

Tips for NGS Data Analysis

次世代DNAシーケンサーのデータ解析技術 (2013/02/01)

著者について

二階堂愛, Ph.D dritoshi+ngstips@gmail.com 理化学研究所 情報基盤センター バイオインフォマティクス研究開発ユニットリーダー

注意

この文章は2013/02時点での情報で、このなかすでに使っていないもの、方法が変更されているものがありますので注意してください。

この文章の著作権は二階堂愛にあります。ファイルのダウンロード、印刷、複製、大量の印刷は自由におこなってよいです。企業、アカデミアに関わらず講義や勉強会で配布してもよいです。ただし販売したり営利目的の集まりで使用してはいけません。ここで許可した行為について二階堂愛に連絡や報告する必要はありません。常に最新版を配布したいのでネット上の再配布や転載は禁止します。内容についての問い合わせはお気軽にメールしてください。

この本について

NGS解析のノウハウをメモしたものを gitbook に変換したもの。あるいはメモに一部解説を加えたもの。節によってはまったく説明がなくメモだけのものもあります。Evernote にある分の多くのノウハウは含まれていないが、暇を見て更新していきます。

今のところ出版の予定はないです。

書くことリスト

- NGSについて
- オープンソースとバイオインフォマティクス
- Tips集である
- 対象者、初心者から上級者のリファレンスとして
- Resequencing (SNP, exome) や De novo genome assembly は扱わない予定
- 読み方、リファレンス的にも、パイプライン的にも。
- \$ が unix コマンド、R> は R のプロンプト。
- 対象シーケンサー、HiSeq, Miseq, GAIIx だが、454, ion torrent, SOLiD, 5500xl などでも fastq や bam/sam などになれば同じこと
- 前提知識、Unix のファイル・ディレクトリ構造、簡単なコマンド。一部 R や Ruby を利用する。

推奨環境

Mac OS X あるいは Linux を推奨する。Windows + Cygwin でも可能なものが多いため、無償の仮想マシン環境 VMware Player と Linux (Ubuntu) を利用することをお勧めする。CentOS はソフトウェアのアップデートが遅いので避けるべきである。または、手元にLinux 環境を持つのが難しい場合は、Amazon EC2 で Linux の Amazon Machine Image を利用する方法もある。

コマンドとしては、git, wget, w3m を利用するのであらかじめインストールしておくとよい。Mac OS X の場合は、MacPorts をインストールし、
(homebrewに書き直す必要あり、RやGCC, Xcodeについても買く必要がある)

```
$ sudo port install wget  
$ sudo port install w3m  
$ sudo port install git-core
```

しておくこと。シェルは zsh を前提としているがほかのシェルでも問題がないように説明している。プログラミングを前提とはしないが、一部 R, Ruby, Java なども利用する。

謝辞

日常的にシーケンス技術について議論させて頂いたマックス・プランク研究所の足立健次郎氏、理化学研究所 情報基盤センターの笹川洋平氏、園野宏樹氏、ラボメンバーに感謝したい。彼らのおかげでシーケンスの実験的側面について深く知ることができた。またここですべての名前を挙げることはできないが、Twitter や Google+, NGS現場の会、オープンバイオ研究会、文部科学省セルイノベーションプロジェクト、そして理研CDBシーケンスプラットームなどを通じ、日々シーケンス技術に関して情報を交換させて頂きたみなさまにも感謝したい。

品質管理

FASTQ formatについて

FASTQはシーケンス配列とそのクオリティを記載するためのファイルフォーマットである。ファイル形式はテキストファイルなので、Unix/Linuxのlessコマンドやテキストエディタで見ることができる。イルミナ社のシーケンサーやNCBI Short Read Archiveなどのデータベースでもこのフォーマットが採用されており、多くの解析ソフトがこの形式を前提としている。FASTQファイルはひとつのシーケンスリードの情報が4行単位で表現され、その単位が一枚のファイルに連なっているファイルである。@で始まる最初の行は、シーケンスリードの情報が含まれたヘッダー行であり、FASTA形式の“>”に値すると考えるとわかりやすい。次の行がリーケンスリードの塩基配列そのもので、ATGCNからなる。3行目も1行目と同じリードの情報を記載する行で+から始まる。大抵の場合は、単に“+”か、最初の行と同じ文字列になっている。4行目がシーケンスクオリティのスコアが符号化されたものである。

イルミナ社のFASTQ fileは1行目の意味について述べる。イルミナ社のFASTQ fileはシーケンサーが出力するバイナリファイルであるBCL fileからCasavaというソフトウェアで生成する。Casavaはイルミナ社が提供するシーケンサーに付属するソフトウェアである。Casavaのバージョンによって、一部フォーマットが異なるのでそれについても解説する。

まず、Casava 1.8より前のFASTQ fileについて解説する。@で始まるヘッダー行は”.”でセパレートされている。まず最初のフィールドHWI-ST554_0072はシーケンサーの機械につけられたIDを示す。次の8はフローセルのレーン番号を示し、HiSeq, GAIixシリーズでは1-8まである。フローセルのひとつのレーンは、tileに分けられているが、1はそのタイル番号を表している。1131, 1901はそれぞれtile内にあるクラスターのx, y座標を表わす。#の次の番号はマルチプレックス番号を示し、0の場合はマルチプレックスされていないことを示す。/の後の数字が1ならsingle-end, 2の場合はpaired-end,あるいは、mate-pair readであることを示す。

```
@HWI-ST554_0072:8:1:1131:1901#0/1
TCCCAAGGAAGGCGTGC GTGTGTTGAGTACTTCAAAACACACTTCCTAC
+HWI-ST554_0072:8:1:1131:1901#0/1
```

```
fcfefffffaefff^edeceZccccddd`dfffefffffef_deffdff
```

図. HiSeqから出力されたFASTQ file の例 (Casava 1.8より前)

Casava 1.8 以降の FASTQ file について解説する。@で始まるヘッダー行の形式が変更されている。まず、HWI-1KL121 がシーケンサにユニークに振られた機器IDである。これは以前のバージョンと変更はない。次の 46 は run ID でシーケンスを実行するたびに割り振られる。D0MVAACXX は flowcell に割り振られた ID である。以降については以前のバージョンと同じ意味である。1.8以降のヘッダーはスペースを挟んで、さらに情報が追加されている。最初の数字が 1 なら single-end, 2 なら paired-end である。次の文字は悪いリードを省くためのフィルターの結果を示しており、Yだと悪いリード、Nだとそれ以外という意味である。その次の数字は、XXX???を示す。最後の塩基はインデックスのシーケンスを表わしている。

```
@HWI-1KL121:46:D0MVAACXX:6:1101:9571:2529 2:N:0:CGATGT  
CAGGCTTAAAATCTGGAAAGGAACACATGAGGGTCTCATCCACANNATCA  
+  
CCCCFFFFHHHHJJJJJJJJJJJJJJJJ# # # 07BB
```

図. HiSeqから出力されたFASTQ file の例 (Casava 1.8以降)

FastQCによるシーケンス実験の品質チェック

FastQC はシーケンス実験の品質をチェックするために便利がプロットを生成するプログラムである。サイクル(リードの塩基)あたりのシーケンスクオリティや、クオリティスコアの分布、同一シーケンスリードの重複率などを計算しプロットしてくれる。シーケンスクオリティはシーケンス実験の成否を、同一リードの重複率は、ライブラリ作成時の PCR バイアスなどを評価するために重要な指標である。重複率が高い場合は、PCRサイクルを減らすことで、ダイナミックレンジの向上を図ることができる。

インストール:

```
$ curl -O http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc  
$ unzip fastqc_v0.10.0.zip  
$ sudo cp fastqc_v0.10.0 /opt  
$ sudo ln -s /opt/fastqc_v0.10.0 /opt/fastqc  
$ emacs -nw ~/.zshenv  
$ export PATH=$PATH:/opt/fastqc
```

FastQCをコマンドラインから利用する

実験が多くなるとGUIから実行するのが面倒になる。FastQCはコマンドラインからの実行にも対応している。以下のコマンドは `test.fastq` に対して、`fastqc` を実行し、結果を `test` ディレクトリに保存するというコマンドである。`-t` オプションは CPU を利用する数を指定する。この場合は 8 CPUs を使って計算する。

```
$ fastqc -t 8 test.fastq -o test
```

FastQC は `fastq file` だけでなく、`gzip (*.gz)`で圧縮されたファイルに対しても実行可能である。結果はウェブブラウザで閲覧できる `html` ファイル、テキストファイル、`html` ファイルが圧縮された `zip` ファイルの3種類出力される。

アダプター配列除去 FASTX_Toolkit fastx_clipper

short-read な FASTQ/A をいろいろいぢるツール群である FASTX_Toolkit に含まれるコマンドのひとつである fastx_clipper。 (→トリムしたい場所が決まっている場合は fastx_trimmer を使う)。後に紹介する Galaxy に組み込み済みである。ちなみにそれぞれのコマンドの動作は Galaxy 内の解説にある図を見るのが良い。

インストール: まず、 http://hannonlab.cshl.edu/fastx_toolkit/download.html から pre-compiled binary 版を落して、 /usr/local/bin などに入れる PATH を通す。本体とは別に実行に必要とされるプログラムは、 libgtextutils-0.6, PerlIO::gzip, GD::Graph::bars, gnuplot version 4.2 or newer であるので、事前に入れておく。 -I でリードの最小サイズ、 -M で一致する最小の長さを指定する。 illumina CASAVA 1.8 以降の FASTQ や Sanger FASTQ には -Q 33 を指定する。それ以前で illumina 1.3 以降の FASTQ の場合は、 -Q 64 を指定すること。トリムした配列だけ出力するには -C を、 trim されなかつた配列を出力するには、 -c を指定する。なにも指定しなければ、両方が表示される。

使いかた:

```
$ fastx_clipper -l 18 -M 12 -Q 33 -v -i input.fastq -a ACACTCTT'
```

アダプター除去 TagDust

概要 理研OSCで開発されたシーケンスアダプター配列除去ソフト。入出力形式は fastq/a が使える。アダプター配列を multi fasta 形式で入力できるのが地味に便利である。なぜなら複数のアダプタ/プライマーを1回の実行で除去できるためである。これに対応しているものがなかなかない。アダプター/プライマー配列の検索には Muth–Manber algorithm (Approximate multiple string search) を利用している。FDRを調節することで除去の強さを指定できる。ライセンスはGPL3

インストール:

```
$ curl -O http://genome.gsc.riken.jp/osc/english/software/src/tagdust.tar.gz
$ cd tagdust/
$ make
$ sudo make install
$ rehash
```

使いかた:

```
$ tagdust adapter.fasta input.fastq -fdr 0.05 -o output.clean.fastq
```

-fdr は除去の強さを調節する。-o にはアダプタ配列を除いたとの fastq, -a には除かれた側の配列が記載された fastq を指定する。

リファレンス <http://bioinformatics.oxfordjournals.org/content/25/21/2839.full.pdf>

アダプター配列のトリミングする Trimmomatic (Paired-end対応) 概要:
paired-end に対応したアダプター配列をトリムするツール。illumina HiSeq は Paired-end の FASTQ は対となる2枚のファイルになっている。大抵の場合は、対となるペアが同じ行に記載されているが、アダプターの除去などの際にリードが除かれると、その順序が一致しなくなる。ツールによっては、IDではなく、出現順序でペアを認識している場合もある。

Trimmomatic はこのようなことが起こらないよう、一对のリードに対してアダプターのトリムする、Paired-end mode が用意されている。

使いかた:

```
$ java -classpath /opt/Trimmomatic/trimmomatic-0.20.jar  
org.usadellab.trimomatic.TrimmaticPE -threads 4 -phred64  
input/fastq/ES-01_1.fastq input/fastq/ES-01_2.fastq  
input/fastq/ES-01_1.trim.fastq input/fastq/ES-01_1.unpaired.fastq  
input/fastq/ES-01_2.trim.fastq input/fastq/ES-01_2.unpaired.fastq  
ILLUMINACLIP:common_data/data/tagdust/solexa_paired_end_primers.fa  
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

adapter 配列の除去を行う。adapter 配列は、
common_data/data/tagdust/solexa_paired_end_primers.fa にFASTA形式で
記載する。2塩基のmismatch を許してアダプタ配列を検索する。リード先
頭の quality < 3 か、N を持つ「配列」を除く。リード末端の quality < 3
か、N を持つ「配列」を除く。4塩基ずつ配列をずらしながら quality の平
均を計算し、quality が 15 以下の「リード」を除く。36塩基以下の長さの
「リード」を除く。

アダプター配列の除去 cutadapt

概要: 入出力ファイルとして fastq/a だけでなく maq や bwa 形式にも対応したアダプタ/プライマー配列除去プログラム。gzip で圧縮されたファイルでもよい。アウトプットとインプットファイルの順序を間違えると大切なデータファイルが上書きされて消えてしまうので注意が必要。アダプタが 3'末端へライゲーションすることを想定している。illumina HiSeq/MiSeq/GA II シリーズ, SOLiD シリーズの場合はそうなる。ただし、-b (--anywhere) を付けると、どの位置にあっても取り除く。アダプタ配列が部分的に存在する場合も取り除く。-e オプションでエラーレートを指定できる。Python で実装されている。Galaxy に組み込むときには、galaxy/ 以下に入れるだけ。MIT ライセンス。

インストール:

```
$ curl -O http://peak.telecommunity.com/dist/ez_setup.py  
$ sudo easy_install cutadapt  
$ rehash
```

使いかた:

```
$ cutadapt -a ACACTCTTCCTACACGACGCTGTTCCATCT -o output.fastq.
```

シーケンスクオリティが低いリードを除く

シーケンスクオリティが低いリードを事前に除去しておくことによって、リファレンスゲノムマッピングやアセンブルの精度や計算コストを軽減することができる。またそれらの解析の精度を純粋に評価するためにも、シーケンスクオリティの問題を分けるのが重要なことである。

ここでは、FASTAX toolbox の fastq_quality_filter を利用して、fastq file からクオリティが低いリードを除く。

```
$ fastq_quality_filter -Q 33 -q 30 -p 75 -i input.fastq -o output.fastq
```

このコマンドでは、quality value が 30 以下の quality value がリード全長の 75% を下回ったときにそのリードを除く。-Q 33 は CASAVA 1.8 で作られた fastq file を扱うためのオプションである。

Quality score については以下を参照せよ。

- [FASTQ format - Wikipedia](#)

なぜアダプタやプライマー配列を除く必要があるのか

シーケンスの際に、読みたいインサートを突き抜けてアダプタ/プライマ配列が現われることがある。特にイルミナ社のシーケンスはエラー少ないので、そのなかでもよくあるエラーがこのアダプタ/プライマ配列の混入である。インサート長が短い場合や、インサートがみあたらず、アダプターダイマーのような配列に由来すると考えられる。

このような配列は mapping の前に除去おいたほうが良いとされている。しかし mapping すると、その配列はゲノムに正しくマップされないため、マッピングスコアなどにより、実質的にはその後の解析に含まれないようにするすることも可能である。マッピングの計算コストがそれほどかからなくなってきた現在、先にこのような配列をフィルタする必要があるのでどうか。

答えとしては、それでも除いたほうが良い、である。なぜなら、マッピングされなかった配列がどのような理由でマッピングできなかったのかを知ることは、ライブラリの質を評価することに繋がるためである。最初にこれらの配列をフィルタすることで、それがどのくらい含まれているのかを知り、それを実験者にフィードバックすることが重要である。仮にアダプタ配列ばかりが読まれると、リードがアダプタDNA断片に奪われてしまうため、期待されるリード読まれる機会が減る。これによりシグナルノイズ比が悪くなる。これは後の解析の成否にかかわることである。このような場合は、読むべきDNA断片とアダプタ量の比率を実験的に調整したり、アダプタダイマーを除くサイズセレクションの精度を上げたり、などの実験的に工夫が必要になる。このような理由に、自分のシーケンス実験のヘルスチェックのために、フィルタリングを行い、その統計情報を持つべきなのである。

ゲノムのリピート領域にマッピングされたリードを除く

ゲノムには大量のリピート配列が含まれておりヒト、マウスのゲノムはその約半分がリピートに関連した領域である。シーケンスされたリードをこのような領域にはマッピングすることは、その位置を一意に決めることができないので、困難である。一般的には、複数箇所にマップされるリードを、重複領域にランダムに配置する方法(リード数に応じて確率的に配置する方法が良いとされている)と、データからリピート領域を除いて解析を進める方法の2通りがある。転写因子の ChIP-seq ではリピート領域にかなり高いピークが形成され、正規化などのノイズになるため、リピート領域を最初から解析に含めない方法が取られることがある。この場合は、マッピング後、peak calling の前にこのような処理を行うことになる。

方法は、RNA-seq データから rRNA を除く方法と同じである。

FASTA/FASTQ/SAM/BAMの詳しいステータスを得る

[samstat](#) はFASTA/FASTQ/SAM/BAMから、nucleotide composition, length distribution, base quality distribution, mapping statistics, mismatch, insertion and deletion error profiles, di-nucleotide and 10-mer over-representation を計算するツールである。

インストール:

```
$ wget http://downloads.sourceforge.net/project/samstat/samstat  
$ tar zxvf samstat.tgz  
$ cd samstat/src  
$ make  
$ sudo cp ./samstat /opt/local/bin/  
$ rehash
```

実行方法:

```
$ samstat test.bam
```

結果は `test.bam.html` というファイルにhtml5形式で出力される。このファイルを閲覧するには通常のWebブラウザを使う。表示されるグラフは JavaScript だけで生成されているので、画像ファイルが出力されるわけではない。

ランダムにリードを取り出す

リード数に応じてシーケンスの深さ (depth) が決まる。この depth は RNA-seq, ChIP-seq などのタグシーケンシングではその精度に関わる。具体的には、depth が深いほど、検出できる遺伝子の数やRNAの発現量やタンパク質-DNA結合量ダイナミックレンジが向上する。自分のサンプルや生物がどのくらいのリード数が必要であるかを知ることは、正確でかつ、無駄なコストをかけずに実験するには重要なことである。そこで、リードをランダムに除いた系列、例えば、0.1, 1, 10, 100M リードなどを用意して解析することで、検出遺伝子やダイナミックレンジの評価を行うことが重要である。ランダムに配列を除くことを、ダウンサンプリング、あるいは、ランダムサンプリングと呼び、取り出されたリード群を、ダウンサンプル、サブサンプルなどと呼ぶ。

BAM file から一定の確率でランダムにリードを取り出す

ここでは、Picard (<http://picard.sourceforge.net/>) の DownsampleSam.jar を使って、bam file からランダムに配列を除く方法を述べる。すでにマッピング済みの配列からファイルを除くにはことを想定する。

```
$ java -jar /opt/picard-tools/DownsampleSam.jar INPUT=input.bam
```

ここでは、全体のリードのうち 20% をランダムに取り出している。注意したいのは、tophat2, bowtie2 などで、ひとつのリードを複数箇所にマッピングされた(いわゆる multi hits)ような bam file を利用する場合は、割合ではなくリードの絶対数を指定しなければならない。それについては次の項を参照すること。

BAM file から一定の数のリードを取り出す

DwonsampleSam.jar はリードを取り出す確率を指定することはできるが、取り出したい数を直接指定することはできない。そこで、bamtools を使って数を直接指定してリードを取り出す。

まず、cmake をインストールする。

```
$ cd ~/src
$ curl -O http://www.cmake.org/files/v2.8/cmake-2.8.8.tar.gz
$ tar zxvf cmake-2.8.8.tar.gz
$ cd cmake-2.8.8/
$ ./configure --prefix=/opt/local
$ gmake
$ sudo gmake install
```

次に bamtools をインストールする。

```
$ cd ~/src
$ git clone git://github.com/pezmaster31/bamtools.git
$ cd bamtools/
$ mkdir build
$ cd build/
$ cmake -DCMAKE_INSTALL_PREFIX=/opt/local ..
$ make
$ sudo make install
```

bamtools を実行して、次のようなエラーがでる場合は、手動でファイルをコピーする。

```
bamtools: error while loading shared libraries:
```

```
$ sudo cp libbamtools-utils.so* /opt/local/lib/bamtools/
$ sudo cp libjsoncpp.so* /opt/local/lib/bamtools/
```

10Mのリードを取り出すには以下のようにする。

```
$ bamtools random -in input.bam -out output..bam -n 100
```

bamtools でリード数を確認してみる。

```
$ bamtools count -in output.bam  
100
```

samtools で数えることができる。

```
$ samtools flagstat output.bam  
100 + 0 in total (QC-passed reads + QC-failed reads)  
0 + 0 duplicates  
100 + 0 mapped (100.00%:nan%)  
:  
:
```

Fastq/a file からランダムにリードを取り出す

次にマッピング前のデータから配列を除くことを想定して、fastq file からのリードのサンプリングについて述べる。ここでは seqtk を利用する。まず seqtk をインストールする。

```
$ git clone git://github.com/lh3/seqtk.git  
$ cd seqtk  
$ make  
$ sudo cp seqtk /opt/local/bin
```

次に sample オプションを利用して、ランダムサンプリングする。

```
$ seqtk sample -s1 input.fastq 30000000 > output.30M.fastq
```

ここでは、30M のリードをランダムに取り出している。-s オプションは乱数を生成するときのシードとなる数値である。シードの数はどのような数でもよい。

シードが同じであれば生成される乱数列はいつも等しくなる。これを利用すれば、Paired-end ファイル、つまり2つの fastq からランダムにペアを選ぶことができる。input_1.fastq と input_2.fastq の pair 同士が同じ並び順になっている場合は、以下のようにする。

```
$ seqtk sample -s1 input_1.fastq 30000000 > output_1.30M.fastq  
$ seqtk sample -s1 input_2.fastq 30000000 > output_1.30M.fastq
```

注意: seqtk は samtools にも付属している (samtools/misc/seqtk) がこれには、sample オプションがない。混同しないよう注意が必要である。

シーケンスリードのカバレッジ(被覆率)を計算する

インストール

```
$ curl -O https://bedtools.googlecode.com/files/BEDTools.v2.17.0.tar.gz
$ tar zxvf BEDTools.v2.17.0.tar.gz
$ cd bedtools-2.17.0/
$ make
$ cd ..
$ sudo cp -a bedtools-2.17.0 /opt
$ cd /opt
$ sudo ln -s bedtools-2.17.0 bedtools
```

設定

```
$ emacs -nw ~/.zshenv
export PATH=$PATH:/opt/bedtools/bin
$ source ~/.zshenv
```

ここでは、`bed` ファイル (`Pou5f1.bed`) に記述された座標にマップされたリードのカバレッジを `bam` ファイルから(`hoge.bam`) 計算する。

```
$ coverageBed -d -abam hoge.bam -b Pou5f1.bed > Pou5f1.TruRNA-Score.txt
```

bam を wig に変換する

wig file はカバレッジのデータを保存するためによく使われるファイル形式である。ここでは、bam から wig を計算する方法を述べる。

```
bam2wig.py -t 1000000000 -i hoge.bam -o hoge.wig -s mm10.info
```

wig を bigwig へ変換する

wig 形式はテキスト形式になっているが、このままではファイルサイズが巨大になる。そこで、バイナリ形式である、bigwig へ変換する。

```
$ wigToBigWig hoge.wig mm10.info hoge.bw
```

DNA配列のGC%を計算する

multi fasta 形式で保存されたDNAの GC content を計算するには、EMBOSSのgeeceeを利用すると簡単である。一般的なコンピュータでなら、マウスの全転写産物のGC contentの計算が、8秒程度で終了する。例では、ensmust.67.fa に保存されているmRNA配列のGC%を計算し、ensmust.67.gc.txt に保存する。

```
$ geecee -sequence ensmust.67.fa -outfile ensmust.67.gc.txt
```

出力の中身は以下のようになっている。

```
$ cat ensmust.67.gc.txt
#Sequence    GC content
ENSMUST00000000096  0.53
ENSMUST00000000137  0.40
ENSMUST00000000109  0.43
ENSMUST00000000127  0.53
```

長さとGC content の両方が欲しい場合は infoseq を利用したほうがよい。

```
$ infoseq -heading N -usa N -database N -type N -description N
```

異なるバージョンのゲノム座標を変換する: liftover

古い論文のデータと比較しなければならない場合などの、ゲノムのバージョンが古い座標系を使っている場合がある。そこで liftover を使い、より新しいバージョンのゲノムの座標系に変換する。

インストール ここでは、マウスゲノムの mm8 から mm9 へ座標変換する。Linux が前提ですが、ほかでもかわりないです。変換されるファイルの形式は bed が基本だが、オプションでほかのファイルにも対応できる。bed ファイルでも browse 行や track 行があるとエラーになるので除いておく。

```
$ curl -O http://hgdownload.cse.ucsc.edu/goldenpath/mm8/liftOver.gz  
$ gzip -d mm8ToMm9.over.chain.gz
```

ほかのゲノム間を比較したい場合は、golden path のダウンロードサイトの下に liftOver というディレクトリがあって、そこに変換テーブル (*.chain) があります。

変換プログラムをダウンロードしてインストールします。

```
$ curl -O http://hgdownload.cse.ucsc.edu/goldenPath/mm8/chain/mm8ToMm9.over.gz  
$ gunzip mm8ToMm9.over.gz  
$ chmod +x ./liftOver  
$ sudo cp ./liftOver /usr/local/bin/
```

使い方 では変換してみましょう。mm8.bed が変換したい mm8 なゲノム座標のファイルで、mm9 な座標に変換後 mm9.bed に保存されます。

unmapped.txt は座標変換が失敗したエントリが保存されます。start と end が同じ座標のエントリは変換が失敗するのであらかじめ、どちらかの座標に +1 しておくこと。

```
$ liftOver mm8.bed mm8ToMm9.over.chain mm9.bed unmapped.txt
```

異なる生物間のゲノム座標を変換する: liftOver

liftOver はもともとゲノムバージョン間の座標を変換するプログラムであったが、オプションを駆使することで種間の座標を変換することも可能である。

ポイントとなるオプションは複数箇所にヒットを許す -multiple、最小エンサイズを指定する -minSizeT, -minSizeQ の2つである。また -minMatch のデフォルトが 0.95 であるが、種間比較で利用するときは 0.01 など低い値を指定する。例えば、ヒトゲノムのENCODE領域 (0.5-1.8MB) では、-minSizeT=4000 とし、哺乳類では -minSizeQ=20000、鳥類や魚類ゲノムでは -minSizeQ=1000 にすると良いとされている。

TopHatでRNA-seq のデータをリファレンスゲノムへマッピングする

RNA-seq で読まれるシーケンスリードはスプライシング後の配列である。そのためリファレンスゲノムにマッピングするときには、リードがスプライスされる部分(スプライスジャンクション)を跨ぐことが多々ある。そのためジャンクションを考慮したマッピングが求められる。一般に spliced mapping と呼ばれる。TopHat は良く使われる spliced alignment に対応したマッピングソフトである。ここでは、TopHat を用いて、fastq file に保存されたリードをリファレンスゲノムにマップする方法について述べる。

```
$ tophat -p 8 -r 100 -o output_dir bowtie2_indexes/mm9 input_1..
```

Paired end の場合は2つの fastq file を順に指定する。また、paired-end の場合は、-r を使う。これは、シーケンスされたリード間にどのくらいの長さDNAがあるかを base pair で指定する。これはシーケンスライブラリを作成するときに、生物学的実験によって設定されているものである。そのため実験担当者に事前に問い合わせておくべきである。具体的には、Agilent社のBioAnalyzerでDNA断片の分布と平均が得られている。またシーケンスを何bpしたのかも聞いておく(これは fastq をみればわかるが)。

-r に指定する数値を計算する方法を述べる。例えば、典型的なライブラリでは、300 bp のDNA断片からライブラリを作成するが、これを 50 bp の Paired-end でシーケンスすると、その間にあるDNAの長さは 200 bp である。よって、-r 200 と指定する。

indexes/mm9 ではリファレンスゲノム配列が検索しやすいうインデックス化されたファイルを指定している。これは [bowtie](#) に含まれる bowtie-build で行う。TopHat はマッピングの計算の部分には、Bowtie を利用している。-p には CPU core の数を指定することで、その数だけ並列計算が可能になる。

TopHat のデフォルトでは、ひとつのリードが複数箇所にマップされる場合

は、mapping score の高い順に 20箇所まで許される設定になっている。
best なスコアで、一箇所 (unique) にマッピングされるようにするには、-g
1 と指定する。一般的にはマルチヒットを許したほうが発現定量の値が良
くなることが知られている。

Bowtie2でRNA-seqデータをトランск립トームへマッピングする

RNA-Seq のデータをトランスクリプトームへマッピングする方法について述べる。RNA-Seqデータはスプライシング後のRNAをシーケンスするのが一般的である。そのためスプライシングされるイントロンとエクソンの境界にあるシーケンスリードは、ゲノムにはマッピングされない。そのため TopHatなどスプライスドアラインメントに対応したアルゴリズムを利用する必要がある。もうひとつ的方法は、トランスクリプトームに対して、シーケンスリードをそのままマッピングすれば、この問題は考慮の必要がない。このデータは、トランスクリプトごとにマッピングされたリードを操作できるようになるので、トランスクリプトごとに、カウントやカバレッジを計算する際などには便利である。ただし、トランスクリプトームデータベースに含まれないトランスクリプトを考慮できない点に注意すること。

まずトランスクリプトームのデータベース (bowtie2 index) を作成する。このステップは一度やればよい。ここではマウスゲノム mm10 の座標系の RefSeq を利用する。

```
$ curl -O http://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.chr1-20.fa.gz  
$ gzip -d mm10.chr1-20.fa.gz  
$ bowtie2-build mm10.chr1-20.fa mm10
```

次にbowtie2でシーケンスリードをトランスクリプトームにマッピングする。

```
$ bowtie2 -p 8 -x mm10 -U hoge.fastq -S hoge.bowtie2.sam
```

参考までにファイルサイズの目安を書いておく。RNA-Seq (4 multiplex/lane, HiSeq 2000, v3)から得られた約11GBの sam ファイルを bam に変換すると、約2.6GBになる。これをソートすると、2.1GB程度の

ファイルになる。インデックスファイルは、15M程度である。

結果が sam で出力されるため、その後の解析や可視化のために、bam へ変換する必要がある。

中間ファイルを残さずにマッピングして sort された bam と bam index を作成する

```
$ bowtie -p 16 --chunkmbs 512 --verbose -m 1 \
-1 hoge_1.fastq -2 -2 hoge_2.fastq mrna -S 2> bowtie.log \
| samtools view -bSu - | samtools sort -m 10G - hoge.sort; \
samtools index hoge.sort.bam
```

RNA-seq データからの rRNA除去

ここではリファレンスゲノムへのマッピングした bam file から、bedtools のなかの intersectBed を利用して、 rRNA 配列を除く方法を述べる。 -abam は入力ファイル形式を、 -b は除くべきゲノム領域を記載した bed file を指定する。ここでは rRNA 遺伝子が存在するゲノム領域の bed file である。 -v は -b で指定した領域に含まれないものだけ出力するという意味である。 grep -v を思い浮べると良いだろう。

RNA-seq は mRNAを転写量を定量するのが目的である。しかし、 RNA-seq のデータには rRNA 由来の配列が含まれる。そもそも抽出した Total RNA には 99 %程度の rRNA が含まれてしまう。これを分子生物学実験的に取り除いてはいるが、そもそもrRNAが大量にあるため、除去効率が高くても多少の rRNA が残る。rRNAが残ってしまうと、その分シーケンスリードが割かれてしまうため、 mRNAの転写量や構造予測をするために必要な有効シーケンスリード数に届かない遺伝子が出てくる可能性がある。よって rRNAが含まれてしまった量を定量することはサンプルの評価に重要である。

使い方

```
$ intersectBed -abam input.bam -b rRNA.ucsc.mm9.bed -v > output
```

遺伝子発現量(FPKM)を計算する

大量のリード配列から、遺伝子ごとにRNAの発現量を見積る必要がある。これには大別して方法が2つある。ひとつは既知の遺伝子領域に含まれたリード数を数え挙げる方法である。もうひとつは、既知の遺伝子領域の情報を使わず、リードを組み立てて(de novo assembly)、転写されたRNAの構造を予測した後、その発現量を定量する方法である。リファレンスゲノムの精度が高く、完全長cDNAやEST (Expressed Sequence Tag)のシークエンスプロジェクトが進んでいる生物種では、前者の方法が簡単である。後者の方針を取るのは、新規の遺伝子構造(アイソフォーム)や融合遺伝子の発見が目的である場合や、リファレンスとなるゲノムやトランスクリプトームが明らかではない場合である。

マッピングデータから遺伝子発現量を定量する

```
$ cuffdiff -p 24 ensembl_gene.gtf
-L sample01,sample02,sample03,control01,control02,control03
-o results
sample01.bam,    sample02.bam,    sample03.bam
control01.bam,  control02.bam,  control03.bam
```

重複した実験をコンマで区切る リファレンストラスクリプトームの GTF が正しいこと -L ではサンプルごとのラベルを指定する。これをちゃんと入れないと cummeRbund で困る

現在のバージョンでは cuffdiff は使わないので注意

発現が有意に異なる遺伝子を探す

cummeRbund つかってかんばれ的なこと書く。

BowtieでChIP-seq のデータをリファレンスゲノムにマッピングする

RNA-Seqと同じ。インデックスファイルを対象のゲノムに置き換えること。

- Bowtie2でRNA-seqデータをトランскriプトームへマッピングする

MACS2を利用してChIP-Seqのピークを探す

MACS2、はおそらくもっとも良く使われている peak caller である MACS のベータバージョンである。

インストール: Python 2.6 以上、Cython 0.14.1以上, numpy が必要。

numpy のインストール:

```
$ wget "http://downloads.sourceforge.net/project/numpy/NumPy/1.6.1/numpy-1.6.1.t
$ tar zxvf numpy-1.6.1.tar.gz
$ cd numpy-1.6.1
$ python setup.py build --fcompiler=gnu
$ sudo python setup.py install
```

MACS2のインストール:

```
$ w3m http://github.com/downloads/taoliu/MACS/MACS-2.0.9-1.tar.gz
$ tar zxvf MACS-2.0.9.tar.gz
$ cd MACS-2.0.9/
$ sudo python setup.py install --prefix=/opt
$ export PYTHONPATH=/opt/lib/python2.6/site-packages/:$PYTHONPA
```

実行例:

```
$ macs2 callpeak -t results/bowtie/Sox2/Sox2.bowtie.sort.rmRepea
```

-t はターゲットとするTFの bam ファイル、-c はネガティブコントロールとなるサンプルの bam ファイル、-f は入力ファイルの形式を指定、-g はゲノムサイズを指定、-n は任意の実験名、-q は q-value の閾値を指定する。-B は extended fragment pileup, local lambda and score tracks を graphBed

`file` に保存するオプションである。`-g` のゲノムサイズ指定には数値による指定と生物名を指定することができ、ヒトゲノムは、hs、マウスゲノムは mm、線虫ゲノムが ce、ハエゲノムが、dm である。

参考URL: <https://github.com/taoliu/MACS>

MACS2のPeak model をプロットする

MACS2はワトソン/クリック鎖にそれぞれマップされたタグをシフトさせてピークを判定する。これはショートリードのシーケンサーはインサートDNAの端を読むため、実際の結合サイトから位置がずれることを考慮するためである。実際にどのように shift させたのかは、Rのコードとして出力される。MACS2の実行の後に、{転写因子名}_model.rのようなファイル名のテキストファイルが出力される。これを実行すると、peak model のプロットが作成される。

```
R -q -f Sox2_model.r
```

この例では、Sox2_model.pdf に保存される。赤とがそれぞれ forward, reverse 鎖にマップされたタグの分布で、black の線がシフトさせたタグの分布になる。

結合サイトの予測 GPS

GPS (Genome Positioning System) はChIP-seqのデータからタンパク質-DNA結合サイトを1塩基解像度で予測するプログラムである。シーケンスリードの分布を混合確率分布と考えて、分布のパラメータ予測を行うことで、結合サイトの位置を予測する。バックグラウンドには実験データのほか、ポアソン分布を仮定したローカルなバックグラウンドを利用した予測も可能である。

使い方

```
$ java -Xmx80G -jar gps.jar --d Read_Distribution_default.txt  
--s 240000000  
--exptCTCF-GM12878 GM12878_chr1_ip.bed  
--exptCTCF-HUVEC HUVEC_chr1_ip.bed  
--ctrlCTCF-GM12878 GM12878_chr1_ctrl.bed  
--ctrlCTCF-HUVEC HUVEC_chr1_ctrl.bed  
--f BED --g hg18_chr1.info --out HumanCTCF
```

-Xmx でメモリを指定する。mulit-condition mode では大量のメモリと計算時間を必要とする。--d には初期条件に利用する read distribution を指定する。read distribution は予測された結合サイト周辺に存在するシーケンスリードの分布に関するデータが含まれている。--s にはマッピングされるリファレンスゲノム領域のサイズを指定する。大抵はゲノムサイズ * 0.8 を指定する。---exptXXX, --ctrlXXX でそれぞれ実験データ、コントロールデータ (例えば IgGやKO, input genome のデータなど)を指定する。--f は入力ファイル形式を指定し、--out には出力ファイル名を指定する。

Guo Y, Papachristoudis G, Altshuler RC, Gerber GK, Jaakkola TS, Gifford DK, Mahony S. Discovering homotypic binding events at high spatial resolution. Bioinformatics. 2010 Dec 15;26(24):3028-34. Epub 2010 Oct 21. PubMed PMID: 20966006; PubMed Central PMCID: PMC2995123.

GPSの結果を BED に変換する: chipgsp chipgsp を利用し、ChIP-seqからタンパク質-DNA結合サイトの予測をするGPS (Genome Positioning

System) の結果を処理し、ゲノムブラウザやほかのツールで利用できる汎用的なファイル形式である BED file に変換する。

インストール ruby 1.9.0 以上、 rubygem が必要となる。

```
$ sudo gem install chipgps
```

使い方 sample ディレクトリに gps2bed.rb がある。これを利用して GPS のファイルを bed file に変換することが可能である。あるいは以下のスクリプトをテキストファイルに入力する。

```
$ ruby gps2bed.rb Demo_2_GPS_significant.txt > Demo_2_GPS_signi:
```

```
#!/usr/bin/env ruby
# gps2bed.rb

$LOAD_PATH.push("../lib")

require 'pp'
require 'chipgps'

#gps_out_file = ARGV.shift
gps_out_file = "Demo_2_GPS_significant.txt"
gps = Chipgps.new(gps_out_file)

for condition_id in 0..(gps.num_of_conditions-1)

  exp_name = "Demo_Day" + condition_id.to_s
  header = "track name=#{exp_name} description=\"GPS: #{exp_name}\""
  puts header
  gps.each_by_condition(condition_id) do |event|

    # output (bed format)
    puts [
      "chr" + event.chr,                      # chr
      event.bp,                                # start
      event.bp + 1,                            # end
      [exp_name, event.chr, event.bp.to_s].join(":") , # name
    ].join("\t")

  end

end
```

```
    event.fold_enrichment          # score
].join("\t")
```

```
end
```

```
end
```


2つのTF ChIP-seq の共通結合領域を探す

異なる2つの転写因子の結合領域がどのくらい重なっているかを検討することで、転写因子の同じ場所で協調、あるいは競合して働いているかどうかを知ることができる。そこで2つの ChIP-seq 解析から得られたピークの重なりを探索する方法について述べる。

```
> library("ChIPpeakAnno")
> oct4.df <- read.table("results/macs2/Oct4_peaks.bed", header=)
> sox2.df <- read.table("results/macs2/Sox2_peaks.bed", header=)
> oct4.gr <- BED2RangedData(oct4.df, header=FALSE)
> sox2.gr <- BED2RangedData(sox2.df, header=FALSE)
> oct4.sox2.overlap <- findOverlappingPeaks(oct4.gr, sox2.gr)
```

まずファイルからbed fileを入力し、そのデータフレームを BED2RangedData で RangedData というデータ構造に変換する。 findOverlappingPeaks は2つのRangedData の共通した行を探す関数である。

2つのTF ChIP-seq の共通結合領域をカウント しベン図を描く

```
> library("ChIPpeakAnno")
> oct4.df <- read.table("results/macs2/Oct4_peaks.bed", header=)
> sox2.df <- read.table("results/macs2/Sox2_peaks.bed", header=)
> oct4.gr <- BED2RangedData(oct4.df, header=FALSE)
> sox2.gr <- BED2RangedData(sox2.df, header=FALSE)
> makeVennDiagram(RangedDataList(oct4.gr, sox2.gr), NameOfPeaks=
```

Peakを近傍に持つ遺伝子によく観察される Gene Ontologyを挙げる

```
> library("ChIPpeakAnno")
> library(org.Mm.eg.db)
>
> oct4.df <- read.table("results/macs2/Oct4_peaks.bed", header=)
> sox2.df <- read.table("results/macs2/Sox2_peaks.bed", header=)
> oct4.gr <- BED2RangedData(oct4.df, header=FALSE)
> sox2.gr <- BED2RangedData(sox2.df, header=FALSE)
>
> oct4.go <- getEnrichedGO(oct4.anno, orgAnn="org.Mm.eg.db", ma:
> oct4.bp.goterm <- unique(oct4.go$bp[order(oct4.go$bp[,10])],c(:)
> oct4.bp.goterm[1:10,]
```

getEnrichedGo の引数 multiAdjMethod で指定できる多重検定補正の手法は以下の通りである。

- Bonferroni: Bonferroni single-step adjusted p-values for strong control of the FWER.
- Holm: Holm (1979) step-down adjusted p-values for strong control of the FWER.
- Hochberg: Hochberg (1988) step-up adjusted p-values for strong control of the FWER (for raw (unadjusted) p-values satisfying the Simes inequality).
- SidakSS: Sidak single-step adjusted p-values for strong control of the FWER (for positive orthant dependent test statistics).
- SidakSD: Sidak step-down adjusted p-values for strong control of the FWER (for positive orthant dependent test statistics).
- BH: adjusted p-values for the Benjamini & Hochberg (1995) step-up FDR controlling procedure (independent and positive regression dependent test statistics).
- BY: adjusted p-values for the Benjamini & Yekutieli (2001) step-up FDR controlling procedure (general dependency structures)

ピーク内のゲノム配列をFASTA形式で得る

ピーク内のDNA配列を得ることができれば、ChIP-qPCRのような確認実験用の PCR primer を設計したり、配列に頻出するDNA motifを計算機で発見することに利用できる。ここでは、ChIPpeakAnno を利用した方法を述べる。

```
> library("ChIPpeakAnno")
> library("BSgenome.Mmusculus.UCSC.mm9")
> oct4.peaksWithSeqs <- getAllPeakSequence(oct4.gr, upstream = :
> write2FASTA(oct4.peaksWithSeqs, file="resutls/oct4.peaksWithSe
```

ローカルディレクトリに oct4.peaksWithSeqs.fa が保存されている。

ピーク内に頻出するDNAモチーフ配列を発見する

タンパク質がDNAに結合するときには、ある特定のDNA配列に結合することが知られている。ChIP-seq のピーク内には、ほとんどの場合、DNAとタンパク質が結合しているはずである。その領域のなかに、ほかの領域では見られないような、特徴的なDNA配列が発見できれば、それがDNA結合配列の候補となる。

```
> library("rGADEM")
> library("BSgenome.Mmusculus.UCSC.mm9")
> oct4.seqs <- read.DNAStringSet("results/oct4.peaksWithSeqs.fa")
> oct4.motif <- GADEM(oct4.seqs, verbose=1, genome=Mmulculs)
> nMotifs(oct4.motif)
[1] 4
> consensus(oct4.motif)
[1] "ywTTswnATGCAAAT" "nyYYYYYYYYCwsyyn" "sCwGswGrnrG"      "sCCC(
> oct4.motif.pwms <- getPWM(oct4.motif)
> oct4.motif.pwms[[1]]
```

Position weight matrix から Sequence Logo を描く

ピーク内もモチーフを発見した後にそのモチーフ配列とその position weight matrix (PWM)が出力される。PWMは"Sequence Logo"という可視化方法によってモチーフ配列を表示するのが一般的である。ここでは「ピーク内に頻出するDNAモチーフ配列を発見する」の PWMを利用する。

```
> library("seqLogo")
> pdf("oct4.motif01.pdf")
> seqLog(oct4.motif.pwms[[1]])
> dev.off()
```

Position weight matrix を既知モチーフデータベースに対して検索する

予測した PWM が既知のモチーフに一致するかを検索する。これによって新規モチーフなのか、既知のモチーフかを分けることができる。またPWM や sequence logo をみていても、それが既知のタンパク質結合サイトかどうかを判断することは難しいため、既知のデータベースと比較するのがよい。ここではデータベースとの比較、比較結果の出力について述べる。

```
> library("MotIV")
> path <- system.file(package="MotIV")
> jaspar <- readPWMfile(paste(path,"/extdata/jaspar2010.txt",sep=""))
> jaspar.scores <- readDBScores(paste(path,"/extdata/jaspar2010_
> oct4.jaspar <- motifMatch(inputPWM = oct4.motif.pwms, align =
    Ungapped Alignment
    Scores read
    Database read
    Motif matches : 5
> summary(oct4.jaspar)
    Number of input motifs : 4
    Input motifs names : m1 m2 m3 m4
    Number of matches per motif: 5
    Matches repartition :
      SP1   EWSR1-FLI1          SPI1          SPIB          INSM1
      3           2             2           2           1
      PPARG::RXRA       Pax4        Pou5f1        RREB1        SOX10
      1           1             1           1           1
      Sox2        Sox5  Tall1::Gata1
      1           1             1
    Arguments used :
      -metric name : PCC
      -alignment : SWU
> pdf("results/oct4.motif.logo.pdf")
> plot(oct4.jaspar, ncol = 2, top = 5, rev = FALSE, main = "Logo")
> dev.off()
```

ピーク内の配列のなかから既知のDNAモチーフ配列を発見する

発見されたモチーフ配列をDNA配列に対して検索する方法を述べる。これを行うモチベーションはいくつか考えられるが、主な理由は、発見したモチーフがどのピークに含まれているか、いくつ含まれているかを知る、ということである。これによって、発見したモチーフ配列とピーク/遺伝子との関係を理解することができる。この方法は、既知のモチーフ配列を収集したデータベースの PWM をゲノム配列に検索する方法と同じである。

```
> hit.fwd <- matchPWM(oct4.motif.pwms$m1, oct4.seqs[[4]])  
> hit.rev <- matchPWM(reverseComplement(oct4.motif.pwms$m1), oct4.seqs[[4]])  
## スコアを計算する  
> PWMscoreStartingAt(oct4.motif.pwms$m1, subject(hit.fwd), start=1)  
> PWMscoreStartingAt(oct4.motif.pwms$m1, subject(hit.rev), start=1)
```

matchPWM はフォワード鎖のみ検索することに注意。配列を reverseComplement() で逆相補鎖に変換する必要がある。hit.rev はヒットがないので、PWMscoreStartingAt() の戻り値は numeric(0) になる。

複数のピーク内DNA配列に対して、まとめて検索するには以下のようにする

```
oct4.motif.search.fwd <- lapply(  
  oct4.seqs,  
  function(x) {  
    matchPWM(oct4.motif.pwms$m1, x)  
  }  
)  
oct4.motif.search.fwd.score <- lapply(  
  oct4.motif.search.fwd,  
  function(hits) {  
    PWMscoreStartingAt(  
      oct4.motif.pwms$m1,  
      subject(hits),
```

```
    start(hits)
)
}
)
```

summit からのモチーフまでの距離分布を描く

summit からモチーフまでの距離を計算し、そのヒストグラムを描く。もし ChIP に関係のないモチーフ配列の場合、その分布は一様分布になるはずである。しかし、実際に結合に関与しているモチーフ配列ならばその分布は submit にその平均値を持つ正規分布になると考えられる。ここでは MACS2 の summit のデータからこのモチーフ分布を描く。以下のコードでは、Peak を上流下流に延長した場合のことは考慮されていないことに注意。

```
## Load summit data
oct4.sm.df <- read.table("results/macs2/Oct4_summits.bed", header=FALSE)
sox2.sm.df <- read.table("results/macs2/Sox2_summits.bed", header=FALSE)
oct4.sm.gr <- BED2RangedData(oct4.sm.df, header=FALSE)
sox2.sm.gr <- BED2RangedData(sox2.sm.df, header=FALSE)

## calc. distance between motif event and summit
num.peaks <- length(oct4.motif.search.fwd)
oct4.motif.summit.dist <- rep(0, num.peaks)

summit.on.peak <- start(oct4.sm.gr) - start(oct4.gr)

num.peaks <- length(oct4.motif.search.fwd)
num <- 0
oct4.motif.summit.dist <- 0
for (peak in 1:num.peaks) {
  if (! identical(start(oct4.motif.search.fwd[[peak]]), integer(1))) {
    dists <- start(oct4.motif.search.fwd[[peak]]) - summit.on.peak[peak]
    if (length(dists) == 1) {
      num <- num + 1
      oct4.motif.summit.dist[num] <- dists
    } else if (length(dists) > 1) {
      for(i in 1:length(dists)) {
        num <- num + 1
        oct4.motif.summit.dist[num] <- dists[i]
      }
    }
  }
}
```

```
pdf("results/oct4.motifdist.pdf")
plot(density(oct4.motif.summit.dist), main="Motif distribution
dev.off()
```

モチーフ間距離分布を描く

2つのDNAモチーフ間の距離の分布をプロットする。この距離が近い場合、分布が単峰性になる。ここから2つの転写因子がヘテロダイマーなどのように近くで働いているという分子メカニズムが想像できる。距離が離れている場合は二峰性の分布になる。これは、一次配列としての距離は遠くとも、ゲノムDNAが折り曲がることにより、クロマチン構造上、近くに存在する場合や、ほかのコファクターとなるタンパク質を介して、タンパク質間相互作用がある場合などが考えられる。

まず2つの転写因子の結合サイトを計算する。

```
library("ChIPpeakAnno")
library("MotIV")

load("results/3rd/03motifdbSearch.rdat")
oct4.sox2.overlappingPeaks.seq <- read.DNAStringSet("results/3rd/03motifdbSearch.rdat")

oct4.motif.pwms <- getPWM(oct4.motif)
oct4.motif.search.fwd <- lapply(
  oct4.sox2.overlappingPeaks.seq,
  function(x) {
    matchPWM(oct4.motif.pwms$m1, x)
  }
)
oct4.motif.search.rev <- lapply(
  oct4.sox2.overlappingPeaks.seq,
  function(x) {
    matchPWM(reverseComplement(oct4.motif.pwms$m1), x)
  }
)

sox2.motif.pwms <- getPWM(sox2.motif)
sox2.motif.search.fwd <- lapply(
  oct4.sox2.overlappingPeaks.seq,
  function(x) {
    matchPWM(sox2.motif.pwms$m1, x)
  }
)
sox2.motif.search.rev <- lapply(
  oct4.sox2.overlappingPeaks.seq,
```

```

function(x) {
  matchPWM(reverseComplement(sox2.motif.pwms$m1), x)
}

save(list = ls(), file = "results/3rd/04motifsearchInOverlappingPeakseq.rdat")

```

この2つの転写因子の距離の関係をプロットする。

```

library("ChIPpeakAnno")
library("MotIV")

load("results/3rd/04motifsearchInOverlappingPeakseq.rdat")

## calc. distance between motif event and summit
num.peaks <- length(oct4.motif.search.fwd)

num <- 0
motif.dist <- 0
for (peak in 1:num.peaks) {
  if (! identical(start(oct4.motif.search.fwd[[peak]]), integer))
    if (! identical(start(sox2.motif.search.fwd[[peak]]), integer))
      oct4 <- start(oct4.motif.search.fwd[[peak]])
      sox2 <- start(sox2.motif.search.fwd[[peak]])
      for (i in 1:length(oct4)) {
        for (j in 1:length(sox2)) {
          num <- num + 1
          motif.dist[num] <- oct4[i] - sox2[j]
        }
      }
    }
  }

save(list = ls(), file = "results/3rd/05twoMotifDistanceDistribution.rdat")

pdf("results/3rd/05twoMotifDistanceDistribution.pdf")
plot(density(motif.dist), main="Oct4-Sox2 / Motif distance distribution")
dev.off()

```

既知のモチーフ配列/PWMのデータを得る: JASPAR

既知のモチーフ配列/PWMを整理したデータベースJASPARの情報は、rGADEM に付属している。以下の例では、名前に Pou5f1 (Oct3/4) の文字列が含まれている PWM を検索し取り出している。

```
jaspar.path <-  
  system.file("extdata/jaspar2009.txt", package = "rGADEM") seeded.pwm  
  <- readPWMfile(jaspar.path) oct4.jaspar.id <- grep("Pou5f1",  
  names(seeded.pwm))
```

これを配列に対して検索するには「ピーク内の配列のなかから既知のDNAモチーフ配列を発見する」を参照せよ。

deeptools

SciPy, matplotlib をそれぞれセットアップをしておく。

インストール

```
pip install deeptools --user
```

~/.local/bin 以下にある。

Rで遺伝子構造を描く

超並列DNAシーケンサーの登場で、遺伝子構造など、ゲノム上のイベントやオブジェクトのデータを大量に得ることができるようになってきました。データ解析にとって可視化は重要ですが、ゲノム上で起きているイベントなので、ゲノム上に配置して可視化したい場面がよくでてきます。しかし、さまざまなゲノム上のオブジェクトをゲノム座標から画像座標に変換して、絵を書くのは意外と面倒な作業です。

例えば遺伝子構造の絵を書こうとします。これは、遺伝子構造をどのように入手するか、遺伝子構造をどのように描くか、の2つの問題に分けることができます。ここでは、遺伝子構造のデータは、R + Bioconductor の biomaRt パッケージを使って、Ensembl Biomart からダウンロードすることで解決します。遺伝子構造をどのように描くかについては、GenomeGraphs パッケージを利用します。

まずはインストールします。

```
$ sudo R
> source("http://www.bioconductor.org/biocLite.R")
> biocLite("biomaRt")
> biocLite("GenomeGraphs")
```

ここでは、Ensembl biomart からヒトの SMN1 という splicing 異常によって疾患になる例が知られている遺伝子名(Gene symbol)の遺伝子構造を描きます。

まず、SMN1 という名前から、Ensembl Gene ID と染色体位置などを入手します。

```
library(GenomeGraphs)
library(biomaRt)

gene.symbol <- "SMN1"
png.file <- paste(gene.symbol, ".png", sep = "")
```

```

## construct an object of Human Ensembl Biomart
human <- useMart(biomart = "ensembl", dataset = "hsapiens_gene_ensembl")

gene <- getBM(
  attributes = c('hgnc_symbol', 'ensembl_gene_id', 'chromosome',
    filters = 'hgnc_symbol',
    values = gene.symbol,
    mart = human
  )

ensgene.id <- gene[,2]
chr.num <- as.character(gene[,3])

```

次に、Ensembl Gene ID から Ensembl Gene の構造情報(染色体名とその exon/intron の位置)を入手します。また、これに対応する Ensembl Transcript のリストとその構造も入手します。

```

## Get annotation of Ensembl Gene
gene <- makeGene(id = ensgene.id, type = "ensembl_gene_id", biomart = human)
transcript <- makeTranscript(
  id = ensgene.id,
  type = "ensembl_gene_id",
  biomart = human,
  dp = DisplayPars(plotId = TRUE, cex = 0.5)
)

```

遺伝子構造を描画するまえに、染色体の大雑把な位置を把握するための ideogram を描くため、Ideogram object を作ります。黒と白の縞々のアレですね。遺伝子の位置には赤い透明なボックスを描きます。

```

## Create an ideogram object for the entire chromosome
ideoog <- new("Ideogram", chromosome = chr.num)

## Create a highlight of the gene position on the ideogram
## using "absolute coordinates"
highlight.position.ideo <- makeRectangleOverlay(
  0.60, 0.65,
  region = c(0.75, 0.82),

```

```
    coords = "absolute",
    dp = DisplayPars(alpha = .2, fill = "red")
)
```

最後に、遺伝子、転写産物、ideogram をまとめて PNG file に出力します。

```
## Create the plot
png(png.file)
gdPlot(
  list(
    makeTitle(gene.symbol),
    "Chr"      = ideog,
    "Gene"     = gene,
    "Transcripts" = transcript
  ),
  overlays = list(highlight.position.ideo)
)
dev.off()
```

完成した図は以下のようになります。

orenogb2 を使って、複数の bam を可視化する

orenogb2 を利用すると、複数の bam と遺伝子名や座標を指定すると、周辺の遺伝子と bam から計算したカバレッジがグラフ化できます。

インストール

```
$ sudo R
```

パッケージをインストールする

```
source("http://bioconductor.org/biocLite.R")
biocLite(c("ggbio", "GenomicRanges", "GenomicAlignments", "devtools"))
biocLite(c("Mus.musculus", "BSgenome.Mmusculus.UCSC.mm10"))
biocLite(c("Homo.sapiens", "BSgenome.Hsapiens.UCSC.hg19"))

library(devtools)
install_github("dritoshi/orenogb2")
```

座標を指定して描画する

```
q01.bam <- system.file("extdata", "Quartz_01.Pou5f1.bam", package = "orenogb2")
q02.bam <- system.file("extdata", "Quartz_02.Pou5f1.bam", package = "orenogb2")

genome.ver <- 'mm10'
zoom.power <- 1
quartz.bam.files <- c(q01.bam, q02.bam)

gb.quartz <- orenogb2$new(
  genome.ver = genome.ver,
  zoom.power = zoom.power,
  bam.files = quartz.bam.files
)

# Coordination
gb.quartz$chr      <- "chr17"
gb.quartz$start.bp <- 35492880
gb.quartz$end.bp   <- 35526079
gb.quartz$plotgb()
```



Semantic Zoom を利用する

ズームの倍率によって描画スタイルが自動的に変更されることを Semantic Zoom と呼びます。例えば拡大していくとDNA配列が表示されたり、ズームアウトするとATGCを色に対応させたヒートマップを表示したりすることです。orenogb2 は semantic zoom に対応しています。

```
genome.ver <- 'mm10'
zoom.power <- 1/200
quartz.bam.files <- c(q01.bam, q02.bam)

gb.quartz <- orenogb2$new(
  genome.ver = genome.ver,
  zoom.power = zoom.power,
  bam.files = quartz.bam.files
```

```
)
```

```
# Coordination  
gb.quartz$chr      <- "chr17"  
gb.quartz$start.bp <- 35492880  
gb.quartz$end.bp   <- 35526079  
gb.quartz$plotgb()
```



Gene Symbol を指定して描画する

座標を指定しなくても遺伝子シンボルを指定すれば描画することができます。

```
s01.bam <- system.file("extdata", "Smart-Seq2_01.CDK2.bam", pac]  
s02.bam <- system.file("extdata", "Smart-Seq2_02.CDK2.bam", pac]  
  
genome.ver <- 'hg19'  
zoom.power <- 1  
smart2.bam.files <- c(s01.bam, s02.bam)  
  
gb.smart2 <- orenogb2$new(  
  genome.ver = genome.ver,  
  zoom.power = zoom.power,  
  bam.files  = smart2.bam.files  
)  
gb.smart2$getPositionBySymbol('CDK2')  
gb.smart2$plotgb()
```



Rでマゼンタ-黒-緑になるバリアフリーなカラーパレットを生成する

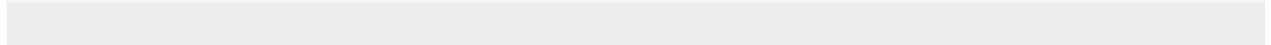
バリアフリーなヒートマップを描くために、マゼンタ-黒-緑のカラーパレットが使いたかったのですが、Rでどうやって良いのかわからなかったので自作しました。簡単な方法を知っている人がいれば教えてください。

colorRampPalette を使う

```
m2b2g <- colorRampPalette(c("#EC008C", "black", "green"))  
pie(rep(1,50), col=m2b2g(50))
```

gplots を使う

```
library(gplots)
m2b2g <- colorpanel(50, low="magenta", mid="black", high="green"
pie(rep(1,50), col=m2b2g)
```



R でソートできるテーブルを作成する

テーブルのタイトル行をクリックするとソートできるテーブルを R で作ります。

まず、インストールします。

```
sudo R  
install.packages('SortableHTMLTables')
```

テーブルを作ってみましょう。デモデータ iris をソートできるHTMLテーブルにしてみます。

```
library('SortableHTMLTables')  
data(iris)  
sortable.html.table(iris, "./index.html", 'iris')
```

iris というディレクトリの中に、index.html と必要なファイルが生成されているはずです。ブラウザで開くと以下のような感じになっているはず。

中身を見てみます。jquery の jquery.tablesorter.js を利用していますね。

```
<html lang="en">  
<head>  
<title>Untitled Page</title>  
<link rel="stylesheet" href="style.css" type="text/css" />  
<script type="text/javascript" src="jquery-1.4.2.js"></script>  
<script type="text/javascript" src="jquery.tablesorter.js"></sc:  
<script type="text/javascript">  
$(document).ready(function() {  
    $("#myTable").tablesorter();  
}  
);  
</script>  
</head>  
<body>  
<table id="myTable" class="tablesorter" border="0" cellspacing='
```

```
<thead>
<tr>
<th>Sepal.Length</th>
<th>Sepal.Width</th>
<th>Petal.Length</th>
<th>Petal.Width</th>
<th>Species</th>
</tr>
</thead>
```

Rでアニメーションするグラフを描く

まずは、非線形モデル $y = \text{mesor} + a \cos(2\pi(t-\text{acrophase})/P)$ に $N(0,0.5)$ のガウシアンノイズを加えたデータを用意し、これを測定されたデータと考える。そのデータに対して非線形回帰するために、Nelder-Mead法で目的関数 $\text{sum}((y-yhat)^2)$ を最大化する。いわゆる cosinor analysis ですね。その計算過程を各ステップごとにプロットし、animation libraryを使ってアニメ化する。

アニメーションの動きが激しくなるように、わざと初期値をおおげざにはずしてある。青が測定データで、緑が予測した回帰曲線。緑のほうの線が動いてみえるはず。収束間近ではあまり動かないでしばらくみつめていると、ループしてまた初期値から計算しなおす。

コードは以下の通り。

```
library(animation)
t <- seq(0, 48, 1)
mesor <- 0
a <- 3
acrophase <- 12
P <- 24
y <- mesor + a * cos(2*pi*(t-acrophase)/P)
y.noise <- mesor + a * cos(2*pi*(t-acrophase)/P) + rnorm(length

resid <- function(par) {
  mesor <- par[1]
  a <- par[2]
  acrophase <- par[3]
  P <- par[4]
  yhat <- mesor + a * cos(2*pi*(t-acrophase)/P)

  matplot(t,
  matrix(c(y.noise, yhat), ncol=2),
  col=c("#1E5692", "#3E9A3B"),
  type="l", lty=1, pch=21, lwd=5, cex=2,
  ylim=c(-5,5))
  sum( (y-yhat)^2 )
}

saveMovie(optim(c(5,10,2,18), resid), interval = 0.01, movietype=
```

ようするにユーザはプロットを何枚か出力するような関数を用意するだけ。それをsaveMovie()関数に渡せば、内部でImageMagickやらSWF Toolsががんばってくれてアニメになる、という仕組みになっている。
ImageMagickを MacPortsあたりで入れておく必要がある。

Flash で出力する

SWF Toolsをインストール

```
sudo port install swf-tools
```

lameやらcmakeやらPythonやらがインストールされる。前回のソース

```
saveMovie(optim(c(5,10,2,18), resid), interval = 0.01, movietype=
```

を

```
saveSWF( optim(c(5,10,2,18), resid), interval = 0.01, dev="png",
```

アニメーションのコントロールボタンを付ける

ソースコードの最後、

```
saveMovie(optim(c(5,10,2,18), resid), interval = 0.01, moviename = "cosinor.html")
```

を

```
ani.options(nmax=200, interval=0.1, title="Cosinor Analysis", optim(c(5,10,2,18), resid))  
ani.start()  
ani.stop()
```

に置きかえるだけでHTMLとアニメ用の画像、再生のコントロールボタンを制御するJavaScriptが自動生成される。

R で家系図を書く

まず次のようなデータ familytree.csv を作成する。

```
"ped","id","father","mother","sex","affected","avail","firstname"
1,101,0,0,1,0,0,"hoge1"
1,102,0,0,2,1,0,"hoge2"
1,103,135,136,1,1,0,"hoge3"
1,104,0,0,2,0,0,"hoge4"
1,105,0,0,1,NA,0,"hoge5"
1,106,0,0,2,NA,0,"hoge6"
1,107,0,0,1,1,0,"hoge7"
1,108,0,0,2,0,0,"hoge8"
1,109,101,102,2,0,1,"hoge9"
1,110,103,104,1,1,1,"hoge10"
1,111,103,104,2,1,0,"hoge11"
1,112,103,104,1,1,0,"hoge12"
1,113,0,0,2,0,1,"hoge13"
1,114,103,104,1,1,0,"hoge14"
1,115,105,106,2,0,0,"hoge15"
1,116,105,106,2,1,1,"hoge16"
1,117,0,0,1,1,0,"hoge17"
1,118,105,106,2,1,1,"hoge18"
1,119,105,106,1,1,1,"hoge19"
1,120,107,108,2,0,0,"hoge20"
1,121,110,109,1,1,0,"hoge21"
1,122,110,109,2,0,0,"hoge22"
1,123,110,109,2,0,0,"hoge23"
1,124,110,109,1,1,1,"hoge24"
1,125,112,118,2,0,1,"hoge25"
1,126,112,118,2,0,1,"hoge26"
1,127,114,115,1,1,1,"hoge27"
1,128,114,115,1,1,1,"hoge28"
1,129,117,116,1,0,1,"hoge29"
1,130,119,120,1,0,1,"hoge30"
1,131,119,120,1,1,0,"hoge31"
1,132,119,120,1,0,0,"hoge32"
1,133,119,120,2,0,1,"hoge33"
1,134,119,120,2,1,0,"hoge34"
1,135,0,0,1,NA,0,"hoge35"
1,136,0,0,2,NA,0,"hoge36"
1,137,0,0,1,NA,0,"hoge37"
1,138,135,136,2,NA,0,"hoge38"
```

```
1,139,137,138,1,1,0,"hoge39"
1,140,137,138,2,0,1,"hoge40"
1,141,137,138,2,0,1,"hoge41"
```

次に familytree.R を作成する。

```
library("kinship2")

# get data filename from command option
args <- commandArgs(TRUE)
file <- args[1]

# make pdf file name
pdf.file <- sub(".csv", ".pdf", file)

# load data from data file
sample.ped <- read.csv(file, header=T)

# make pedigree object
pedAll <- pedigree(
  id = sample.ped$id,
  dadid = sample.ped$father,
  momid = sample.ped$mother,
  sex   = sample.ped$sex,
  famid = sample.ped$ped
)

# get a family
ped1basic <- pedAll["1"]

# make id strings
ids <- paste(sample.ped$id, sample.ped$firstname, sep ="\n")

# plot and save pdf file
pdf(pdf.file)
plot(ped1basic, id = ids, status = sample.ped$avail,
      affected = sample.ped$affected, cex = 0.7)
dev.off()
```

実行する。

```
R -q -f familytree.R --args familytree.csv
```

その他

雑多なこと書く

NGSの情報の集めかた

- 講演&チュートリアル
 - 統合TV
- QAサイト
 - Biostars
 - SEQanswers
 - LSQA
- コミュニティ
 - NGS-field (NGS現場の会)

大量の計算を高速化するには

シーケンステータの計算で、最もボトルネックになるのはハードディスクやSSDなどの記憶装置にアクセスする部分、disk I/Oの部分である。最近のコンピュータはたくさんのCPUコアを持つため、たくさんの計算を同時にこなすことができる。しかしたくさんの計算を同時になると、巨大なファイルの入力や出力に時間がかかる。一定の量は、メモリに蓄えられるが、メモリが足りなくなると、OSは Swap 領域というストレージ装置を利用しはじめる。メモリへのアクセスに対して、ストレージ装置へのアクセスは遅いので、計算スピードも下がる。計算を投入したら、`top` コマンドを利用して、Swap の空き容量を確認しながら、無理のない量の計算をすることが重要である。

図. Swap の値をチェックする

Swap を使わないようにするには、物理的にメモリを大きいものにする方法が根本的な解決ではある。また、なるべくディスクに書き込む回数を減らすのも重要である。そのためには、大量のデータをファイルに出力するときはなるべく圧縮しながら出力する、という方法がある。CPUが比較的空いているのであれば、圧縮にCPUをメモリを使いながら、ストレージ装置に書き込まれる量を減らすことで、速度を稼ぐという考え方である。例えば、、、する場合は、シェルのパイプ機能を使えば、ファイルを圧縮してから、ストレージに書き込むことができるため、ストレージ装置へのアクセスを減らすことが可能である。

バイオインフォマティクス解析のための SQLite入門

SQLite3 はリレーションナルデータベース管理システムのひとつ。サーバクライアントモデルではなく、データベースがファイルになっています。ライセンスがパブリックドメインであることもあります、組み込みDBとしてよく使われます。iPhone の位置情報などのデータを格納しているのも SQLite3 です。ウェブアプリケーションのフレームワークである、Ruby on Rails では標準の組み込みDBとして使われています。また中規模のデータベースならMySQLなどと比較しても遜色ない速度で利用できます。

バイオ業界でも、マイクロアレイやNGSデータを格納する際によく利用されており、Bioconductor の annotation package に含まれるデータも SQLite3 のデータベースになっています。

インストール

Mac OS X には標準でインストールされており、設定なしでターミナルから利用できます。Linuxでもほとんどのディストリビューションでパッケージマネジメントシステムでインストール可能です。Windows は知りません。

使いかた

SQLite3 が正常にインストールされていれば、以下のようにして対話型のクライアントを起動することができます。

```
sqlite3
```

データベースを作成するには (ここでは `expressions.sqlite` という名前のデータベースを作成する)

```
sqlite3 expressions.sqlite
```

とする。

テーブルを作りスキーマを確認する。スキーマとはテーブルの定義のこと。

```
sqlite> create table transcripts(id varchar(10), sample varchar  
sqlite> .schema  
CREATE TABLE transcripts(id varchar(10), sample varchar(10), fp[  
sqlite>
```

データを格納する。

```
sqlite> insert into transcripts values("Nanog", "ESC", 1000);  
sqlite> insert into transcripts values("Pou5f1", "ESC", 1500);
```

データを検索する

```
sqlite> select * from transcripts;  
Nanog|ESC|1000.0
```

```
Pou5f1|ESC|1500.0
sqlite> select * from transcripts where id = "Nanog";
Nanog|ESC|1000.0
sqlite> select fpkm from transcripts where id = "Nanog";
1000.0
```

ヘルプと終了は以下の通り。

```
sqlite> .help
sqlite> .exit
```

ローカルディレクトリに `expressions.sqlite` が作成されています。次回利用したい場合は、

```
sqlite3 expressions.sqlite
```

とすればよいだけです。

様々な bulk 処理

バイオインフォマティクスでは、データを機器や公的データベースから入手し、一度にデータベースに登録するのが主です。ここではその方法について解説します。

まず、スキーマをテキストファイルに書いておいて一気にテーブルを作成します。スキーマファイル `expressions.schema` を作成します。これはテキストエディタで作ります。中身は以下の通りです。

RNA-seq で `gene` と `isoform` レベルの遺伝子発現のデータを格納することを想定しています。テーブル名は英語の複数形で作りましょう。

```
DROP TABLE IF EXISTS genes;
CREATE TABLE genes (
    id      varchar(10),
    sample  varchar(10),
```

```
fpkm      varchar(10)
);

DROP TABLE IF EXISTS isoforms;
CREATE TABLE isoforms (
    id      varchar(10),
    sample  varchar(10),
    fpkm    varchar(10)
);
```

このスキーマにあわせてデータベースを作成します。

```
sqlite3 expressions.sqlite < expressions.schema
```

確認してみましょう。データベースに接続します。

```
sqlite3 expressions.sqlite
```

スキーマを確認します。

```
sqlite> .schema
CREATE TABLE genes (
    id      varchar(10),
    sample  varchar(10),
    fpkm    varchar(10));
CREATE TABLE isoforms (
    id      varchar(10),
    sample  varchar(10),
    fpkm    varchar(10)
);
```

次に、カンマ区切りテキストファイル(CSV)からテーブルを作成してみます。まずデータが含まれるCSVは以下のようになっています。

`expressions.isoforms.csv`

```
Nanog,ESC,1000  
Pou5f1,ESC,1500
```

expressions.genes.csv

```
Nanog,ESC,1200  
Pou5f1,ESC,1700
```

この2つのファイルをそれぞれ、`isoforms`, `genes` というテーブルに取り込みます。

```
sqlite3 expressions.sqlite  
sqlite> .separator ,  
sqlite> .import expressions.isoforms.csv isoforms  
sqlite> .import expressions.genes.csv genes
```

試しに検索してみます。

```
sqlite> select fpkm from transcripts where id = "Nanog";
```

DNA配列をRで操作する

基本的なDNA配列の操作方法や、FASTA/FASTQ fileを取り込む方法を解説します。また全ゲノム配列を読み込み操作する方法についても述べます。

準備

```
source("http://bioconductor.org/biocLite.R")
biocLite("Biostrings")
```

基本的なDNA, アミノ酸配列の操作

```
library("Biostrings")

x <- DNAString("actttGtag")
is(x)
## [1] "DNAString" "XString"    "XRaw"       "XVector"    "Vector"
x
## 9-letter "DNAString" instance
## seq: ACTTTGTAG
```

XString クラスを継承した DNAString オブジェクトを作成しています。このオブジェクトにしておくと、いろいろな配列操作が可能になります。RNA, アミノ酸は、それぞれ、RNAString and AAString を使います。

```
x[1:2]
## 2-letter "DNAString" instance
## seq: AC

aa <- translate(x)
is(aa)
## [1] "AAString" "XString"    "XRaw"       "XVector"    "Vector"
aa
## 3-letter "AAString" instance
## seq: TL*
```

```
is(x.freq)
## [1] "integer"           "numeric"            "vector"
## [4] "data.frameRowLabels" "EnumerationValue"    "atomic"
## [7] "vectorORfactor"

x.freq
##      A      C      G      T other
##      2      1      2      4      0
```

DNA配列やコード表のデータ

もちろんDNAやコードのデータも用意されています。

```
DNA_BASES
## [1] "A" "C" "G" "T"

DNA_ALPHABET
## [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B"

IUPAC_CODE_MAP
##      A      C      G      T      M      R      W      S
##    "A"    "C"    "G"    "T"    "AC"   "AG"   "AT"   "CG"
##      V      H      D      B      N
##    "ACG"   "ACT"   "AGT"   "CGT"  "ACGT"
```

FASTA/FASTQ ファイルの読み込み

multi FASTA 形式のテキストファイルを読み込みます。デモデータは、

```
system.file("extdata", "someORF.fa", package="Biostrings")
```

있습니다。

```
file <- system.file("extdata", "someORF.fa", package = "Biostrings")
seqs <- read.DNAStringSet(file, "fasta")
is(seqs)
## [1] "DNAStringSet" "XStringSet"      "XRawList"       "XVectorList"
## [5] "List"          "Vector"         "Annotated"

seqs
##      A DNAStringSet instance of length 7
##      width seq                                     names
## [1] 5573 ACTTGTAATATATCTTTAT...ATCGACCTTATTGTTGATAT YAL001
## [2] 5825 TTCCAAGGCCGATGAATTCGA...AAATTTTCTATTCTCTT YAL002
## [3] 2987 CTTCATGTCAGCCTGCACTTC...TACTCATGTAGCTGCCTCAT YAL003
## [4] 3929 CACTCATATCGGGGTCTTAC...CCCGAACACGAAAAAGTAC YAL004
## [5] 2648 AGAGAAAGAGTTCACTTCTT...TAATTATGTGTGAACATAG YAL005
## [6] 2597 GTGTCCGGGCCTCGCAGGCGT...TTTGGCAGAATGTACTTT YAL006
## [7] 2780 CAAGATAATGTCAAAGTTAGT...AAGGAAGAAAAAAAATCAC YAL007
```

次に、デモ用の fastq file が `system.file("extdata", "s_1_sequence.txt", package = "Biostrings")` にありますので、これを取り込んでみます。

```
filepath <- system.file("extdata", "s_1_sequence.txt", package = "Biostrings")
fastq.geometry(filepath) # 統計情報を表示
## [1] 256 36

fastq <- read.DNAStringSet(filepath, format = "fastq")
```

全ゲノムDNA配列の操作

全ゲノム配列をネットからダウンロードします。時間がかかるので、日本ミラーを設定することをお勧めします。

参考: [bioconductor のパッケージを高速にインストールする
\(bioconductor.jp の設定\)](#)

ここではマウスゲノム mm10 をダウンロードします。

```
source("http://bioconductor.org/biocLite.R")
biocLite("BSgenome.Mmusculus.UCSC.mm10")
```

ゲノムデータを読み込みます。

```
library("BSgenome.Mmusculus.UCSC.mm10")

Mmusculus
## Mouse genome
## |
## | organism: Mus musculus (Mouse)
## | provider: UCSC
## | provider version: mm10
## | release date: Dec. 2011
## | release name: Genome Reference Consortium GRCm38
## |
## | sequences (see '?seqnames'):
## |   chr1           chr2           chr3
## |   chr4           chr5           chr6
## |   chr7           chr8           chr9
## |   chr10          chr11          chr12
## |   chr13          chr14          chr15
## |   chr16          chr17          chr18
## |   chr19          chrX            chrY
## |   chrM           chr1_GL456210_random  chr1_GL456211_random
## |   chr1_GL456212_random  chr1_GL456213_random  chr1_GL456221_random
## |   chr4_GL456216_random  chr4_GL456350_random  chr4_JH584291_random
## |   chr4_JH584293_random  chr4_JH584294_random  chr4_JH584292_random
## |   chr5_GL456354_random  chr5_JH584296_random  chr5_JH584293_random
## |   chr5_JH584298_random  chr5_JH584299_random  chr7_GL456211_random
```

```

## |   chrX_GL456233_random    chrY_JH584300_random    chrY_JH584301
## |   chrY_JH584302_random    chrY_JH584303_random    chrUn_GL45620
## |   chrUn_GL456359          chrUn_GL456360          chrUn_GL45630
## |   chrUn_GL456367          chrUn_GL456368          chrUn_GL45631
## |   chrUn_GL456372          chrUn_GL456378          chrUn_GL45632
## |   chrUn_GL456381          chrUn_GL456382          chrUn_GL45633
## |   chrUn_GL456385          chrUn_GL456387          chrUn_GL45634
## |   chrUn_GL456390          chrUn_GL456392          chrUn_GL45635
## |   chrUn_GL456394          chrUn_GL456396          chrUn_JH58430
## |
## | (use the '$' or '[' operator to access a given sequence)

```

```

is(Mmusculus)
## [1] "BSgenome"           "GenomeDescription"

seqnames(Mmusculus)
##  [1] "chr1"                  "chr2"                  "chr3"
##  [4] "chr4"                  "chr5"                  "chr6"
##  [7] "chr7"                  "chr8"                  "chr9"
## [10] "chr10"                 "chr11"                 "chr12"
## [13] "chr13"                 "chr14"                 "chr15"
## [16] "chr16"                 "chr17"                 "chr18"
## [19] "chr19"                 "chrX"                  "chrY"
## [22] "chrM"                  "chr1_GL456210_random" "chr1_GL456210_random"
## [25] "chr1_GL456212_random" "chr1_GL456213_random" "chr1_GL456213_random"
## [28] "chr4_GL456216_random" "chr4_GL456350_random" "chr4_JH584293_random"
## [31] "chr4_JH584293_random" "chr4_JH584294_random" "chr4_JH584294_random"
## [34] "chr5_GL456354_random" "chr5_JH584296_random" "chr5_JH584296_random"
## [37] "chr5_JH584298_random" "chr5_JH584299_random" "chr7_GL456233_random"
## [40] "chrX_GL456233_random" "chrY_JH584300_random" "chrY_JH584300_random"
## [43] "chrY_JH584302_random" "chrY_JH584303_random" "chrUn_GL456359_random"
## [46] "chrUn_GL456359"        "chrUn_GL456360"        "chrUn_GL456360"
## [49] "chrUn_GL456367"        "chrUn_GL456368"        "chrUn_GL456368"
## [52] "chrUn_GL456372"        "chrUn_GL456378"        "chrUn_GL456378"
## [55] "chrUn_GL456381"        "chrUn_GL456382"        "chrUn_GL456382"
## [58] "chrUn_GL456385"        "chrUn_GL456387"        "chrUn_GL456387"
## [61] "chrUn_GL456390"        "chrUn_GL456392"        "chrUn_GL456392"
## [64] "chrUn_GL456394"        "chrUn_GL456396"        "chrUn_JH584302_random"
## |

## chr19 の情報を取り出す
mm10.chr19 <- Mmusculus$chr19
is(mm10.chr19)

```

```
## [1] "MaskedDNAString" "MaskedXString"
```

4種類のゲノム配列マスクデータが含まれている。AGAPS は、いわゆる UCSCゲノムブラウザの Gap Track の部分が N でマスクされていることを示す。コンティグの間をNで埋めている。つまりアセンブリギャップ。 AMB は、コンティグ内で読むことができなかった塩基を IUPAC ambiguity letter でマスクしている。RM は RepeatMasker でマスクしている。TRF は tandem repeat をマスクしたもの。

データを読み込んだときのデフォルトは、AGAPSとAMBだけが有効になっているゲノム配列が得られる。例えば、RM を有効にするには、

```
active(masks(mm10.chr19))["RM"] <- TRUE
## Warning: 置き換えるべき項目数が、置き換える数の倍数ではありませんでした
countPattern("AAGAACAT", mm10.chr19) # pattern をカウント
## [1] 2352
```

パターンマッチング

```
mm10.chr19.match <- matchPattern("AAGAACAT", mm10.chr19, max.mis=0)
is(mm10.chr19.match)
## [1] "XStringViews" "Views"           "List"          "Vector"
## [5] "Annotated"

mm10.chr19.match
##   Views on a 61431566-letter DNAString subject
## subject: NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN...NNNNNNNNNNNNNNNNN
## views:
##      start     end width
## [1] 3000189 3000196    8 [AAGAAAAT]
## [2] 3005005 3005012    8 [AAGAACCT]
## [3] 3005091 3005098    8 [AAGAACCT]
## [4] 3006251 3006258    8 [AAGATCAT]
## [5] 3009789 3009796    8 [AACAACAT]
## [6] 3009818 3009825    8 [AAGAATAT]
## [7] 3010972 3010979    8 [AAGAACAG]
## [8] 3011103 3011110    8 [CAGAACAT]
## [9] 3011138 3011145    8 [AAGAACAT]
## ...
## [49007] 61323881 61323888    8 [AAGAAAAT]
## [49008] 61324346 61324353    8 [AAAAACAT]
## [49009] 61324511 61324518    8 [AAGAATAT]
## [49010] 61324707 61324714    8 [AACAACAT]
## [49011] 61325474 61325481    8 [AAGATCAT]
## [49012] 61325642 61325649    8 [AAGAGCAT]
## [49013] 61327589 61327596    8 [AACAACAT]
## [49014] 61329266 61329273    8 [ATGAACAT]
## [49015] 61330773 61330780    8 [AAGGACAT]
```

配列を取り出す

```
mm10.chr19.dna <- as(mm10.chr19, "XStringViews")
is(mm10.chr19.dna)
## [1] "XStringViews" "Views"           "List"            "Vector"
## [5] "Annotated"

mm10.chr19.dna
##   Views on a 61431566-letter DNAString subject
## subject: NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN...NNNNNNNNNNNNNNN
## views:
##       start     end    width
## [1] 3000001 6685768 3685768 [GATCATACTCCTCATGCTGG...GCAAGC]
## [2] 6685869 6688105   2237 [GCACACACACATAACACACAC...ACCTGT]
## [3] 6813616 6829746 16131 [GCCTGCTCTCCTAGGAGTAC...GAGTGG]
## [4] 6829847 61331566 54501720 [GATCAGAGTGGAGCCGCTG...AGGGTT]

start(mm10.chr19.dna[1]) # コンティグのスタートの座標
## [1] 3000001

mm10.chr19.dna[[1]][1:100] # 3000001 bp から 100 bp 取り出す
##   100-letter "DNAString" instance
## seq: GATCATACTCCTCATGCTGGACATTCTGGTTCT...TCAACTTCCCTTTCCT
```

Rから Refseq や UniProt-KBの情報を高速に検索する

ここでは、R package の `genesearchR` を使って、NCBI RefSeq と UniProt-KB の全文検索を高速に行う 方法を述べます。

インストール

Rを起動します。

```
sudo R
```

関連するパッケージをインストールします。

```
source("http://bioconductor.org/biocLite.R")
biocLite("GenomicRanges")

install.packages("devtools")
install.packages("roxygen2")

library(devtools)
install_github("genesearchr", "dritoshi")
```

使い方

```
library("genesearchr")

## RefSeq
my.hs.gene <- refSeqSearch(query = "NM_001518", species = "hs")
my.mm.gene <- refSeqSearch(query = "homeobox", species = "mm")

## UniProt-KB
my.gid.gene <- uniprotSearch(query = "GeneID:93986")
my.eco.gene <- uniprotSearch(query = "eco:b2699")

## Genome DNA Sequence
my.mm10.dna <- genomeSearch(query = "TTCATTGACAACATTGCGT", db =
```

Search オプション

検索ワードの指定の仕方は以下のサイトを参考にしてください。

- [G-links](#), UniProt-KB search
- [GGRNA](#), RefSeq search
- [GGGenome](#), Ultrafast sequence search

仕組み

genesearchR は以下のサービスを検索した結果をRに取り込みます。

- [GGRNA](#)
- [GGGenome](#)
- [G-links](#)

R で EnsEMBL Gene ID を NCBI Entrez Gene ID へ変換する

org.Mm.eg.db というアノテーションパッケージを利用する。Bioconductor のアノテーションパッケージは、AnnDbBimap オブジェクトになっている。これはID変換のテーブルを 1:1 で扱うオブジェクトで、これを操作してIDを変換する。

いや別に Mm じゃなくてもいいし、EnsEMBL じゃなくても共通する話です、はい。

まず、この AnnDbBimap をベクターにしてから変換してみる。

```
library("org.Mm.eg.db")
my.ensmusg <- c("ENSMUSG00000000275", "ENSMUSG00000002844", "ENSMUSG00000003868")
my.ensmusg <- unlist(as.list(org.Mm.egENSEMBL2EG))
# convert EnsMUSG -> Entrez Gene ID
my.eg <- ensmusg[my.ensmusg]
my.eg
ENSMUSG00000000275 ENSMUSG00000002844 ENSMUSG00000003868
"217069"           "11544"           "20174"
```

最後のIDが複数の Entrez ID に対応しているので、変換がうまくいっていない。これは、単に操作ミスによって対応のつくIDが少なくなり、その後の解析結果に影響を及ぼす可能性が高いのでよろしくない。

次は、mget を利用して、AnnDbBimap オブジェクトを直接検索してみる。

```
my.eg2 <- mget(my.ensmusg, org.Mm.egENSEMBL2EG)
R> my.eg2
$ENSMUSG00000000275
[1] "217069"
$ENSMUSG00000002844
[1] "11544"
```

```
$ENSMUSG00000003868
```

```
[1] "20174"
```

```
$ENSMUSG00000091821
```

```
[1] "100151772" "100502688" "100502744" "100502884" "100502947"  
[7] "100503346" "100503401" "100503451" "100504531" "100504596"
```

この方法だと、list でデータが返ってくるので 1:n の対応でも正しくデータを得ることができる。AnnDbBimap はいわゆる環境のようなものなので、exists, ls, eapply なども使える。[Rの環境について](#)はこちらの資料を。

ただし、これを vector に変換するとえらいことになる。

```
R> unlist(my.eg2)  
ENSMUSG00000000275 ENSMUSG00000002844 ENSMUSG00000003868  
"217069" "11544" "20174"  
ENSMUSG000000918211 ENSMUSG000000918212 ENSMUSG000000918213  
"100151772" "100502688" "100502744"  
ENSMUSG000000918214 ENSMUSG000000918215 ENSMUSG000000918216  
"100502884" "100502947" "100502996"  
ENSMUSG000000918217 ENSMUSG000000918218 ENSMUSG000000918219  
"100503346" "100503401" "100503451"  
ENSMUSG0000009182110 ENSMUSG0000009182111 ENSMUSG0000009182112  
"100504531" "100504596" "100504635"
```

4つ目の EnsEMBL Gene ID は ENSMUSG00000091821 で複数の Entrez Gene ID に対応しているが、リストの要素名が勝手にインクリメントされている。ENSMUSG000000918211, ENSMUSG000000918212, ...

これは list の要素名がユニークでなければならないからであるが、これに気付かずに解析を進めてしまうと、EnsEMBLに実在しないIDを含んだまま解析するにことにあるので注意が必要である。

これを data frame や matrix にするとどうなるか？

```
R> as.matrix(my.eg2)
```

```

[,1]
ENSMUSG00000000275 "217069"
ENSMUSG00000002844 "11544"
ENSMUSG00000003868 "20174"
ENSMUSG00000091821 Character,12
R> as.data.frame(my.eg2)
ENSMUSG00000000275 ENSMUSG00000002844 ENSMUSG00000003868 ENSMUSG00000091821
1          217069          11544          20174
2          217069          11544          20174
3          217069          11544          20174
4          217069          11544          20174
5          217069          11544          20174
6          217069          11544          20174
7          217069          11544          20174
8          217069          11544          20174
9          217069          11544          20174
10         217069          11544          20174
11         217069          11544          20174
12         217069          11544          20174

```

結論としては list のまま扱うのがよい、ということになるだろう。

ID変換は、解析中のミスが入りやすいところなので、なるべくID空間を変更しなくてよいように、最初によく考えて選択するのが大切だと思う。RNA-seqの場合などはリファレンスのトランスクリプトームの選択と同義である。

さて、mget の話に戻る。

mget は一致しないIDがあった場合、エラーを出して止まってしまう。大人しく、NAにしてほしいものだ。これを解決するには、exists で先に存在を確認しておくのがよさそう。

```

R> my.ensmusg <- c("ENSMUSG00000000275", "ENSMUSG00000002844", "HOGE")
R> a <- sapply(my.ensmusg, function(x) exists(x, org.Mm.egENSEMBA))
ENSMUSG00000000275 ENSMUSG00000002844 ENSMUSG00000003868 ENSMUSG00000091821
                  TRUE                  TRUE                  TRUE
                  HOGE
                  FALSE

```

```
R> my.enSMusg.existed <- my.enSMusg[a]
my.enSMusg.existed
[1] "ENSMUSG00000000275" "ENSMUSG00000002844" "ENSMUSG0000000386"
[4] "ENSMUSG00000091821"
```

Bioconductor 日本ミラーを利用する

日本ミラーの `bioconductor.jp` を選択するには、以下のように、

```
chooseBioCmirror()
```

で 6 番を選ぶだけです。

```
options("BioC_mirror")
```

で切り替わっていることを確認できます。試しに、`BrainStars` package をインストールしてみると、ちゃんと `bioconductor.jp` からインストールされているのがわかります。

Bioconductorのソースコードを検索する

[Bioconductor Code Search](#)

Vagrant を使って Bioconductor Devel の 解析・開発環境をAWSに構築する

vagrant のインストール

<http://www.vagrantup.com/> から dmg ダウンロードし、インストールする。

vagrant をセットアップする

aws にプロビジョニングできるプラグインをインストールする。

```
$ vagrant plugin install vagrant-aws
```

AMIを起動するとは言え、ダミーの仮想マシンが必要。ちょっとわかりにくい。
\$ vagrant box add dummy <https://github.com/mitchellh/vagrant-aws/raw/master/dummy.box>

Vagrantfile を作る

Bioconductor 公式のBioC-Devel入りの AMI を利用する。リージョンはバージニアだけ。知る必要はないがアカウント名は root です。

まず適当なディレクトリを作る。

```
$ mkdir bioc-devel  
$ cd bioc-devel
```

初期化する。

```
$ vagrant init
```

Vagrantfile にいろいろ書く。

```
$ jed Vagrantfile  
# -*- mode: ruby -*-  
# vi: set ft=ruby :  
  
require 'yaml'  
require 'pp'  
  
aws_conf = YAML.load_file('../.aws.yaml')  
# pp aws_conf  
  
VAGRANTFILE_API_VERSION = "2"  
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|  
  config.vm.box = "dummy"  
  
  config.vm.provider :aws do |aws, override|  
    aws.access_key_id      = aws_conf['access_key_id']  
    aws.secret_access_key  = aws_conf['secret_access_key']  
    aws.keypair_name       = aws_conf['keypair_name']  
  
    aws.instance_type      = aws_conf['instance_type']  
    aws.ami                 = aws_conf['ami']  
    aws.region              = aws_conf['region']
```

```
aws.security_groups      = aws_conf['security_groups']
aws.tags = {
  'Name'          => aws_conf['tags']['Name'],
  'Description'   => aws_conf['tags']['Description']
}
aws.elastic_ip = true

override.ssh.username      = aws_conf['ssh_username']
override.ssh.private_key_path = aws_conf['ssh_private_key_pa
end

# shell
config.vm.provision :shell, :path => "bootstrap.sh"

end
```

プロビジョニングされたときにAMI上で実行されるシェルスクリプトを作る。今回はなにも実行しない。

```
$ echo "#!/bin/sh" > bootstrap.sh
```

AWSに作るインスタンスの設定を作る

Keypair を作る

AWSにログインし keypair を作る。ダウンロードされた *.pem* を
~/.ssh/*pem* にコピー後、400にする

```
$ cp ~/Downloads/*.pem ~/.ssh  
$ chmod 400 ~/.ssh/*.pem
```

Security Group を設定する

default の Inbound で SSH の source を 0.0.0.0/0 にする。注意: 本来はIP制限すべき。アカウントの access key id や secret access key を調べておく。

AWSの設定ファイルを作る

注意: 以下をうっかり github とかにアップしないように!! .gitignore に書いておこう。AWSの情報をYAMLで書いておく。Vagrantfile と切り分けるためです。

```
$ jed .aws.yaml  
access_key_id: XXXX  
secret_access_key: XXXXXX  
keypair_name: XXXX  
ssh_username: root  
ssh_private_key_path: ~/.ssh/XXXX  
instance_type: m1.xlarge  
region: us-east-1  
ami: ami-81acace8  
security_groups:  
- default  
tags:  
Name: bioc-devel  
Description: bioc-devel
```

プロビジョニングして、SSHでログインする

```
$ vagrant up --provider=aws  
$ vagrant ssh
```

Rを実行して、Bioconductor Devel が使えることを確認する

```
$ R
R Under development (unstable) (2014-02-24 r65070) -- "Unsuffered
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Bioconductor version 2.14 (BiocInstaller 1.13.3), ?biocLite for
```

Python環境と整える

Python 2.7 系が前提。

LAPACK/BLAS をセットアップ

```
cd ~/src
curl -O http://www.netlib.org/lapack/lapack-3.5.0.tgz
tar zxvf lapack-3.5.0.tgz
cd lapack-3.5.0
cp make.inc.example make.inc
```

make.inc の以下の2行を書き換える

```
OPTS      = -O2 -frecursive -m64 -fPIC
NOOPT     = -O0 -frecursive -m64 -fPIC
```

```
make blaslib
make
cp *.a ~/opt/lib
```

~/.zshenv に以下を追記する

```
# BLAS/LAPACK
export BLAS=~/opt/lib/librefblas.a
export LAPACK=~/opt/lib/liblapack.a
```

scipy をインストール

```
pip install --user scipy
```

matplotlib をセットアップ

```
git clone https://github.com/matplotlib/matplotlib.git  
cd matplotlib  
python setup.py install
```

執筆環境について

MacOS X, homebrew, npm, [gitbook](#)

以下 bluesky にて。

setup

```
cd /Users/itoshi/Projects/test  
brew install npm  
npm install gitbook -g
```

Calibreをダウンロードしてインストールする。

```
sudo ln -s /Applications/calibre.app/Contents/console.app/Contents/MacOS/calibre /usr/local/bin/calibre
```

writing

gitbook を起動する

```
gitbook serve ./repository
```

<http://localhost:4000/> へアクセス。エディタを起動する。

```
subl ./repository
```

./repository 以下に md を作っていく。 ファイルを保存するとブラウザ側でコンテンツが自動的にリロードされる。 コンテンツを編集したら、 SUMMARY.md を編集して、新規追加したコンテンツを追加する。

ファイルを作る

PDf

```
gitbook pdf ./repository  
open ./repository/book.pdf
```

ePub

```
gitbook epub ./repository  
open ./repository/book.epub
```

ToDo

- rmd をコンパイルできるように plugin を書く

目次

はじめに	2
品質管理・前処理	2
FASTQ format	9
FASTQCによるシーケンス実験の品質管理	11
余分な配列の除去	13
シーケンスクオリティが低いリードを除く	17
なぜアダプタやプライマー配列を除く必要があるのか	18
ゲノムのリピート領域にマッピングされたリードを除く	18
FASTA/FASTQ/SAM/BAMの詳しいステータスを得る	20
FASTA/BAMからランダムにリードを取り出す	21
シーケンスリードのカバレッジを計算する	26
DNA配列のGC%を計算する	26
異なるバージョンのゲノム座標を変換する: liftover	30
異なる生物間のゲノム座標を変換する: liftover	30
TopHatでRNA-seq のデータをリファレンスゲノムへマッピングする	33
Bowtie2でRNA-seqデータをトランскriプトームへマッピングする	35
RNA-seq データからの rRNA除去	35
遺伝子発現量(FPKM)を計算する	39
発現が有意に異なる遺伝子を探す	39
Bowtie2でChIP-Seqデータをゲノムへマッピングする	39
MACS2による結合サイトの予測	42
GPSによる結合サイトの予測	45
結合領域を近傍の遺伝子に割り当てる	45
ピークがゲノムのどのような場所に存在しているのかを円グラフにする	45
2つのTF ChIP-seq の共通結合領域を探す	49
Peakを近傍に持つ遺伝子によく観察されるGene Ontologyを挙げる	51
ピーク内のゲノム配列をFASTA形式で得る	51
ピーク内に頻出するDNAモチーフ配列を発見する	51
Position weight matrix から Sequence Logo を描く	51
Position weight matrix を既知モチーフデータベースに対して検索する	55
ピーク内の配列のなかから既知のDNAモチーフ配列を発見する	55
summit からのモチーフまでの距離分布を描く	58
モチーフ間距離分布を描く	60
既知のモチーフ配列/PWMのデータを得る: JASPAR	60
deepToolsのセットアップ	63
Rで遺伝子構造を描く	65
orenogb2 を使って、複数の bam を可視化する	68
Rでマゼンタ-黒-緑になるバリアフリーなカラーパレットを生成する	72
R でソートできるテーブルを作成する	75
R でアニメーションするグラフを描く	77

R で家系図を書く	81
その他	81
NGSの情報の集めかた	81
大量の計算を高速化するには	86
バイオインフォマティクス解析のためのSQLite入門	87
DNA配列をRで操作する	93
Rから Refseq や UniProt-KB の情報を高速に検索する	103
R で Ensembl Gene ID を NCBI Entrez Gene ID へ変換する	107
Bioconductor 日本ミラーを利用する	107
Bioconductor のソースコードを検索する	107
Vagrant を使って Bioconductor Devel の解析・開発環境をAWSに構築する	113
SciPyをセットアップする	120
matplotlibをセットアップする	120
執筆環境	124