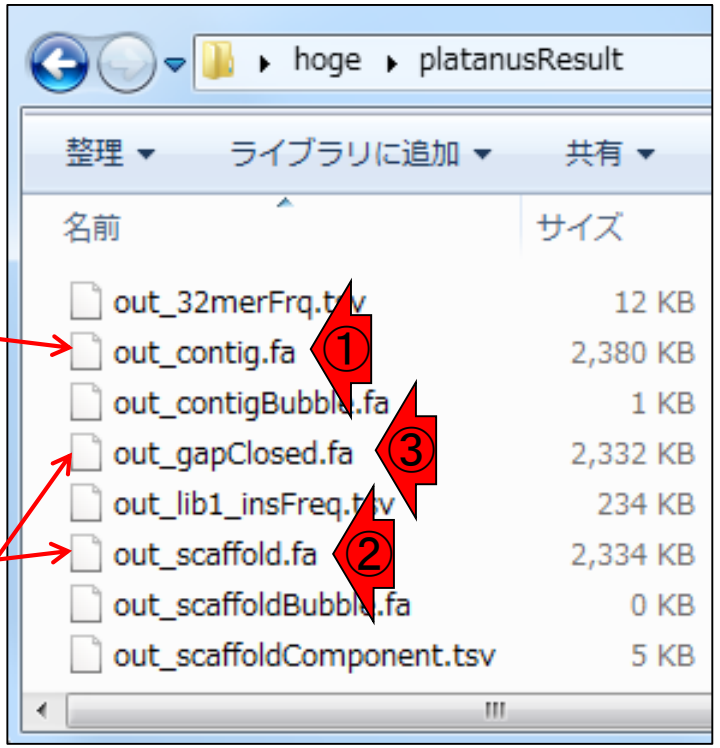


Step2: Scaffold



Step3: Gap close





一気に結果を得る

実際の利用時は、hogeフォルダ直下にある①rcode1.txtのように、無駄なコメントを除いてスリムにした一連のスクリプトを作成しておき、一気にコピペ

rcode1.txt

①

```
param_base <- c("A", "C", "G", "T", "N")#出力させたい塩基を指定↓
library(Biostrings)                     #パッケージの読み込み↓
↓
#################↓
###  Step 1↓
#################↓
in_f <- "out_contig.fa"                 #入力ファイル名を指定してin_fに格納↓
out_f <- "result_step1.txt"             #出力ファイル名を指定してout_fに格納↓
↓
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み
hoge <- alphabetFrequency(fasta)        #A,C,G,T,..の数を各配列ごとにカウントした
obj <- is.element(colnames(hoge), param_base)#条件を満たすかどうかを判定した結果
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ごとの総和をoutに格納↓
write.table(out, out_f, sep="¥t", append=F, quote=F, row.names=T, col.names=F)#t
↓
#################↓
###  Step 2↓
#################↓
in_f <- "out_scaffold.fa"               #入力ファイル名を指定してin_fに格納↓
out_f <- "result_step2.txt"             #出力ファイル名を指定してout_fに格納↓
↓
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み
hoge <- alphabetFrequency(fasta)        #A,C,G,T,..の数を各配列ごとにカウントした
obj <- is.element(colnames(hoge), param_base)#条件を満たすかどうかを判定した結果
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ごとの総和をoutに格納↓
write.table(out, out_f, sep="¥t", append=F, quote=F, row.names=T, col.names=F)#t
↓
#################↓
###  Step 3↓
#################↓
```

一気に結果を得る

hogeフォルダ直下にある、①rcode1.txtのように無駄なコメントを除いてスリムにした一連のスクリプトを作成しておき、一気にコピペ。②コピペ後に自分が指定した出力ファイルができていることを確認

rcode1.txt

①

```
param_base <- c("A", "C", "G", "T", "N")#出力させたい塩基を指定
library(Biostrings)                     #パッケージの読み込み

################
###  Step 1
################
in_f <- "out_contig.fa"                 #入力ファイル名を指定してin_fに格納
out_f <- "result_step1.txt"             #出力ファイル名を指定してout_fに格納

fasta <- readDNAStringSet(in_f, format="fasta")#in
hoge <- alphabetFrequency(fasta)        #A,C,G,T,..
obj <- is.element(colnames(hoge), param_base)#条件
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ご
write.table(out, out_f, sep="¥t", append=F, quote=

################
###  Step 2
################
in_f <- "out_scaffold.fa"               #入力ファイ
out_f <- "result_step2.txt"             #出力ファイ

fasta <- readDNAStringSet(in_f, format="fasta")#in
hoge <- alphabetFrequency(fasta)        #A,C,G,T,..
obj <- is.element(colnames(hoge), param_base)#条件
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ご
write.table(out, out_f, sep="¥t", append=F, quote=

################
###  Step 3
################
```

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge/platanusResult"
> list.files()
 [1] "hoge7.txt"
 [2] "out_32merFrq.tsv"
 [3] "out_contig.fa"
 [4] "out_contigBubble.fa"
 [5] "out_gapClosed.fa"
 [6] "out_lib1_insFreq.tsv"
 [7] "out_scaffold.fa"
 [8] "out_scaffoldBubble.fa"
 [9] "out_scaffoldComponent.tsv"
[10] "result_step1.txt"
[11] "result_step2.txt"
[12] "result_step3.txt"
>
```

②

①result_step*.txtの結果をまとめたものが②

結果のまとめ

rcode1.txt

```
param_base <- c("A", "C", "G", "T", "N")#出力させたい塩基を指定
library(Biostrings)                    #パッケージの読み込み

################
###  Step 1
################
in_f <- "out_contig.fa"                #入力ファイル名を指定してin_fに格納
out_f <- "result_step1.txt"            #出力ファイル名を指定してout_fに格納

fasta <- readDNAStringSet(in_f, format="fasta")#in
hoge <- alphabetFrequency(fasta)       #A,C,G,T,..
obj <- is.element(colnames(hoge), param_base)#条件
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ご
write.table(out, out_f, sep="¥t", append=F, quote=

################
###  Step 2
################
in_f <- "out_scaffold.fa"              #入力ファイ
out_f <- "result_step2.txt"            #出力ファイ

fasta <- readDNAStringSet(in_f, format="fasta")#in
hoge <- alphabetFrequency(fasta)       #A,C,G,T,..
obj <- is.element(colnames(hoge), param_base)#条件
out <- apply(as.matrix(hoge[, obj]), 2, sum)#列ご
write.table(out, out_f, sep="¥t", append=F, quote=

################
###  Step 3
################
```

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge/platanusResult"
> list.files()
 [1] "hoge7.txt"
 [2] "out_32merFrq.tsv
 [3] "out_contig.fa"
 [4] "out_contigBubble
 [5] "out_gapClosed.fa
 [6] "out_lib1_insFreq
 [7] "out_scaffold.fa"
 [8] "out_scaffoldBubb
 [9] "out_scaffoldComponent.tsv"
[10] "result_step1.txt"
[11] "result_step2.txt"
[12] "result_step3.txt"
>
```

②

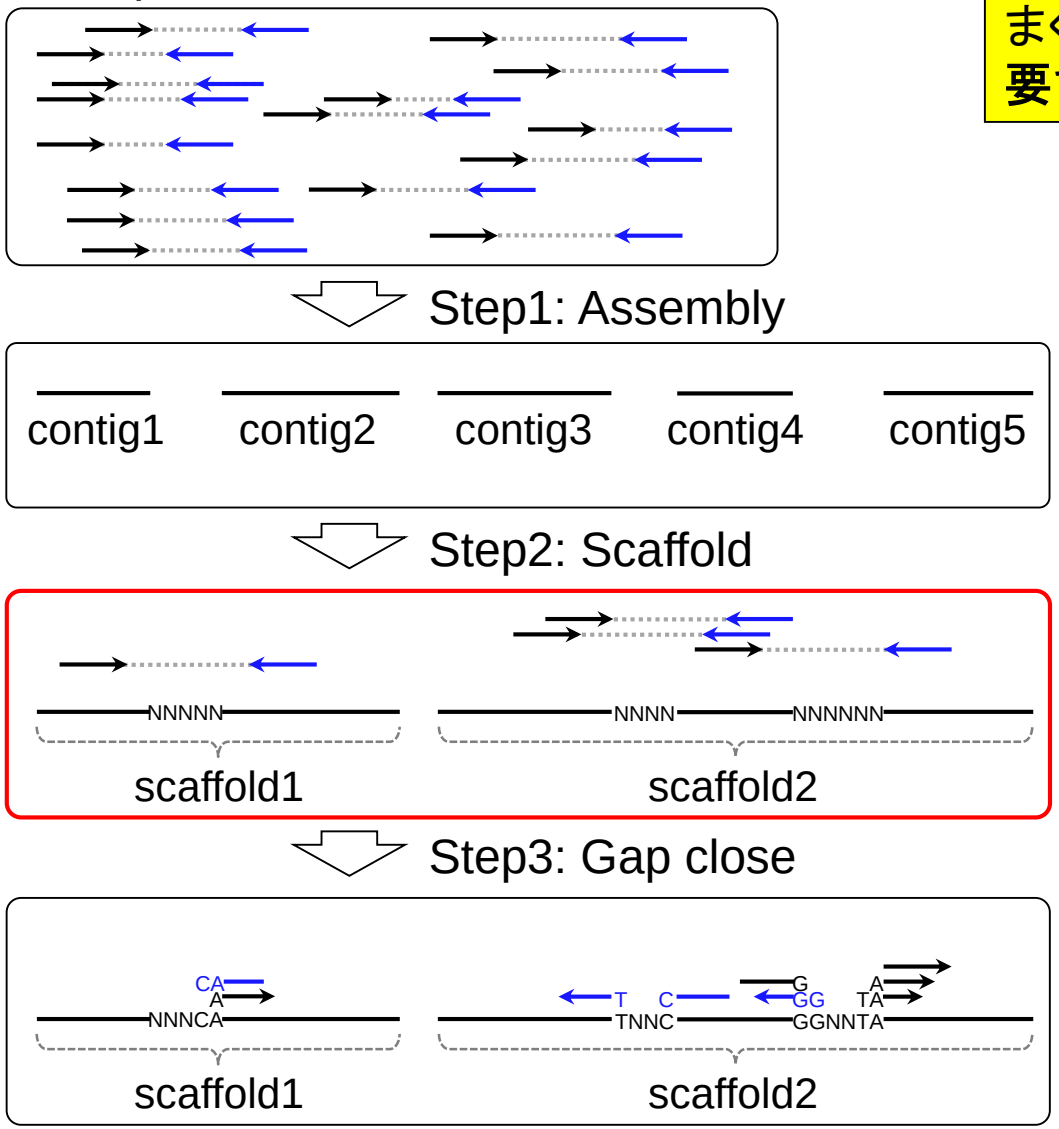
①

base	Step1	Step2	Step3
A	739836	729635	729772
C	469377	458119	458211
G	446806	440510	440585
T	742714	727306	727451
N	0	491	0

結果の解釈

①Step1実行後はNが0。②Step2実行後にNが491個生成されたということは、いくつかのcontigsがまとめられてscaffoldsになったのだろう。③Step3でNが0個になったのはおそらくたまたまうまくいっただけ。**491個よりも減ったということが重要**で、gap closeがうまく機能したと判断できる

入力：paired-end FASTQファイル

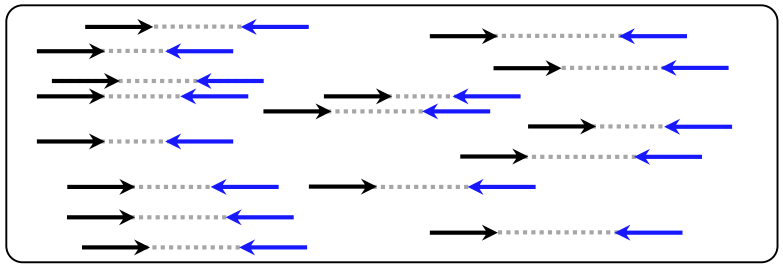


base	Step1	Step2	Step3
A	739836	729635	729772
C	469377	458119	458211
G	446806	440510	440585
T	742714	727306	727451
N	0	491	0

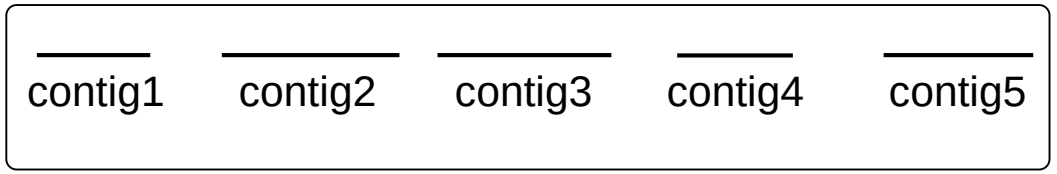
配列数

Rを使うことで、アセンブリプログラムの内部挙動の把握や理解ができる。他の例は、**配列数**（contig数やscaffold数と書くと説明しづらいので配列数に統一）。配列数は、①Step1 → Step2で減り、②Step2 → Step3では不変だろうと予想

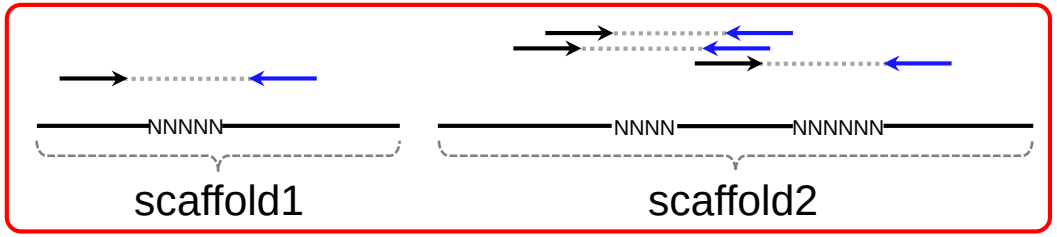
入力：paired-end FASTQファイル



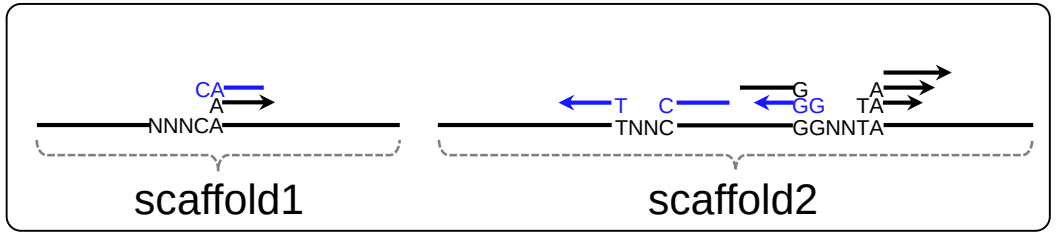
Step1: Assembly

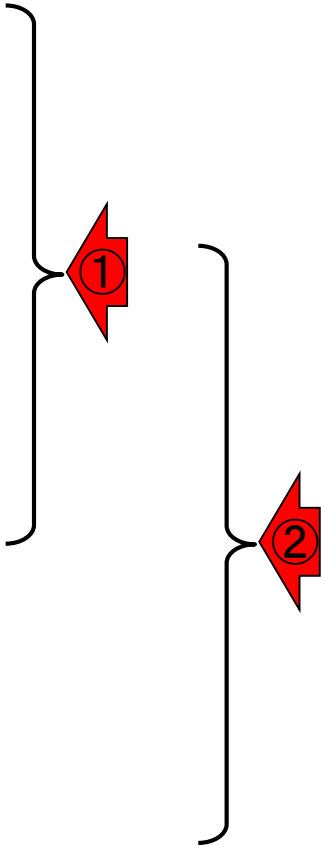


Step2: Scaffold



Step3: Gap close





配列数

配列数は、①Step1 → Step2で減り、②Step2 → Step3では不変だろうと予想。③(hogeフォルダ直下の)rcode2.txtは、配列数をカウントする必要最小限のコード。349 → 117 → 117で予想通りの結果

③

rcode2.txt

```
library(Biostrings)                    #パッケージの読み込み
↓
###  Step 1  ###
in_f <- "out_contig.fa"                #入力ファイ
fasta <- readDNAStringSet(in_f, format="fasta")#ir
length(fasta)                          #配列数を表
↓
###  Step 2  ###
in_f <- "out_scaffold.fa"              #入力ファイ
fasta <- readDNAStringSet(in_f, format="fasta")#ir
length(fasta)                          #配列数を表
↓
###  Step 3  ###
in_f <- "out_gapClosed.fa"             #入力ファイ
fasta <- readDNAStringSet(in_f, format="fasta")#ir
length(fasta)                          #配列数を表
```

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge/platanusResult"
> list.files(pattern="*.fa")
[1] "out_contig.fa"          "out_contigBubble.fa"
[3] "out_gapClosed.fa"       "out_scaffold.fa"
[5] "out_scaffoldBubble.fa"
> ###  Step 1  ###
> in_f <- "out_contig.fa"                    #入力フ$
> fasta <- readDNAStringSet(in_f, format="fasta")
> length(fasta)                              #配列数$
[1] 349
>
> ###  Step 2  ###
> in_f <- "out_scaffold.fa"                  #入力フ$
> fasta <- readDNAStringSet(in_f, format="fasta")
> length(fasta)                              #配列数$
[1] 117
>
> ###  Step 3  ###
> in_f <- "out_gapClosed.fa"                 #入力フ$
> fasta <- readDNAStringSet(in_f, format="fasta")
> length(fasta)                              #配列数$
[1] 117
```

配列数

①DDBJ Pipeline最終結果の数値（117個）と同じことがわかります。ここまでの話は、2016.03.03の講習会資料のダイジェスト版でした。「2016.03.03」でページ内検索

ID
21211

Tool (Version)
Platanus (1.2.2)

RunAccession or Filename	Download	Read length	Alias
QC.1.trimmed.fastq.gz	QC.1.trimmed.fastq.gz	N.A. bp	L.hokkaidonensis_MiSeq_denovo

Download modified queries

- QC.1.trimmed.fastq.gz (Original size 189.4 MB)
- QC.2.trimmed.fastq.gz (Original size 189.6 MB)

Download wgs file

- out_WGS.fasta.gz (Original size 2.3 MB)

Assembly statistics

Contig # : 117
Total contig size : 2,356,019
Maximum contig size : 257,728
Minimum contig size : 101
N50 contig size : 92,304

①

Time

Wait time	Start time	End time
0: 0:9	2016-01-20 18:33:36	2016-01-21 10:10:06

Command	Start time	End time	Log1	Log2	Result	MD5
platanus assemble -m 120 -f QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-20 18:33:37	2016-01-21 10:08:51		View	Download(2.2 MB)	MD5
platanus scaffold -c out_contig.fa -b out_contigBubble.fa -IP1 QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-21 10:09:02	2016-01-21 10:09:12		View	Download(2.2 MB)	MD5
platanus gap_close -c out_scaffold.fa -IP1 QC.1.trimmed.fastq QC.2.trimmed.fastq	2016-01-21 10:09:23	2016-01-21 10:09:34		View	Download(2.2 MB)	MD5

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

k-mer解析(k=2)

ヒトゲノムを調べると、①CGという連続塩基の出現確率が他（特にCC, GC, GG）に比べて少ないという有名な話を調べる。②染色体番号。実際に行うことは、2連続塩基の出現頻度解析 (k=2のときのk-mer解析に相当)。2016.04.25の講義資料（スライド35~）をベースに作成

②　①

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
1	9.5%	5.0%	7.1%	7.4%	7.3%	5.4%	1.0%	7.1%	6.0%	4.4%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
2	10.0%	5.0%	7.0%	7.9%	7.2%	5.0%	0.9%	7.0%	5.9%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
3	10.1%	5.0%	6.9%	8.0%	7.2%	4.9%	0.8%	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
4	10.6%	5.0%	6.7%	8.5%	7.1%	4.5%	0.8%	6.7%	5.9%	3.8%	4.5%	5.0%	7.3%	5.8%	7.1%	10.6%
5	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	0.9%	6.9%	5.9%	4.0%	4.8%	5.1%	6.9%	5.9%	7.2%	10.3%
6	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	0.9%	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
7	9.8%	5.0%	7.0%	7.7%	7.3%	5.1%	1.0%	7.0%	6.0%	4.2%	5.1%	5.1%	6.5%	5.9%	7.3%	10.0%
8	10.0%	5.1%	6.9%	7.9%	7.2%	5.0%	0.9%	6.9%	6.0%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
9	9.7%	5.1%	7.0%	7.6%	7.3%	5.3%	1.0%	7.0%	6.0%	4.3%	5.3%	5.0%	6.4%	6.0%	7.3%	9.7%
10	9.6%	5.0%	7.1%	7.5%	7.3%	5.3%	1.0%	7.1%	6.0%	4.4%	5.3%	5.1%	6.3%	6.0%	7.4%	9.7%
11	9.5%	5.1%	7.1%	7.5%	7.3%	5.3%	1.0%	7.1%	6.1%	4.3%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
12	9.8%	5.0%	7.0%	7.7%	7.2%	5.1%	1.0%	7.0%	6.0%	4.2%	5.2%	5.1%	6.6%	6.0%	7.2%	9.9%
13	10.5%	5.0%	6.8%	8.4%	7.1%	4.5%	0.9%	6.7%	5.9%	3.8%	4.6%	5.0%	7.2%	5.8%	7.1%	10.6%
14	9.7%	5.0%	7.0%	7.7%	7.2%	5.1%	1.0%	7.0%	6.0%	4.2%	5.2%	5.1%	6.6%	5.9%	7.3%	9.9%
15	9.4%	5.1%	7.1%	7.3%	7.3%	5.4%	1.1%	7.1%	6.0%	4.5%	5.5%	5.1%	6.1%	6.0%	7.4%	9.5%
16	8.6%	5.1%	7.3%	6.7%	7.5%	6.1%	1.4%	7.2%	6.1%	5.0%	6.1%	5.1%	5.4%	6.1%	7.6%	8.8%
17	8.5%	5.1%	7.3%	6.4%	7.4%	6.3%	1.5%	7.4%	6.2%	5.1%	6.4%	5.0%	5.2%	6.1%	7.5%	8.6%
18	10.1%	5.1%	7.0%	7									6.7%	5.9%	7.3%	10.3%

Lander et al., *Nature*, **409**: 860-921, 2001

k-mer解析(k=2)

②平成27年度NGSハンズオン講習会(2015.07.29のスライド94～)で用いたhoge4.faを入力として、配列ごと(この場合コンティグごと)に16種類の2連続塩基の出現頻度解析の概念を説明します

①

イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings

Biostringsパッケージを用いて、multi-FASTA形式ファイルを読み込んで、"AA", "AC", "AG", "AT", "CA", "CC", "CG", "CT", "GA", "GC", "GG", "GT", "TA", "TC", "TG", "TT" の計$4^2 = 16$通りの2連続塩基の出現頻度を調べるやり方を示します。k-mer解析のk=2の 場合に相当します。ヒトゲノムで"CG"の 割合が 期待値よりも 低い(Lander et al., 2001; Saxonov et al., 2006)ですが、それを簡単に検証できます。
「ファイル」-「ディレクトリの 変更」で解析したいファイルを置いてあるディレクトリに移動し以下をコピペ。

1. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の 場合:

タイトル通りの 出現頻度です。

②

```
in_f <- "hoge4.fa"                    #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.txt"                  #

#必要なパッケージをロード
library(Biostrings)                   #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="
fasta                                 #

#本番
out <- dinucleotideFrequency(fasta)   #

#ファイルに保存
tmp <- cbind(names(fasta), out)       #
write.table(tmp, out_f, sep="\t", append
```

hoge4.fa - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
```

k-mer解析(k=2)

①例題1。②出力ファイルは、配列ごと(この場合コンティグごと)に16種類の2連続塩基の出現**頻度**をカウントしたもの。(作業ディレクトリにhoge4.faがないというエラーに何人か遭遇するだろうが、基本自力で解決せよ)

① ② 1. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られた

タイトル通りの出現頻度です。

```
in_f <- "hoge4.fa"                          #入力ファ
out_f <- "hoge1.txt"                        #出力ファ

#必要なパッケージをロード
library(Biostrings)                         #パッケー

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")
fasta                                       #確認して

#本番
out <- dinucleotideFrequency(fasta)         #連続塩基

#ファイルに保存
tmp <- cbind(names(fasta), out)             #保存した
write.table(tmp, out_f, sep="\t", append=F, quo
```

hoge4.fa - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
```

出力:hoge1.txt

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
contig_1	0	1	1	2	2	2	3	2	2	2	3	0	0	3	0	0
contig_2	4	6	9	1	11	11	5	6	4	9	10	8	1	8	6	3
contig_3	2	4	5	4	4	2	5	2	4	3	7	6	6	4	3	3
contig_4	3	6	2	3	5	3	3	4	3	3	1	2	3	2	4	1

k-mer解析(k=2)

①例題2。②出力ファイルは、配列ごとに2連続塩基の出現確率をカウントしたもの。③as.probオプションをTRUEにしているだけ

2. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の場合:

出現頻度ではなく、出現確率を得るやり方です。

```
in_f <- "hoge4.fa"                          #入力ファ
out_f <- "hoge2.txt"                        #出力ファ

#必要なパッケージをロード
library(Biostrings)                         #パッケー

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")
fasta                                       #確認して

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#

#ファイルに保存
tmp <- cbind(names(fasta), out)             #保存した
write.table(tmp, out_f, sep="\t", append=F, quo
```

hoge4.fa - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
```

出力:hoge2.txt

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
contig_1	0.0%	4.3%	4.3%	8.7%	8.7%	8.7%	13.0%	8.7%	8.7%	8.7%	13.0%	0.0%	0.0%	13.0%	0.0%	0.0%
contig_2	3.9%	5.9%	8.8%	1.0%	10.8%	10.8%	4.9%	5.9%	3.9%	8.8%	9.8%	7.8%	1.0%	7.8%	5.9%	2.9%
contig_3	3.1%	6.3%	7.8%	6.3%	6.3%	3.1%	7.8%	3.1%	6.3%	4.7%	10.9%	9.4%	9.4%	6.3%	4.7%	4.7%
contig_4	6.3%	12.5%	4.2%	6.3%	10.4%	6.3%	6.3%	8.3%	6.3%	6.3%	2.1%	4.2%	6.3%	4.2%	8.3%	2.1%

k-mer解析(k=2)

①例題7。②ヒトゲノムRパッケージを入力とすることもできます。③fastaオブジェクト以降は同じ。平成27年度講習会(2015.07.30のスライド40あたり)にもあり

2. イントロ | 一般 | ランダムな塩基配列を作成の4.を実行して得られたmulti-FASTAファイル(hoge4.fa)の場合:

出現頻度ではなく、出現確率を得るやり方です。

```
in_f <- "hoge4.fa"                    #入力ファイル名を指定してin_fに格納
out_f <- "hoge2.txt"                  #出力ファイル名を指定してout_fに格納

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f,
fasta

#本番
out <- dinucleotideFrequency(fast

#ファイルに保存
tmp <- cbind(names(fasta), out)
write.table(tmp, out_f, sep="\t",
```

①

7. ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

2013年12月にリリースされたGenome Reference Consortium GRCh38です。出力は出現確率です。

```
out_f <- "hoge7.txt"                  #出力ファイル名を指定してout_fに格納
param_bsgenome <- "BSgenome.Hsapiens.NCBI.GRCh38"#パッケージ名を指定(BSgenome系のゲノム

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み
library(param_bsgenome, character.only=T)#指定したパッケージの読み込み

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能なオブ
genome <- eval(parse(text=tmp))       #文字列tmpをRオブジェクトとしてgenomeに格納(パッ
fasta <- getSeq(genome)               #ゲノム塩基配列情報を抽出した結果をfastaに格納
names(fasta) <- seqnames(genome)      #description情報を追加している
fasta                                 #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#連続塩基の出現確率情報をoutに格納

#ファイルに保存
tmp <- cbind(names(fasta), out)       #保存したい情報をtmpに格納
write.table(tmp, out_f, sep="\t", append=F, quote=F, row.names=F)#tmpの中身を指定した
```

②

③

k-mer解析(k=2)

例題7実行結果ファイル。約3分。CGの連続塩基が他に比べて確かに低いことがわかる。出現確率は染色体ごとにばらつきがある。私ならbox plotを採用

出力：hoge7.txt

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
1	9.5%	5.0%	7.1%	7.4%	7.3%	5.4%	1.0%	7.1%	6.0%	4.4%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
2	10.0%	5.0%	7.0%	7.9%	7.2%	5.0%	0.9%	7.0%	5.9%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
3	10.1%	5.0%	6.9%	8.0%	7.2%	4.9%	0.8%	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
4	10.6%	5.0%	6.7%	8.5%	7.1%	4.5%	0.8%	6.7%	5.9%	3.8%	4.5%	5.0%	7.3%	5.8%	7.1%	10.6%
5	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	0.9%	6.9%	5.9%	4.0%	4.8%	5.1%	6.9%	5.9%	7.2%	10.3%
6	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	0.9%	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
7	9.8%	5.0%	7.0%	7.7%	7.3%	5.1%	1.0%	7.0%	6.0%	4.2%	5.1%	5.1%	6.5%	5.9%	7.3%	10.0%
8	10.0%	5.1%	6.9%	7.9%	7.2%	5.0%	0.9%	6.9%	6.0%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
9	9.7%	5.1%	7.0%	7.6%	7.3%	5.3%	1.0%	7.0%	6.0%	4.3%	5.3%	5.0%	6.4%	6.0%	7.3%	9.7%
10	9.6%	5.0%	7.1%	7.5%	7.3%	5.3%	1.0%	7.1%	6.0%	4.4%	5.3%	5.1%	6.3%	6.0%	7.4%	9.7%
11	9.5%	5.1%	7.1%	7.5%	7.3%	5.3%	1.0%	7.1%	6.1%	4.3%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
12	9.8%	5.0%	7.0%	7.7%	7.2%	5.1%	1.0%	7.0%	6.0%	4.2%	5.2%	5.1%	6.6%	6.0%	7.2%	9.9%
13	10.5%	5.0%	6.8%	8.4%	7.1%	4.5%	0.9%	6.7%	5.9%	3.8%	4.6%	5.0%	7.2%	5.8%	7.1%	10.6%
14	9.7%	5.0%	7.0%	7.7%	7.2%	5.1%	1.0%	7.0%	6.0%	4.2%	5.2%	5.1%	6.6%	5.9%	7.3%	9.9%
15	9.4%	5.1%	7.1%	7.3%	7.3%	5.4%	1.1%	7.1%	6.0%	4.5%	5.5%	5.1%	6.1%	6.0%	7.4%	9.5%
16	8.6%	5.1%	7.3%	6.7%	7.5%	6.1%	1.4%	7.2%	6.1%	5.0%	6.1%	5.1%	5.4%	6.1%	7.6%	8.8%
17	8.5%	5.1%	7.3%	6.4%	7.4%	6.3%	1.5%	7.4%	6.2%	5.1%	6.4%	5.0%	5.2%	6.1%	7.5%	8.6%
18	10.1%	5.1%	7.0%	7.9%	7.2%	4.7%	0.9%	6.9%	6.1%	4.0%	4.9%	5.1%	6.7%	5.9%	7.3%	10.3%

作図(box plot):基本形

②例題10。box plotをPNG形式ファイルで出力するやり方の基本形。エアーハンズオン(やったつもり)でよい

- イントロ | 一般 | 逆相補鎖(reverse complement)を取得 (last modified 2013/06/14)
- イントロ | 一般 | 逆鎖(reverse)を取得 (last modified 2013/06/14)
- イントロ | 一般 | k-mer解析 | k=1(塩基ごとの出現頻度解析) | Biostrings (last modified 2016/02/07)
- イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=3(3連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- (削除予定)イン
- (削除予定)イン
- (削除予定)イン
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般

①

イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings

Biostringsパッケージを用いて、multi-FASTA形式ファイルを読み込んで、"AA", "AC", "AG", "AT", "CA", "CC", "CG", "CT", "GA", "GC", "GG", "GT", "TA", "TC", "TG", "TT"の計4^2 = 16通りの2連続塩基の出現頻度を調べるやり方を示します。k-mer解析のk=2の場合に相当します。ヒトゲノムで"CG"の割合が期待値よりも低い(Lander et al., 2001; Saxonov et al., 2006)ですが、それを簡単に検証できます。

「ファイル」-「ディレクト

②

1. イントロ | 一般 | ランダム

タイトル通りの出現頻度で

```
in_f <- "hoge4.fa"
out_f <- "hoge1.tx

#必要なパッケージをロ
library(Biostrings

#入力ファイルの読み込
fasta <- readDNASt
fasta
```

10.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

7.と基本的に同じですが、box plotのPNGファイルも出力しています。

```
out_f1 <- "hoge10.txt"                    #出力ファイル名を指定してout_f1に格納
out_f2 <- "hoge10.png"                    #出力ファイル名を指定してout_f2に格納
param_bsgenome <- "BSgenome.Hsapiens.NCBI.GRCh38"#パッケージ名を指定(BSgenome系のゲノ
param_fig <- c(700, 400)                  #ファイル出力時の横幅と縦幅を指定(単位はピクセル

#必要なパッケージをロード
library(Biostrings)                       #パッケージの読み込み
library(param_bsgenome, character.only=T)#指定したパッケージの読み込み

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能なオ
genome <- eval(parse(text=tmp))           #文字列tmpをRオブジェクトとしてgenomeに格納(パッ
fasta <- getSeq(genome)                   #ゲノム塩基配列情報を抽出した結果をfastaに格納
names(fasta) <- seqnames(genome)          #description情報を追加している
fasta                                     #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#連続塩基の出現確率情報をoutに格納
```

作図(box plot):基本形

①例題10実行結果。②CGの連続塩基は、当然縦軸上で0.01付近に位置する。染色体全体の分布をbox plotで眺めても、確かに低い

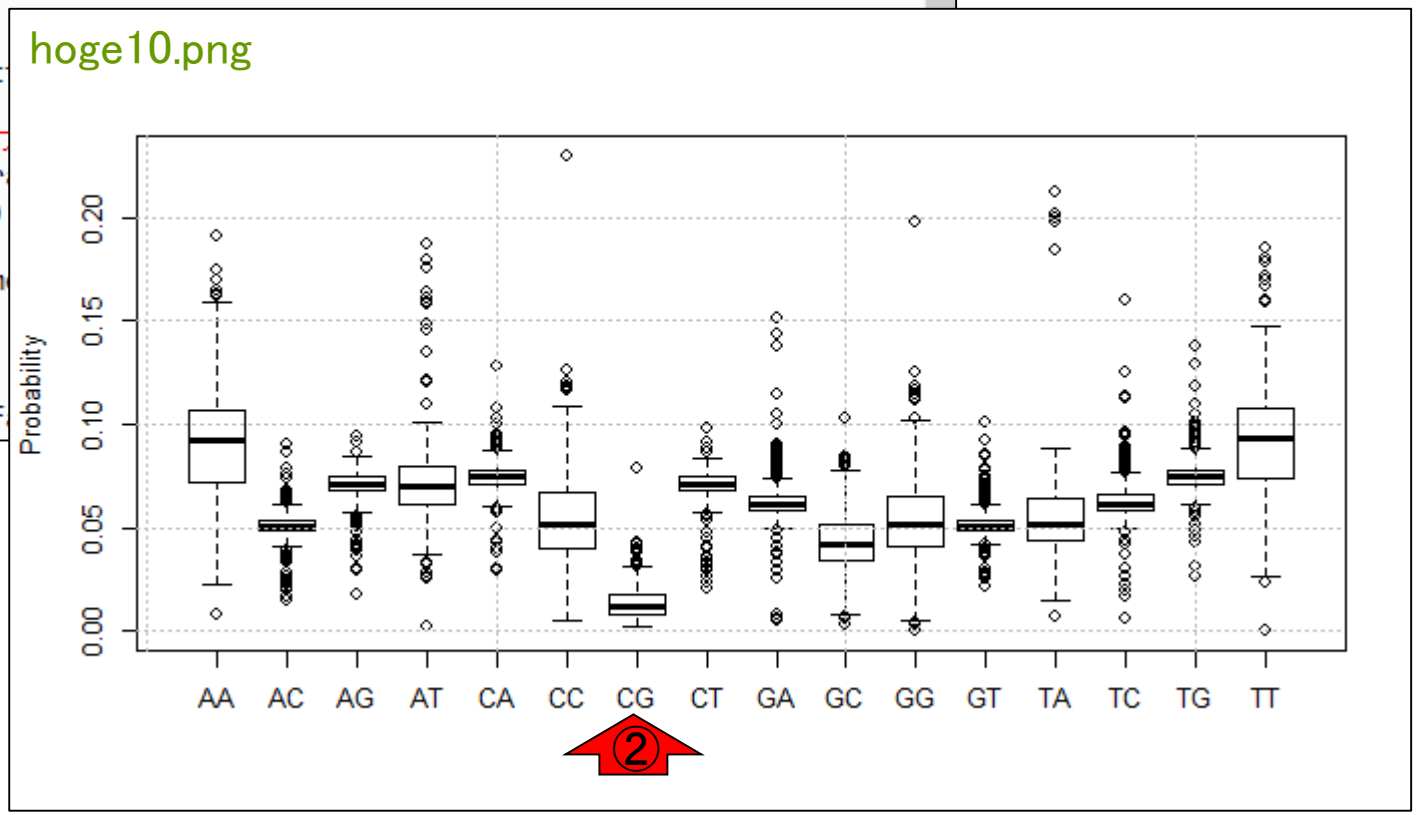
①

10.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

7.と基本的に同じですが、box plotのPNGファイルも出力しています。

```
out_f1 <- "hoge10.txt"                    #出力ファイル名を指定してout_f1に格納
out_f2 <- "hoge10.png"                    #出力ファイル名を指定してout_f2に格納
param_bsgenome <- "BSgenome.Hsapiens.NCBI.GRCh38"#パッケージ名を指定(BSgenome系のゲノ
param_fig <- c(700, 400)                  #ファイル出力時の横幅と縦幅を指定(単位はピクセル

#必要なパッケージをロード
library(Biostrings)
library(param_bsgenome, charac

#前処理(指定したパッケージ中のオ
tmp <- ls(paste("package", par
genome <- eval(parse(text=tmp)
fasta <- getSeq(genome)
names(fasta) <- seqnames(genom
fasta

#本番
out <- dinucleotideFrequency(f
```

hoge10.png



②作図（PNGファイル作成）部分のコード

作図(box plot)：基本形

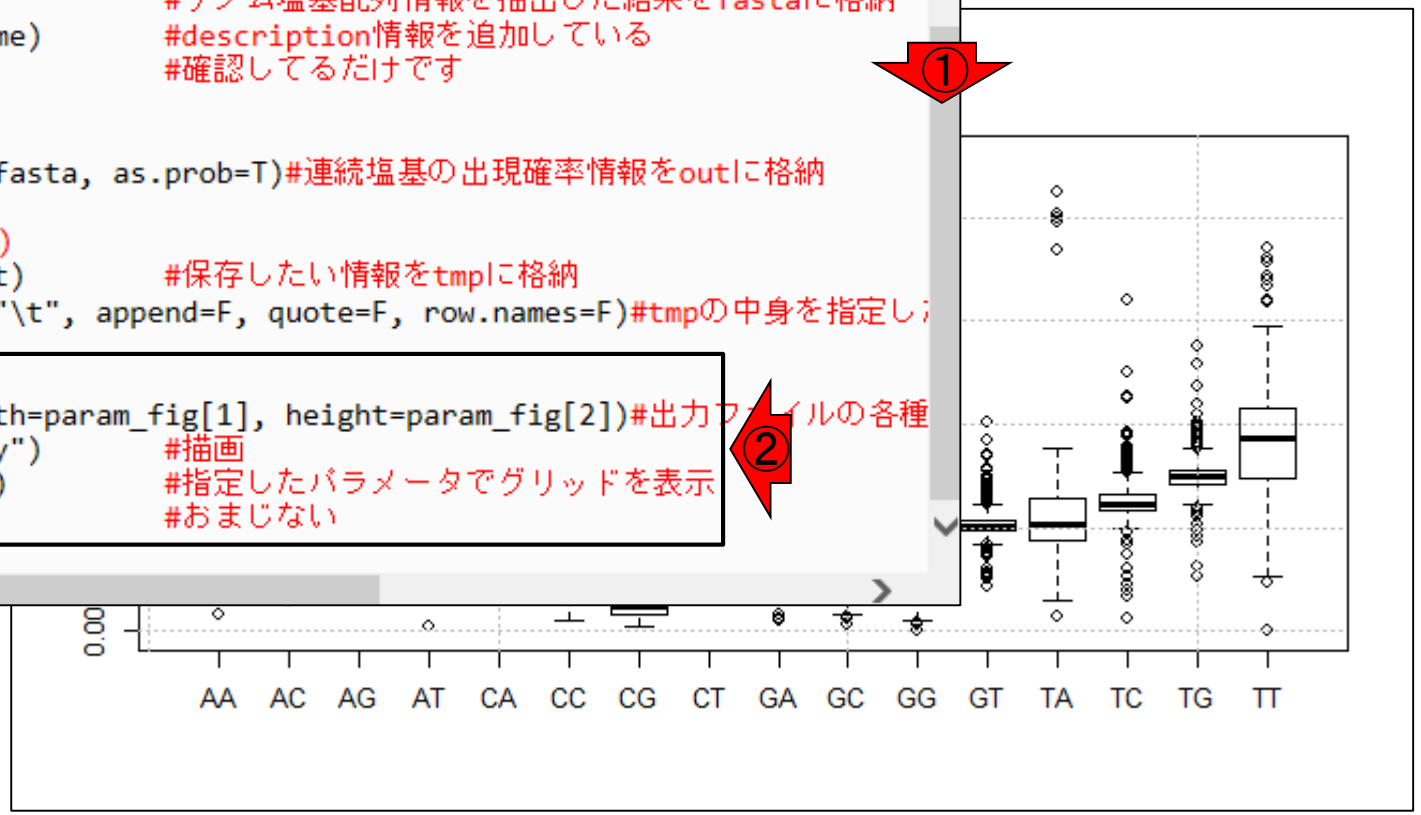
10.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

7.と基本的に同じですが、box plotのPNGファイルも出力しています。

```
library(param_bsgenome, character.only=T)#指定したパッケージの読み込み

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能なオ
genome <- eval(parse(text=tmp))          #文字列tmpをRオブジェクトとしてgenomeに格納(パッ
fasta <- getSeq(genome)                  #ゲノム塩基配列情報を抽出した結果をfastaに格納
names(fasta) <- seqnames(genome)         #description情報を追加している
fasta                                    #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#連続塩基の出現確率情報をoutに格納

#ファイルに保存(テキストファイル)
tmp <- cbind(names(fasta), out)          #保存したい情報をtmpに格納
write.table(tmp, out_f1, sep="\t", append=F, quote=F, row.names=F)#tmpの中身を指定し

#ファイルに保存(pngファイル)
png(out_f2, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイルの各種
boxplot(out, ylab="Probability")         #描画
grid(col="gray", lty="dotted")           #指定したパラメータでグリッドを表示
dev.off()                                #おまじない
```

①

②



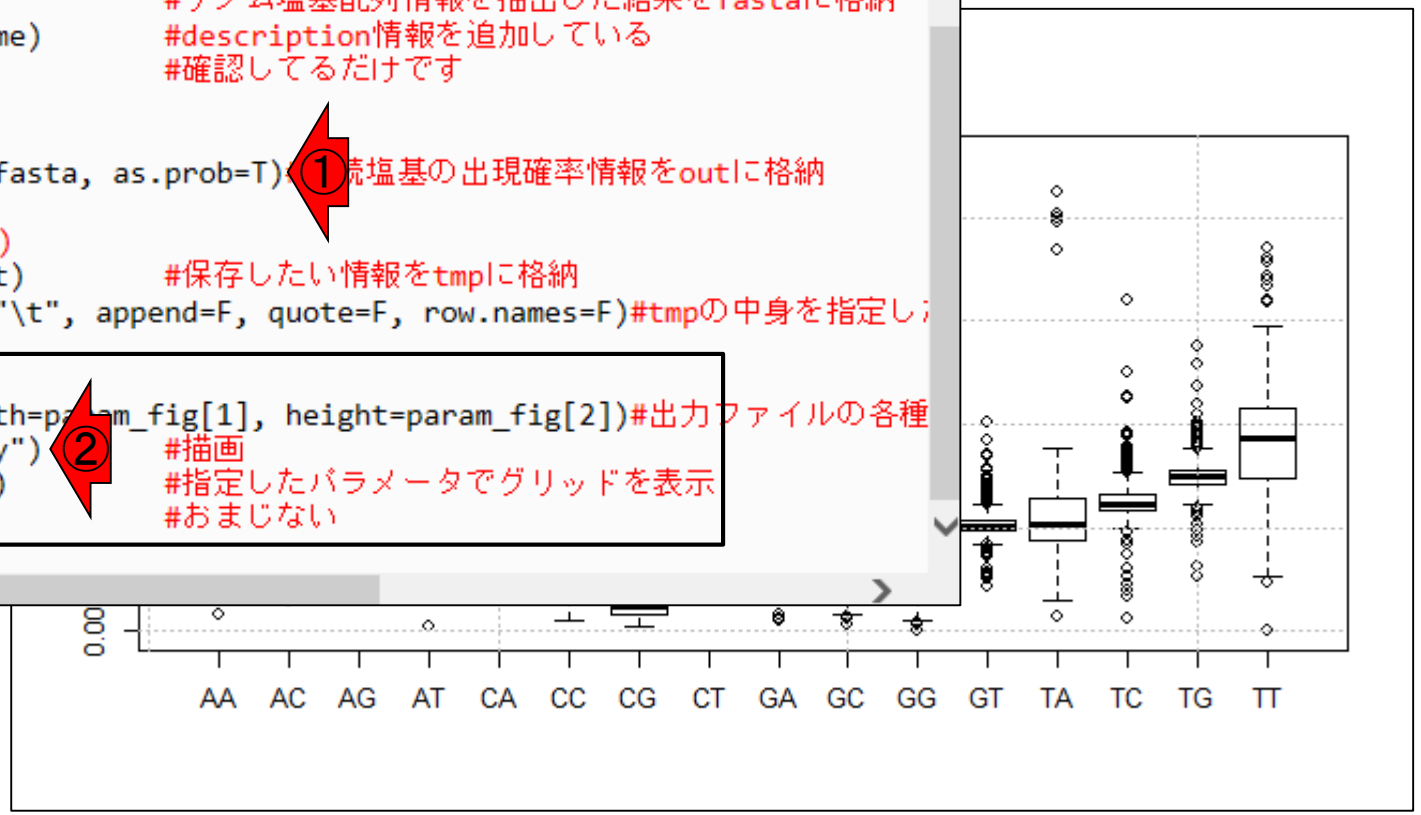
作図(box plot):基本形

①dinucleotideFrequency関数実行結果のoutオブジェクトを入力として、
②boxplot関数を実行しているだけ

10.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

7.と基本的に同じですが、box plotのPNGファイルも出力しています。

```
library(param_bsgenome, character.only=T)#指定したパッケージの読み込み

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能なオ
genome <- eval(parse(text=tmp))          #文字列tmpをRオブジェクトとしてgenomeに格納(パッ
fasta <- getSeq(genome)                  #ゲノム塩基配列情報を抽出した結果をfastaに格納
names(fasta) <- seqnames(genome)         #description情報を追加している
fasta                                    #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#①2連続塩基の出現確率情報をoutに格納

#ファイルに保存(テキストファイル)
tmp <- cbind(names(fasta), out)          #保存したい情報をtmpに格納
write.table(tmp, out_f1, sep="\t", append=F, quote=F, row.names=F)#tmpの中身を指定し

#ファイルに保存(pngファイル)
png(out_f2, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイルの各種
boxplot(out, ylab="Probability")         #②描画
grid(col="gray", lty="dotted")           #指定したパラメータでグリッドを表示
dev.off()                                #おまじない
```



注意点

②ヒトゲノムのGC含量は50%未満なので、塩基ごとの出現確率で見ると、A or TのほうがC or Gに比べて高い。そのため、2連続塩基の出現確率の期待値は③A or Tのみ、④C or Gのみ、⑤混合、の種類ごとに異なるという点に注意

① 解析したパッケージ名：BSgenome.Hsapiens.NCBI.GRCh38

② ヒトゲノムの全体のGC含量：約41%

各塩基(A, C, G, T)の出現確率：(0.295, 0.205, 0.205, 0.295)

③ AA, AT, TA, TTの出現確率の期待値 = 0.295 × 0.295 = 8.7%

④ CC, **CG**, GC, GGの出現確率の期待値 = 0.205 × 0.205 = 4.2%

⑤ AC, AG, CA, CT, GA, GT, TC, TGの出現確率の期待値 = 0.205 × 0.295 = 6.0%

期待値	8.7	6.0	6.0	8.7	6.0	4.2	**4.2**	6.0	6.0	4.2	4.2	6.0	8.7	6.0	6.0	8.7
連続塩基	AA	AC	AG	AT	CA	CC	**CG**	CT	GA	GC	GG	GT	TA	TC	TG	TT
1	9.5%	5.0%	7.1%	7.4%	7.3%	5.4%	**1.0%**	7.1%	6.0%	4.4%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
2	10.0%	5.0%	7.0%	7.9%	7.2%	5.0%	**0.9%**	7.0%	5.9%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
3	10.1%	5.0%	6.9%	8.0%	7.2%	4.9%	**0.8%**	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
4	10.6%	5.0%	6.7%	8.5%	7.1%	4.5%	**0.8%**	6.7%	5.9%	3.8%	4.5%	5.0%	7.3%	5.8%	7.1%	10.6%
5	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	**0.9%**	6.9%	5.9%	4.0%	4.8%	5.1%	6.9%	5.9%	7.2%	10.3%
6	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	**0.9%**	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
7	9.8%	5.0%	7.0%	7.7%	7.3%	5.1%	**1.0%**	7.0%	6.0%	4.2%	5.1%	5.1%	6.5%	5.9%	7.3%	10.0%
8	10.0%	5.1%	6.9%	7.9%	7.2%	5.0%	**0.9%**	6.9%	6.0%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
9	9.7%	5.1%	7.0%	7.6%	7.3%	5.3%	**1.0%**	7.0%	6.0%	4.3%	5.3%	5.0%	6.4%	6.0%	7.3%	9.7%
10	9.6%	5.0%	7.1%	7.5%	7.3%	5.3%	**1.0%**	7.1%	6.0%	4.4%	5.3%	5.1%	6.3%	6.0%	7.4%	9.7%
11	9.5%	5.1%	7.1%	7.5%	7.3%	5.3%	**1.0%**	7.1%	6.1%	4.3%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%

作図(box plot):色づけ

①例題11。②colorという列名のところに2連続塩基の種類ごとに色を指定した、③タブ区切りファイル(human_2mer.txt)を与えて利用。④このファイルの情報を利用しているのは、コードの下のほう

11.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

10.と基本的に同じですが、連続塩基の種類ごとの期待値とボックスプロット(box plot)上での
ファイル(human_2mer.txt)を入力として利用し、色情報のみを取り出して利用しています。

```
in_f <- "human_2mer.txt"                #入力ファイル名を指定してin_fに格納
out_f1 <- "hoge11.txt"                  #出力ファイル名を指定してout_f1に格納
out_f2 <- "hoge11.png"                  #出力ファイル名を指定してout_f2に格納
param_bsgenome <- "BSgenome.Hsapiens.NCBI.GRCh38"#パッケージ名を指定(BSgenome系の
param_fig <- c(700, 400)                #ファイル出力時の横幅と縦幅を指定(単位はピク

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み
library(param_bsgenome, character.only=T)#指定したパッケージの読み込み

#入力ファイルの読み込み
data <- read.table(in_f, header=TRUE, row.names=1, sep="\t", quote="")#in_fで指定

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能な
genome <- eval(parse(text=tmp))         #文字列tmpをRオブジェクトとしてgenomeに格納
fasta <- getSeq(genome)                 #ゲノム塩基配列情報を抽出した結果をfastaに格
names(fasta) <- seqnames(genome)        #description情報を追加している
fasta                                   #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#連続塩基の出現確率情報をoutに格納
```

type	expected	color
AA	0.087025	red
AC	0.060475	skyblue
AG	0.060475	skyblue
AT	0.087025	red
CA	0.060475	skyblue
CC	0.042025	black
CG	0.042025	black
CT	0.060475	skyblue
GA	0.060475	skyblue
GC	0.042025	black
GG	0.042025	black
GT	0.060475	skyblue
TA	0.087025	red
TC	0.060475	skyblue
TG	0.060475	skyblue
TT	0.087025	red

作図(box plot):色づけ

①boxplot関数実行時の②colオプション部分で③color列の情報を利用していることがわかる。④expected列情報は、例題11では利用していない

11.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

10.と基本的に同じですが、連続塩基の種類ごとの期待値とボックスプロット(box plot)上での色情報を含むファイル(human_2mer.txt)を入力として利用し、色情報のみを取り出して利用しています。

```
data <- read.table(in_f, header=TRUE, row.names=1, sep="\t", quote="")#in_fで指定

#前処理(指定したパッケージ中のオブジェクト名をgenomeに統一)
tmp <- ls(paste("package", param_bsgenome, sep=":"))#指定したパッケージで利用可能な
genome <- eval(parse(text=tmp))          #文字列tmpをRオブジェクトとしてgenomeに格納
fasta <- getSeq(genome)                  #ゲノム塩基配列情報を抽出した結果をfastaに格
names(fasta) <- seqnames(genome)         #description情報を追加している
fasta                                    #確認してるだけです

#本番
out <- dinucleotideFrequency(fasta, as.prob=T)#連続塩基の出現確率情報をoutに格納

#ファイルに保存(テキストファイル)
tmp <- cbind(names(fasta), out)          #保存したい情報をtmpに格納
write.table(tmp, out_f1, sep="\t", append=F, quote=F, row.names=F)#tmpの中身を指

#ファイルに保存(pngファイル)
png(out, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイルの
boxplot(out, ylab="Probability", col=as.character(data$color))#描画
grid(col="gray", lty="dotted")           #指定したパラメータでグリッドを表示
dev.off()                                #おまじない
```

type	expected	color
AA	0.087025	red
AC	0.060475	skyblue
AG	0.060475	skyblue
AT	0.087025	red
CA	0.060475	skyblue
CC	0.042025	black
CG	0.042025	black
CT	0.060475	skyblue
GA	0.060475	skyblue
GC	0.042025	black
GG	0.042025	black
GT	0.060475	skyblue
TA	0.087025	red
TC	0.060475	skyblue
TG	0.060475	skyblue
TT	0.087025	red

作図(box plot):色づけ

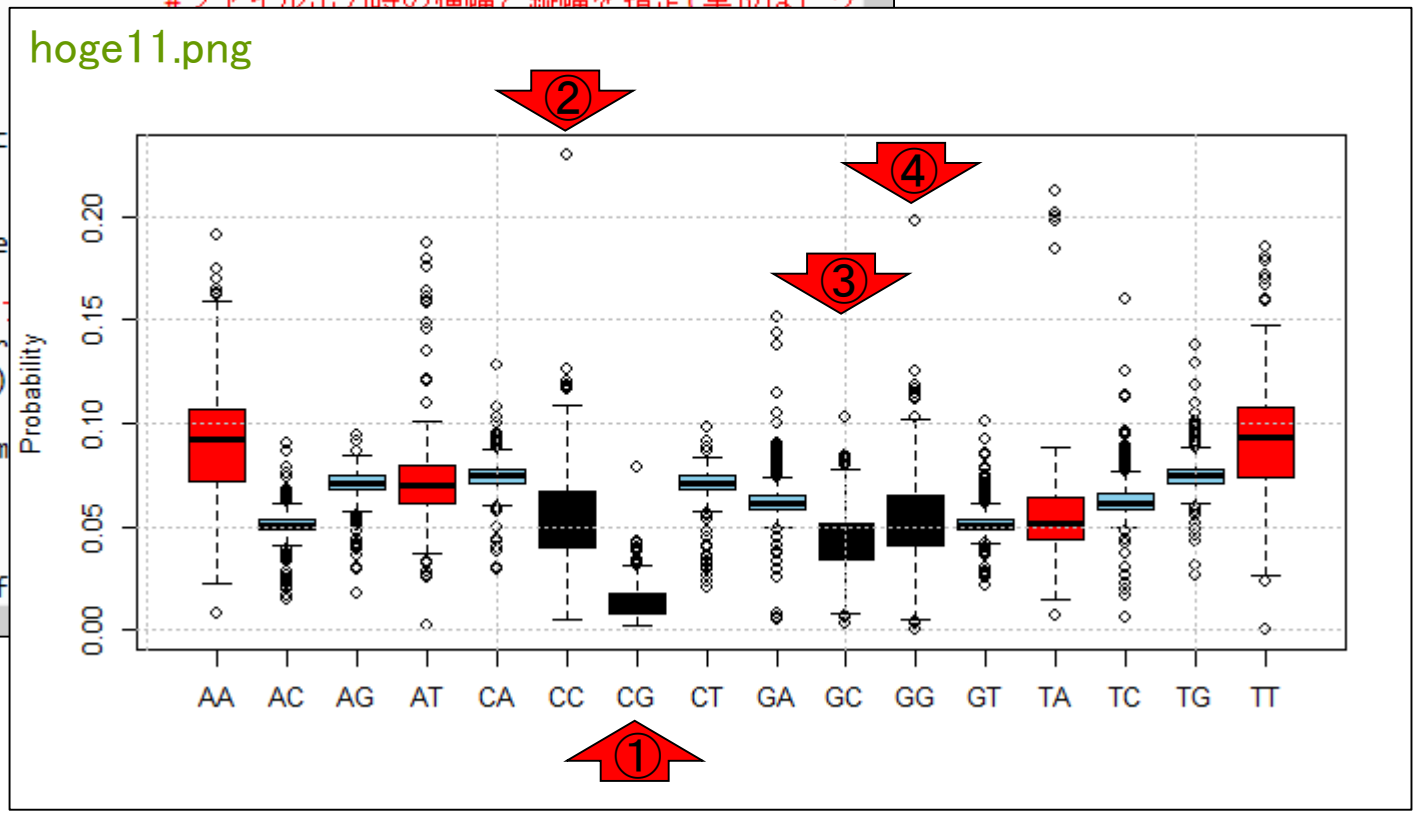
色づけできている。同じ期待値(4.2%)の4種類間で比較しても、①CGは、他の②CC, ③GC, ④GGと比べて低い

11.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

10.と基本的に同じですが、連続塩基の種類ごとの期待値とボックスプロット(box plot)上での色情報を含むファイル(human_2mer.txt)を入力として利用し、色情報のみを取り出して利用しています。

```
in_f <- "human_2mer.txt"                    #入力ファイル名を指定してin_fに格納
out_f1 <- "hoge11.txt"                      #出力ファイル名を指定してout_f1に格納
out_f2 <- "hoge11.png"                      #出力ファイル名を指定してout_f2に格納
param_bsgenome <- "BSgenome.Hsapiens.NCBI.GRCh38"#パッケージ名を指定(BSgenome系の
param_fig <- c(700, 400)                    #ファイル出力時の横幅と縦幅を指定(単位はピク

#必要なパッケージをロード
library(Biostrings)
library(param_bsgenome, charac

#入力ファイルの読み込み
data <- read.table(in_f, heade

#前処理(指定したパッケージ中のオ
tmp <- ls(paste("package", par
genome <- eval(parse(text=tmp))
fasta <- getSeq(genome)
names(fasta) <- seqnames(genom
fasta

#本番
out <- dinucleotideFrequency(f
```

hoge11.png



400 pixels

700 pixels

期待値との差分

2連続塩基の出現確率の期待値は③A or Tのみ、④C or Gのみ、⑤混合、の種類ごとに異なるため、期待値を揃えてbox plotを描きたい

① 解析したパッケージ名：BSgenome.Hsapiens.NCBI.GRCh38

② ヒトゲノムの全体のGC含量：約41%

各塩基(A, C, G, T)の出現確率：(0.295, 0.205, 0.205, 0.295)

③ AA, AT, TA, TTの出現確率の期待値 = 0.295 × 0.295 = 8.7%

④ CC, **CG**, GC, GGの出現確率の期待値 = 0.205 × 0.205 = 4.2%

⑤ AC, AG, CA, CT, GA, GT, TC, TGの出現確率の期待値 = 0.205 × 0.295 = 6.0%

期待値	8.7	6.0	6.0	8.7	6.0	4.2	**4.2**	6.0	6.0	4.2	4.2	6.0	8.7	6.0	6.0	8.7
連続塩基	AA	AC	AG	AT	CA	CC	**CG**	CT	GA	GC	GG	GT	TA	TC	TG	TT
1	9.5%	5.0%	7.1%	7.4%	7.3%	5.4%	**1.0%**	7.1%	6.0%	4.4%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%
2	10.0%	5.0%	7.0%	7.9%	7.2%	5.0%	**0.9%**	7.0%	5.9%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
3	10.1%	5.0%	6.9%	8.0%	7.2%	4.9%	**0.8%**	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
4	10.6%	5.0%	6.7%	8.5%	7.1%	4.5%	**0.8%**	6.7%	5.9%	3.8%	4.5%	5.0%	7.3%	5.8%	7.1%	10.6%
5	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	**0.9%**	6.9%	5.9%	4.0%	4.8%	5.1%	6.9%	5.9%	7.2%	10.3%
6	10.2%	5.0%	6.9%	8.1%	7.2%	4.8%	**0.9%**	6.9%	5.9%	4.0%	4.9%	5.0%	6.9%	5.9%	7.2%	10.2%
7	9.8%	5.0%	7.0%	7.7%	7.3%	5.1%	**1.0%**	7.0%	6.0%	4.2%	5.1%	5.1%	6.5%	5.9%	7.3%	10.0%
8	10.0%	5.1%	6.9%	7.9%	7.2%	5.0%	**0.9%**	6.9%	6.0%	4.1%	5.0%	5.0%	6.7%	5.9%	7.2%	10.0%
9	9.7%	5.1%	7.0%	7.6%	7.3%	5.3%	**1.0%**	7.0%	6.0%	4.3%	5.3%	5.0%	6.4%	6.0%	7.3%	9.7%
10	9.6%	5.0%	7.1%	7.5%	7.3%	5.3%	**1.0%**	7.1%	6.0%	4.4%	5.3%	5.1%	6.3%	6.0%	7.4%	9.7%
11	9.5%	5.1%	7.1%	7.5%	7.3%	5.3%	**1.0%**	7.1%	6.1%	4.3%	5.4%	5.0%	6.3%	6.0%	7.3%	9.6%

作図(box plot):発展形

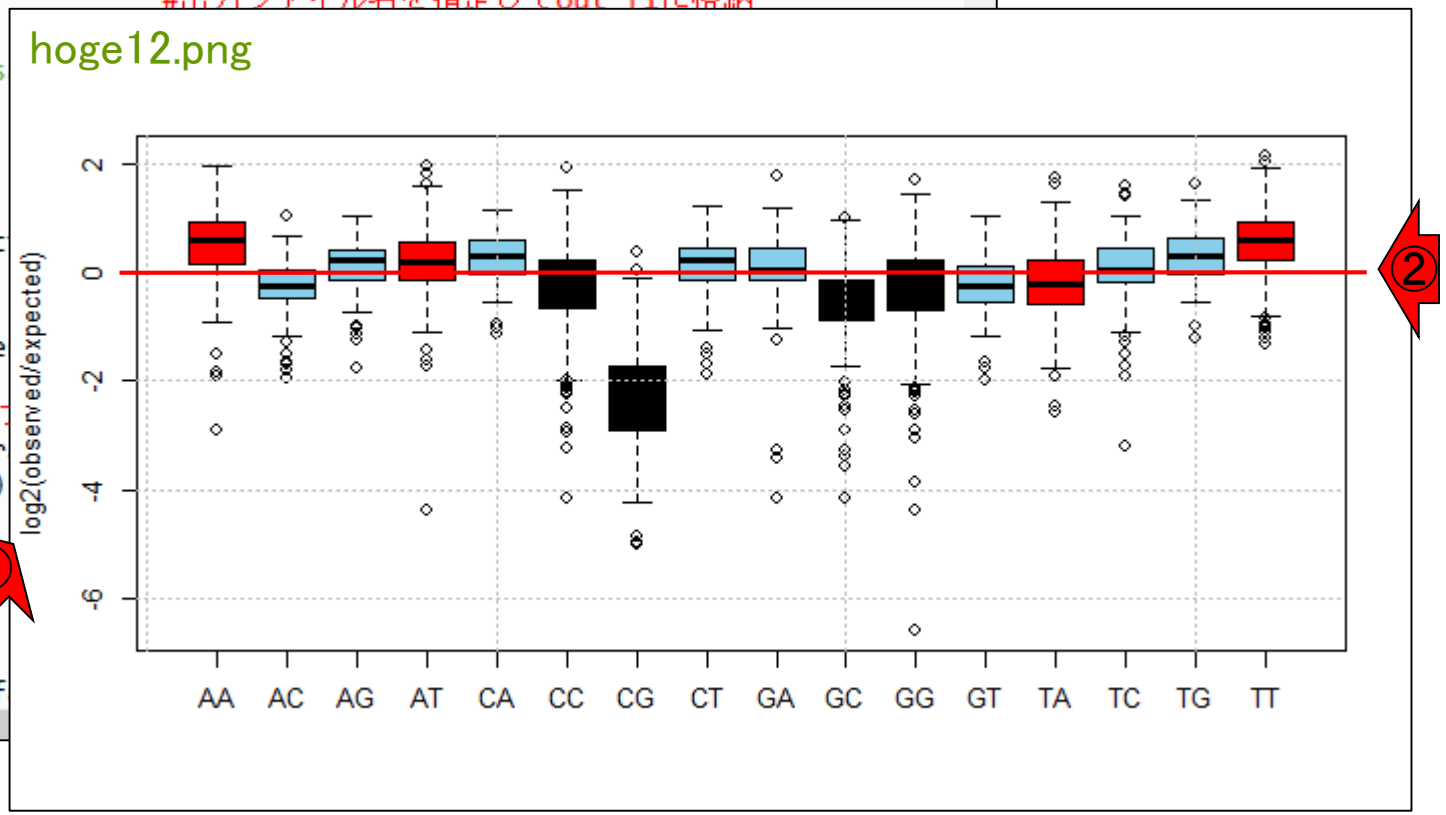
期待値との差分を評価すべく、①縦軸をlog(観測値/期待値)としてプロット。②0付近にある2連続塩基は、観測値(実測された出現確率)が期待値とほぼ同じことを意味する。この縦軸のような表現方法は一般的です

12.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

11.と基本的に同じですが、human_2mer.txtというファイルを入力として与えて、連続塩
クスプロット(box plot)上での色情報を利用しています。また、重要なのは期待値からの
待値(expected)と同程度の観測値(observed)であればゼロ、観測値のほうが大きければプラス、観測値のほうが小
さければマイナス」といった具合で表現したほうがスマートです。それゆえ、box plotの縦軸をlog(observed/expected)
として表現しています。CG以外の連続塩基は縦軸上でが0近辺に位置していることが分かります。

```
in_f <- "human_2mer.txt"                    #入力ファイル名を指定してin_fに格納
out_f1 <- "hoge12.txt"                      #出力ファイル名を指定してout_f1に格納
out_f2 <- "hoge12.png"
param_bsgenome <- "BSgenome.Hs
param_fig <- c(700, 400)

#必要なパッケージをロード
library(Biostrings)
library(param_bsgenome, charac

#入力ファイルの読み込み
data <- read.table(in_f, heade

#前処理(指定したパッケージ中のオ
tmp <- ls(paste("package", par
genome <- eval(parse(text=tmp)
fasta <- getSeq(genome)
names(fasta) <- seqnames(ge
fasta

#本番
out <- dinucleotideFrequency(f
```

hoge12.png



作図(box plot):発展形

①コピペ実行時に警告メッセージが出た。これは、どこかに-Inf(マイナス無限大のこと)があり、それはプロットされないということ。
②プロット対象はlogratioオブジェクトであり、
③でoutオブジェクトを期待値(data$expected)で割ることで作成している

12.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

11.と基本的に同じですが、human_2mer.txtというファイルを入力として与えて、連続塩
クスプロット(box plot)上での色情報を利用しています。また、重要なのは期待値からの
待値(expected)と同程度の観測値(observed)であればゼロ、観測値のほうが大きければ
さければマイナス」といった具合で表現したほうがスマートです。それゆえ、box plotの縦軸
として表現しています。CG以外の連続塩基は縦軸上でが0近辺に位置していることが分かります。

```
in_f <- "human_2mer.txt"                    #入力ファイル名を指定してin_fに格納
out_f1 <- "hoge12.txt"
out_f2 <- "hoge12.png"
param_bsgenome <- "BSgenome.Hsapiens
param_fig <- c(700, 400)

#必要なパッケージをロード
library(Biostrings)
library(param_bsgenome, character.on

#入力ファイルの読み込み
data <- read.table(in_f, header=TRUE

#前処理(指定したパッケージ中のオブジェク
tmp <- ls(paste("package", param_bsg
genome <- eval(parse(text=tmp))
fasta <- getSeq(genome)
names(fasta) <- seqnames(genome)
fasta

#本番
out <- dinucleotideFrequency(fasta,
```

R Console

```
> logratio <- log2(out/data$expected)     #log(observed/expe$
>
> #ファイルに保存(テキストファイル)
> tmp <- cbind(names(fasta), logratio)    #保存したい情報をt$
> write.table(tmp, out_f1, sep="\t", append=F, quote=F, row$
>
> #ファイルに保存(pngファイル)
> png(out_f2, poi  ize=13, width=param_fig[1], height=para$
> boxplot(logratio, ylab="log2(observed/expected)",#描画
+         col=as.character(data$color))   #描画
 警告メッセージ:
 bplt(at[i], wid = width[i], stats = z$stats[, i], out = z$$
  Outlier (-Inf) in boxplot 16 is not drawn
> grid(col="gray", lty="dotted")          #指定したパラメ-$
> dev.off()                               #おまじない
null device
          1
>
```

作図(box plot):発展形

12.ヒトゲノム配列パッケージ(BSgenome.Hsapiens.NCBI.GRCh38)の場合:

11.と基本的に同じですが、human_2mer.txtというファイルを入力として与えて、連
クスプロット(box plot)上での色情報を利用しています。また、重要なのは期待値か
待値(expected)と同程度の観測値(observed)であればゼロ、観測値のほうが大き
さければマイナス」といった具合で表現したほうがスマートです。それゆえ、box pl
として表現しています。CG以外の連続塩基は縦軸上でが0近辺に位置しているこ

```
in_f <- "human_2mer.txt"                    #入力ファイル名を指定してin_fに格納
out_f1 <- "hoge12.txt"
out_f2 <- "hoge12.png"
param_bsgenome <- "BSgenome.Hsapiens
param_fig <- c(700, 400)

#必要なパッケージをロード
library(Biostrings)
library(param_bsgenome, character.on

#入力ファイルの読み込み
data <- read.table(in_f, header=T

#前処理(指定したパッケージ中のオブジ
tmp <- ls(paste("package", param_bsg
genome <- eval(parse(text=tmp))
fasta <- getSeq(genome)
names(fasta) <- seqnames(genome)
fasta

#本番
out <- dinucleotideFrequency(fast
```

①

R Console

```
> apply(logratio, MARGIN=2, max)
       AA        AC        AG        AT        CA        CC
1.9553181 1.0365555 1.0380107 1.9660966 1.1534243 1.9262110
       CG        CT        GA        GC        GG        GT
0.3787232 1.2207788 1.7721007 1.0111684 1.7080309 1.0141437
       TA        TC        TG        TT
1.7346511 1.5812906 1.0247335 2.1372668
> apply(logratio, MARGIN=2, min)
       AA        AC        AG        AT        CA        CC
-2.906679 -1.947401 -1.766627 -4.381588 -1.123647 -4.159398
       CG        CT        GA        GC        GG        GT
-5.021894 -1.888509 -4.157054 -4.172619 -6.606857 -1.981918
       TA        TC        TG        TT
-2.574233 -3.219269 -1.207795      -Inf
> dim(logratio)
[1] 455  16
>
```

②

③

-Inf(マイナス無限大のこと)なので、①apply関数でlogratioオブジェクトの②各列(MARGIN=2)に対してmin関数を実行し、どの列に-Infがあったかを確認しているだけ。③logratioの行数と列数を表示。ヒトゲノムとはいえ、断片を合わせると455配列あるため、TTの実測値が0ということもあるのだろう。スライドを見るだけ

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

ランダムデータ生成

この後に用いるランダムな塩基配列、および仮想NGSデータの作成について解説します。①サンプルデータの、②例題32。コピペはスライド75で行う

- インストール | Rパッケージ | 必要最小限(数GB?!) (last modified 2015/05/25)
- インストール | Rパッケージ | 個別 (last modified 2015/06/10)
- (削除予定)Rのインストールと起動 (last modified 2016/02/21)
- (削除予定)個別パッケージのインストール (last modified 2015/02/20)
- 基本的な利用法 (last modified 2015/04/03)
- サンプルデータ (last modified 2015/06/15)
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2016
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2015
- バイオインフォマテ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ

①

サンプルデータ

1. Illumina/36bp/single-end/human (SRA000299) data (Marioni et al., Genome Res., 2008)
   「Kidney 7 samples vs Liver 7 samples」のRNA-seqの遺伝子発現行列データ(SupplementaryTable2.txt)です。サンプルは二つの濃度(1.5 pM and 3 pM)でシーケンスされており、「3 pMのものが5 samples vs. 5 samples」、「1.5 pMのものが
   に発現データ
   す):

   7列目：R1L1K
   8列目：R1L2L
   9列目：R1L3K
   10列目：R1L4
   11列目：R1L6
   12列目：R1L7
   13列目：R1L8

②

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGIset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
```

ランダムデータ生成

①例題32。ここではまず②A, C, G, Tの並びで、③塩基の存在比率が22, 28, 28, 22のランダム塩基配列を生成させようとしている。④最終的に得られるのは2つのFASTA形式ファイル

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

仮想リファレンス

①はsingle-FASTA形式の仮想リファレンス(ゲノム)ファイル。②配列長は50 bp、③description部分はkkk。④この部分で作成しており、referenceという名前のオブジェクトを最後に①のファイル名で出力している

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sa

50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リードからランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

仮想リファレンス

赤枠部分をコピペ。①ランダムデータ生成と言いつつ、得られるreferenceオブジェクトは、②全員同じGAA…GCAという配列になっているはず。この理由は③set.seed関数で1010という同じタネ番号を指定しているから。この数字はなんでもよい

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample3
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列
す。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています
sample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成から と基本的に同じです。

```
out_f1 <- "sample32_ref.f
out_f2 <- "sample32_ngs.f
param_len_ref <- 50
narabi <- c("A","C","G","
param_composition <- c(2
param_len_ngs <- 20
param_num_ngs <- 10
param_desc <- "kkk"

#必要なパッケージをロード
library(Biostrings)

#本番(リファレ  ③ 配列生成)
set.seed(1010)
ACGTset <- rep(narabi, pa
reference <- paste(sample
reference <- DNAStringSet
names(reference) <- para
reference

#本番(シミュレーションデー
s_posi <- sample(1:(paran
```

①

①

R Console

```
    setdiff, sort, table, tapply, union, unique, unlist,
    unsplit

 要求されたパッケージ S4Vectors をロード中です
 要求されたパッケージ stats4 をロード中です
 要求されたパッケージ IRanges をロード中です
 要求されたパッケージ XVector をロード中です
>
> #本番(リファレンス配列生成)
> set.seed(1010)                                  #おまじない(同じ乱数にな$
> ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam$
> reference <- paste(sample(ACGTset, param_len_ref, replace=T), c$
> reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させ$
> names(reference) <- param_desc          #description行に相当する$
> reference                               #確認してるだけです
  A DNAStringSet instance of length 1
    width seq                                         names
[1]    50 GAAGTTGTCTTTTGAA...GAAGACGCGGACGGCA kkk
>
```

②

仮想NGSリード

①はmulti-FASTA形式の仮想NGSリードファイル。②長さは20 bp、③リード数は10。これらのリードは、仮想リファレンス(ゲノム)ファイルとなるreferenceオブジェクトから乱数を発生させて、部分配列として切り出している。④下のほうに移動

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を
す。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。
sample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

仮想NGSリード

①sample関数で乱数を発生させて、得られた乱数s_posiを②subseq関数でreferenceから切り出すときの④start位置として利用している。⑤20 bpというparam_len_ngsはwidthオプションとして与えている

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(samp
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10データ分だけランダム抽出するプログラムです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
①s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
s_posi                                  #確認してるだけです
hoge <- NULL                            ③#最終的に得る結果を格納するためのプレースホルダhoge
for(i in 1:length(s_posi)){             #length(s_posi)回だけループを回す⑤
    hoge <- append(hoge, subseq(reference, start=s_posi[i], width=param_len_ngs))#sub
}
②fasta <- hoge                          ④#hogeの中身をfastaに格納

#後処理(description部分の作成)
description <- paste(param_desc, s_posi, (s_posi+param_len_ngs-1), sep="_")#param_des
names(fasta) <- description             #description行に相当する記述を追加している
fasta                                   #確認してるだけです

#ファイルに保存(仮想リファレンス配列と仮想NGS配列)
writeXStringSet(reference, file=out_f1, format="fasta", width=50)#referenceの中身を指定
writeXStringSet(fasta, file=out_f2, format="fasta", width=50)#fastaの中身を指定したファ
```

仮想NGSリード

①出力ファイルのdescription部分にリファレンス配列のどの領域由来リードかを書き込んでいる

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
s_posi                                  #確認してるだけです
hoge <- NULL                            #最終的に得る結果を格納するためのプレースホルダhoge
for(i in 1:length(s_posi)){             #length(s_posi)回だけループを回す
    hoge <- append(hoge, subseq(reference, start=s_posi[i], width=param_len_ngs))#sub
}
fasta <- hoge                           #hogeの中身をfastaに格納

#後処理(description部分の作成)
description <- paste(param_desc, s_posi, (s_posi+param_len_ngs-1), sep="_")#param_des
names(fasta) <- description             #description行に相当する記述を追加している
fasta                                   #確認してるだけです

#ファイルに保存(仮想リファレンス配列と仮想NGS配列)
writeXStringSet(reference, file=out_f1, format="fasta", width=50)#referenceの中身を指定
writeXStringSet(fasta, file=out_f2, format="fasta", width=50)#fastaの中身を指定したファ
```

①

最後までコピペ実行すると、こんな感じになります

最後までコピペ

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
reference <- DNAStringSet
names(reference) <- param
reference

#本番(シミュレーションデー
s_posi <- sample(1:(param
s_posi
hoge <- NULL
for(i in 1:length(s_posi)
    hoge <- append(hoge,
}
fasta <- hoge

#後処理(description部分の
description <- paste(para
names(fasta) <- descripti
fasta

#ファイルに保存(仮想リファ
writeXStringSet(reference
writeXStringSet(fasta, fi
```

R Console

```
> names(fasta) <- description                #description行に相当する$
> fasta                                      #確認してるだけです
  A DNAStringSet instance of length 10
     width seq                                    names
 [1]    20 CACCAGGACATGAAGACGCG                   kkk_24_43
 [2]    20 AGGACATGAAGACGCGGACG                   kkk_28_47
 [3]    20 TTGAACTCACTACACCAGGA                   kkk_12_31
 [4]    20 GTTGTCTTTTGAACTCACTA                   kkk_4_23
 [5]    20 TTGTCTTTTGAACTCACTAC                   kkk_5_24
 [6]    20 GTCTTTTGAACTCACTACAC                   kkk_7_26
 [7]    20 TTGAACTCACTACACCAGGA                   kkk_12_31
 [8]    20 ACACCAGGACATGAAGACGC                   kkk_23_42
 [9]    20 GAAGTTGTCTTTTGAACTCA                   kkk_1_20
[10]    20 AGTTGTCTTTTGAACTCACT                   kkk_3_22
>
> #ファイルに保存(仮想リファレンス配列と仮想NGS配列)
> writeXStringSet(reference, file=out_f1, format="fasta", width=5$
> writeXStringSet(fasta, file=out_f2, format="fasta", width=50)#f$
>
```

出力ファイル

得られる出力ファイルはこんな感じ。
①仮想リファレンスゲノムは②50 bp

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

①

②

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Bios

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

①

```
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA
```

出力ファイル

①multi-FASTA形式の仮想NGSリードファイル。②長さは20 bp、③リード数は10

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

①

```
>kkk_24_43
CACCAGGACATGAAGACGCG
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

①

出力ファイル

仮想NGSリードファイルの①description部分を見れば、②そのリードが③仮想リファレンス配列中のどの領域由来かがわかる

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverage相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Bios
```

③
```
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA
```

```
#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

①
②
```
>kkk_24_43
CACCAGGACA[illegible]AGACGCG
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

出力ファイル

総塩基数は仮想リファレンス配列が50 bp、仮想NGSリードが20 bp × 10リード = 200 bp。NGSデータはリファレンスの4倍の塩基数、つまり**4X coverage**。そしてsequence errorはゼロという解釈が可能

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Bios
```

```
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA
```

```
#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

```
>kkk_24_43
CACCAGGACATGAAGACGCG
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

①大きめのk値を用いたアセンブリ関連の話に移行

k-mer解析

- 比較ゲノム解析
  - k=3 or 4付近の値を用いてゲノムごとの頻度情報を取得し、類似性尺度として利用
- アセンブリ（ゲノムやトランスクリプトーム）
  - k=25～200付近の値を用いてde Bruijnグラフを作成
  - k-mer頻度グラフを作成して眺め、Heterozygosityの有無などを調査

- モチーフ解析
  - 転写開始点の上流配列解析。古細菌の上流50塩基に絞ってk=4で出現頻度解析すると、おそらくTATAが上位にランクイン
- 発現量推定
  - RNA-seq解析で、リファレンスにリードをマップしてリード数をカウントするのが主流だが、マッピング作業をすっ飛ばして*k*-merに基づく方法で定量。Sailfish (Patro et al., *Nat Biotechnol.*, 2014)やRNA-Skim (Zhang and Wang, *Bioinformatics*, 2014)。

ゲノムアセンブリ概略

大きく分けて4つの手順からなる。k-merは、主にIlluminaに代表される～数百塩基程度のリード長のNGSデータに適用される。2014.06.25の講義資料をベースに作成

1. 前処理(pre-processing filtering)
   - クオリティの低いリードやコンタミを除去するステップ。塩基置換(substitution)やインデル(indels; insertion/deletion)を含むリードの除去や補正(error correction)。
   - 4つのアプローチ：**k-mer**, suffix tree/array, multiple sequence alignment, hybrid
2. グラフ構築(graph construction)
   - 前処理後のリードを用いて、リード間のオーバーラップ(overlap)を頼りにつなげていくステップ。シークエンスエラー(sequencing error)と多型(polymorphism)の違いを見るべく、グラフ構築時にエラー補正を行うものもある。
   - 4つのアプローチ：OLC, de Bruijn graph (**k-mer**), greedy, hybrid
3. グラフ簡易化(graph simplification)
   - グラフ構築後に、複雑化したグラフをシンプルにしていくステップ。連続したノード(nodes; 頂点) やバブルのマージ作業に相当。
4. 後処理(post-processing)
   - contigsやscaffoldsを得るステップ。ミスアセンブリの同定も含む。

El-Metwally et al., *PLoS Comput Biol.*, **9**: e1003345, 2013

k-mer出現頻度解析

アセンブリ時の前処理部分でよく用いられる。kの値は概ねリード長の半分～2/3程度が採用されるが、解析戦略は多様化してきているので参考程度

- 入力はNGSデータ
  - リード長より短いk連続塩基からなる部分文字列を発生させるのが最初のステップ。発生させたk-merの出現頻度情報をもとに、カバレッジや**ゲノムサイズ推定**、コンタミリードの除去などを行う
  - 例1：20塩基長のリードをk=19で分割すると、2個のk-merを発生可能

```
CACCAGGACATGAAGACGCG
        ⇩
CACCAGGACATGAAGACGC
 ACCAGGACATGAAGACGCG
```

  - 例2：20塩基長のリードをk=17で分割すると4個のk-merを発生可能

```
CACCAGGACATGAAGACGCG
        ⇩
CACCAGGACATGAAGAC
 ACCAGGACATGAAGACG
  CCAGGACATGAAGACGC
   CAGGACATGAAGACGCG
```

  - 例3：L塩基長のリードをk-merに分割すると(L − k + 1)個のk-merを発生可能

ゲノムサイズ推定

ここでのゲノムサイズ推定とは、①NGSリードファイルを入力としてk-mer解析を行い、②元のゲノム配列のサイズ(総塩基数)を予想すること。通常、*de novo*アセンブリの前処理段階で行う。この場合50 bpが正解

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sa
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リードぶんだけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Bios

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

②

```
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA
```

```
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

①

ゲノムサイズ推定

意義：*de novo*アセンブリの前処理段階で行うため、アセンブリ結果が推定ゲノムサイズ（や近縁種のゲノムサイズ）と似た結果になっているかどうかや、PacBioデータのアセンブリ実行時に事前情報としても利用

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sa
50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リードだけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Bios

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

②

```
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA
```

```
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

①

k-mer解析(k=2)

NGSデータからのゲノムサイズ推定の考え方を説明すべく、k=2の場合のk-mer出現頻度解析から順に述べる。①項目名に注意！②例題7。③入力ファイル

- イントロ | 一般 | 逆鎖(reverse)を取得 (last modified 2013/06/14)
- イントロ | 一般 | k-mer解析 | k=1(塩基ごとの 出現頻度解析) | Biostrings (last modified 2016/04/27) NEW
- イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=3(3連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings (last modified 2016/04/30) NEW
- (削除予定)イントロ | 一般 | 2連続塩基の出現頻度情報を取得 (last modified 2015/04/20)

①

イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings NE

Biostringsパッケージを用いて、4^n通りの任意のn連続塩基の 出現頻度を調べるやり方を示します。k-mer解析のk=n

②

7. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sample32_ngs.fas

2連続塩基(k=2)の出現頻度情報を得るやり方です。4^2 = 16通りのk-merの 出現頻度を毎に出現頻度を算出しています。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_f
out_f <- "hoge7.txt"                  #出力ファイル名を指定してout_
param_kmer <- 2                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファ

#本番
out <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度情報をoutに格納

#ファイルに保存
tmp <- cbind(names(fasta), out)       #最初の列にID情報、そのあとに出現頻度情報のoutを結合したtm
write.table(tmp, out_f, sep="\t", append=F, quote=F, row.names=F)#tmpの中身を指定したファイル名
```

③

③

```
>kkk_24_43
CACCAGGACATGAAGACGCG
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

k-mer解析(k=2)

①例題7コピペ実行後に、②出力ファイルの中身に相当するoutオブジェクトを表示。これは、4^2 = 16種類の2連続塩基の出現頻度をリードごとに算出したもの。例えば③は3番目のリードの結果

①

7. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(san

2連続塩基(k=2)の出現頻度情報を得るやり方です。4^2 = 16通りのk-merの出現頻度を計算することになります。リ
毎に出現頻度を算出しています。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge7.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 2                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
out <- oligonucleotideFrequency(fast

②
#ファイルに保存
tmp <- cbind(names(fasta), out)
write.table(tmp, out_f, sep="\t", apper
```

```
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
```

③

R Console

```
> #ファイルに保存
> tmp <- cbind(names(fasta), out)          #最初の列にID情$
> write.table(tmp, out_f, sep="\t", append=F, quote=F, r$
> out
       AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
  [1,]  1  3  2  1  3  1  2  0  3  1  1  0  0  0  1  0
  [2,]  1  3  2  1  1  0  3  0  4  1  2  0  0  0  1  0
  [3,]  1  4  1  0  3  1  0  2  2  0  1  0  1  1  1  1
  [4,]  1  2  0  0  1  0  0  3  1  0  0  2  1  2  2  4
  [5,]  1  3  0  0  1  0  0  3  1  0  0  1  1  2  2  4
  [6,]  1  4  0  0  2  0  0  3  1  0  0  1  1  2  1  3
  [7,]  1  4  1  0  3  1  0  2  2  0  1  0  1  1  1  1
  [8,]  1  4  2  1  3  1  1  0  3  1  1  0  0  0  1  0
  [9,]  2  1  1  0  1  0  0  2  2  0  0  2  0  2  2  4
 [10,]  1  2  1  0  1  0  0  3  1  0  0  2  0  2  2  4
>
```

k-mer解析(k=2)

ゲノムサイズ推定の場合は、①リードごとではなく全体をまとめて考える。②例えばTTという2連続塩基は、入力ファイル中に全部で21回出現したと解釈する

7. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sample32_ngs.fasta)の場合:

2連続塩基(k=2)の出現頻度情報を得るやり方です。4^2 = 16通りのk-merの出現頻度を計算することになります。リード毎に出現頻度を算出しています。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge7.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 2                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
out <- oligonucleotideFrequency(fasta,

#ファイルに保存
tmp <- cbind(names(fasta), out)
write.table(tmp, out_f, sep="\t", apper
```

R Console

```
> out
       AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
 [1,]   1  3  2  1  3  1  2  0  3  1  1  0  0  0  1  0
 [2,]   1  3  2  1  1  0  3  0  4  1  2  0  0  0  1  0
 [3,]   1  4  1  0  3  1  0  2  2  0  1  0  1  1  1  1
 [4,]   1  2  0  0  1  0  0  3  1  0  0  2  1  2  2  4
 [5,]   1  3  0  0  1  0  0  3  1  0  0  1  1  2  2  4
 [6,]   1  4  0  0  2  0  0  3  1  0  0  1  1  2  1  3
 [7,]   1  4  1  0  3  1  0  2  2  0  1  0  1  1  1  1
 [8,]   1  4  2  1  3  1  1  0  3  1  1  0  0  0  1  0
 [9,]   2  1  1  0  1  0  0  2  2  0  0  2  0  2  2  4
[10,]   1  2  1  0  1  0  0  3  1  0  0  2  0  2  2  4
> colSums(out)
AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
11 30 10  3 19  4  6 18 20  3  6  8  5 12 14 21
>
```

①

②

k-mer解析(k=2)

①例題8は、全リードの合算結果を出力するコード。②outオブジェクトが合算後の数値ベクトル。③4^2 = 16通り全ての2連続塩基が④1回以上出現していることがわかる

8. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sample32_ngs.fasta)の場合:

2連続塩基(k=2)の出現頻度情報を得るやり方です。4^2 = 16通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge8.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 2                       #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)                  #列ご

#ファイルに保存
write.table(out, out_f, sep="\t", apper
```

R Console

```
> #必要なパッケージをロード
> library(Biostrings)                          #パッケージの読$
>
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")#in_fで$
>
> #本番
> hoge <- oligonucleotideFrequency(fasta, width=param_km$
> out <- colSums(hoge)                    #列ごとの総和をoutに$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F, r$
> out
AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
11 30 10  3 19  4  6 18 20  3  6  8  5 12 14 21
>
```

ゲノムサイズ推定

ゲノムサイズ推定は、①kの値を大きくして、1回以上出現しているk連続塩基(k-mer)の種類数をカウントするのが基本

8. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sample32_ngs.fasta)の場合:

2連続塩基(k=2)の出現頻度情報を得るやり方です。4^2 = 16通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge8.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 2                       #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)                  #列ご

#ファイルに保存
write.table(out, out_f, sep="\t", apper
```

①

R Console

```
> #必要なパッケージをロード
> library(Biostrings)                          #パッケージの読$
>
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")#in_fで$
>
> #本番
> hoge <- oligonucleotideFrequency(fasta, width=param_km$
> out <- colSums(hoge)                    #列ごとの総和をoutに$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F, r$
> out
AA AC AG AT CA CC CG CT GA GC GG GT TA TC TG TT
11 30 10  3 19  4  6 18 20  3  6  8  5 12 14 21
>
```

k-mer解析(k=3)

①例題9は、②k=3のときの、4^3 = 64通りの3連続塩基ごとの出現頻度。いくつかの3連続塩基(例えばAAA)は1回も出現していないことがわかる

9. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sample32_ngs.fasta)の場合:

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge9.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)

#ファイルに保存
write.table(out, out_f, sep="\t", apper
```

R Console

```
> out <- colSums(hoge)                #列ごとの総和を$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F, r$
> out
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC
  0   7   4   0   7   4   4  13   3   0   5   2   0   0
ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT
  3   0   0  11   4   3   4   0   0   0   0   3   1   0
CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC
  5   7   0   5  11   7   0   0   0   0   2   0   6   0
GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT TCA TCC TCG TCT
  0   0   0   5   0   3   0   4   0   0   7   0   0   5
TGA TGC TGG TGT TTA TTC TTG TTT
 10   0   0   4   0   0  11  10
>
```

k-mer解析(k=3)

当たり前だが、①outオブジェクトの要素数は64。②1回以上出現した3連続塩基(k-mer)の種類数は32。この種類数が、ゲノムサイズの推定値に相当する。この場合、推定値は32 bpで正解は50 bp

9. サンプルデータの例題32を実行して得られたmulti-FASTAファイル(sam

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample32_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge9.txt"                  #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)

#ファイルに保存
write.table(out, out_f, sep="\t", apper
```

R Console

```
> out
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA ATC
  0   7   4   0   7   4   4  13   3   0   5   2   0   0
ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC CGG CGT
  3   0   0  11   4   3   4   0   0   0   0   3   1   0
CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG GCT GGA GGC
  5   7   0   5  11   7   0   0   0   0   2   0   6   0
GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT TCA TCC TCG TCT
  0   0   0   5   0   3   0   4   0   0   7   0   0   5
TGA TGC TGG TGT TTA TTC TTG TTT
 10   0   0   4   0   0  11  10
① > length(out)
[1] 64
② > sum(out > 0)
[1] 32
>
```

この仮想データは

リマインド。①入力のNGSリードファイルは、②50 bpの③仮想リファレンスゲノム配列からランダム抽出して得られたもの。①の総塩基数は200 bpなので4X coverage

32. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample32_ref.fastaとsample32_ngs.fasta)です。50塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を10リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample32_ref.fastaで、10リードからなる仮想NGSデータがsample32_ngs.fastaです。リード長20塩基で10リードなのでトータル200塩基となり、50塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample32_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample32_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 50                     #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにする
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 10                     #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケ
library(Biost
>kkk
GAAGTTGTCTTTTGAACTCACTACACCAGGACATGAAGACGCGGACGGCA

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetから
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用し
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配
```

```
>kkk_24_43
CACCAGGACATGAAGACGCG
>kkk_28_47
AGGACATGAAGACGCGGACG
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_4_23
GTTGTCTTTTGAACTCACTA
>kkk_5_24
TTGTCTTTTGAACTCACTAC
>kkk_7_26
GTCTTTTGAACTCACTACAC
>kkk_12_31
TTGAACTCACTACACCAGGA
>kkk_23_42
ACACCAGGACATGAAGACGC
>kkk_1_20
GAAGTTGTCTTTTGAACTCA
>kkk_3_22
AGTTGTCTTTTGAACTCACT
```

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

ランダムデータ生成

NGSリードファイル中に存在するk-merの種類数でゲノムサイズ推定ができることの実感を持ってもらうべく、1,000 bpの仮想ゲノム配列、および仮想NGSデータを作成する。①サンプルデータの、②例題33

- インストール | Rパッケージ | 必要最小限(数GB?!) (last modified 2015/05/25)
- インストール | Rパッケージ | 個別 (last modified 2015/06/10)
- (削除予定)Rのインストールと起動 (last modified 2016/02/21)
- (削除予定)個別パッケージのインストール (last modified 2015/02/20)
- 基本的な利用法 (last modified 2015/04/03)
- サンプルデータ (last modified 2015/06/15)
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2016
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2015
- バイオインフォマテ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ

①

サンプルデータ

1. Illumina/36bp/single-end/human (SRA000299) data (Marioni et al., Genome Res., 2008)
   「Kidney 7 samples vs Liver 7 samples」のRNA-seqの遺伝子発現行列データ(SupplementaryTable2.txt)です。サンプルは二つの濃度(1.5 pM and 3 pM)でシーケンスされており、「3 pMのものが5 samples vs. 5 samples」、「1.5 pMのもの
   に発現デ
   す):

   7列目：R1L1
   8列目：R1L2
   9列目：R1L3
   10列目：R1I
   11列目：R1I
   12列目：R1I
   13列目：R1I

②

33. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample33_ref.fastaとsample33_ngs.fasta)です。
    1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を200リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample33_ref.fastaで、200リードからなる仮想NGSデータがsample33_ngs.fastaです。リード長20塩基で200リードなのでトータル4,000塩基となり、1,000塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成から と基本的に同じです。

```
out_f1 <- "sample33_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample33_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 1000                   #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにするた
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 200                    #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数だ
```

ランダムデータ生成

①例題33。②1000 bpの仮想リファレンスゲノム配列、および③仮想NGSデータを作成する。20 bpのリードが200個なので、計4,000 bp、つまり4X coverageのデータを作成していることになる

33. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample33_
1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を200リード分だけランダム抽出のためです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample33_ref.fastaで、200リードからなる仮想NGSデータがsample33_ngs.fastaです。リード長20塩基で200リードなのでトータル4,000塩基となり、1,000塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample33_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample33_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 1000                   #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにするた
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 200                    #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数だ
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetからpa
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用した
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配列
```

ランダムデータ生成

コピペ実行後の状態。①set.seed関数があるので、みんな同じ結果になっているはず

33. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample33_ref.fastaとsample33_ngs.fasta)です。1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を200リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample33_ref.fastaで、200リードからなる仮想NGSデータがsample33_ngs.fastaです。リード長20塩基で200リードなのでトータル4,000塩基となり、1,000塩基からなる元のゲノム配列の4倍シーケンスしていることになります(4X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample33_ref.fasta"
out_f2 <- "sample33_ngs.fasta"
param_len_ref <- 1000
narabi <- c("A","C","G","T")
param_composition <- c(22, 28, 28, 22
param_len_ngs <- 20
param_num_ngs <- 200
param_desc <- "kkk"

#必要なパッケージをロード
library(Biostrings)

#本番(リファレ　①　配列生成)
set.seed(1010)
ACGTset <- rep(narabi, param_composit
reference <- paste(sample(ACGTset, pa
reference <- DNAStringSet(reference)
names(reference) <- param_desc
reference

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-par
```

R Console

```
> fasta                                    #確認してるだけ$
  A DNAStringSet instance of length 200
      width seq                            names
  [1]    20 GGTACGGTTCCGGTTGCCGA           kkk_658_677
  [2]    20 TTATCCGGCAGTCCTATATG           kkk_197_216
  [3]    20 CAATAGACACCACGCGCCAC           kkk_559_578
  [4]    20 ACCGAAAGTTCTGGAGGTCA           kkk_55_74
  [5]    20 TTGGGCCCGATATCTATAAA           kkk_681_700
  ...   ... ...
[196]    20 GGACACGGAACGGGGCTGAG           kkk_283_302
[197]    20 TACGGTTCCGGTTGCCGATC           kkk_660_679
[198]    20 TCACAACGGCCGTCTGCATT           kkk_101_120
[199]    20 TTAGGAAGGTACGGTTCCGG           kkk_651_670
[200]    20 CTTTATTGGGACGCTTTTCA           kkk_925_944
>
> #ファイルに保存(仮想リファレンス配列と仮想NGS配列)
> writeXStringSet(reference, file=out_f1, format="fasta"$
> writeXStringSet(fasta, file=out_f2, format="fasta", wi$
>
```

k-mer解析(k=3)

k=3の場合のk-mer出現頻度解析を行う。①項目名に注意！②例題10。③入力ファイル

- イントロ | 一般 | 逆鎖(reverse)を取得 (last modified 2013/06/14)
- イントロ | 一般 | k-mer解析 | k=1(塩基ごとの 出現頻度解析) | Biostrings (last modified 2016/04/27) NEW
- イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=3(3連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings (last modified 2016/04/30) NEW
- (削除予定)イントロ | 一般 | 2連続塩基の出現頻度情報を取得 (last modified 2015/04/20)

①

イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings NEW

Biostringsパッケージを用いて、4^n通りの任意のn連続塩基の 出現頻度を調べるやり方を示します。k-mer解析のk=nの場合に相当します。例え

②

「ファイル」-「ディレ

1. イントロ | 一般 | ラン

4連続塩基(k=4)の出

```
in_f <- "hoge4.
out_f <- "hoge1
param_kmer <- 4

#必要なパッケージ
library(Biostri

#入力ファイルの読
fasta <- readDN
```

10. サンプルデータの例題33を実行して得られたmulti-FASTAファイル(sample33_ngs.fasta)の場合:

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通りのk-merの 出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

③

```
in_f <- "sample33_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge10.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み

#本番
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度情報をhogeに格
out <- colSums(hoge)                  #列ごとの総和をoutに格納

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=T, col.names=F)#outの中身
```

k-mer解析(k=3)

①例題10は、②k=3のときの、③4^3 = 64通りの3連続塩基ごとの出現頻度。④全ての3連続塩基(例えばAAA)は1回以上出現していることがわかる

10. サンプルデータの例題33を実行して得られたmulti-FASTAファイル(sample33_ngs.fasta)の場合:

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通りのk-merの出現頻度を計算することになります。
全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample33_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge10.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format=

#本番
hoge <- oligonucleotideFrequency(fasta, w
out <- colSums(hoge)                  #歹

#ファイルに保存
write.table(out, out_f, sep="\t", append=
```

R Console

```
> out
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
 36  52  42  44  58  89  59  45  36  47  55  39  39
ATC ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC
 49  62  32  34  90  62  18  56 124 121  54  66  70
CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG
 85  47  50  35  93  49  55  75  53  44  30  68  64
GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
 94  66  85  90  53  54  52  39  43  51  42  21  76
TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
 63  61  19  32  61  49  67  56  46  45  40  68
> length(out)
[1] 64
> sum(out > 0)
[1] 64
>
```

ゲノムサイズ推定

ゲノムサイズ推定は、①kの値を大きくして、②1回以上出現しているk連続塩基(k-mer)の種類数をカウントするのが基本。しかし、③k=3だと種類数の最大は4^3 = 64しかないため、このデータの正解である1000 bpにはなりようがない。つまり①kの値を大きくする必要がある

10. サンプルデータの例題33を実行して得られたmulti-FASTAファ

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通り
全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample33_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge10.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="f

#本番
hoge <- oligonucleotideFrequency(fasta, w
out <- colSums(hoge)                  #歹

#ファイルに保存
write.table(out, out_f, sep="\t", append=
```

① ②

R Console

```
> out
AAA AAC AAG AAT ACA ACC ACG ACT AGA AGC AGG AGT ATA
 36  52  42  44  58  89  59  45  36  47  55  39  39
ATC ATG ATT CAA CAC CAG CAT CCA CCC CCG CCT CGA CGC
 49  62  32  34  90  62  18  56 124 121  54  66  70
CGG CGT CTA CTC CTG CTT GAA GAC GAG GAT GCA GCC GCG
 85  47  50  35  93  49  55  75  53  44  30  68  64
GCT GGA GGC GGG GGT GTA GTC GTG GTT TAA TAC TAG TAT
 94  66  85  90  53  54  52  39  43  51  42  21  76
TCA TCC TCG TCT TGA TGC TGG TGT TTA TTC TTG TTT
 63  61  19  3   61  49  67  56  46  45  40  68
> length(out)
[1] 64
> sum(out > 0)
[1] 64
>
```

③

ゲノムサイズ推定

kをどの程度まで大きくする必要があるかの感覚をつかむ。まず、1KB (1,000 bp)のゲノムサイズを推定するためには、①最低でもk=5以上にする必要がある。②k=10にすると1MB (1,000,000 bp)だが、これだと原理的に3GBのヒトゲノム(3,000,000,000 bp)には適用できない

10. サンプルデータの例題33を実行して得られたmulti-FASTAファ

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通り
全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample33_ngs.fasta"            #入力ファイル名を指定してin_fに格納
out_f <- "hoge10.txt"                   #出力ファイル名を指定してout_fに格納
param_kmer <- 3                         #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fa

#本番
hoge <- oligonucleotideFrequency(fasta, wi
out <- colSums(hoge)                    #列

#ファイルに保存
write.table(out, out_f, sep="\t", appen
```

①
②

R Console

```
> length(out)
[1] 64
> sum(out > 0)
[1] 64
> 4^5
[1] 1024
> 4^10
[1] 1048576
> 4^15
[1] 1073741824
> 4^16
[1] 4294967296
> 4^20
[1] 1.099512e+12
>
```

ゲノムサイズ推定

3GBのヒトゲノム(3,000,000,000 bp)に適用可能な下限は、①k=16であることがわかる。②このデータの大元はゲノムサイズ1,000 bpなので、とりあえず③k=10にして1回以上出現するk-merの種類数を調べる

10. サンプルデータの例題33を実行して得られたmulti-FASTAファイル

3連続塩基(k=3)の出現頻度情報を得るやり方です。4^3 = 64通りのk-merの出現頻度を計算することになります。
全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample33_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge10.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 3                       #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fa

#本番
hoge <- oligonucleotideFrequency(fasta, wi
out <- colSums(hoge)                  #列

#ファイルに保存
write.table(out, out_f, sep="\t", append=F
```

②

③

①

R Console

```
> length(out)
[1] 64
> sum(out > 0)
[1] 64
> 4^5
[1] 1024
> 4^10
[1] 1048576
> 4^15
[1] 1073741824
> 4^16
[1] 4294967296
> 4^20
[1] 1.099512e+12
>
```

k-mer解析(k=10)

①例題11は、②k=10のときの、4^10 = 1,048,576通りの10連続塩基ごとの出現頻度を返すコード。③1回以上出現している10連続塩基は904個。つまり推定ゲノムサイズは904 bp。正解は1,000 bpなので近い値

①

11. サンプルデータの例題33を実行して得られたmulti-FASTAファイル

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample33_ngs.fasta"            #入力ファイル名を指定してin_fに格納
out_f <- "hoge11.txt"                   #出力ファイル名を指定してout_fに格納
param_kmer <- 10                        #k-merのkの値を指定

#必要なパッケージのロード
library(Biostrings)                     #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fa

#本番
hoge <- oligonucleotideFrequency(fasta, wi
out <- colSums(hoge)                    #列

#ファイルに保存
write.table(out, out_f, sep="\t", append=F
length(out)                             #4^
sum(out > 0)                            #1[
```

②

③

```
R Console
>
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")#in_$
>
> #本番
> hoge <- oligonucleotideFrequency(fasta, width=param$
> out <- colSums(hoge)                              #列ごとの総$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F$
> length(out)                                       #4^param_kme$
[1] 1048576
> sum(out > 0)                                      #1回以上出現$
[1] 904
>
```

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

ランダムデータ生成

例題33は4X coverageデータで904 bpだった（正解は1,000 bp）。Coverageの違いによるゲノムサイズ推定精度の違いを把握するべく、10X coverageデータを作成する。①サンプルデータの、②例題34

- インストール | Rパッケージ | 必要最小限(数GB?!) (last modified 2015/05/25)
- インストール | Rパッケージ | 個別 (last modified 2015/06/10)
- (削除予定)Rのインストールと起動 (last modified 2016/02/21)
- (削除予定)個別パッケージのインストール (last modified 2015/02/20)
- 基本的な利用法 (last modified 2015/04/03)
- サンプルデータ (last modified 2015/06/15)
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2016
- バイオインフォマティクス人材育成カリキュラム(次世代シークエンサ) | NGSハンズオン講習会2015
- バイオインフォマテ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ
- 書籍 | トランスクリプ

①

サンプルデータ

1. Illumina/36bp/single-end/human (SRA000299) data (Marioni et al., Genome Res., 2008)

   「Kidney 7 samples vs Liver 7 samples」のRNA-seqの遺伝子発現行列データ(SupplementaryTable2.txt)です。サンプルは二つの濃度(1.5 pM and 3 pM)でシーケンスされており、「3 pMのものが5 samples vs. 5 samples」、「1.5 pMのもの
   に発現デ
   す):

   7列目：R1L
   8列目：R1L
   9列目：R1L
   10列目：R1
   11列目：R1
   12列目：R1
   13列目：R1

②

34. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample34_ref.fastaとsample34_ngs.fasta)です。1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を500リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample34_ref.fastaで、500リードからなる仮想NGSデータがsample34_ngs.fastaです。リード長20塩基で500リードなのでトータル10,000塩基となり、1,000塩基からなる元のゲノム配列の10倍シーケンスしていることになります(10X coverageに相当)。イントロ | NGS | 配列取得 | シミュレーションデータ | ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample34_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample34_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 1000                   #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにするた
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 500                    #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数だ
```

ランダムデータ生成

①例題34。②1000 bpの仮想リファレンスゲノム配列、および③仮想NGSデータを作成する。20 bpのリードが500個なので、計10,000 bp、つまり10X coverageのデータを作成していることになる

34. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample34_r
1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を500
す。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)が
sample34_ref.fastaで、500リードからなる仮想NGSデータがsample34_ngs.fastaです。リード長20塩基で500リードなので
トータル10,000塩基となり、1,000塩基からなる元のゲノム配列の10倍シーケンスしていることになります(10X coverage
に相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample34_ref.fasta"          #出力ファイル名を指定してout_f1に格納
out_f2 <- "sample34_ngs.fasta"          #出力ファイル名を指定してout_f2に格納
param_len_ref <- 1000                   #リファレンス配列の長さを指定
narabi <- c("A","C","G","T")            #以下の数値指定時にACGTの並びを間違えないようにするた
param_composition <- c(22, 28, 28, 22) #(A,C,G,Tの並びで)各塩基の存在比率を指定
param_len_ngs <- 20                     #リード長を指定
param_num_ngs <- 500                    #リード数を指定
param_desc <- "kkk"                     #FASTA形式ファイルのdescription行に記述する内容

#必要なパッケージをロード
library(Biostrings)                     #パッケージの読み込み

#本番(リファレンス配列生成)
set.seed(1010)                          #おまじない(同じ乱数になるようにするため)
ACGTset <- rep(narabi, param_composition)#narabi中の塩基がparam_compositionで指定した数だけ
reference <- paste(sample(ACGTset, param_len_ref, replace=T), collapse="")#ACGTsetからpar
reference <- DNAStringSet(reference)    #DNA塩基配列だと認識させるDNAStringSet関数を適用した
names(reference) <- param_desc          #description行に相当する記述を追加している
reference                               #確認してるだけです

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-param_len_ngs), param_num_ngs, replace=T)#部分塩基配列抽
```

ランダムデータ生成

コピペ実行後の状態。①set.seed関数があるので、みんな同じ結果になっているはず

34. k-mer解析用のランダム配列から生成したFASTA形式ファイル(sample34_ref.fastaとsample34_ngs.fasta)です。1000塩基の長さのリファレンス配列を生成したのち、20塩基長の部分配列を500リード分だけランダム抽出したものです。塩基の存在比はAが22%, Cが28%, Gが28%, Tが22%にしています。リファレンス配列(仮想ゲノム配列)がsample34_ref.fastaで、500リードからなる仮想NGSデータがsample34_ngs.fastaです。リード長20塩基で500リードなのでトータル10,000塩基となり、1,000塩基からなる元のゲノム配列の10倍シーケンスしていることになります(10X coverageに相当)。イントロ|NGS|配列取得|シミュレーションデータ|ランダムな塩基配列の生成からと基本的に同じです。

```
out_f1 <- "sample34_ref.fasta"
out_f2 <- "sample34_ngs.fasta"
param_len_ref <- 1000
narabi <- c("A","C","G","T")
param_composition <- c(22, 28, 28, 22
param_len_ngs <- 20
param_num_ngs <- 500
param_desc <- "kkk"

#必要なパッケージをロード
library(Biostrings)

#本番(リファレ ①配列生成)
set.seed(1010)
ACGTset <- rep(narabi, param_composit
reference <- paste(sample(ACGTset, pa
reference <- DNAStringSet(reference)
names(reference) <- param_desc
reference

#本番(シミュレーションデータ生成)
s_posi <- sample(1:(param_len_ref-para
```

R Console

```
> fasta                                    #確認してるだ$
  A DNAStringSet instance of length 500
      width seq                                  names          $
  [1]    20 GGTACGGTTCCGGTTGCCGA                 kkk_658_677
  [2]    20 TTATCCGGCAGTCCTATATG                 kkk_197_216
  [3]    20 CAATAGACACCACGCGCCAC                 kkk_559_578
  [4]    20 ACCGAAAGTTCTGGAGGTCA                 kkk_55_74
  [5]    20 TTGGGCCCGATATCTATAAA                 kkk_681_700
  ...   ... ...
[496]    20 GAGCTATCAATGTACCCCAG                 kkk_433_452
[497]    20 AACCTAATCGCCCCGTGGCG                 kkk_805_824
[498]    20 CGCCGCCTTCGAGCTCGCTT                 kkk_140_159
[499]    20 GACAGCGGGAGTGATCCGGC                 kkk_401_420
[500]    20 TGTAAGCCGTACCCTACCCC                 kkk_904_923
>
> #ファイルに保存(仮想リファレンス配列と仮想NGS配列)
> writeXStringSet(reference, file=out_f1, format="fasta$
> writeXStringSet(fasta, file=out_f2, format="fasta", w$
>
```

k-mer解析(k=10)

k=10の場合のk-mer出現頻度解析を行う。①項目名に注意！②例題12。③入力ファイル

- イントロ | 一般 | 逆鎖(reverse)を取得 (last modified 2013/06/14)
- イントロ | 一般 | k-mer解析 | k=1(塩基ごとの 出現頻度解析) | Biostrings (last modified 2016/04/27) NEW
- イントロ | 一般 | k-mer解析 | k=2(2連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=3(3連続塩基の出現頻度解析) | Biostrings (last modified 2016/01/28)
- イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings (last modified 2016/04/30) NEW
- (削除予定)イントロ | 一般 | 2連続塩基の出現頻度情報を取得 (last modified 2015/04/20)

①

イントロ | 一般 | k-mer解析 | k=n(n連続塩基の出現頻度解析) | Biostrings NEW

Biostringsパッケージを用いて、4^n通りの任意のn連続塩基の 出現頻度を調べるやり方を示します。k-mer解析のk=nの場合に相当します。例

「ファイル」-「

1. イントロ | 一般 |

4連続塩基(k=4)

②

12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル(sample34_ngs.fasta)の場合:

10連続塩基(k=10)の 出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの 出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

③

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #パッケージの読み込み

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み

#本番
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度情報をhogeに格納
out <- colSums(hoge)                  #列ごとの総和をoutに格納

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=T, col.names=F)#outの中身を
length(out)                           #4^param_kmerの値を表示
sum(out > 0)                          #1回以上出現したk-merの種類数を表示
```

k-mer解析(k=10)

①例題12は、②k=10のときの、4^10 = 1,048,576通りの10連続塩基ごとの出現頻度を返すコード。③1回以上出現している10連続塩基は985個。つまり10X coverageのときの推定ゲノムサイズは985 bp。4X coverageのときの推定値(904 bp)に比べて、より正解(1,000 bp)に近い値になっていることがわかる

①

12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,
す。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample34_ngs.fasta"                #入力ファイル
out_f <- "hoge12.txt"                       #出力ファイル名を指定してout_fに格納
param_kmer <- 10                            #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                         #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)                        #

#ファイルに保存
write.table(out, out_f, sep="\t", appen
length(out)                                 #
sum(out > 0)                                #
```

②

R Console

```
> param_kmer <- 10                                  #k-merのkの値$
>
> #必要なパッケージをロード
> library(Biostrings)                               #パッケージの$
>
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")#in_f$
>
> #本番
> hoge <- oligonucleotideFrequency(fasta, width=param_k$
> out <- colSums(hoge)                              #列ごとの総和$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F, $
> length(out)                                       #4^param_kmer$
[1] 1048576
> sum(out > 0)                                      #1回以上出現し$
[1] 985
>
```

③

k-mer出現頻度分布への切り替わり部分(スライド115-118)は、きっちり整理して理解しておこう

Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析(k個の連続塩基に基づく各種解析)の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列(仮想ゲノムおよび仮想NGSデータ)の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定(1,000 bpの仮想ゲノムの場合)
  - ゲノムサイズ推定(1,000 bpの仮想ゲノム;4X → 10X coverageの場合)
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

k-mer出現頻度分布

①outベクトルの要素数は1,048,576個で、ほとんどの要素(1,048,576 − 985)が0。②1回以上出現している10連続塩基(10-mer)のみで取り扱いたい場合は、out[out > 0]とすればよい

12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル(sample34

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)                  #

#ファイルに保存
write.table(out, out_f, sep="\t", app
length(out)                           #
sum(out > 0)                          #
```

R Console

```
> out <- colSums(hoge)                          #列ごとの総和$
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F, $
> length(out)                                   #4^param_kmer$
[1] 1048576
> sum(out > 0)                                  #1回以上出現し$
[1] 985
> head(out)
AAAAAAAAAA AAAAAAAAAC AAAAAAAAAG AAAAAAAAAT AAAAAAAACA
         0          0          0          0          0
AAAAAAAACC
         0
> head(out[out > 0])
AAAACTTGGC AAAAGTTTTT AAACAGCTGC AAACGGGAGC AAACTTGGCG
         8          6          7          2          7
AAAGAGCTAT
         8
>
```

k-mer出現頻度分布

①1回以上出現している10連続塩基(10-mer)のみをkmerとして取り扱う。②当然kmerの要素数は985

12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル(sample34_ngs.fasta)の場合:

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)                  #

#ファイルに保存
write.table(out, out_f, sep="\t", appen
length(out)
sum(out > 0)
```

R Console

```
> head(out)
AAAAAAAAAA AAAAAAAAAC AAAAAAAAAG AAAAAAAAAT AAAAAAAACA
         0          0          0          0          0
AAAAAAAACC
         0
> head(out[out > 0])
AAAACTTGGC AAAAGTTTTT AAACAGCTGC AAACGGGAGC AAACTTGGCG
         8          6          7          2          7
AAAGAGCTAT
         8
> kmer <- out[out > 0]
> head(kmer)
AAAACTTGGC AAAAGTTTTT AAACAGCTGC AAACGGGAGC AAACTTGGCG
         8          6          7          2          7
AAAGAGCTAT
         8
> length(kmer)
[1] 985
>
```

①

②

k-mer出現頻度分布

k-mer出現頻度分布は、①hist(kmer)を実行して得られる、②横軸が出現回数、縦軸が出現頻度からなるヒストグラムのこと

12. サンブルデータの例題34を実行して得られたmulti-FASTAファイル(sample34_ngs.fasta)の場合:

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

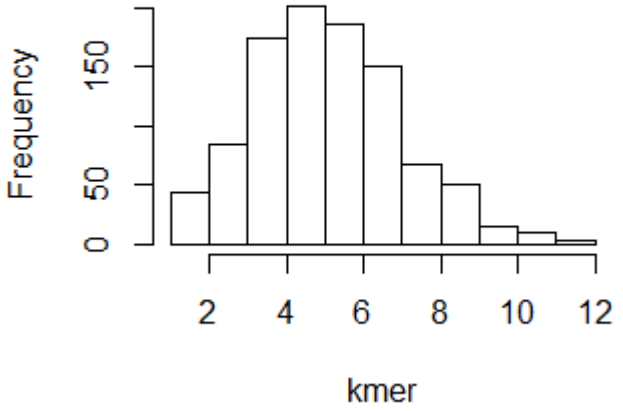
```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="                    #

#本番
hoge <- oligonucleotideFrequency(fasta,                     #
out <- colSums(hoge)                  #

#ファイルに保存
write.table(out, out_f, sep="\t", appen                     #
length(out)                           #
sum(out > 0)                          #
```

R Console

```
AAAAAAAAAA AAAAAAAAAC AAAAA
         0          0
AAAAAAAACC
         0
> head(out[out > 0])
AAAACTTGGC AAAAGTTTTT AAACA
         8          6
AAAGAGCTAT
         8
> kmer <- out[out > 0]
> head(kmer)
AAAACTTGGC AAAAGTTTTT AAACAGCTGC AAACGGGAGC AAACTTGGCG
         8          6          7          2          7
AAAGAGCTAT
         8
> length(kmer)
[1] 985
> hist(kmer)
>
```

①

②



k-mer出現頻度分布

①ヒストグラムでピンとこないヒトは、①table(kmer)と同じものという理解でよい。例えば②は、12回出現したk-merが3個（3種類）あったということ。気になるヒトは③のようにして確認

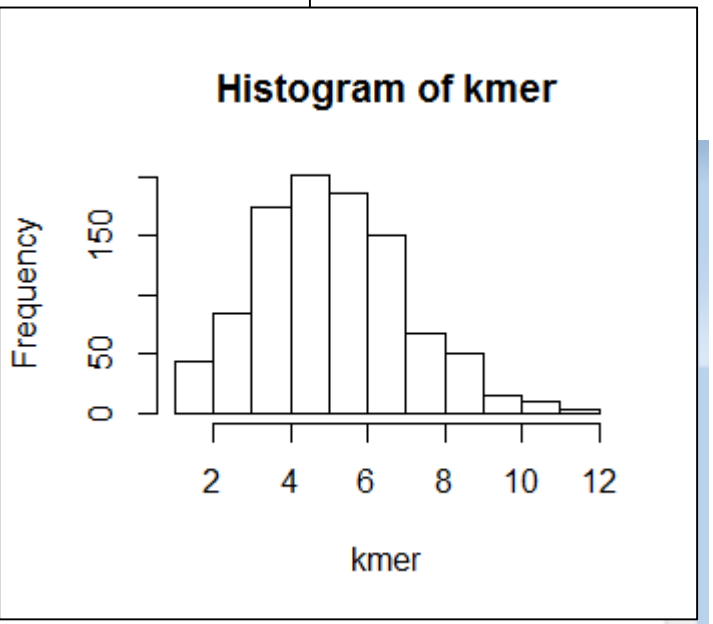
12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル(sampl

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-merの出現頻度を計算することになります。全リードを合算した出現頻度を出力するやり方です。

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)                   #

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="                    #

#本番
hoge <- oligonucleotideFrequency(fasta,                     #
out <- colSums(hoge)                  #

#ファイルに保存
write.table(out, out_f, sep="\t", app                       #
length(out)                           #
sum(out > 0)                          #
```

R Console

```
> head(kmer)
AAAACTTGGC AAAAGTTTTT AAACA
         8          6
AAAGAGCTAT
         8
> length(kmer)
[1] 985
> hist(kmer)
> table(kmer)
kmer
  1   2   3   4   5   6   7   8   9  10  11  12
  3  40  84 174 201 186 151  67  51  15  10   3
> kmer[kmer == 1]
GACGTGTAAA TAAACGGGAG TGACGTGTAA
         1          1          1
> kmer[kmer == 12]
CACACGACCG CACCCTCAGA TCACACGACC
        12         12         12
>
```



k-mer出現頻度分布

ちなみに②k-mer出現回数の中央値(median)は5。③入力ファイルは、ゲノムサイズ1,000 bpのリファレンス配列から、20 bpのリードを500個ランダム抽出したもの。計10,000 bpなので、10X coverageのデータ。おかしくない?

12. サンプルデータの例題34を実行して得られたmulti-FASTAファイル(sample34

10連続塩基(k=10)の出現頻度情報を得るやり方です。4^10 = 1,048,576通りのk-
す。全リードを合算した出現頻度を出力するやり方です。

③

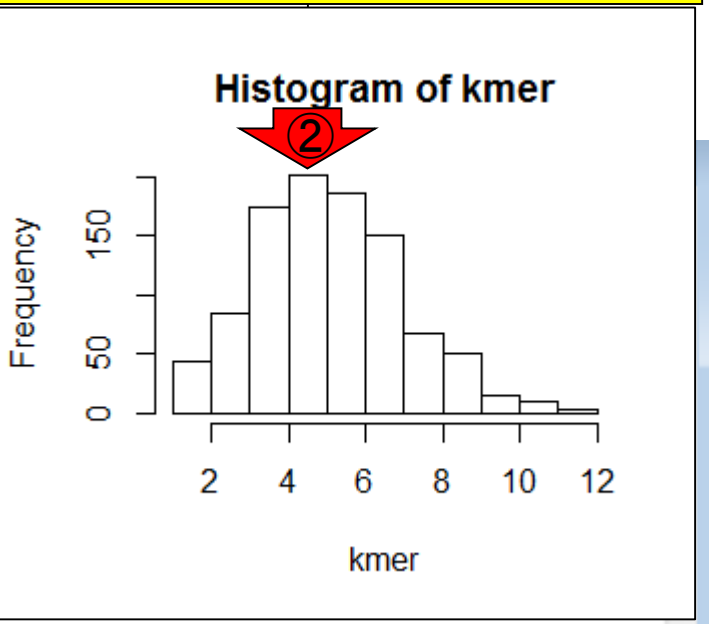
```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
out_f <- "hoge12.txt"                 #出力ファイル名を指定してout_fに格納
param_kmer <- 10                      #k-merのkの値を指定

#必要なパッケージをロード
library(Biostrings)

#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="

#本番
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)

#ファイルに保存
write.table(out, out_f, sep="\t", appen
length(out)
sum(out > 0)
```

R Console

```
          8
> length(kmer)
[1] 985
> hist(kmer)
> table(kmer)
kmer
  1   2   3   4   5   6   7
  3  40  84 174 201 186 151
> kmer[kmer == 1]
GACGTGTAAA TAAACGGGAG TGACG
         1          1          1
> kmer[kmer == 12]
CACACGACCG CACCCTCAGA TCACACGACC
        12         12         12
> mean(kmer)
[1] 5.583756
> median(kmer)
[1] 5
>
```

①

②



k=6, 8, 10, 12

k値の違いによるk-mer出現頻度分布の傾向を把握。①k=6のときの②k-mer出現回数の中央値(median)は8。③最多出現回数は39回。作業としては、赤枠部分のみコピペしていくのが基本だが、全部一度にコピペ実行してもよい。尚、k=6, 8, 10まではエラーは出ないが、12でエラーが出る(後述)

- k=6, 8, 10, 12

必要最小限のコードにして、k値の違い(k=6, 8, 10, and 12)による影響の全体像を

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定し
library(Biostrings)                   #パッケージの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定し

param_kmer <- 6
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 8
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 10
hoge <- oligonucleotideFrequency(fasta,
```

①

R Console

```
> in_f <- "sample34_ngs.fasta"                    #入力ファイル$
> library(Biostrings)                             #パッケージの$
> fasta <- readDNAStringSet(in_f, format="fasta")#in_f$
>
> param_kmer <- 6                                 #k-merのkの値$
> hoge <- oligonucleotideFrequency(fasta, width=param_k$
> out <- colSums(hoge)                            #列ごとの総和$
> kmer <- out[out > 0]                            #1回以上出現し$
> length(kmer)                                    #1回以上出現し$
[1] 862
> table(kmer)                                     #(k-merの種類$
kmer
  2   3   4   5   6   7   8   9  10  11  12  13  14  15
  5  10  36  74 109 134 146 112  68  43  29  11  17  11
 16  17  18  19  20  21  22  24  26  27  28  29  30  39
 13   7  10   4   6   4   4   1   2   2   1   1   1   1
> median(kmer)                                    #出現回数の中$
[1] 8
>
```

②

③

k=6, 8, 10, 12

k値の違いによるk-mer出現頻度分布の傾向を把握。①k=8のときの②k-mer出現回数の中央値(median)は7。③最多出現回数は16回

- k=6, 8, 10, 12
  必要最小限のコードにて、k値の違い(k=6, 8, 10, and 12)による影響の全体像を把握。

```
param_kmer <- 8                                  #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                             #列ごとの総和をoutに格納
kmer <- out[out > 0]                             #1回以上出現したk-merのみをkmerに格納
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 10
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 12
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
```

①

②

③

R Console

```
   1    2    3    4    5    6    7    8    9   10   11   12
   3   40   84 174 201 186 151   67   51   15   10    3
> median(kmer)                                   #出現回数の中$
[1] 5
> param_kmer <- 8                                #k-merのkの値$
> hoge <- oligonucleotideFrequency(fasta, width=param_k$
> out <- colSums(hoge)                           #列ごとの総和$
> kmer <- out[out > 0]                           #1回以上出現し$
> length(kmer)                                   #1回以上出現し$
[1] 975
> table(kmer)                                    #(k-merの種類$
kmer
  2   3   4   5   6   7   8   9  10  11  12  13  14  15
 18  35  99 142 185 178 148  78  47  22  12   4   4   1
 16
  2
> median(kmer)                                   #出現回数の中$
[1] 7
>
```

k=6, 8, 10, 12

k値の違いによるk-mer出現頻度分布の傾向を把握。①k=10のときの②k-mer出現回数の中央値(median)は5。③最多出現回数は12回

- k=6, 8, 10, 12
  必要最小限のコードにして、k値の違い(k=6, 8, 10, and 12)による影響の全体像を把握。

```
out <- colSums(hoge)                    #列ごとの総和をoutに格納
kmer <- out[out > 0]                    #1回以上出現したk-merのみをkmerに格納
length(kmer)                            #1回以上出現したk-merの種類数を表示
table(kmer)                             #(k-merの種類は問わずに)k-merの出現回数
median(kmer)

①
param_kmer <- 10
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 12
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)
```

R Console

```
  2   3   4   5   6   7   8   9  10  11  12  13  14  15
 18  35  99 142 185 178 148  78  47  22  12   4   4   1
 16
  2
> median(kmer)                          #出現回数の中$
[1] 7
> param_kmer <- 10                      #k-merのkの値$
> hoge <- oligonucleotideFrequency(fasta, width=param_k$
> out <- colSums(hoge)                  #列ごとの総和$
> kmer <- out[out > 0]                  #1回以上出現し$
> length(kmer)                          #1回以上出現し$
[1] 985
> table(kmer)                           #(k-merの種類$
kmer
  1   2   3   4   5   6   7   8   9  10  11  12
  3  40  84 174 201 186 151  67  51  15  10   3
②
> median(kmer)                          #出現回数の中$
[1] 5
>
```

③

k=6, 8, 10, 12

k値の違いによるk-mer出現頻度分布の傾向を把握。①k=12のときはメモリ不足の問題でエラーが出ていることがわかる。それゆえ、後半の結果はk=10のときのものがそのまま表示されている点に注意!

- k=6, 8, 10, 12

必要最小限のコードにして、k値の違い(k=6, 8, 10, and 12)による影響の全体像を

```
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 10
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)

param_kmer <- 12
hoge <- oligonucleotideFrequency(fasta,
out <- colSums(hoge)
kmer <- out[out > 0]
length(kmer)
table(kmer)
median(kmer)
```

①

②

R Console

```
> param_kmer <- 12                                  #k-merのkの値$
> hoge <- oligonucleotideFrequency(fasta, width=param_k$
 エラー:  サイズ 31.2 Gb のベクトルを割り当てることがで$
 追加情報:  警告メッセージ:
1:  .Call2("XStringSet_oligo_frequency", x, width, step$
  Reached total allocation of 8106Mb: see help(memory.s$
2:  .Call2("XStringSet_oligo_frequency", x, width, step$
  Reached total allocation of 8106Mb: see help(memory.s$
3:  .Call2("XStringSet_oligo_frequency", x, width, step$
  Reached total allocation of 8106Mb: see help(memory.s$
4:  .Call2("XStringSet_oligo_frequency", x, width, step$
  Reached total allocation of 8106Mb: see help(memory.s$
> out <- colSums(hoge)                               #列ごとの総和$
> kmer <- out[out > 0]                                #1回以上出現し$
> length(kmer)                                        #1回以上出現し$
[1] 985
> table(kmer)                                         #(k-merの種類$
kmer
  1   2   3   4   5   6   7   8   9  10  11  12
  3  40  84 174 201 186 151  67  51  15  10   3
> median(kmer)                                        #出現回数の中$
[1] 5
>
```

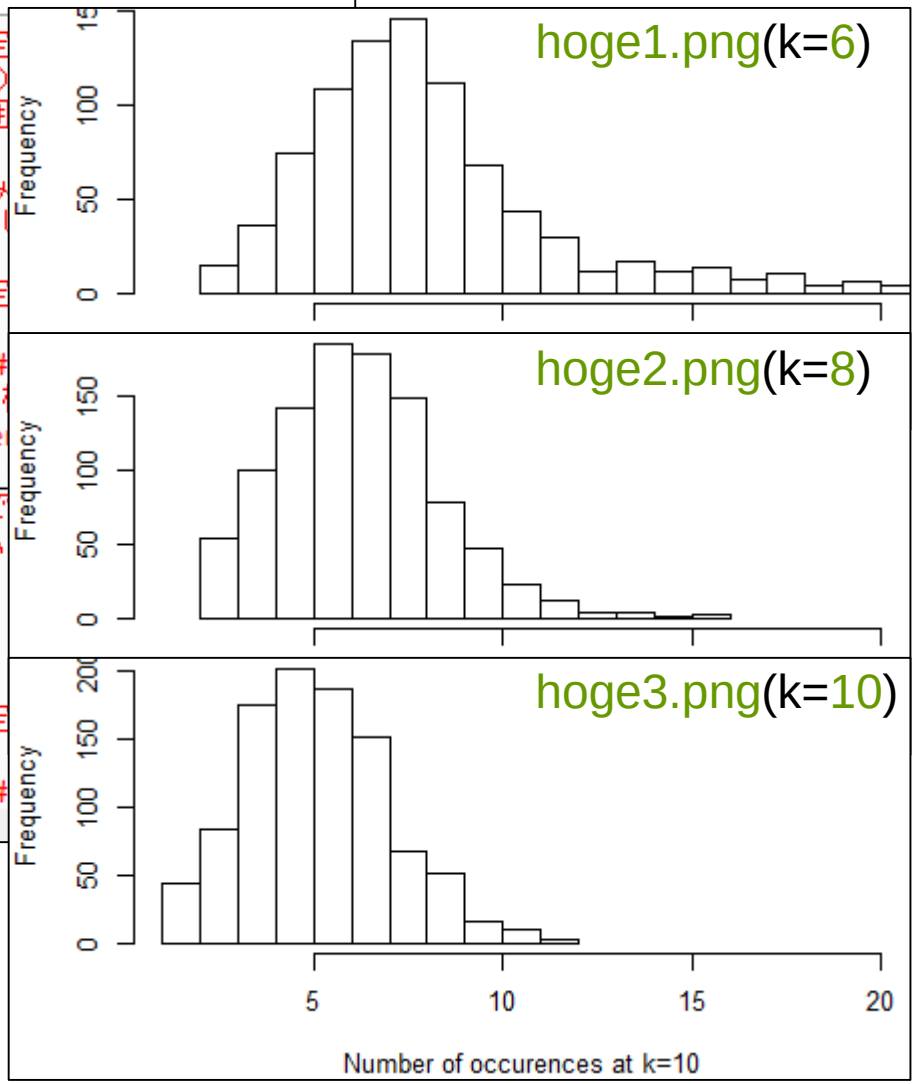
作図（k=6, 8, 10）

k値が小さくなる（図の下から上）にしたがって、分布が右にシフトしていることがわかる。つまり、出現回数の中央値は、k-merのk値次第だということ。横軸の範囲を揃えないとわからないので、①1-20に揃えるべく、②のxlimオプション部分で指定

- 作図(k=6, 8, 10)

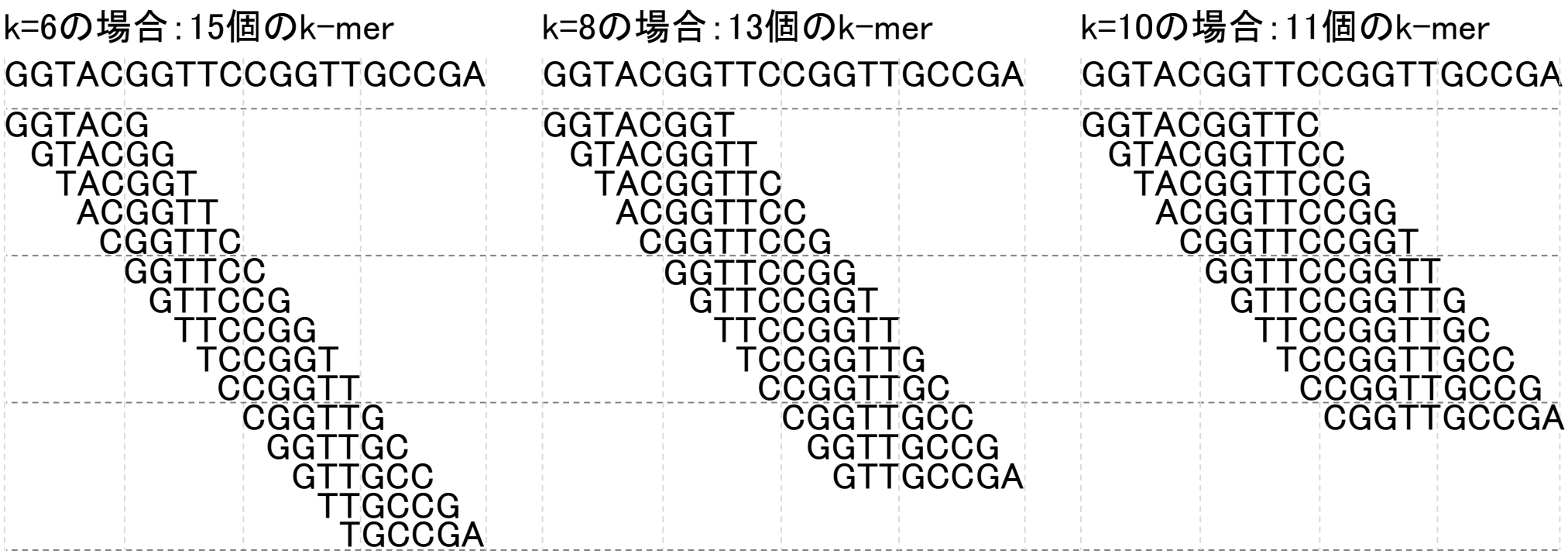
上記のk=6, 8, 10の結果より、横軸の最大値は39であることがわかってい
遷が把握しづらいので 1から20の範囲を指定している。このあたりは結果
合わせて微調整する。

```
in_f <- "sample34_ngs.fasta"                #入力ファイル名を指定
param_fig <- c(450, 210)                    #ヒストグラム描画時の
param_range <- c(1, 20)                     #表示したい横軸の範囲

library(Biostrings)                         #パッケージの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定

out_f <- "hoge1.png"                        #出力ファイル名を指定
param_kmer <- 6                             #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#
out <- colSums(hoge)                        #列ごとの総和をoutに
kmer <- out[out > 0]                        #1回以上出現したk-me
png(out_f, pointsize=13, width=param_fig[1], height=param_
par(mar=c(4, 4, 0, 0))                      #下、左、上、右の順で
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム
     xlab=paste("Number of occurences at k=", param_kmer,
     xlim=param_range, main="")             #ヒストグラムを描画
dev.off()                                   #おまじない

out_f <- "hoge2.png"                        #出力ファイル名を指定
param_kmer <- 8                             #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#
```



おさらい

k-mer出現頻度解析は、このように1塩基づつずらして部分配列を発生させて出現回数を解析しています。L塩基長のリードをk-merで分割すると、(L − k + 1) 個のk-merを発生可能



Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

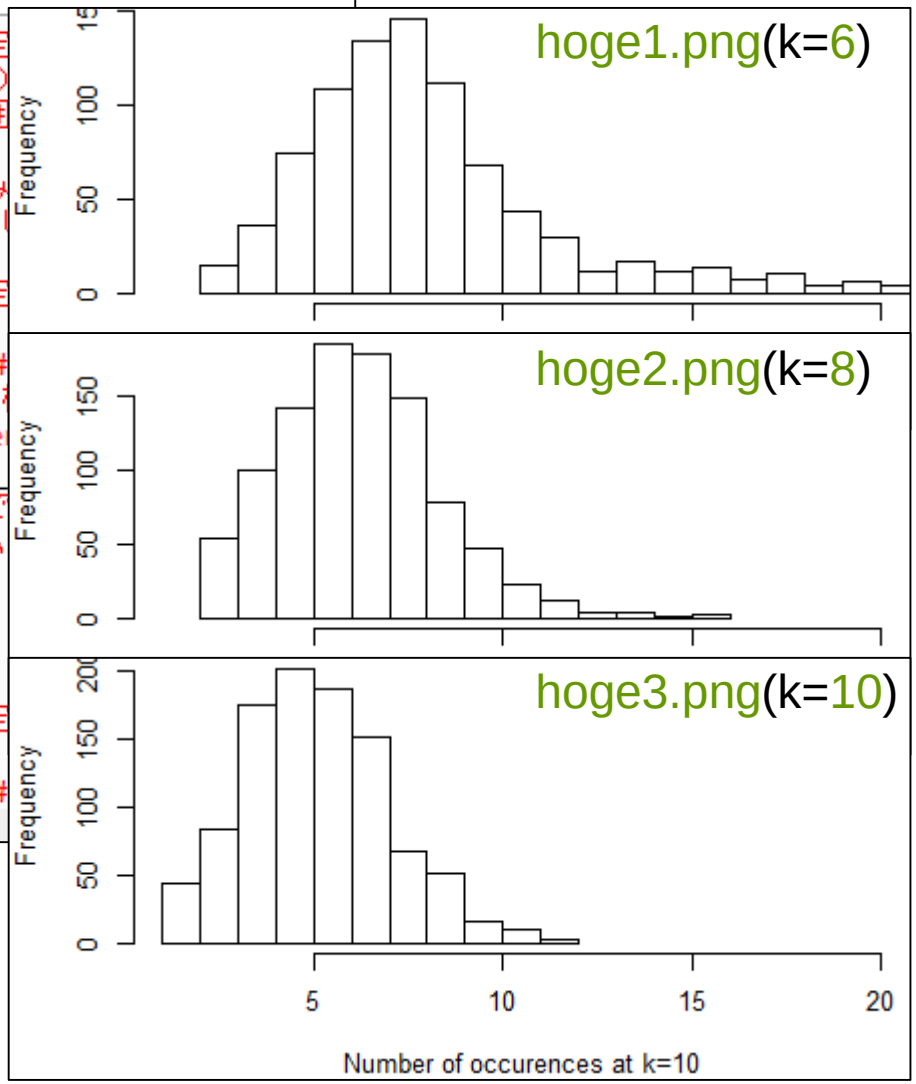
作図（k=6, 8, 10）

①は1000 bpの仮想リファレンス配列からランダム抽出した、長さ20 bpのリードが500個からなるシークエンスエラーのない仮想データ

- 作図(k=6, 8, 10)

上記のk=6, 8, 10の結果より、横軸の最大値は39であることがわかっているが、ヒストグラム全体の変遷が把握しづらいので 1から20の範囲を指定している。このあたりは結果を眺めながら自分の好みに合わせて微調整する。

```
in_f <- "sample34_ngs.fasta"                    #入力ファイル名を指定
param_fig <- c(450, 210)                        #ヒストグラム描画時の
param_range <- c(1, 20)                         #表示したい横軸の範囲

library(Biostrings)                             #パッケージの読み込み
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指定

out_f <- "hoge1.png"                            #出力ファイル名を指定
param_kmer <- 6                                 #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#
out <- colSums(hoge)                            #列ごとの総和をoutに
kmer <- out[out > 0]                            #1回以上出現したk-me
png(out_f, pointsize=13, width=param_fig[1], height=param_
par(mar=c(4, 4, 0, 0))                          #下、左、上、右の順で
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム
     xlab=paste("Number of occurences at k=", param_kmer,
     xlim=param_range, main="")                 #ヒストグラムを描画
dev.off()                                       #おまじない

out_f <- "hoge2.png"                            #出力ファイル名を指定
param_kmer <- 8                                 #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#
```



塩基置換

赤枠内をコピペした結果のR Console画面。①は長さ20 bpのリードが500個からなるシークエンスエラーのない仮想データ。500個の全リードについて、②18番目のポジションに1塩基だけ「TならA、CならG」となるような塩基置換（仮想シークエンスエラー）を入れて、k-mer出現頻度分布がどうなるかをみる

- 塩基置換

GならC、AならTのような塩基置換を行うenkichikan関数
の塩基置換を行う。「書籍 | トランスクリプトーム解析 | 2.
す。

①

```
in_f <- "sample34_ngs.fasta"                #入力ファイル名を指定してin_fに格納
param_fig <- c(450, 210)                    #ヒストグラム描画時の横幅と縦幅を指定
param_range <- c(1, 20)                     #表示した

library(Biostrings)                         #パッケー
fasta <- readDNAStringSet(in_f, format="fasta")
fasta                                       #確認して

#塩基置換関数の作成
enkichikan <- function(fa, p) {             #関数名や
  t <- substring(fa, p, p)                  #置換した
  t_c <- chartr("CGAT", "GCTA", t)          #置換後の
  substring(fa, p, p) <- t_c                #置換
  return(fa)                                #置換後の
}

#enkichikan関数の動作確認
as.character(fasta[500])                      #500番
enkichikan(as.character(fasta[500]), 18) #18番目
```

②

```
#塩基置換本番
fasta_org <- fasta                          #塩基置換
```

R Console

```
[499]     20 GACAGCGGGAGTGATCCGGC     kkk_401_420
[500]     20 TGTAAGCCGTACCCTACCCC     kkk_904_923
>
> #塩基置換関数の作成
> enkichikan <- function(fa, p) {           #関数名$
+   t <- substring(fa, p, p)                #置換し$
+   t_c <- chartr("CGAT", "GCTA", t)        #置換後$
+   substring(fa, p, p) <- t_c              #置換
+   return(fa)                              #置換後$
+ }
>
> #enkichikan関数の動作確認
> as.character(fasta[500])                      #500$
          kkk_904_923
"TGTAAGCCGTACCCTACCCC"
> enkichikan(as.character(fasta[500]), 18) #18番$
[1] "TGTAAGCCGTACCCTACGCC"
>
```

塩基置換

①はfastaオブジェクトの500番目の要素を文字列として(as.character)表示させた結果。つまり、②と同じもの

- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍 |トランスクリプトーム解析 | 2.3.4 マッピング(準備)」のp81と同じ関数です。

```
in_f <- "sample34_ngs.fasta"              #入力ファイル名を指定してin_fに格納
param_fig <- c(450, 210)                  #ヒストグラム描画時の横幅と縦幅を指定
param_range <- c(1, 20)                   #表示した

library(Biostrings)                       #パッケー
fasta <- readDNAStringSet(in_f, format="fasta")
fasta                                     #確認して

#塩基置換関数の作成
enkichikan <- function(fa, p) {           #関数名や
  t <- substring(fa, p, p)                #置換した
  t_c <- chartr("CGAT", "GCTA", t)        #置換後の
  substring(fa, p, p) <- t_c              #置換
  return(fa)                              #置換後の
}

#enkichikan関数の動作確認
as.character(fasta[500])                  #500番
enkichikan(as.character(fasta[500]), 18) #18番目

#塩基置換本番
fasta_org <- fasta                        #塩基置換
```

①

```
R Console

[499]     20 GACAGCGGGAGTGATCCGGC     kkk_401_420
[500]     20 TGTAAGCCGTACCCTACCCC     kkk_904_923
>
> #塩基置換関数の作成
> enkichikan <- function(fa, p) {          #関数名$
+   t <- substring(fa, p, p)               #置換し$
+   t_c <- chartr("CGAT", "GCTA", t)       #置換後$
+   substring(fa, p, p) <- t_c             #置換
+   return(fa)                             #置換後$
+ }
>
> #enkichikan関数の動作確認
> as.character(fasta[500])                 #500$
            kkk_904_923
"TGTAAGCCGTACCCTACCCC"
> enkichikan(as.character(fasta[500]), 18) #18番$
[1] "TGTAAGCCGTACCCTACGCC"
>
```

②

①

塩基置換

①の部分で作成しておいたenkichikan関数を用いて、②18番目の塩基を置換。③確かにCがGに置換されている

- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍 | トランスクリプトーム解析 | 2.3.4 マッピング(準備)」のp81と同じ関数です。

```
in_f <- "sample34_ngs.fasta"                #入力ファイル名を指定してin_fに格納
param_fig <- c(450, 210)                    #ヒストグラム描画時の横幅と縦幅を指定
param_range <- c(1, 20)                     #表示した

library(Biostrings)                         #パッケー
fasta <- readDNAStringSet(in_f, format="fasta")
fasta                                       #確認して

#塩基置換関数の作成
enkichikan <- function(fa, p) {             #関数名や
  t <- substring(fa, p, p)                  #置換した
  t_c <- chartr("CGAT", "GCTA", t)          #置換後の
  substring(fa, p, p) <- t_c                #置換
  return(fa)                                #置換後の
}

#enkichikan関数の動作確認
as.character(fasta[500])                    #500番
enkichikan(as.character(fasta[500]), 18)    #18番目

#塩基置換本番
fasta_org <- fasta                          #塩基置換
```

①

②

```
R Console

[499]     20 GACAGCGGGAGTGATCCGGC     kkk_401_420
[500]     20 TGTAAGCCGTACCCTACCCC     kkk_904_923
>                                          ③
> #塩基置換関数の作成
> enkichikan <- function(fa, p) {          #関数名$
+   t <- substring(fa, p, p)               #置換し$
+   t_c <- chartr("CGAT", "GCTA", t)       #置換後$
+   substring(fa, p, p) <- t_c             #置換
+   return(fa)                             #置換後$
+ }
>
> #enkichikan関数の動作確認
> as.character(fasta[500])                  #500$
          kkk_904_923
"TGTAAGCCGTACCCTACCCC"
> enkichikan(as.character(fasta[500]), 18)  ②8番$
[1] "TGTAAGCCGTACCCTAGCC"
>                                    ③
```

塩基置換

①コード下部に移動。②全リードについて18番目の塩基を置換したい場合には、入力部分をas.character(fasta)とすればよい

- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍|トランスクリプトーム解析|2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換前のfastaをfasta_orgに格納
hoge <- enkichikan(as.character(fasta), 18) #18番目の塩基を置換
fasta <- DNAStringSet(hoge)                 #DNA塩基配列だと認識させるDNAString
names(fasta) <- names(fasta_org)            #fasta_orgのdescription情報をコピー
fasta                                       #確認してるだけです

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファイル名を指定してout_fに格納
param_kmer <- 8                             #k-merのkの値を指定
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻
out <- colSums(hoge)                        #列ごとの総和をoutに格納
kmer <- out[out > 0]                        #1回以上出現したk-merのみをkmerに格納
png(out_f, pointsize=13, width=param_fig[1], height=param_fig[2])#出力フ
par(mar=c(4, 4, 0, 0))                      #下、左、上、右の順で余白（行）を指
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラムを描画
     xlab=paste("Number of occurences at k=", param_kmer, sep=""),#ヒスト
     xlim=param_range, main="")             #ヒストグラムを描画
dev.off()                                   #おまじない
```

塩基置換

②実行後のhogeオブジェクトは、文字列ベクトル。③それゆえDNAStringSet関数を実行して、見慣れたDNAStringSet形式のfastaオブジェクトに変換している

- 塩基置換

GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍 | トランスクリプトーム解析 | 2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換
hoge <- enkichikan(as.character(fasta), 18) #置換
fasta <- DNAStringSet(hoge)                 #DNA塩基配
names(fasta) <- names(fasta_org)            #fasta
fasta                                       #確認し

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファ
param_kmer <- 8                             #k-merの
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                        #列ごとの
kmer <- out[out > 0]                        #1回以上
png(out_f, pointsize=13, width=param_fig[1],
par(mar=c(4, 4, 0, 0))                      #下、左
hist(kmer, breaks=max(kmer), ylab="Frequency",
     xlab=paste("Number of occurences at k=", pa
     xlim=param_range, main="")             #ヒストグ
dev.off()                                   #おまじな
```

③

②

②

③

R Console

```
> #塩基置換本番
> fasta_org <- fasta                          #塩基置$
> hoge <- enkichikan(as.character(fasta), 18)#18$
> head(hoge)
[1] "GGTACGGTTCCGGTTGCGGA" "TTATCCGGCAGTCCTATTTG"
[3] "CAATAGACACCACGCGCGAC" "ACCGAAAGTTCTGGAGGACA"
[5] "TTGGGCCCGATATCTATTAA" "TATGATATGGGCCGCTGGGG"
> length(hoge)
[1] 500
> fasta <- DNAStringSet(hoge)                 #DNA塩$
> fasta
  A DNAStringSet instance of length 500
      width seq
  [1]    20 GGTACGGTTCCGGTTGCGGA
  [2]    20 TTATCCGGCAGTCCTATTTG
  [3]    20 CAATAGACACCACGCGCGAC
  [4]    20 ACCGAAAGTTCTGGAGGACA
  [5]    20 TTGGGCCCGATATCTATTAA
```

塩基置換

ただし、変換直後の①fastaオブジェクトはdescription情報がついていない。(k-mer出現頻度解析自体には無関係だが、気持ち悪いと思うヒトのために)②で塩基置換前の情報を保持するfasta_orgのdescription情報部分を③でコピーしている

- 塩基置換

GならC、AならTのような塩基置換を行うenkichikan関数(孫[illegible])を利用して任意の位置の塩基置換を行う。「書籍|トランスクリプトーム解析|2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換
hoge <- enkichikan(as.character(fasta), 18)#18番
fasta <- DNAStringSet(hoge)                 #DNA塩基配
names(fasta) <- names(fasta_org)            #fasta_or
fasta                                       #確認して

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファ
param_kmer <- 8                             #k-merのk
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                        #列ごとの
kmer <- out[out > 0]                        #1回以上出
png(out_f, pointsize=13, width=param_fig[1], he
par(mar=c(4, 4, 0, 0))                      #下、左
hist(kmer, breaks=max(kmer), ylab="Frequency"
     xlab=paste("Number of occurences at k=",
     xlim=param_range, main="")             #ヒストグ
dev.off()                                   #おまじな
```

②

③

①

R Console

```
> #塩基置換本番
> fasta_org <- fasta                          #塩基置$
> hoge <- enkichikan(as.character(fasta), 18)#18$
> head(hoge)
[1] "GGTACGGTTCCGGTTGCGGA" "TTATCCGGCAGTCCTATTTG"
[3] "CAATAGACACCACGCGCGAC" "ACCGAAAGTTCTGGAGGACA"
[5] "TTGGGCCCGATATCTATTAA" "TATGATATGGGCCGCTGGGG"
> length(hoge)
[1] 500
> fasta <- DNAStringSet(hoge)                 #DNA塩$
> fasta
  A DNAStringSet instance of length 500
      width seq
  [1]    20 GGTACGGTTCCGGTTGCGGA
  [2]    20 TTATCCGGCAGTCCTATTTG
  [3]    20 CAATAGACACCACGCGCGAC
  [4]    20 ACCGAAAGTTCTGGAGGACA
  [5]    20 TTGGGCCCGATATCTATTAA
```

塩基置換後

①description情報をコピーした塩基置換後のfastaオブジェクトの中身は②

- 塩基置換

GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍|トランスクリプトーム解析|2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換
hoge <- enkichikan(as.character(fasta), 18)#18番
fasta <- DNAStringSet(hoge)                 #DNA塩基配
names(fasta) <- names(fasta_org)            #fasta_or
fasta                                       #確認して

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファ
param_kmer <- 8                             #k-merのk
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                        #列ごとの
kmer <- out[out > 0]                        #1回以上出
png(out_f, pointsize=13, width=param_fig[1], hei
par(mar=c(4, 4, 0, 0))                      #下、左、
hist(kmer, breaks=max(kmer), ylab="Frequency",#
     xlab=paste("Number of occurences at k=", pa
     xlim=param_range, main="")             #ヒストグ
dev.off()                                   #おまじな
```

①

②

R Console

```
[499]    20 GACAGCGGGAGTGATCCCGC
[500]    20 TGTAAGCCGTACCCTACGCC
> names(fasta) <- names(fasta_org)        #fasta_$
> fasta                                   #確認し$
  A DNAStringSet instance of length 500
      width seq                 names
  [1]    20 GGTACGG...TGCGGA kkk_658_677
  [2]    20 TTATCCG...TATTTG kkk_197_216
  [3]    20 CAATAGA...CGCGAC kkk_559_578
  [4]    20 ACCGAAA...AGGACA kkk_55_74
  [5]    20 TTGGGCC...TATTAA kkk_681_700
  ...   ... ...
[496]    20 GAGCTAT...CCCGAG kkk_433_452
[497]    20 AACCTAA...GTGCCG kkk_805_824
[498]    20 CGCCGCC...TCGGTT kkk_140_159
[499]    20 GACAGCG...TCCCGC kkk_401_420
[500]    20 TGTAAGC...TACGCC kkk_904_923
>
```

塩基置換前

①塩基置換前のfasta_orgオブジェクトの中身は②。意図通りに置換できていることがわかる

- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍|トランスクリプトーム解析|2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換
hoge <- enkichikan(as.character(fasta), 18)#18番
fasta <- DNAStringSet(hoge)                 #DNA塩基配
names(fasta) <- names(fasta_org)            #fasta_or
fasta                                       #確認して

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファ
param_kmer <- 8                             #k-merの
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                        #列ごとの
kmer <- out[out > 0]                        #1回以上出
png(out_f, pointsize=13, width=param_fig[1], hei
par(mar=c(4, 4, 0, 0))                      #下、左、
hist(kmer, breaks=max(kmer), ylab="Frequency",#
     xlab=paste("Number of occurences at k=", pa
     xlim=param_range, main="")             #ヒストグ
dev.off()                                   #おまじな
```

①

②

R Console

```
[498]     20 CGCCGCC...TCGGTT kkk_140_159
[499]     20 GACAGCG...TCCCGC kkk_401_420
[500]     20 TGTAAGC...TACGCC kkk_904_923
> fasta_org
  A DNAStringSet instance of length 500
      width seq                 names
   [1]     20 GGTACGG...TGCCGA kkk_658_677
   [2]     20 TTATCCG...TATATG kkk_197_216
   [3]     20 CAATAGA...CGCCAC kkk_559_578
   [4]     20 ACCGAAA...AGGTCA kkk_55_74
   [5]     20 TTGGGCC...TATAAA kkk_681_700
   ...    ... ...
[496]     20 GAGCTAT...CCCCAG kkk_433_452
[497]     20 AACCTAA...GTGGCG kkk_805_824
[498]     20 CGCCGCC...TCGCTT kkk_140_159
[499]     20 GACAGCG...TCCGGC kkk_401_420
[500]     20 TGTAAGC...TACCCC kkk_904_923
>
```

塩基置換後の…

塩基置換後のfastaオブジェクトを入力として、①k-mer出現頻度分布図を作成

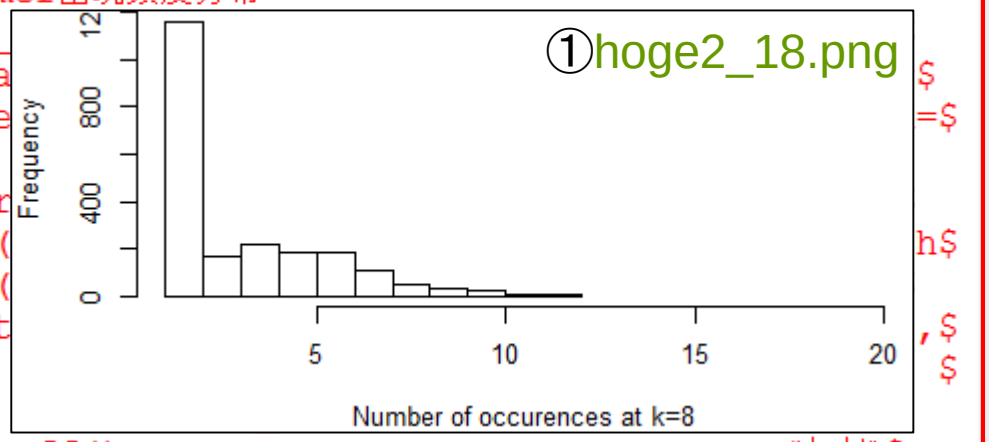
- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数(孫 建強氏作成)を利用して任意の位置の塩基置換を行う。「書籍 | トランスクリプトーム解析 | 2.3.4 マッピング(準備)」のp81と同じ関数です。

```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置換

#塩基置換本番
fasta_org <- fasta                          #塩基置換
hoge <- enkichikan(as.character(fasta), 18)#18番
fasta <- DNAStringSet(hoge)                 #DNA塩基配
names(fasta) <- names(fasta_org)            #fasta_or
fasta                                       #確認して

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                     #出力ファ
param_kmer <- 8                             #k-merのk
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                        #列ごとの
kmer <- out[out > 0]                        #1回以上出
png(out_f, pointsize=13, width=param_fig[1], hei
par(mar=c(4, 4, 0, 0))                      #下、左、
hist(kmer, breaks=max(kmer), ylab="Frequency",#
     xlab=paste("Number of occurences at k=", pa
     xlim=param_range, main="")             #ヒストグ
dev.off()                                   #おまじな
```

①

R Console

```
[498]     20 CGCCGCC...TCGCTT kkk_140_159
[499]     20 GACAGCG...TCCGGC kkk_401_420
[500]     20 TGTAAGC...TACCCC kkk_904_923
> #k-mer出現頻度分布
> out_
> para
> hoge
> out
> kmer
> png(
> par(
> hist
+
+
> dev.off()                              #おまじ$
null device
          1
>
```



塩基置換後の...

①塩基置換による仮想シークエンスエラー導入後と②導入前(同じk=8)の主な違いは、①のシークエンスエラーを含むデータにおいて、③1-2回しか出現しないk-merが1,200個弱存在する点。これがリアルデータでもよく見られるk-mer出現頻度分布

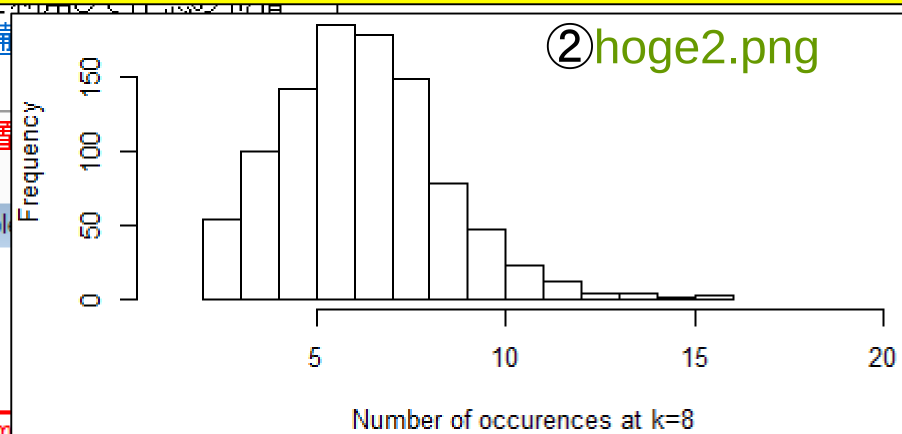
- 塩基置換
  GならC、AならTのような塩基置換を行うenkichikan関数... の塩基置換を行う。「書籍 | トランスクリプトーム解析 | 2.3.4 マッピング(準備... す。

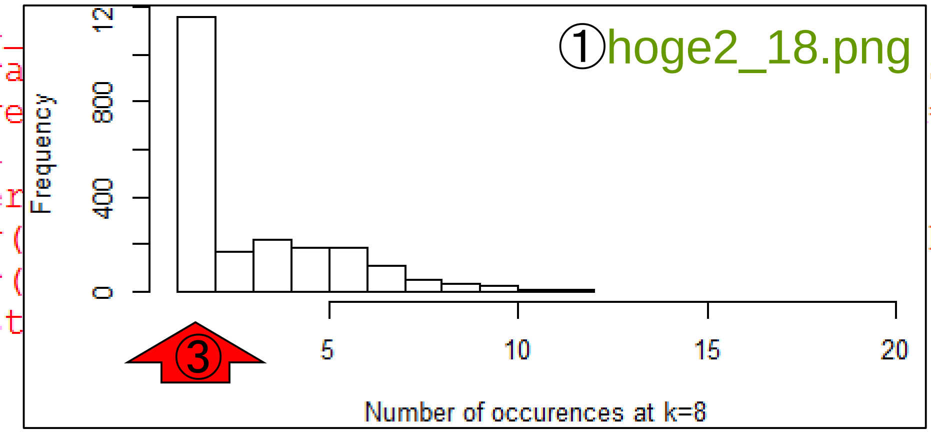
```
enkichikan(as.character(fasta[500]), 18) #18番目の塩基を置

#塩基置換本番
fasta_org <- fasta                              #塩基置換
hoge <- enkichikan(as.character(fasta), 18)#18番
fasta <- DNAStringSet(hoge)                     #DNA塩基
names(fasta) <- names(fasta_org)                #fasta_or
fasta                                           #確認して

#k-mer出現頻度分布
out_f <- "hoge2_18.png"                         #出力ファ
param_kmer <- 8                                 #k-merのk
hoge <- oligonucleotideFrequency(fasta, width=pa
out <- colSums(hoge)                            #列ごとの
kmer <- out[out > 0]                            #1回以上出
png(out_f, pointsize=13, width=param_fig[1], hei
par(mar=c(4, 4, 0, 0))                          #下、左、
hist(kmer, breaks=max(kmer), ylab="Frequency",#
     xlab=paste("Number of occurences at k=", pa
     xlim=param_range, main="")                 #ヒストグ
dev.off()                                       #おまじな
```

①

R Console

```
[498]
[499]
[500]
> #k-mer出現頻度分布
> out_
> para
> hoge
> out
> kmer
> png(
> par(
> hist
+
+
> dev.off()                              #おまじ$
null device
          1
>
```





縦軸の範囲を揃える

縦軸の範囲が違いすぎるので、①0-250に統一して眺める

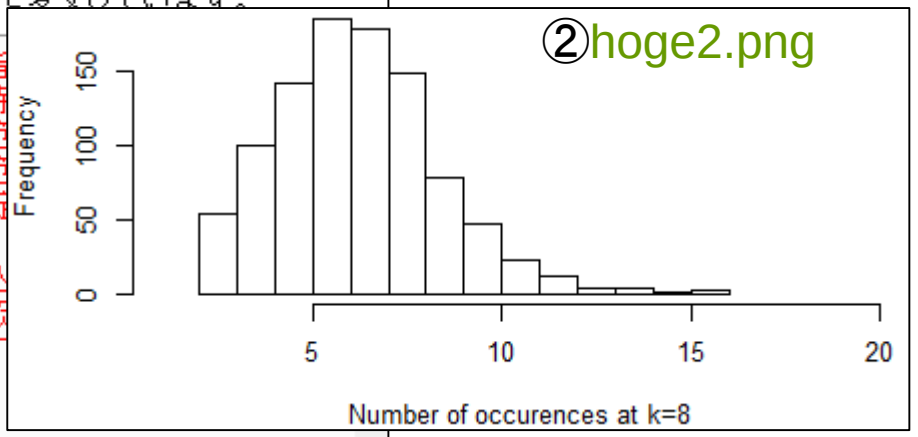
- 縦軸の範囲を揃える

ついでに、x軸の範囲指定のところの名前をparam_rangeからparam_xrangeに変更しています。

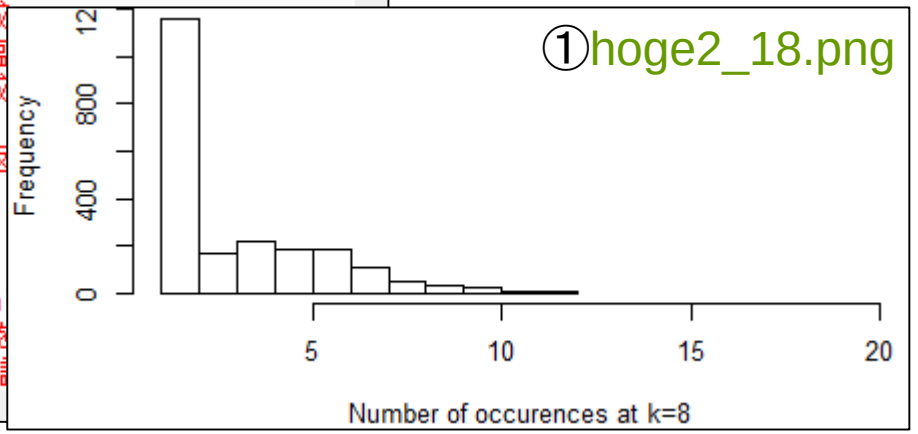
```
in_f <- "sample34_ngs.fasta"              #入力ファイル名を指
param_fig <- c(450, 210)                  #ヒストグラム描画時
param_xrange <- c(1, 20)                  #表示したい横軸の範
param_yrange <- c(0, 250)                 #表示したい縦軸の範
param_kmer <- 8                           #k-merのkの値を指定

library(Biostrings)                       #パッケージの読み込
fasta <- readDNAStringSet(in_f, format="fasta")#in_fで指
fasta                                     #確認してるだけです

#塩基置換関数の作成
enkichikan <- function(fa, p) {           #関数名や引数の作成
  t <- substring(fa, p, p)                #置換したい位置の塩
  t_c <- chartr("CGAT", "GCTA", t)        #置換後の塩基を作成
  substring(fa, p, p) <- t_c              #置換
  return(fa)                              #置換後のデータを返
}

#塩基置換本番
fasta_org <- fasta                        #塩基置換前のfasta
hoge <- enkichikan(as.character(fasta), 18)#18番目の塩基を
fasta <- DNAStringSet(hoge)               #DNA塩基配列だと認
```

①

②hoge2.png



①hoge2_18.png



縦軸の範囲を揃える

①コード下部はこんな感じ。②塩基置換後と③置換前のpngファイルを一気に作成して眺める。コード全体をコピペ

- 縦軸の範囲を揃える

  ついでに、x軸の範囲指定のところの名前をparam_rangeからparam_xrangeに変更しています。

```
#k-mer出現頻度分布(塩基置換後)
out_f <- "hoge2_18_250.png"                    #出力ファイル名を指定してout_fに格納
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                           #列ごとの総和をoutに格納
kmer <- out[out > 0]                           #1回以上出現したk-merのみをkmerに格納
png(out_f, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイル
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順で余白（行）を指定
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラムを描画
     xlab=paste("Number of occurences at k=", param_kmer, sep=""),#ヒストグラ
     xlim=param_xrange, ylim=param_yrange, main="")#ヒストグラムを描画
dev.off()                                      #おまじない

#k-mer出現頻度分布(塩基置換前)
fasta <- fasta_org                             #塩基置換前のfasta_orgをfastaに格納
out_f <- "hoge2_250.png"                       #出力ファイル名を指定してout_fに格納
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                           #列ごとの総和をoutに格納
kmer <- out[out > 0]                           #1回以上出現したk-merのみをkmerに格納
png(out_f, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイル
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順で余白（行）を指定
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラムを描画
     xlab=paste("Number of occurences at k=", param_kmer, sep=""),#ヒストグラ
```

縦軸の範囲を揃える

①塩基置換後と②置換前。シークエンスエラーに相当する置換塩基を含むk-merは③のあたりに位置する。それ以外は置換塩基を含まないものたち

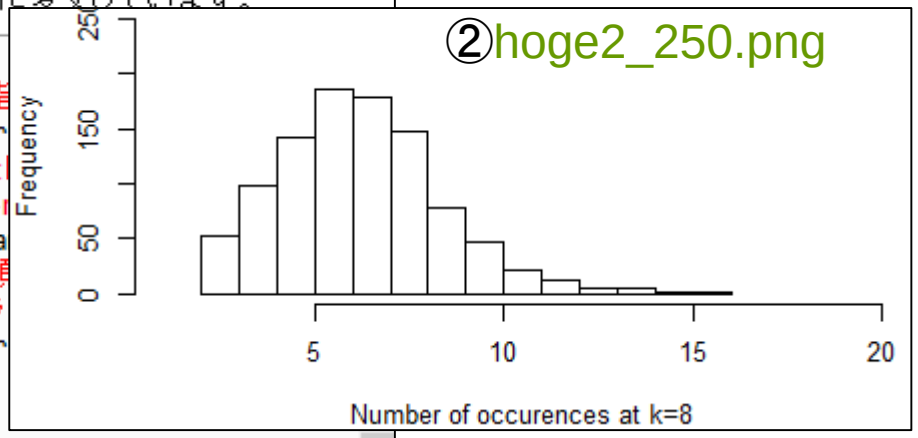
- 縦軸の範囲を揃える

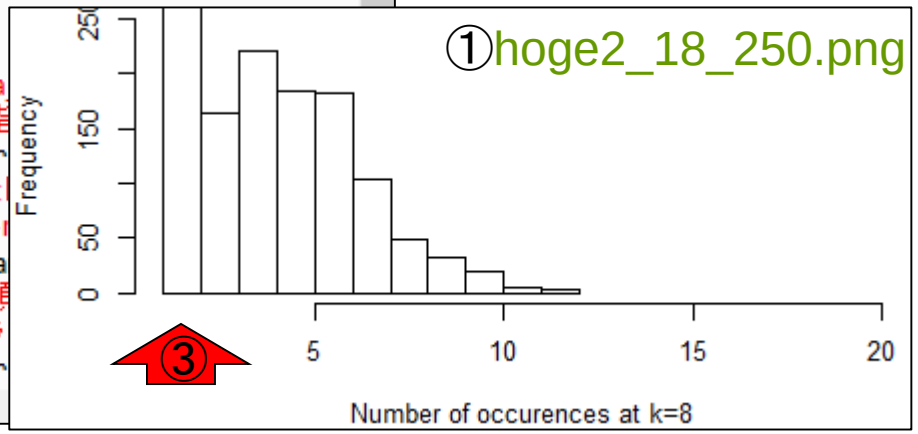
ついでに、x軸の範囲指定のところの名前をparam_rangeからparam_xrangeに変更しています。

```
#k-mer出現頻度分布(塩基置換後)
out_f <- "hoge2_18_250.png"                    #出力ファイル名を指
hoge <- oligonucleotideFrequency(fasta, width=param_kmer
out <- colSums(hoge)                           #列ごとの総和をout
kmer <- out[out > 0]                           #1回以上出現したk-
png(out_f, pointsize=13, width=param_fig[1], height=para
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラ
     xlab=paste("Number of occurences at k=", param_kmer
     xlim=param_xrange, ylim=param_yrange, main="")#ヒス
dev.off()                                      #おまじない

#k-mer出現頻度分布(塩基置換前)
fasta <- fasta_org                             #塩基置換前のfasta
out_f <- "hoge2_250.png"                       #出力ファイル名を指
hoge <- oligonucleotideFrequency(fasta, width=param_kmer
out <- colSums(hoge)                           #列ごとの総和をout
kmer <- out[out > 0]                           #1回以上出現したk-
png(out_f, pointsize=13, width=param_fig[1], height=para
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラ
     xlab=paste("Number of occurences at k=", param_kmer
```

①

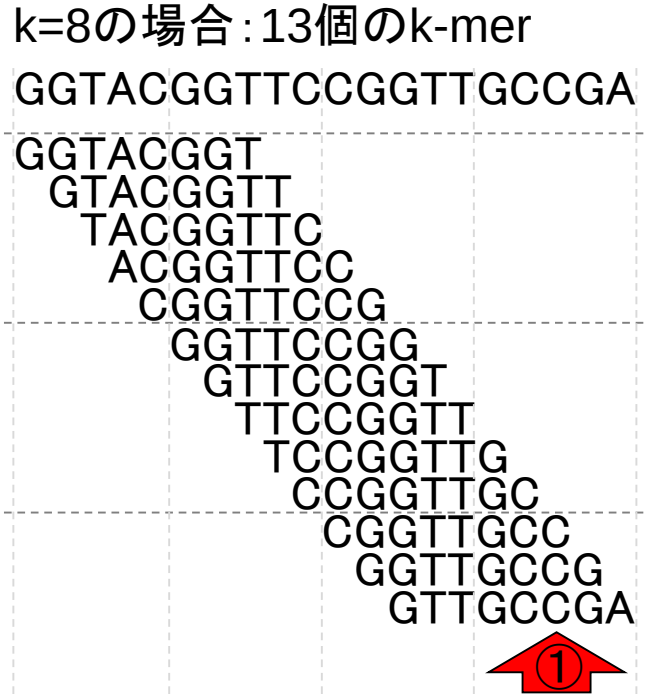
②

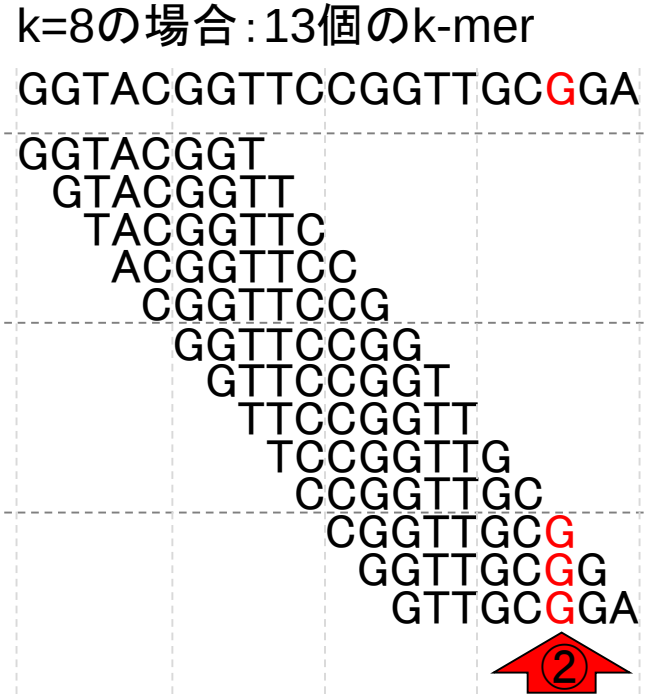




おさらい

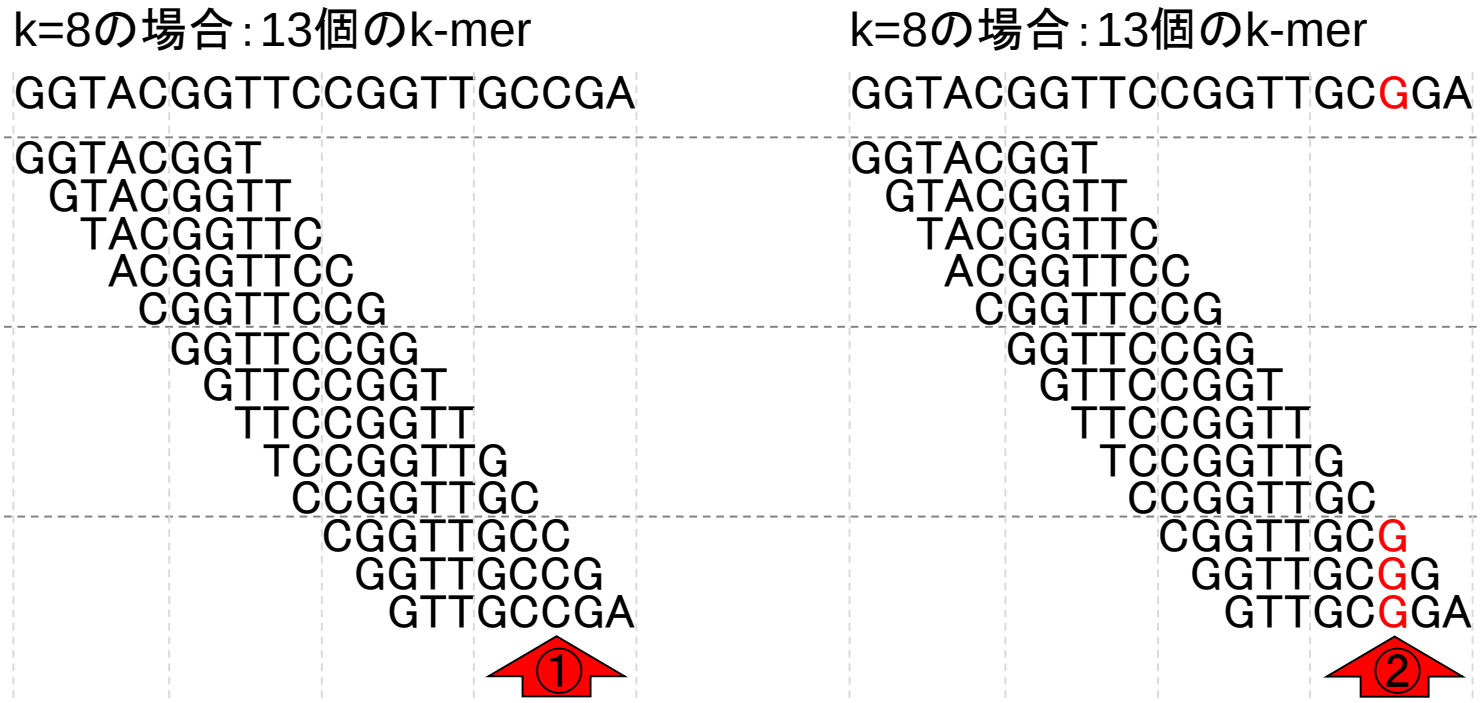
今は、長さ20 bpのリードが500個からなる仮想データに対して、18番目の塩基を置換してk-mer分布がどう変化するかを調べている





おさらい

k=8の場合は、リードあたり13個のk-merが生成できる。500リードのトータルで、500 × 13 = 6,500個のk-merとなる。そのうち、置換を行った18番目の塩基を含む、仮想シークエンスエラー由来k-merは、リードあたり3個。トータルで500 × 3 = 1,500個



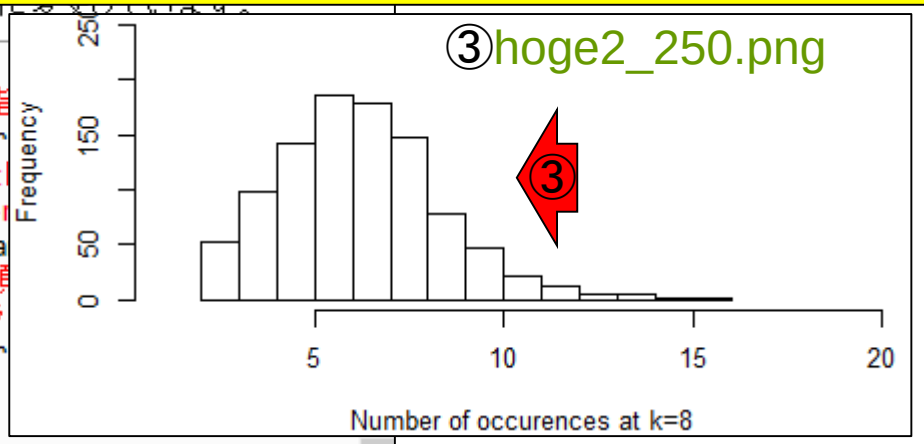
縦軸の範囲を揃え

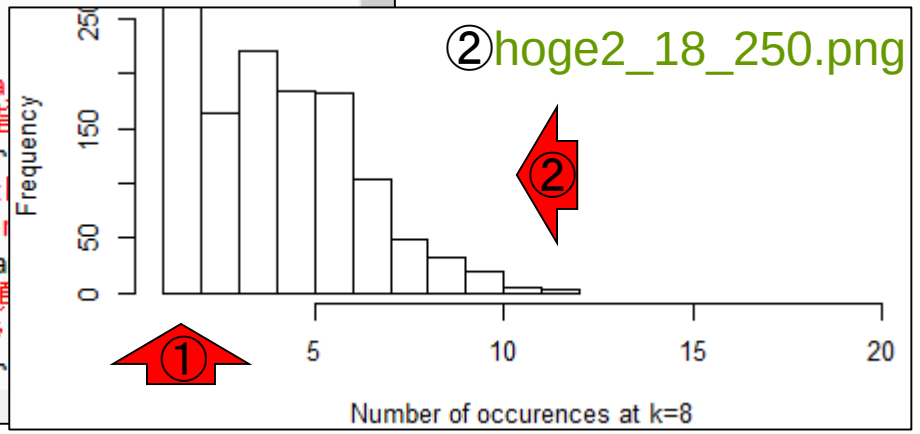
①の総数が1,200弱程度だったのは、シークエンスエラー由来k-merが1,500個程度だったことに起因する。②塩基置換後の分布が③置換前に比べて左にずれているのは、エラーなしが6,500→ 5,000個に減っているため

- 縦軸の範囲を揃える

ついでに、x軸の範囲指定のところの名前をparam_rangeからparam_xrangeに変えておく

```
#k-mer出現頻度分布(塩基置換後)
out_f <- "hoge2_18_250.png"                    #出力ファイル名を指
hoge <- oligonucleotideFrequency(fasta, width=param_kmer
out <- colSums(hoge)                           #列ごとの総和をout
kmer <- out[out > 0]                           #1回以上出現したk-
png(out_f, pointsize=13, width=param_fig[1], height=para
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラ
     xlab=paste("Number of occurences at k=", param_kmer
     xlim=param_xrange, ylim=param_yrange, main="")#ヒス
dev.off()                                      #おまじない

#k-mer出現頻度分布(塩基置換前)
fasta <- fasta_org                             #塩基置換前のfasta
out_f <- "hoge2_250.png"                       #出力ファイル名を指
hoge <- oligonucleotideFrequency(fasta, width=param_kmer
out <- colSums(hoge)                           #列ごとの総和をout
kmer <- out[out > 0]                           #1回以上出現したk-
png(out_f, pointsize=13, width=param_fig[1], height=para
par(mar=c(4, 4, 0, 0))                         #下、左、上、右の順
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラ
     xlab=paste("Number of occurences at k=", param_kmer
```



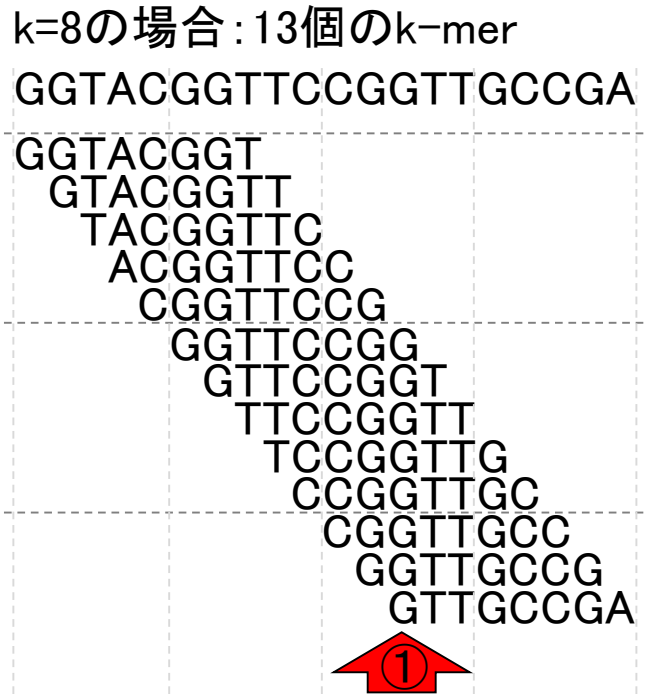


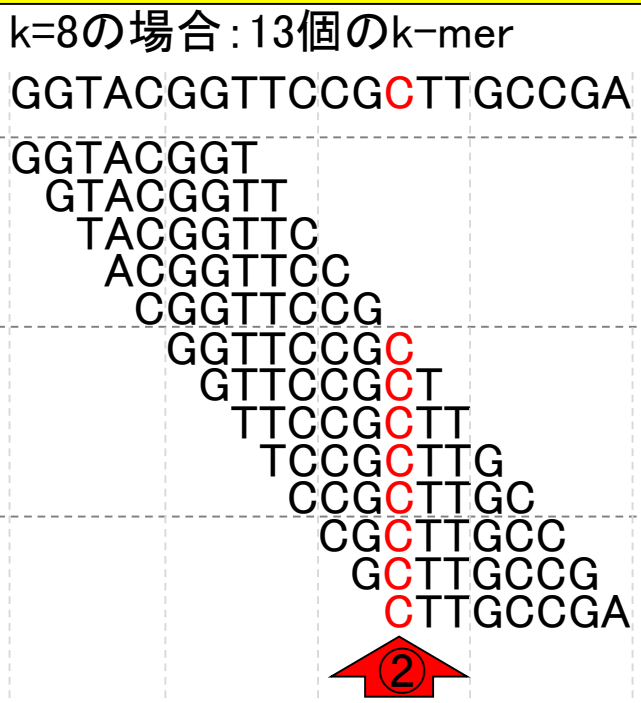
Contents

- NGS解析手段、ウェブツール(DDBJ Pipeline)との連携
- DDBJ PipelineでPlatanusを実行
- k-mer解析（k個の連続塩基に基づく各種解析）の基礎
  - 塩基ごとの出現頻度解析(k=1)、一気に計算
  - 2連続塩基の出現頻度解析(k=2)、基本スキルの復習や作図
- de novoアセンブリ時のエラー補正やゲノムサイズ推定の基本的な考え方
  - ランダムな塩基配列（仮想ゲノムおよび仮想NGSデータ）の作成
  - k-mer解析の応用、ゲノムサイズ推定の基礎
  - ゲノムサイズ推定（1,000 bpの仮想ゲノムの場合）
  - ゲノムサイズ推定（1,000 bpの仮想ゲノム；4X → 10X coverageの場合）
  - k-mer出現頻度分布
  - シークエンスエラーを含む場合
  - 最終確認

最終確認

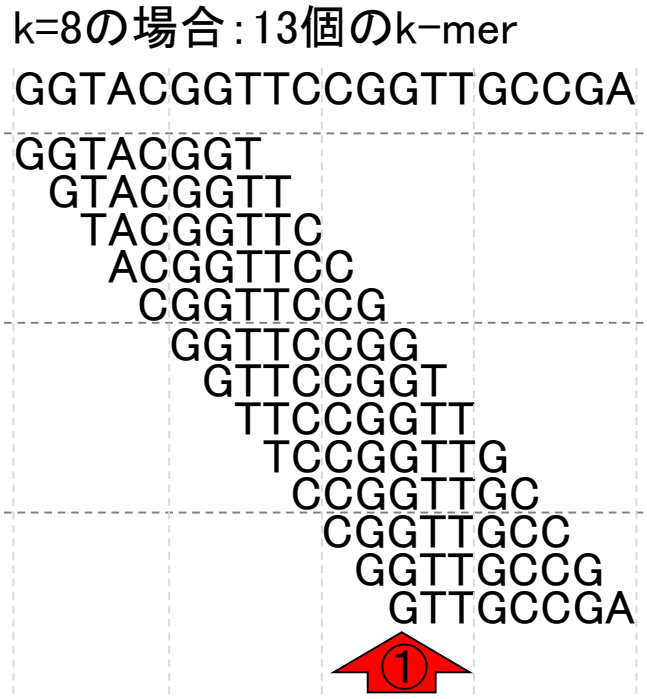
k=8の場合は、(8番目から13番目の範囲であればどこでも同じであるが)もし②13番目の塩基に置換を入れれば、仮想シークエンスエラー由来k-merは、リードあたり8個。最後の50リード(451-500番目のリード)に対してこの置換を導入すると、トータルで50×8 = 400個の仮想シークエンスエラー由来k-merを発生させることになる

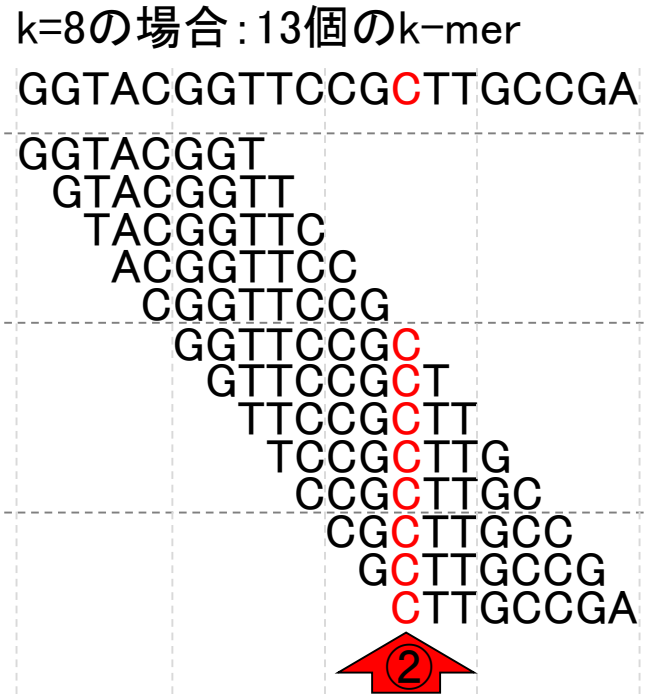




狙い

エラーを全く含まない1-450番目のリードのみのk-mer分布と、エラーを多く含む451-500番目のリードのk-mer出現頻度分布を独立に作成して眺め、エラーを含まないk-merのみ取り扱える（フィルタリングできる）ことを学ぶ





最終確認

塩基置換を行わない最初の450リード分の情報は①fasta_first450としている

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
in_f <- "sample34_ngs.fasta"          #入力ファイル名を指定してin_fに格納
param_fig <- c(450, 210)              #ヒストグラム描画時の横幅と縦幅を指定(単
param_xrange <- c(1, 20)              #表示したい横軸の範囲を指定
param_yrange <- c(0, 350)             #表示したい縦軸の範囲を指定
param_kmer <- 8                       #k-merのkの値を指定

library(Biostrings)
fasta <- readDNAStringSet(in_f,
fasta
fasta_first450 <- fasta[1:450]
fasta_first450

#塩基置換関数の作成
enkichikan <- function(fa, p) {
  t <- substring(fa, p, p)
  t_c <- chartr("CGAT", "GCTA",
  substring(fa, p, p) <- t_c
  return(fa)
}

#塩基置換本番(451:500のみ)
fasta_org <- fasta
```

①

R Console

```
[499]    20 GACAGCGGGAGTGATCCGGC          kkk_401_420
[500]    20 TGTAAGCCGTACCCTACCCC          kkk_904_923
> fasta_first450 <- fasta[1:450]         #塩基置換を行わない$
> fasta_first450                         #確認してるだけです
  A DNAStringSet instance of length 450
      width seq                           names
  [1]    20 GGTACGGTTCCGGTTGCCGA          kkk_658_677
  [2]    20 TTATCCGGCAGTCCTATATG          kkk_197_216
  [3]    20 CAATAGACACCACGCGCCAC          kkk_559_578
  [4]    20 ACCGAAAGTTCTGGAGGTCA          kkk_55_74
  [5]    20 TTGGGCCCGATATCTATAAA          kkk_681_700
  ...   ... ...
[446]    20 ATATGATATGGGCCGCTGCG          kkk_212_231
[447]    20 ATTTCGTTTCCGCATGACAC          kkk_118_137
[448]    20 GCTCGCTTTTTACTTCCGGA          kkk_152_171
[449]    20 TTTTGAACTCACTACACCAG          kkk_10_29
[450]    20 TGAAGACGCGGACGGCAGTC          kkk_34_53
>
```

①

最終確認

①ちょっと下に移動。②赤枠部分で451-500番目のリードのみに対して13番目の塩基を置換した結果を③fasta_last50としている

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#塩基置換関数の作成
enkichikan <- function(fa, p) {          #関数名や引数の作成
  t <- substring(fa, p, p)               #置換したい位置の塩基を取りだす
  t_c <- chartr("CGAT", "GCTA", t)       #置換後の塩基を作成
  substring(fa, p, p) <- t_c             #置換
  return(fa)                             #置換後のデータを返す
}

#塩基置換本番(451:500のみ)
fasta_org <- fasta                       #塩基置換前のfastaをfasta_orgに格納
hoge <- enkichikan(as.character(fasta[451:500]), 13)#13番目の塩基を置換
fasta <- DNAStringSet(hoge)              #DNA塩基配列だと認識させるDNAStringSet
names(fasta) <- names(fasta_org[451:500])#fasta_orgのdescription情報をコピー
fasta_last50 <- fasta                    #fasta_last50として取り扱う
fasta_last50                             #確認してるだけです

#k-mer出現頻度分布(original)
fasta <- fasta_org                       #fasta_orgをfastaに格納
out_f <- "hoge2_original.png"              #出力ファイル名を指定してout_fに格納
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                     #列ごとの総和をoutに格納
kmer <- out[out > 0]                     #1回以上出現したk-merのみをkmerに格納
```

最終確認

①13番目の塩基置換後のfasta_last50の中身。②置換された塩基ポジション

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#塩基置換関数の作成
enkichikan <- function(fa, p) {          #関数名や引数の作成
  t <- substring(fa, p, p)               #置換したい位置の塩基を取りだす
  t_c <- chartr("CGAT", "GCTA", t)       #置換後の塩基を作成
  substring(fa, p, p) <- t_c             #置換
  return(fa)
}

#塩基置換本番(451:500のみ)
fasta_org <- fasta
hoge <- enkichikan(as.characte
fasta <- DNAStringSet(hoge)
names(fasta) <- names(fasta_or
fasta_last50 <- fasta
fasta_last50

#k-mer出現頻度分布(original)
fasta <- fasta_org
out_f <- "hoge2_original.png"
hoge <- oligonucleotideFrequenc
out <- colSums(hoge)
kmer <- out[out > 0]
```

①

R Console

```
> fasta <- DNAStringSet(hoge)                #DNA塩基配列だと認識$
> names(fasta) <- names(fasta_org[451:500])#fasta_orgのdescri$
> fasta_last50 <- fasta                      #fasta_last50として$
> fasta_last50                               #確認してるだけです
  A DNAStringSet instance of length 50
      width seq                                names
 [1]    20 CTGCGACCGTTCTCTAAATC                kkk_746_765
 [2]    20 TGGGCCTCCGGAACCCAGTT                kkk_956_975
 [3]    20 GGACGCTTTTCATGTCCAGC                kkk_933_952
 [4]    20 CAGTTTTGCTAACTGGTAGG                kkk_265_284
 [5]    20 GTACGGTTCCGGATGCCGAT                kkk_659_678
 ...   ... ...
[46]    20 GAGCTATCAATGAACCCCAG                kkk_433_452
[47]    20 AACCTAATCGCCGCGTGGCG                kkk_805_824
[48]    20 CGCCGCCTTCGACCTCGCTT                kkk_140_159
[49]    20 GACAGCGGGAGTCATCCGGC                kkk_401_420
[50]    20 TGTAAGCCGTACGCTACCCC                kkk_904_923
>
```

②

最終確認

①置換前のfasta_orgオブジェクトの同じ部分を表示。②確かに13番目の塩基を意図通りに置換できていることがわかる

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#塩基置換関数の作成
enkichikan <- function(fa, p) {          #関数名や引数の作成
  t <- substring(fa, p, p)               #置換したい位置の塩基を取りだす
  t_c <- chartr("CGAT", "GCTA", t)       #置換後の塩基を作成
  substring(fa, p, p) <- t_c             #置換
  return(fa)
}

#塩基置換本番(451:500のみ)
fasta_org <- fasta
hoge <- enkichikan(as.character
fasta <- DNAStringSet(hoge)
names(fasta) <- names(fasta_org
fasta_last50 <- fasta
fasta_last50

#k-mer出現頻度分布(original)
fasta <- fasta_org
out_f <- "hoge2_original.png"
hoge <- oligonucleotideFrequenc
out <- colSums(hoge)
kmer <- out[out > 0]
```

①

R Console

```
[48]     20 CGCCGCCTTCGACCTCGCTT            kkk_140_159
[49]     20 GACAGCGGGAGTCATCCGGC            kkk_401_420
[50]     20 TGTAAGCCGTACGCTACCCC            kkk_904_923
> fasta_org[451:500]
  A DNAStringSet instance of length 50
      width seq                              names
 [1]     20 CTGCGACCGTTCACTAAATC             kkk_746_765
 [2]     20 TGGGCCTCCGGATCCCAGTT             kkk_956_975
 [3]     20 GGACGCTTTTCAAGTCCAGC             kkk_933_952
 [4]     20 CAGTTTTGCTAAGTGGTAGG             kkk_265_284
 [5]     20 GTACGGTTCCGGTTGCCGAT             kkk_659_678
 ...    ... ...
[46]     20 GAGCTATCAATGTACCCCAG             kkk_433_452
[47]     20 AACCTAATCGCCCCGTGGCG             kkk_805_824
[48]     20 CGCCGCCTTCGAGCTCGCTT             kkk_140_159
[49]     20 GACAGCGGGAGTGATCCGGC             kkk_401_420
[50]     20 TGTAAGCCGTACCCTACCCC             kkk_904_923
>
```

②

塩基置換前の全リード

まずはおさらいとして、①オリジナルの全500リードからなる塩基置換前のfasta_orgのk-mer出現頻度分布や周辺情報を見ておく

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

①

```
#k-mer出現頻度分布(original)
fasta <- fasta_org                          #fasta_orgをfastaに格納
out_f <- "hoge2_original.png"               #出力ファイル名を指定してout_fに格納
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                        #列ごとの総和をoutに格納
kmer <- out[out > 0]                        #1回以上出現したk-merのみをkmerに格納
length(kmer)                                #1回以上出現したk-merの種類数を表示
table(kmer)                                 #(k-merの種類は問わずに)k-merの出現回数
png(out_f, pointsize=13, width=param_fig[1], height=param_fig[2])#出力ファイ
par(mar=c(4, 4, 0, 0))                      #下、左、上、右の順で余白（行）を指定
hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラムを描画
     xlab=paste("Number of occurences at k=", param_kmer, sep=""),#ヒストグラ
     xlim=param_xrange, ylim=param_yrange, main="")#ヒストグラムを描画
dev.off()                                   #おまじない
kmer_org <- kmer                            #kmerをkmer_orgに格納

#k-mer出現頻度分布(最初の450リードのみ；塩基置換なし)
fasta <- fasta_first450                     #fasta_first450をfastaに格納
out_f <- "hoge2_first450.png"               #出力ファイル名を指定してout_fに格納
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k連続塩基の出現頻度
out <- colSums(hoge)                        #列ごとの総和をoutに格納
kmer <- out[out > 0]                        #1回以上出現したk-merのみをkmerに格納
```

塩基置換前の全リード

①の結果は、②k-merの種類数は975個であり、これが推定ゲノムサイズ（正解は1,000 bp）。③ヒストグラムはbin次第で見栄えが異なりうる。④table(kmer)で元データも眺める

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#k-mer出現頻度分布(original)
fasta <- fasta_org                    #fasta_orgをfastaに格納
out_f <- "hoge2_original.png"         #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                  #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer),
     xlab=paste("Number of oc
     xlim=param_xrange, ylim=
dev.off()
kmer_org <- kmer
```

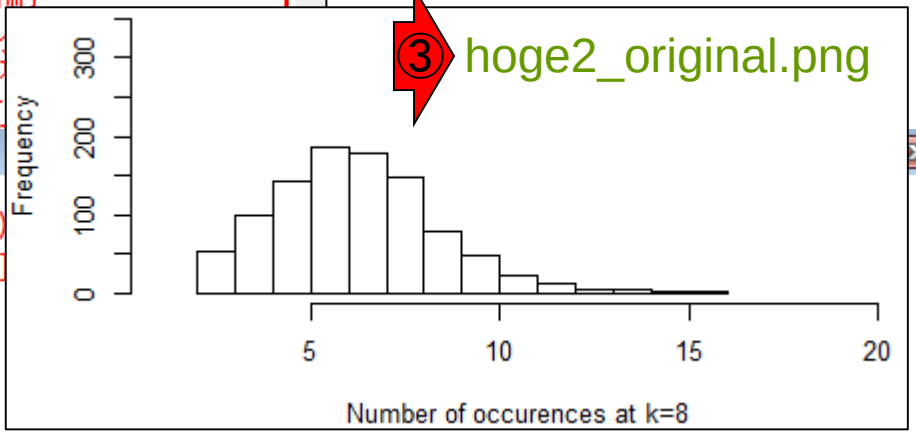
```
#k-mer出現頻度分布(最初の450リー
fasta <- fasta_first450
out_f <- "hoge2_first450.png"
hoge <- oligonucleotideFrequenc
out <- colSums(hoge)
kmer <- out[out > 0]
```

① ② ④

R Console

```
> out <- colSums(hoge)
> kmer <- out[out > 0]
> length(kmer)
[1] 975
> table(kmer)
kmer
  2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
 18  35  99 142 185 178 148  78  47  22  12   4   4   1   2
> png(out_f, pointsize=13, width=param_fig[1], height=param_f$
> par(mar=c(4, 4, 0, 0))                          #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                       #おまじない
null device
          1
> kmer_org <- kmer                                #kmerをkmer_orgに格納
>
```

③ hoge2_original.png



最初の450リードのみ

①の結果は、②k-merの種類数は975個であり、これが推定ゲノムサイズ（正解は1,000 bp）。③ヒストグラムはbin次第で見栄えが異なりうる。④table(kmer)で元データも眺める。シークエンスエラーはないが、リード数が減っているので、分布も左に少しずれている

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#k-mer出現頻度分布(最初の450リードのみ；塩基置換なし)
fasta <- fasta_first450                    #fasta_first450をfasta
out_f <- "hoge2_first450.png"              #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                       #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer),
     xlab=paste("Number of oc
     xlim=param_xrange, ylim=
dev.off()
kmer_first450 <- kmer

#k-mer出現頻度分布(最後の50リード
fasta <- fasta_last50
out_f <- "hoge2_last50.png"
hoge <- oligonucleotideFrequenc
out <- colSums(hoge)
kmer <- out[out > 0]
```

①

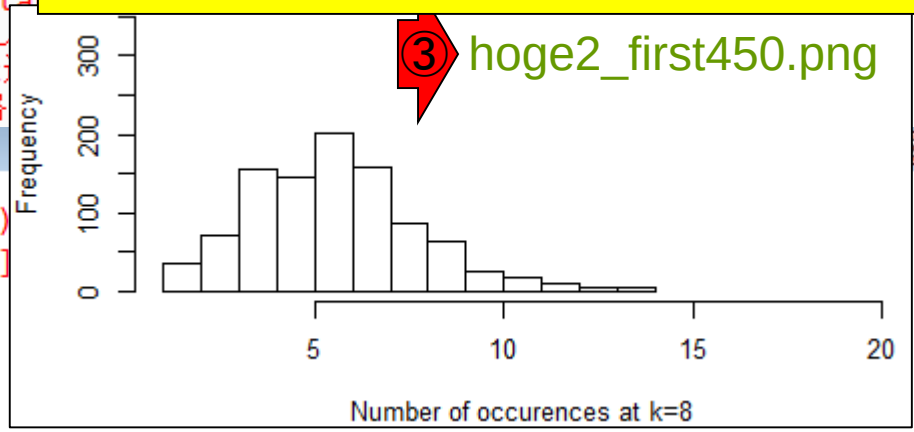
②

④

R Console

```
> out <- colSums(hoge)
> kmer <- out[out > 0]
> length(kmer)
[1] 975
> table(kmer)
kmer
  1   2   3   4   5   6   7   8   9  10  11  12  13  14
  5  31  71 155 145 201 158  87  64  24  17  10   4   3
> png(out_f, pointsize=13, width=param_fig[1], height=param_f$
> par(mar=c(4, 4, 0, 0))                          #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                       #おまじない
null device
          1
> kmer_first450 <- kmer                           #kmerをkmer_first450$
>
```

③ hoge2_first450.png



最後の50リードのみ

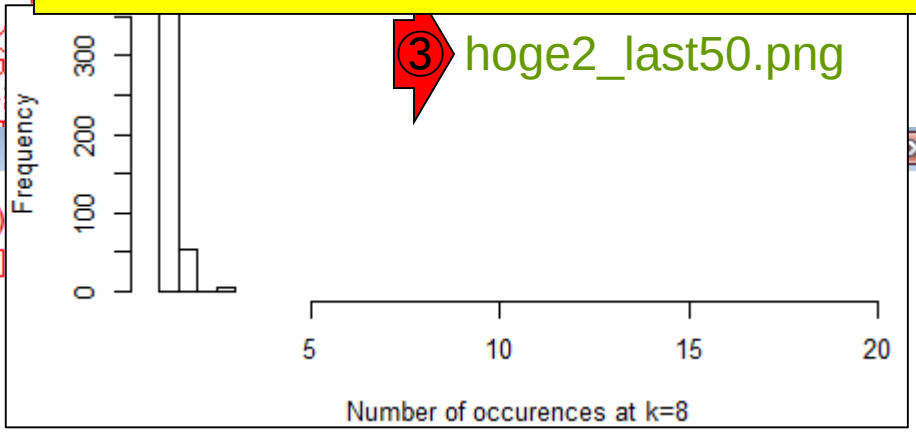
①の結果は、②k-merの種類数は590個であり、これが推定ゲノムサイズ（正解は1,000 bp）。③ヒストグラムはbin次第で見栄えが異なりうる。④table(kmer)で元データも眺める。シークエンスエラー由来k-merがほとんどなので、出現回数が最大でも3回になってる

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#k-mer出現頻度分布(最後の50リードのみ；塩基置換あり)
fasta <- fasta_last50                    #fasta_last50をfastaに
out_f <- "hoge2_last50.png"              #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                     #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer),
     xlab=paste("Number of oc
     xlim=param_xrange, ylim=
dev.off()
kmer_last50 <- kmer

#k-mer出現頻度分布(merged)
fasta <- c(fasta_first450, fast
out_f <- "hoge2_merged.png"
hoge <- oligonucleotideFrequenc
out <- colSums(hoge)
kmer <- out[out > 0]
```

① ② ③ ④



③ hoge2_last50.png

R Console

```
> out <- colSums(hoge)
> kmer <- out[out > 0]
> length(kmer)
[1] 590
> table(kmer)
kmer
  1   2   3
533  54   3
> png(out_f, pointsize=13, width=param_fig[1], height=param_f$
> par(mar=c(4, 4, 0, 0))                    #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                 #おまじない
null device
          1
> kmer_last50 <- kmer                       #kmerをkmer_last50に$
>
```

マージ後のデータ

①の結果は、②k-merの種類数は1,341個であり、これが推定ゲノムサイズ（正解は1,000 bp）。③ヒストグラムはbin次第で見栄えが異なりうる。④table(kmer)で元データも眺める

- 最終確認

500リードのうち、451:500番目の[illegible]についてのみ13番目の塩基を置換。

①

```
#k-mer出現頻度分布(merged)
fasta <- c(fasta_first450, fasta_last50)#マージしたものをfastaに格納
out_f <- "hoge2_merged.png"                #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                       #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer),
     xlab=paste("Number of oc
     xlim=param_xrange, ylim=
dev.off()
kmer_merged <- kmer
```

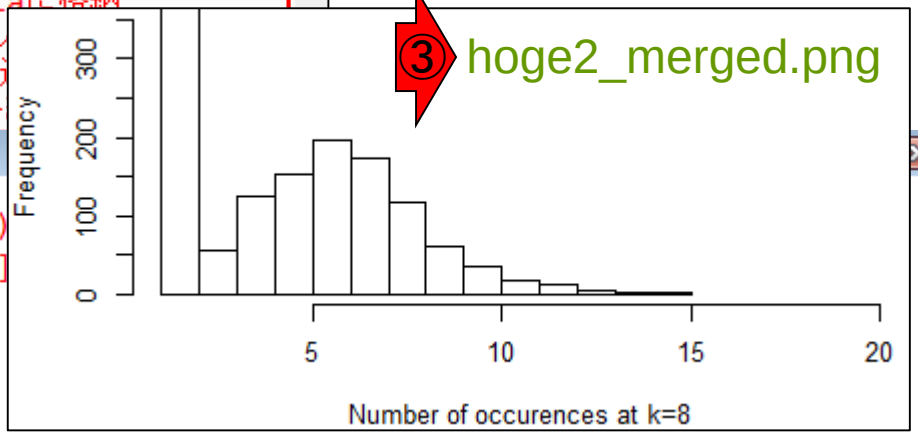
②

④

R Console

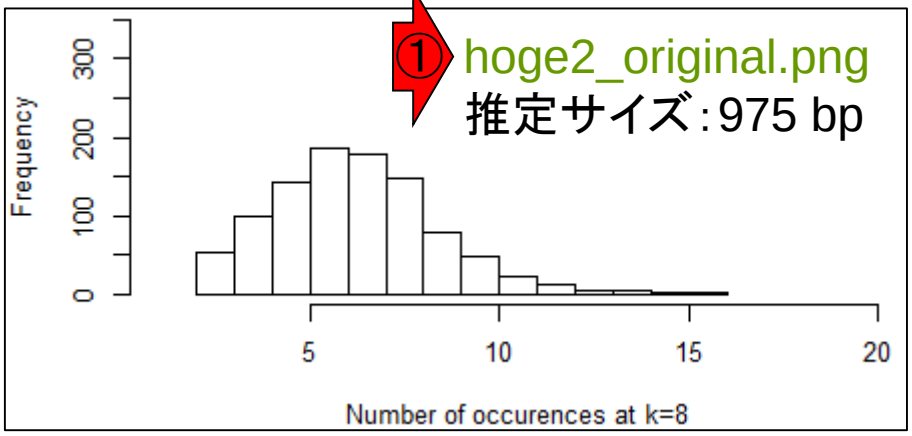
```
> out <- colSums(hoge)
> kmer <- out[out > 0]
> length(kmer)
[1] 1341
> table(kmer)
kmer
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
342  51  56 125 153 195 172 116  60  35  16  11   5   2   2
> png(out_f, pointsize=13, width=param_fig[1], height=param_f$
> par(mar=c(4, 4, 0, 0))                          #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                       #おまじない
null device
          1
> kmer_merged <- kmer                             #kmerをkmer_mergedに$
>
```

③ hoge2_merged.png

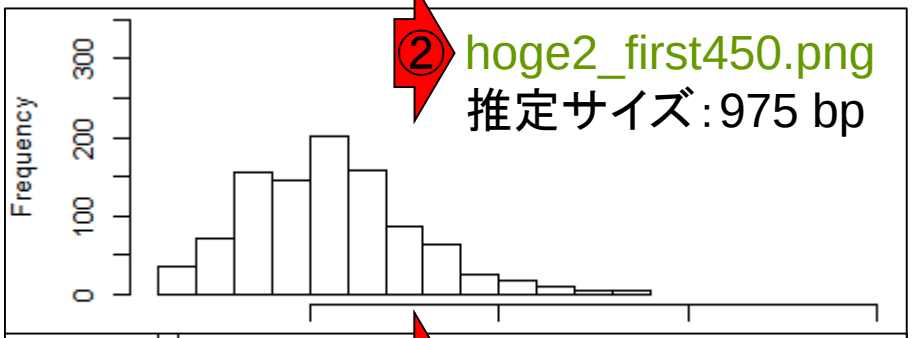


比較

①オリジナルの塩基置換前の全500リードの結果。②最初の450リードの結果、③13番目の塩基置換後の最後の50リードの結果。④は、②と③を合わせた結果。③の結果はシークエンスエラー由来k-merが多くを占めていたことを思い出そう

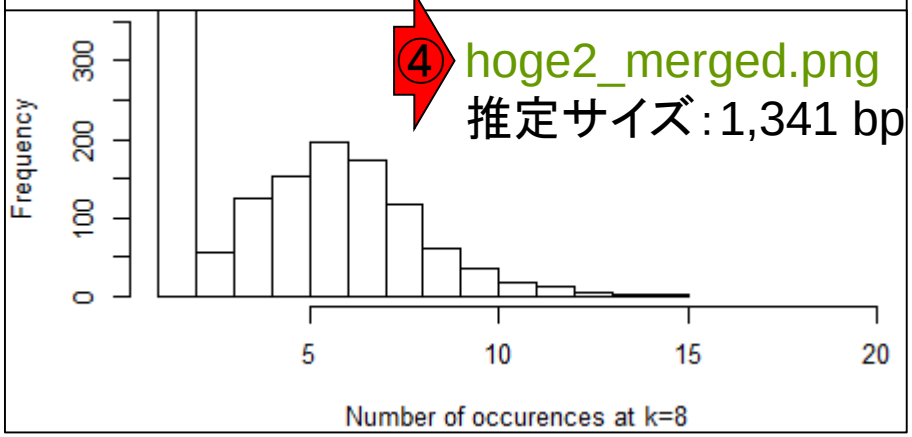


① hoge2_original.png
推定サイズ：975 bp



② hoge2_first450.png
推定サイズ：975 bp

③ hoge2_last50.png
推定サイズ：590 bp



④ hoge2_merged.png
推定サイズ：1,341 bp

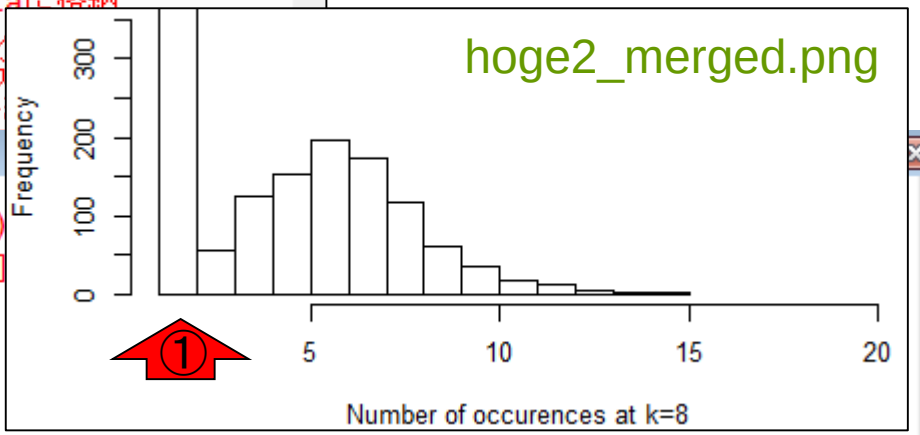
マージ後のデータ

①シークエンスエラー由来k-merが多くを占めていた部分まで含めて算出した②推定ゲノムサイズは1,341 bp。正解（1,000 bp）からのずれは、③シークエンスエラー由来k-merの結果部分に起因するので除いちゃえばいい

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#k-mer出現頻度分布(merged)
fasta <- c(fasta_first450, fasta_last50)#マージしたものをfastaに格納
out_f <- "hoge2_merged.png"              #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                     #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer),
     xlab=paste("Number of occ
     xlim=param_xrange, ylim=pa
dev.off()
kmer_merged <- kmer
```



②

R Console

```
> out <- colSums(hoge)
> kmer <- out[out > 0]
> length(kmer)
[1] 1341
> table(kmer)
kmer
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
342  51  56 125 153 195 172 116  60  35  16  11   5   2   2
> png(out_f, pointsize=13, width=param_fig[1], height=param_f$
> par(mar=c(4, 4, 0, 0))                          #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                       #おまじない
null device
          1
> kmer_merged <- kmer                             #kmerをkmer_mergedに$
>
```

③

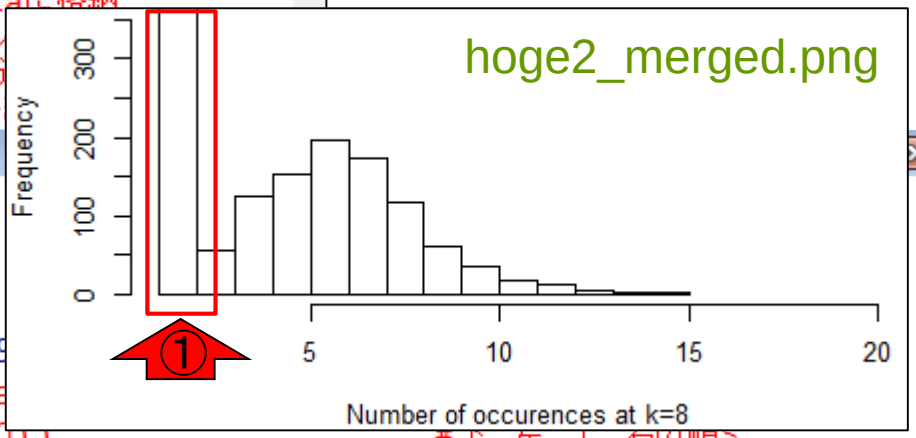
マージ後のデータ

①シークエンスエラーを多く含むk-mer部分を除いて考えるのは、この場合は②2回以上、あるいは③3回以上出現するk-merの種類数のみをカウントすることに相当

- 最終確認

500リードのうち、451:500番目のリードについてのみ13番目の塩基を置換。

```
#k-mer出現頻度分布(merged)
fasta <- c(fasta_first450, fasta_last50)#マージしたものをfastaに格納
out_f <- "hoge2_merged.png"              #出力ファイル名を指定し
hoge <- oligonucleotideFrequency(fasta, width=param_kmer)#k
out <- colSums(hoge)                     #列ごとの総和をoutに格
kmer <- out[out > 0]
length(kmer)
table(kmer)
png(out_f, pointsize=13, width=
par(mar=c(4, 4, 0, 0))
hist(kmer, breaks=max(kmer), yl
     xlab=paste("Number of occu
     xlim=param_xrange, ylim=pa
dev.off()
kmer_merged <- kmer
```



R Console

```
> table(kmer)
kmer
  1   2   3   4   5
342  51  56 125 153 19
> png(out_f, pointsize
> par(mar=c(4, 4, 0, 0))                    #下、左、上、右の順$
> hist(kmer, breaks=max(kmer), ylab="Frequency",#ヒストグラム$
+      xlab=paste("Number of occurences at k=", param_kmer, s$
+      xlim=param_xrange, ylim=param_yrange, main="")#ヒスト$
> dev.off()                                 #おまじない
null device
          1
> kmer_merged <- kmer                       #kmerをkmer_mergedに$
> sum(kmer >= 2)
[1] 999
> sum(kmer >= 3)
[1] 948
>
```

①
②
③