

ゲノム情報解析基礎

～ Rで塩基配列解析1～

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/>

講義予定

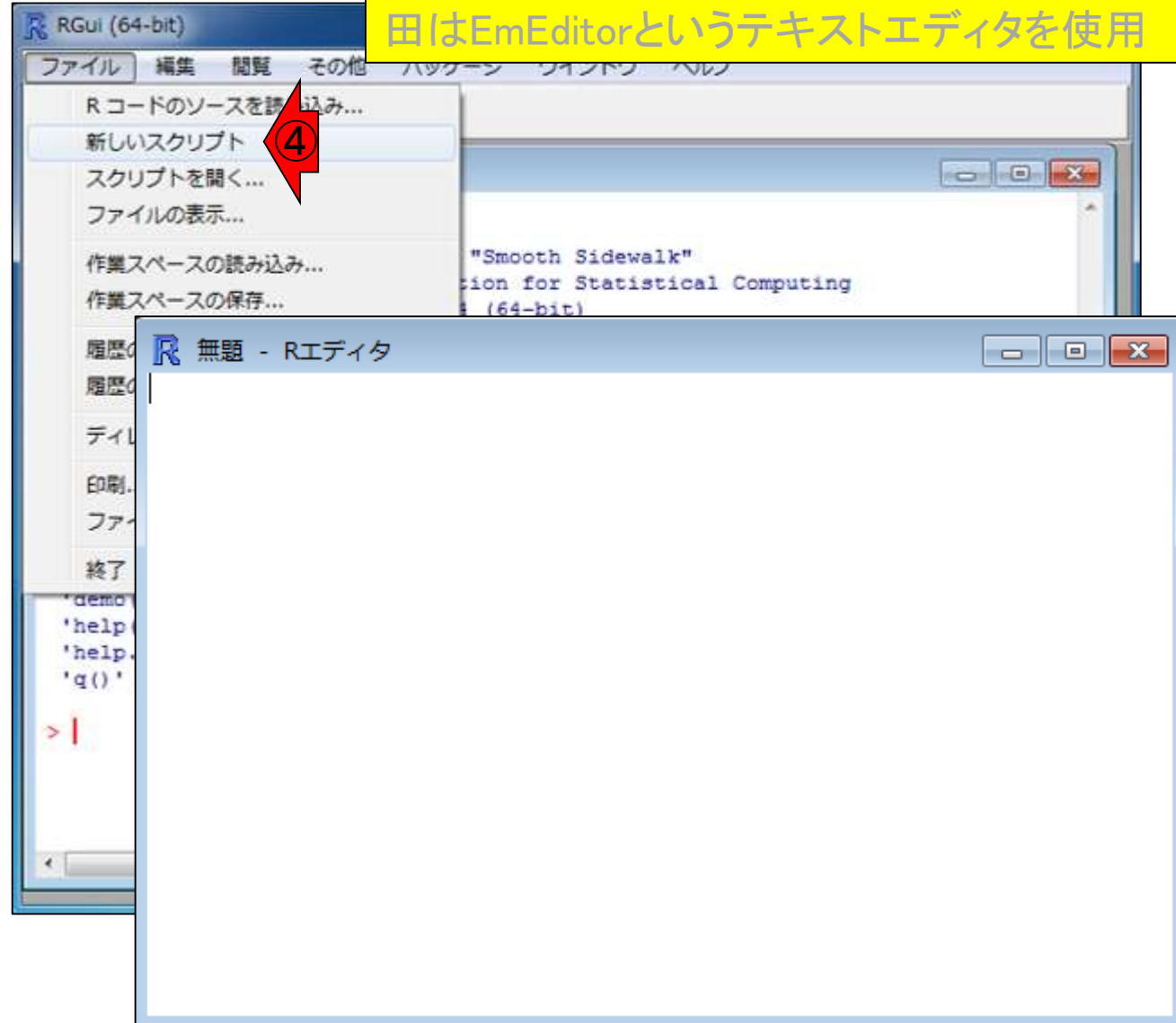
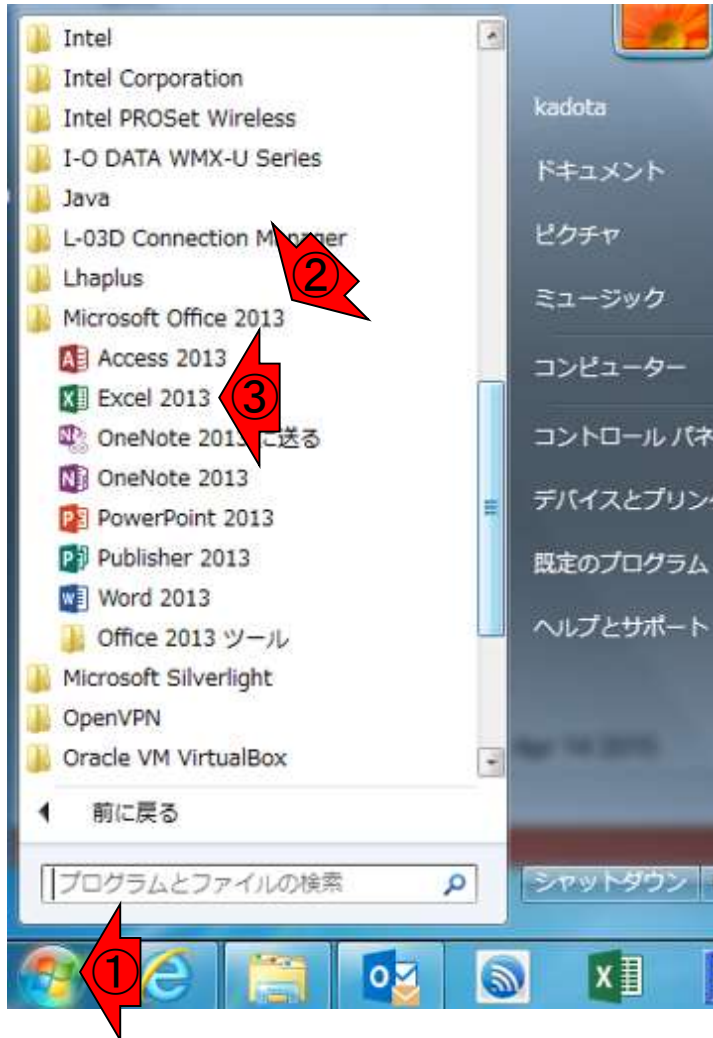
- 4月17日月曜日(17:15-20:30)
 - 嶋田透:ゲノムからの遺伝子予測
 - 門田幸二:バイオインフォマティクス基礎知識、Rのイントロダクション
- 4月18日火曜日(17:15-20:30)
 - 門田幸二:Rで塩基配列解析1、multi-FASTAファイルの各種解析
- 4月24日月曜日(17:15-20:30)
 - 嶋田透:ゲノムアノテーション、遺伝子の機能推定、RNA-seqなどによる発現解析、比較ゲノム解析
 - 門田幸二:Rで塩基配列解析2、Rパッケージ、k-mer解析の基礎
- 5月01日月曜日(17:15-19:00頃)
 - 勝間進:非コードRNA、小分子RNA、エピジェネティクス
 - 講義後、小テスト

Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

各種ソフトの場所

①②③Excelは行列データファイルの確認用。④エディタはR付属のものを推奨。主目的は二重クォーテーション問題の回避。門田はEmEditorというテキストエディタを使用



各講義科目へのアクセス

①教育プログラム、②各講義のページ、③「ゲノム情報解析基礎」の場合



(Rで)塩基配列解析

①「(Rで)塩基配列解析」を用いて講義を進めます。レポート課題で用いるhoge6.faは、②からダウンロードしてください

2. ゲノム情報解析基礎

授業の目標・概要

次世代シーケンサーの普及に
るスキルが要求される時代に
フリーソフトウェアRを用いて
を目指した実習を含む講義を
また、ウェブツールなどを用
は解析するための手法につい

担当教員

嶋田 透 (東大・農・生産・環
勝間 進 (東大・農・生産・環
門田幸二 (東大・農・アグリ

参考図書

講義日程 (平成29年度)

1. 平成29年04月17日 (PC使用)

講師：嶋田 透

講師：門田幸二

バイオインフォマティクス基礎

講義資料PDF(Win版; 完全版)

講義資料PDF(Mac版; Rの説明

2. 平成29年04月18日 (PC使用)

講師：門田幸二

講義資料PDF

(Rで)塩基配列解析

hoge6.fa (課題用)

3. 平成29年04月24日 (PC使用)

講師：嶋田 透

講師：門田幸二

4. 平成29年05月01日 (PC使用)

講師：勝間 進

(Rで)塩基配列解析

(last modified 2017/04/10, since 2010)

このウェブページのR関連部分は、[インストール](#)についての推奨手順 ([Windows2015.04.04版](#)と[Macintosh2015.04.03版](#))に従ってフリーソフトRと必要なパッケージをインストール済みであるという前提で記述しています。初心者の方は[基本的な利用法](#)([Windows2015.04.03版](#)と[Macintosh2015.04.03版](#))で自習してください。本ウェブページを体系的にまとめた[書籍](#)もあります。(2015/04/03)

What's new?

- Galaxyのウェブサイトのリンク先を<http://usegalaxy.org>から<https://galaxyproject.org/>に変更しました。(2017/03/17) **NEW**
- [日本乳酸菌学会誌](#)のNGS関連連載の[第9回分原稿PDF](#)を公開しました。(2017/03/13) **NEW**
- 「[書籍](#) | [日本乳酸菌学会誌](#) | [第7回ロンドンリードアセンブリ](#)」でgrep -Aの説明が間違っていたので修正しました。「検索文字列の行を含む後ろx行を表示」みたいにかいていましたが、検索文字列の行は含まないようですm(_)_m(2017/01/26)

- [門田からメール返信をもらえない場合は](#) (last modified 2016/08/23)

- [はじめに](#) (last modified 2015/03/31)

- 参考資料 | [書籍、学会誌](#) (last modified 2017/03/13) **NEW**

- 参考資料 | [講習会、講義、講演資料](#) (last modified 2016/12/07)

- [過去のお知らせ](#) (last modified 2017/04/10) **NEW**

- [インストール](#) | [インストール](#) (last modified 2015/11/12)

- インストール | R本体 | [最新版](#) | [Win用](#) (last modified 2015/03/22) 推奨

- インストール | R本体 | [最新版](#) | [Mac用](#) (last modified 2015/04/22) 推奨

- インストール | R本体 | [過去版](#) | [Win用](#) (last modified 2015/03/22)

- インストール | R本体 | [過去版](#) | [Mac用](#) (last modified 2015/03/22)

- インストール | Rパッケージ | [ほぼ全て\(20GB以上?\)](#) (last modified 2015/05/25)

- インストール | Rパッケージ | [必要最小限プラスアルファ\(数GB?\)](#) (last modified 2017/03/13) 推奨 **NEW**

- インストール | Rパッケージ | [必要最小限プラスアルファ\(アグリバイオ居室のみ\)](#) (last modified 2015/06/16)

- インストール | Rパッケージ | [必要最小限\(数GB?\)](#) (last modified 2015/05/25)

(遺伝子)アノテーション

遺伝子ごとに、どの染色体のどの座標上に存在するのかなどの情報を含むタブ区切りテキストファイル。①GFF/GTF形式ファイルが有名

- ・ 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](#) (last modified 2015/01/18)
- ・ イントロ | NGS | アノテーション情報取得 | [|について](#) (last modified 2014/03/26)
- ・ イントロ | NGS | アノテーション情報取得 | [GFF/GTF形式ファイル](#) ① (last modified 2014/04/11)
- ・ イントロ | NGS | アノテーション情報取得 | [refFlat形式ファイル](#) (last modified 2013/09/25)
- ・ イントロ | NGS | アノテーション情報取得 | [biomaRt\(Durinck 2009\)](#) (last modified 2013/09/26)
- ・ イントロ | NGS | アノテーション情報取得 | [TxDb|について](#) (last modified 2014/03/28)

イントロ | NGS | アノテーション情報取得 | GFF/GTF形式ファイル **NEW**

多くの生物種について [Ensembl \(Yates et al., Nucleic Acids Res., 2016\)](#) の [FTPサイト](#) から [GTF形式\(GFF ver. 2\)](#) の遺伝子アノテーションファイルを得ることができます。2017年4月10日現在、上記の [FTPサイト](#) へのアクセス時にユーザ名とパスワードを聞かれますが、「匿名でログオンする」にチェックを入れればOKです。GTFはGeneral Transfer FormatまたはGene Transfer Formatの略で、GTFの派生版として [GTF2](#) というフォーマットもあるようです。また、[General Feature Format ver. 3 \(GFF3\)](#) という形式も存在するなど、GFF/GTF形式として総称されている中で様々なバリエーションがあります。いずれもrefFlat形式同様の領域にどの遺伝子があるのかという座標(Coordinates)情報を含みます。ゲノム配列のバージョンと同じであることを確認した上で用いましょう。

- ・ [Ensembl \(Yates et al., Nucleic Acids Res., 2016\)](#)
圧縮(gzip)ファイル形式です。基本は[FTPサイト](#)です。代表的なものを以下にリストアップしています。
 - [ヒト; Human \(H.sapiens\)](#)
 - [ラット; Rat \(R.norvegicus\)](#)
 - [ネコ; Cat \(F.catus\)](#)
 - [ウサギ; Rabbit \(O.cuniculus\)](#)
 - [ニワトリ; Chicken \(G.gallus\)](#)
 - [イヌ; Dog \(C.familiaris\)](#)
 - [ウマ; Horse \(E.caballus\)](#)
 - [ゼブラフィッシュ; Zebrafish \(D.rerio\)](#)
 - ...
- ・ イネ: [RAP-DB \(Sakai et al., Plant Cell Physiol., 2013\)](#)
 - 「[ダウンロード](#)」-「Gene set」-「Gene structure and function information in GFF format」-「[Download](#)」。
IRGSP-1.0_representative_2014-03-05.tar.gz (12.4MB程度)の圧縮ファイルが得られます。
- ・ シロイヌナズナ: [The Arabidopsis Information Resource \(TAIR\) \(Lamesch et al., Nucleic Acids Res., 2012\)](#)
 - 「[ダウンロード](#)」-「[Genes](#)」-「[TAIR10 genome release](#)」-「[TAIR10 gff3](#)」の [TAIR10 GFF3 genes.gff](#) (42MB程度)

GFF/GTF形式ファイルの例

GFF3形式 (シロイヌナズナ; TAIR10_GFF3_genes.gff)

▲	A	B	C	D	E	F	G	H	I
1	Chr1	TAIR10	chromosome	1	30427671	.	.	.	ID=Chr1;Name=Chr1
2	Chr1	TAIR10	gene	3631	5899	.	+	.	ID=AT1G01010;Note=protein_coding_gene;Name=AT1G01010
3	Chr1	TAIR10	mRNA	3631	5899	.	+	.	ID=AT1G01010.1;Parent=AT1G01010;Name=AT1G01010.1;Index=1
4	Chr1	TAIR10	protein	3760	5630	.	+	.	ID=AT1G01010.1-Protein;Name=AT1G01010.1;Derives_from=AT1G01010.1
5	Chr1	TAIR10	exon	3631	3913	.	+	.	Parent=AT1G01010.1
6	Chr1	TAIR10	five_prime_UTR	3631	3759	.	+	.	Parent=AT1G01010.1
7	Chr1	TAIR10	CDS	3760	3913	.	+	0	Parent=AT1G01010.1,AT1G01010.1-Protein;
8	Chr1	TAIR10	exon	3996	4276	.	+	.	Parent=AT1G01010.1
9	Chr1	TAIR10	CDS	3996	4276	.	+	2	Parent=AT1G01010.1,AT1G01010.1-Protein;
10	Chr1	TAIR10	exon	4486	4605	.	+	.	Parent=AT1G01010.1
11	Chr1	TAIR10	CDS	4486	4605	.	+	0	Parent=AT1G01010.1,AT1G01010.1-Protein;
12	Chr1	TAIR10	exon	4706	5095	.	+	.	Parent=AT1G01010.1

他にrefFlat形式など様々なファイル形式が存在します。このようなファイルを入力として、任意の(染色体上にある遺伝子の)サブセットを抽出することができます

GTF形式 (ゼブラフィッシュ; Danio_rerio.Zv9.75.gtf)

▲	A	B	C	D	E	F	G	H	I
1									#!genome-build Zv9
2									#!genome-version Zv9
3									#!genome-date 2010-04
4									#!genome-build-accession NCBI:GCA_000002035.2
5									#!genebuild-last-updated 2014-02
6	7	protein_coding	gene	100958	101715	.	+	.	gene_id "ENSDARG00000076051"; gene_name "CABZ01062994.1"; gene
7	7	protein_coding	transcript	100958	101715	.	+	.	gene_id "ENSDARG00000076051"; transcript_id "ENSDART00000113409
8	7	protein_coding	exon	100958	100975	.	+	.	gene_id "ENSDARG00000076051"; transcript_id "ENSDART00000113409
9	7	protein_coding	CDS	100958	100975	.	+	0	gene_id "ENSDARG00000076051"; transcript_id "ENSDART00000113409
10	7	protein_coding	exon	101077	101715	.	+	.	gene_id "ENSDARG00000076051"; transcript_id "ENSDART00000113409
11	7	protein_coding	CDS	101077	101715	.	+	0	gene_id "ENSDARG00000076051"; transcript_id "ENSDART00000113409
12	7	protein_coding	gene	116160	117573	.	+	.	gene_id "ENSDARG00000088691"; gene_name "BX511027.1"; gene_sour
13	7	protein_coding	transcript	116160	117573	.	+	.	gene_id "ENSDARG00000088691"; transcript_id "ENSDART00000129330


解析基礎2

目的: アノテーションファイル(annotation.txt)中の第1列目に対して、リストファイル(genelist1.txt)中の文字列と一致する行を抜き出して、hoge1.txtというファイル名で出力したい

入力1: アノテーションファイル(annotation.txt)

	A	B	C	D
1	gene1	hoge01	plasma_mem	nuclear
2	gene2	hoge02	hohinu	membrane
3	gene3	hoge03	agribio	endoplasmic
4	gene4	hoge04	genesis	endoplasmic
5	gene5	hoge05	kamo	membrane
6	gene6	hoge06	netteba	humei
7	gene7	hoge07	tebasaki	nuclear
8	gene8	hoge08	biiru	nuclear
9	gene9	hoge09	nihonshu	nuclear
10	gene10	hoge10	agene1	membrane
11	gene11	hoge11	iyaaaa	endoplasmic

出力:hoge1.txt



	A	B	C	D
1	gene1	hoge01	plasma_mem	nuclear
2	gene7	hoge07	tebasaki	nuclear
3	gene9	hoge09	nihonshu	nuclear

入力2: リストファイル(genelist1.txt)

	A
1	gene1
2	gene7
3	gene9

解析基礎2

目的: アノテーションファイル(annotation.txt)中の第1列目に対して、リストファイル(genelist1.txt)中の文字列と一致する行を抜き出して、hoge1.txtというファイル名で出力したい

- ・ イントロ | 一般 | [ランダムに行を抽出 \(last modified 2014/07/17\)](#)
- ・ イントロ | 一般 | [任意の文字列を行の最初に挿入 \(last modified 2014/07/17\)](#)
- ・ イントロ | 一般 | [任意のキーワードを含む行を抽出\(基礎\)](#) (last modified 2014/04/11)
- ・ イントロ | 一般 | [ランダムな塩基配列を生成 \(last modified 2014/06/16\)](#)
- ・ イントロ | 一般 | [任意の長さの可能な全ての塩基配列を作成 \(last modified 2015/02/19\)](#)
- ・ イントロ | 一般 | [任意の位置の塩基を置換 \(last modified 2014/06/16\)](#)
- ・ イントロ | 一般 | [指定した範囲の配列を取得 \(last modified 2014/06/16\)](#)
- ・ イントロ | 一般 | [指定したID\(染色体やdescription\)の配列を取得 \(last modified 2014/06/16\)](#)
- ・ イントロ | 一般 | [翻訳配列\(translate\)を取得 \(last modified 2014/06/16\)](#)
- ・ イントロ | 一般 | [翻訳配列\(translate\)を取得 \(last modified 2014/06/16\)](#)

イントロ | 一般 | 任意のキーワードを含む行を抽出(基礎)

例えばタブ区切りテキストファイルが手元があり、この中からリストファイル中の文字列を含む行を抽出するやり方を示します。Linux (UNIX)のgrepコマンドのようなものであり、perlのハッシュのようなものです。
「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ー。

1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リストファイル(genelist1.txt)中のものが含まれる行全体を出力したい場合:

```

in_f1 <- "annotation.txt" #入力ファイル名を指定してin_f1に格納(アノテーションファイル)
in_f2 <- "genelist1.txt" #入力ファイル名を指定してin_f2に格納(リストファイル)
out_f <- "hoge1.txt" #出力ファイル名を指定してout_fに格納
param <- 1 #アノテーションファイル中の検索したい列番号を指定

#入力ファイルの読み込み
data <- read.table(in_f1, header=TRUE, sep="\t", quote="")#in_f1で指定したファイルの読み込み
keywords <- readLines(in_f2) #in_f2で指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示

#本番
obj <- is.element(as.character(data[,param]), keywords)#条件を満たすかどうかを判定した結果
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F)#outの中身を指定したファイルに保存

```



解析基礎2

①作業ディレクトリは「デスクトップ - hoge」。hogeフォルダ中にannotation.txtとgenelist1.txtが存在するという前提。②貸与PCはkadotaではなくiu

イントロ | 一般 | 任意のキーワードを含む行を抽出(基礎)

例えばタブ区切りテキストファイルが手元があり、この中からリストファイル中の文字列を含む行を抽出するやり方を示します。Linux (UNIX)のgrepコマンドのようなものであり、perlのハッシュのようなものです。「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。

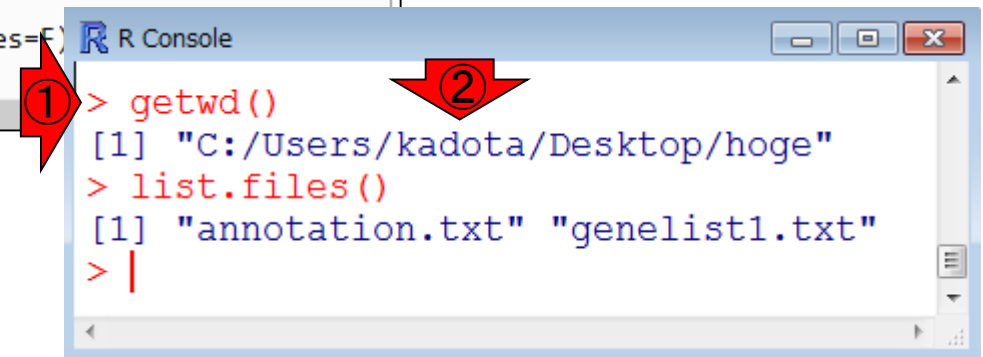
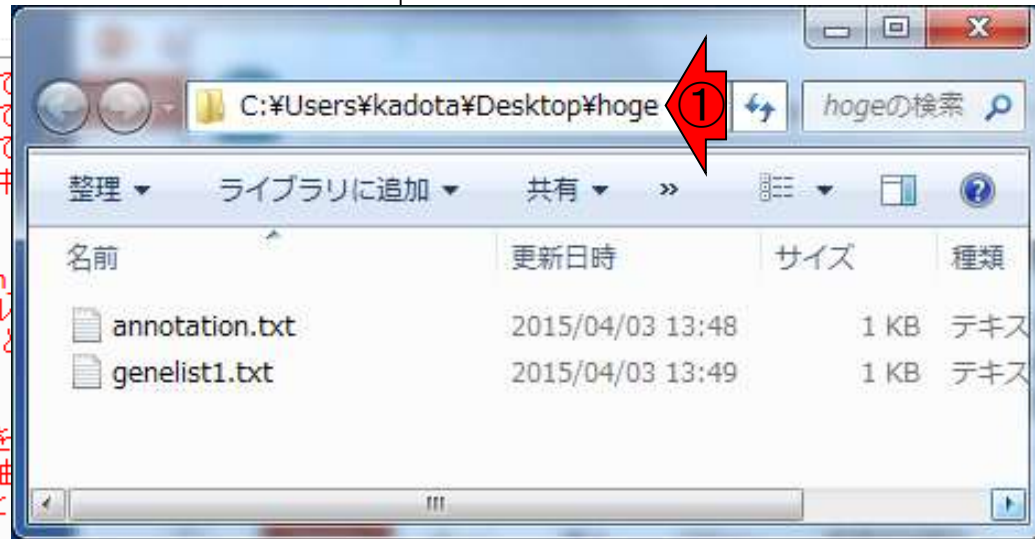
1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リストファイル(genelist1.txt)中のものが含まれる行全体を出力したい場合:

```
in_f1 <- "annotation.txt" #入力ファイル名を指定して
in_f2 <- "genelist1.txt" #入力ファイル名を指定して
out_f <- "hoge1.txt" #出力ファイル名を指定して
param <- 1 #アンテーションファイル中

#入力ファイルの読み込み
data <- read.table(in_f1, header=TRUE, sep="\t", quote="") #in
keywords <- readLines(in_f2) #in_f2で指定したファイル
dim(data) #オブジェクトdataの行数と

#本番
obj <- is.element(as.character(data[,param]), keywords) #条件を
out <- data[obj,] #objがTRUEとなる行のみ抽
dim(out) #オブジェクトoutの行数と

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



基本はコピー

イントロ | 一般 | 任意のキーワードを含む行を抽出

①一連のコマンド群をコピーして②R Console画面上でペースト。ブラウザがInternet Explorerの場合は、CTRLとALTキーを押しながらコードの枠内で左クリックすると、全選択できます。トリプルクリックでも全選択可能

例えばタブ区切りテキストファイルが手元があり、この中からリストファイル中の文字列を含む行を抽出するやり方を示します。Linux (UNIX)のgrepコマンドのようなものであり、perlのハッシュのようなものです。

「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ー。

1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リストファイル(genelist.txt)中のものが含まれる行全体を出力したい場合:

```

in_f1 <- "annotation.txt"
in_f2 <- "genelist.txt"
out_f <- "hogel.txt"
param <- 1

#入力ファイルの読み込み
data <- read.table(in_f1,
keywords <- readLines(in_f2)
dim(data)

#本番
obj <- is.element(as.character(data[,1]), keywords)
out <- data[obj,]
dim(out)

#ファイルに保存
write.table(out, out_f, sep="\t", as.is=TRUE)

```

```

'demo()' と入力すればデモを実行することができます。
'help()' とすればヘルプを表示します。
'help.start()' とすればヘルプのウェブページを開きます。
'q()' と入力すればRを終了します。

> getwd()
[1] "C:/Users/..."
> list.files()
[1] "annotation.txt"
>

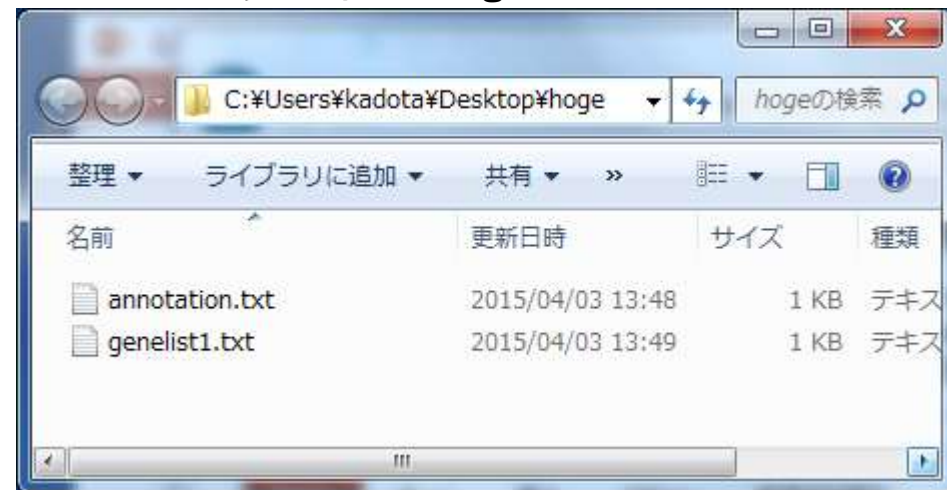
```

①コピー実行後にlist.files()。②出力ファイル名として指定したhoge1.txtが生成されているのがわかる。「list.files()で表示される結果」と「実行後のhogeフォルダの中身」は当然同じ

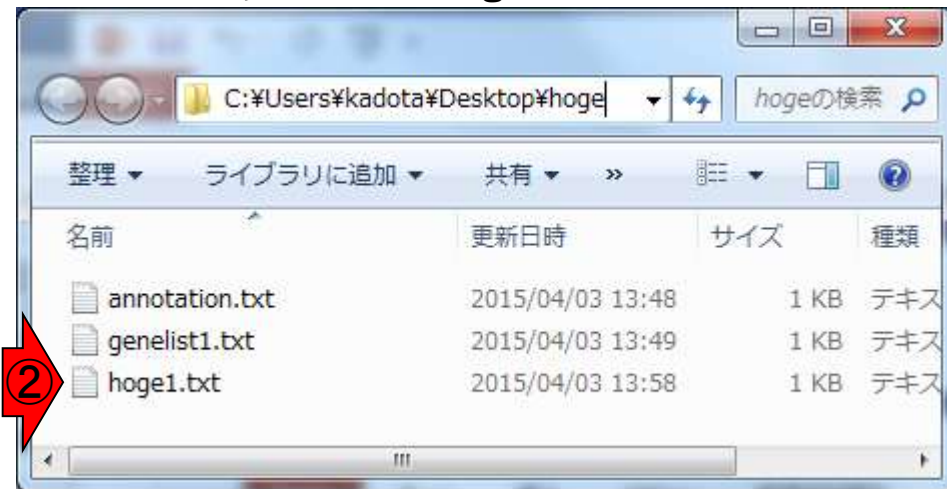
実行結果

```
R Console  
> #本番  
> obj <- is.element(as.character(data[,pa$  
> out <- data[obj,] $  
> dim(out) $  
[1] 3 4  
>  
> #ファイルに保存  
> write.table(①, out_f, sep="\t", appen$  
> list.files()  
[1] "annotation.txt" "genelist1.txt"  
[3] "hoge1.txt"  
> |
```

実行前のhogeフォルダ



実行後のhogeフォルダ



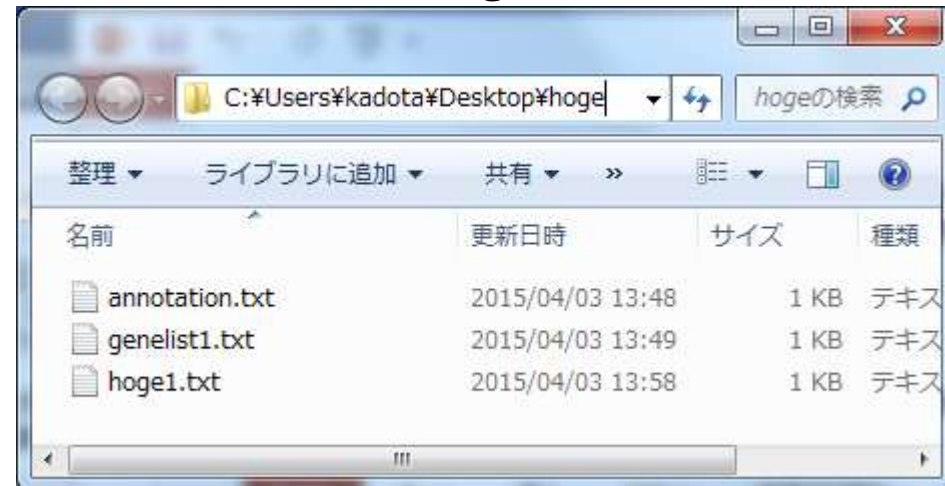
実行結果

①outというオブジェクトの中身をwrite.tableという関数でファイルに出力しています。この場合、出力ファイルhoge1.txtの中身は、Rコンソール画面中でoutと打ち込むことで見られる

```
R Console  
> #ファイルに保存  
> write.table(out, out_f, sep="\t", append=F, quote=F$  
> list.files()  
[1] "annotation.txt" "genelist1.txt" "hoge1.txt"  
> out  
  gene_name  accession  description  subcellular_location  
1    gene1    hoge01    plasma_mem      nuclear  
7    gene7    hoge07    tebasaki        nuclear  
9    gene9    hoge09    nihonshu        nuclear  
> |
```

	A	B	C	D
1	gene_name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene7	hoge07	tebasaki	nuclear
4	gene9	hoge09	nihonshu	nuclear

実行後のhogeフォルダ



色の説明

(Rで)塩基配列解析

(last modified 2017/04/10, since 2010)

このウェブページのR関連部分は、[インストール](#)についての推奨手順 ([Windows2015.04.04版](#)と[Macintosh2015.04.03版](#))に従ってフリーソフトRと必要なパッケージをインストール済みであるという前提で記述しています。初心者の方は[基本的な利用法](#)([Windows2015.04.03版](#)と[Macintosh2015.04.03版](#))で自習してください。本ウェブページを体系的にまとめた[書籍](#)もあります。(2015/04/03)

What's new?

- Galaxyのウェブサイトのリンク先を<http://usegalaxy.org>から<https://galax>
- [日本乳酸菌学会誌](#)のNGS関連連載の[第9回分原稿PDF](#)を公開しました。(2017/03/13) **NEW**
- 「[書籍 | 日本乳酸菌学会誌 | 第7回ロングリードアセンブリ](#)」で `grep -A`の説明が間違っていたので修正しました。「検索文字列の行を含む後ろx行を表示」みたいに書いていましたが、検索文字列の行は含まないようです `m(_)m`(2017/01/26)



コメント

特にやらなくてもいいコマンド
プログラム実行時に目的に応じて変更すべき箇所

- [門田からメール返信をもらえない場合は](#) (last modified 2016/08/23)
- [はじめに](#) (last modified 2015/03/31)
- 参考資料 | [書籍、学会誌](#) (last modified 2017/03/13) **NEW**
- 参考資料 | [講習会、講義、講演資料](#) (last modified 2016/12/07)
- [過去のお知らせ](#) (last modified 2017/04/10) **NEW**
- [インストール](#) | [インストールについて](#) (last modified 2015/11/12)
- インストール | R本体 | [最新版](#) | [Win用](#) (last modified 2015/03/22) 推奨
- インストール | R本体 | [最新版](#) | [Mac用](#) (last modified 2015/04/22) 推奨
- インストール | R本体 | [過去版](#) | [Win用](#) (last modified 2015/03/22)
- インストール | R本体 | [過去版](#) | [Mac用](#) (last modified 2015/03/22)
- インストール | Rパッケージ | [ほぼ全て\(20GB以上?\)](#) (last modified 2015/05/25)
- インストール | Rパッケージ | [必要最小限プラスアルファ\(数GB?\)](#) (last modified 2017/03/13) 推奨 **NEW**
- インストール | Rパッケージ | [必要最小限プラスアルファ\(アグリバイオ居室のみ\)](#) (last modified 2015/06/16)
- インストール | Rパッケージ | [必要最小限\(数GB?\)](#) (last modified 2015/05/25)

応用

このサンプルコードは①1列目でキーワード検索する場合。別のリストファイルを読み込んで4列目で検索したい場合のやり方を示します

1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リストファイル(genelist.txt)中のものが含まれる行全体を出力したい場合:

```

in_f1 <- "annotation.txt" #入力ファイル名を指定してin_f1に格納(アノテーションファイル)
in_f2 <- "genelist.txt"  #入力ファイル名を指定してin_f2に格納(リストファイル)
out_f <- "hoge1.txt"     #出力ファイル名を指定してout_fに格納
param <- 1              #アノテーションファイル中の検索したい列番号を指定

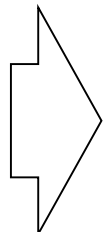
#入力ファイルの読み込み
data <- read.table(in_f1, header=TRUE, sep="\t", quote="")#
keywords <- readLines(in_f2) #in_f2で指定したファイルの各行を読み込み
dim(data) #オブジェクトdataの行数と列数を表示

#本番
obj <- is.element(as.character(data[,param]), keywords)#条件を満たす行のみ抽出した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F)#outの中身を指定したファイル名で保存
    
```

コメント
特にやらなくてもいいコマンド
プログラム実行時に目的に応じて変更すべき箇所

	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

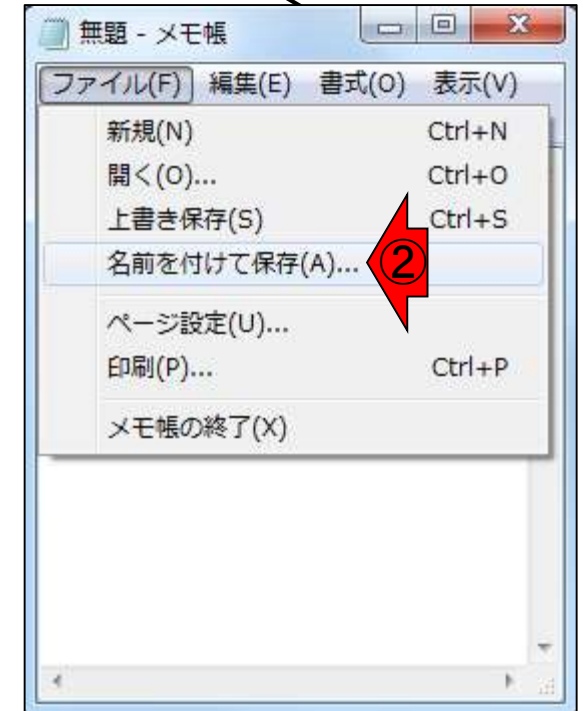
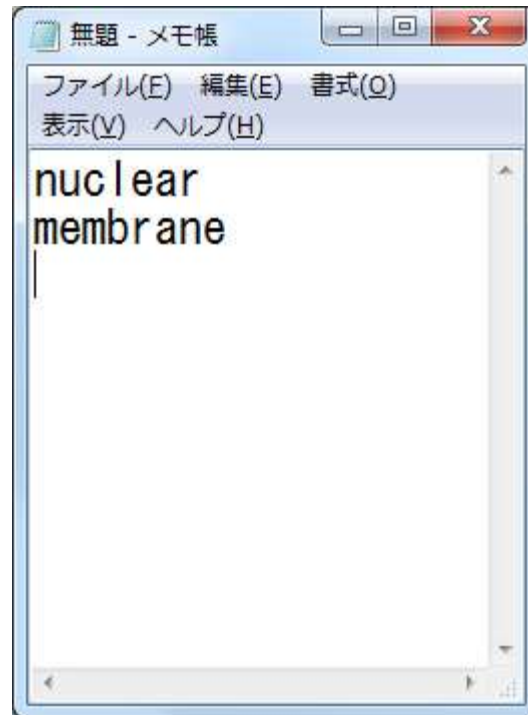
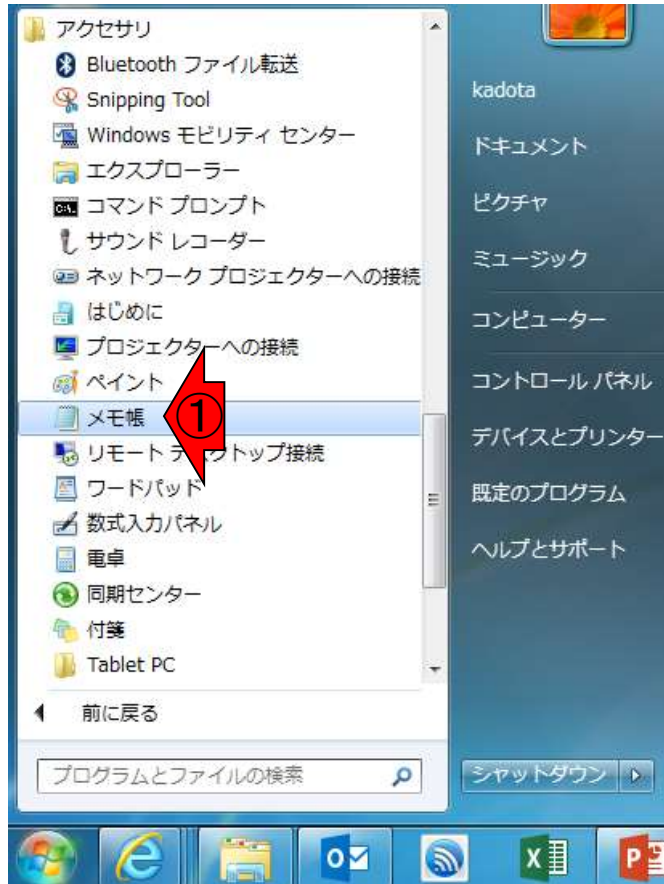


	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

①メモ帳など任意のエディタでリストファイル(list.txt)を作成し、②「デスクトップ - hoge」フォルダ上で保存

解答例

1. 目的のキーワードリストを含むファイルを作成し(例: list.txt)
2. 該当箇所を変更し、Rコンソール画面上でコピー



一連の作業手順を記述したスクリプトを1つのファイルとして保存することをお勧め

解答例

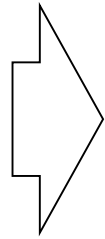
1. 目的のキーワードリストを含むファイルを作成し(例: list.txt)
2. 該当箇所を変更し、Rコンソール画面上でコピー

```
nuclear↓
membrane↓
↓
```

```
run1.txt - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V)
in_f1 <- "annotation.txt"
in_f2 <- "genelist1.txt"
out_f <- "hogel.txt"
param <- 1

#ファイルの読み込み
data <- read.table(in_f1, head
keywords <- readLines(in_f2)
dim(data)

#本番
obj <- is.element(as.character
out <- data[obj,]
dim(out)
write.table(out, out_f, sep="¥
```



```
run1.txt - メモ帳
ファイル(F) 編集(E) 書式(O) 表示(V)
in_f1 <- "annotation.txt"
in_f2 <- "list.txt"
out_f <- "hogel.txt"
param <- 4

#ファイルの読み込み
data <- read.table(in_f1, head
keywords <- readLines(in_f2)
dim(data)

#本番
obj <- is.element(as.character
out <- data[obj,]
dim(out)
write.table(out, out_f, sep="¥
```

Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

警告メッセージ

list.txtファイル作成時に、membraneと打った後に改行を①入れた場合と②入れない場合の挙動の違いを把握し、後学のために警告メッセージの意味を理解しておくとい。この場合は結果には影響していないことがわかる。Rは警告メッセージの記述内容が比較的分かりやすいのでよく読むべし

```

R Console
> in_f1 <- "annotation.txt"
> in_f2 <- "list.txt"
> out_f <- "hogel.txt"
> param <- 4
>
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="\"")
> keywords <- readLines(in_f2)
> dim(data)
[1] 11 4
>
> #本番
> obj <- is.element(as.character(data[,param]), keywords)
> out <- data[obj,]
> dim(out)
[1] 7 4
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", as.is=TRUE)
>

```

nuclear↓
membrane↓
←

①

```

R Console
> in_f1 <- "annotation.txt"
> in_f2 <- "list.txt"
> out_f <- "hogel.txt"
> param <- 4
>
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="\"")
> keywords <- readLines(in_f2)
警告メッセージ:
In readLines(in_f2) : 'list.txt' で不完全な最終行が見つかりました
> dim(data)
[1] 11 4
>
> #本番
> obj <- is.element(as.character(data[,param]), keywords)
> out <- data[obj,]
> dim(out)
[1] 7 4
>
> #ファイルに保存

```

nuclear↓
membrane←

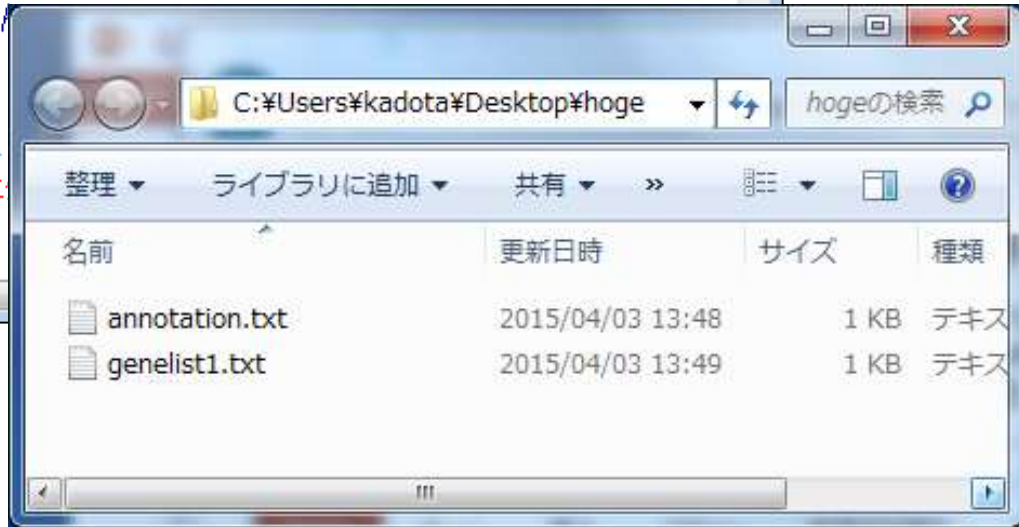
②

ありがちなミス1

作業ディレクトリの変更を忘れていたため、in_f1で指定した最初のファイルの読み込み段階でエラーが出る。つまり、実際に行ったフォルダ中にはannotation.txtというファイルは存在しないということ

```
R Console
> getwd()
[1] "C:/Users/kadota/Documents"
> in_f1 <- "annotation.txt"
> in_f2 <- "genelist1.txt"
> out_f <- "hogel.txt"
> param <- 1
>
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="") #in_f1で指定したフ$
以下にエラー file(file, "rt") : コネクションを開くことができません
追加情報: 警告メッセージ:
In file(file, "rt") :
  ファイル 'annotation.txt' を開くことができません: No such file or directory
> keywords <- readLines(in_f2) #in_f2で指定したファイルの読み込み
以下にエラー file(con, "r") : コネクションを開くことができません
追加情報: 警告メッセージ:
In file(con, "r") :
  ファイル 'genelist1.txt' を開くことができません: No such
> dim(data) #オブジェ
NULL
```

#入力ファイル名を指定してin_f1に格納(\$
#入力ファイル名を指定してin_f2に格納(\$
#出力ファイル名を指定してout_fに格納
#アノテーションファイル中の検索したい\$
#in_f2で指定したファイルの読み込み
#オブジェ



ありがちなミス2

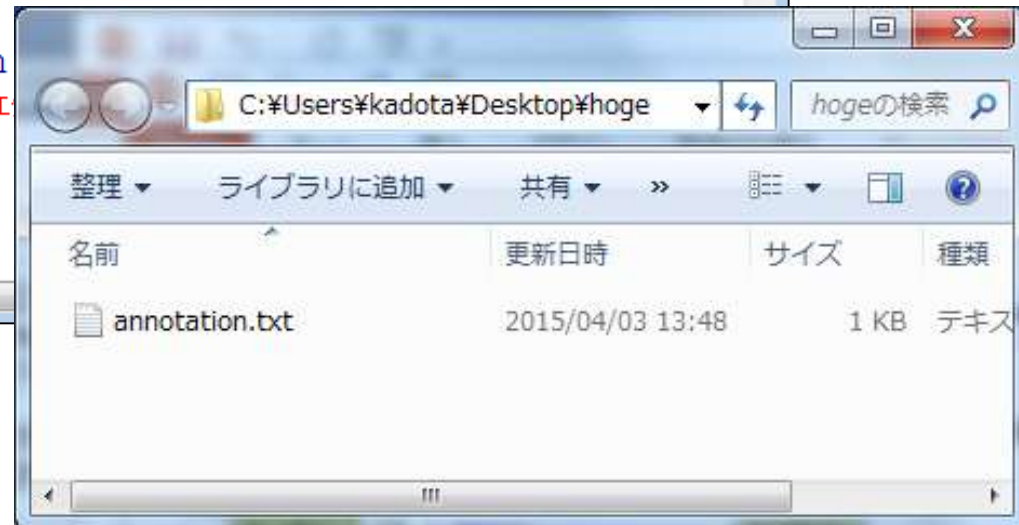
必要な入力ファイルが作業ディレクトリ中に存在しない。この場合、in_f2で指定したgenelist1.txtが存在しないため、その読み込み段階でエラーが出ている。それゆえ、その情報を用いているコマンド部分でエラーが出ている

```
R Console
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "annotation.txt"
> in_f1 <- "annotation.txt"
> in_f2 <- "genelist1.txt"
> out_f <- "hoge1.txt"
> param <- 1
>
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="") #in_f1で指定したフ$
> keywords <- readLines(in_f2) #in_f2で指定したファイルの読み込み
以下にエラー file(con, "r") : コネクションを開くことができません
追加情報: 警告メッセージ:
In file(con, "r") :
  ファイル 'genelist1.txt' を開くことができません: No such
> dim(data) #オブジェ
[1] 11 4
>
> #本番
```

#入力ファイル名を指定してin_f1に格納(\$
#入力ファイル名を指定してin_f2に格納(\$
#出力ファイル名を指定してout_fに格納
#アノテーションファイル中の検索したい\$

#in_f2で指定したファイルの読み込み

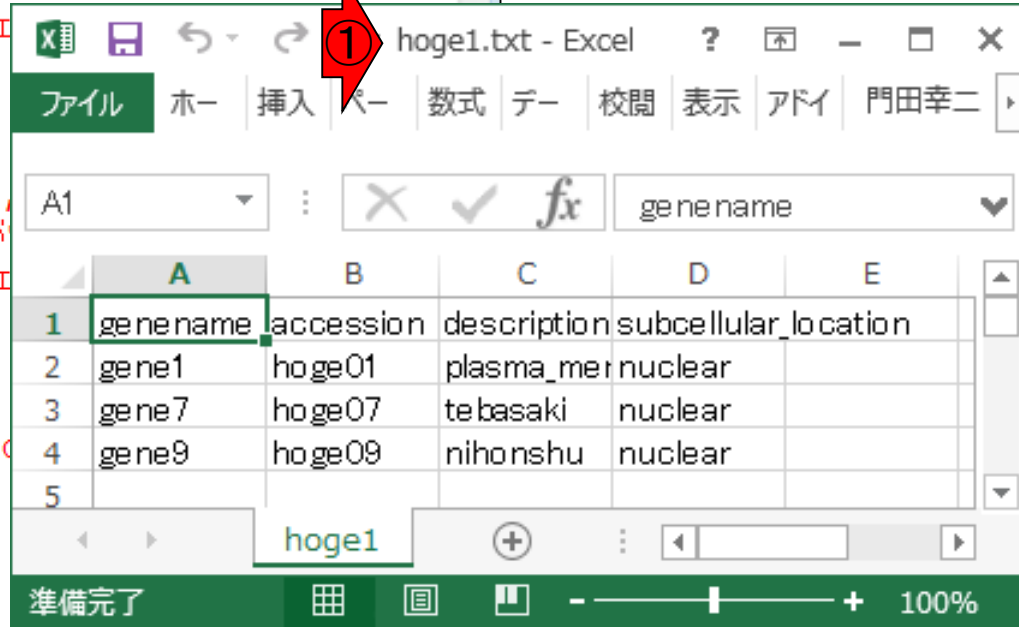
#オブジェ



ありがちなミス3

①出力予定のファイル名(hoge1.txt)と同じものをエクセルなど別のプログラムで開いているため、②最後のwrite.table関数のところでエラーが出る。対処法は、出力ファイル名を変更するか、開いている別のプログラムを閉じる。Macはエクセルで開いていても保存できる

```
R Console
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="")#in_f1$
> keywords <- readLines(in_f2) #in_f2で指定したファイル$
> dim(data) #オブジェ
[1] 11 4
>
> #本番
> obj <- is.element(as.character(data[,param]), keywords) #objが
> out <- data[obj,] #オブジェ
> dim(out)
[1] 3 4
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, col.names=T)
以下にエラー file(file, ifelse(append, "a", "w"))
コネクションを開くことができません
追加情報: 警告メッセージ:
In file(file, ifelse(append, "a", "w")) :
ファイル 'hoge1.txt' を開くことができません: Permission denied
> |
```



ありがちなミス4

①実行スクリプトをコピーする際、②最後の行のところで改行を含ませずに③R Console画面上でペーストしたため、最後のコマンドが実行されない(出力ファイルが生成されない)。これも比較的ありがちなパターンです。コピー後に無意識にリターンキーを押すことを心がけるだけでもよいでしょう

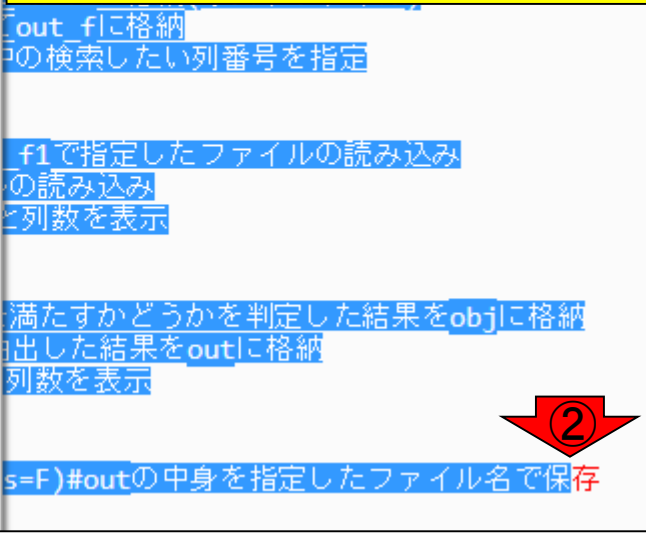
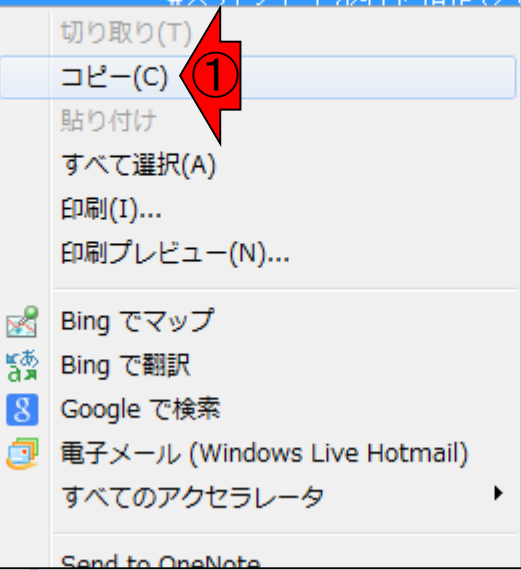
1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リ
体を出力したい場合:

```
in_f1 <- "annotation.txt"
in_f2 <- "genelist1.txt"
out_f <- "hoge1.txt"
param <- 1

#入力ファイルの読み込み
data <- read.table(in_f1, header=T, sep="\t", as.is=T)
keywords <- readLines(in_f2)
dim(data)

#本番
obj <- is.element(as.character(data[,1]), keywords)
out <- data[obj,]
dim(out)

#ファイルに保存
write.table(out, out_f, sep="\t", as.is=T, quote=F, row.names=F) #outの中身を指定したファイル名で保存
```



```
$param]), keywords) #条件を満たすかどうかを判定した結果をobjに格納
$ #objがTRUEとなる行のみ抽出した結果をoutに格納
$ #オブジェクトoutの行数と列数を表示
```

```
$send=F, quote=F, row.names=F) #outの中身を指定したファイル名で保
```



Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

コード内部の説明

イントロ | 一般 | [任意のキーワードを含む行を抽出\(基礎\)](#)

例えばタブ区切りテキストファイルが手元があり、この中からリストファイル中の文字列を含む行を抽出するやり方を示します。Linux (UNIX)の grep コマンドのようなものであり、perlのハッシュのようなものです。

「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ー。

1. 目的のタブ区切りテキストファイル([annotation.txt](#))中の第1列目をキーとして、リストファイル([genelist1.txt](#))中のものが含まれる行全体を出力したい場合:

```
in_f1 <- "annotation.txt"
in_f2 <- "genelist1.txt"
out_f <- "hogel.txt"
param <- 1
```

```
#入力ファイルの読み込み
data <- read.table(in_f1, header=TRUE, sep="\t", quote="\"
keywords <- readLines(in_f2)
dim(data)
```

```
#本番
obj <- is.element(as.character(
out <- data[obj,]
```

```
R Console
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "annotation.txt" "genelist1.txt"
> in_f1 <- "annotation.txt" #入力ファイル名を$
> in_f2 <- "genelist1.txt" #入力ファイル名を$
> out_f <- "hogel.txt" #出力ファイル名を$
> param <- 1 #アノテーションフ$
>
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="\"
> keywords <- readLines(in_f2) #in_f2で指定した$
> dim(data) #オブジェクトdata$
[1] 11 4
> |
```

読み込み

- ① in_f1で指定したファイルを読み込め
- ② 読み込むファイルの最初の行はヘッダ一部分
- ③ ファイルの区切り文字はタブです
- ④ 読み込んだ結果をdataという名前で取り扱う

#入力ファイル名を指定してin_f2に格納
 #出力ファイル名を指定してout_f1に格納
 #アノテーションファイル中の検索したい

```
in_f1 <- "annotation.txt"
in_f2 <- "genelist1.txt"
out_f <- "hoge1.txt"
param <- 1
```

#出力ファイルの読み込み

```
data <- read.table(in_f1, header=TRUE,
keywords <- readLines(in_f2)
dim(data)
```

#in_f1で指定した
 #in_f2で指定したファイルの読み込み

	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

dataと打ってリターン。入力ファイルの中身を正しく読み込めていることがわかる。②header=TRUEとしているので、③このように見えて列名として認識される

行列data

```
R Console  
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="$"  
> keywords <- readLines(in_f2) #in_f2で指定した$  
> dim(data) #オブジェクトdata$  
[1] 11 4  
> data  
  gene name  accession  description  subcellular_location  
1  gene1      hoge01    plasma_mem  nuclear  
2  gene2      hoge02      hohinu     membrane  
3  gene3      hoge03    agribio    endoplasmic  
4  gene4      hoge04    genesis    endoplasmic  
5  gene5      hoge05      kamo      membrane  
6  gene6      hoge06    netteba    humei  
7  gene7      hoge07    tebasaki   nuclear  
8  gene8      hoge08      biiru     nuclear  
9  gene9      hoge09    nihonshu   nuclear  
10 gene10     hoge10    agene1     membrane  
11 gene11     hoge11    iyaaaa     endoplasmic  
> |
```

	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

dimで行数と列数を表示

- ①オブジェクトdataの行数と列数は11と4。
- ②ウェブページ中の表記が灰色なのは、特にやらなくてもいいコマンドだから

```
in_f1 <- "annotation.txt"  
in_f2 <- "genelist1.txt"  
out_f <- "hoge1.txt"  
param <- 1
```

#入力ファイル名を指定してin_f1に格納
#入力ファイル名を指定してin_f2に格納
#出力ファイル名を指定してout_f1に格納
#アノテーションファイル中の検索したし

```
#入力ファイルの読み込み  
data <- read.table(in_f1,  
keywords <- readLines(in_f2)  
dim(data)
```



```
#本番  
obj <- is.element(as.char  
out <- data[obj,]  
dim(out)
```

```
#ファイルに保存  
write.table(out, out_f, s
```

```
R Console  
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="$"  
> keywords <- readLines(in_f2) #in_f2で指定した$  
> dim(data) #オブジェクトdata$  
[1] 11 4  
> data  
  genename accession description subcellular_location  
1     gene1   hoge01   plasma_mem          nuclear  
2     gene2   hoge02     hohinu            membrane  
3     gene3   hoge03   agribio            endoplasmic  
4     gene4   hoge04   genesis            endoplasmic  
5     gene5   hoge05     kamo              membrane  
6     gene6   hoge06   netteba           humei  
7     gene7   hoge07   tebasaki          nuclear  
8     gene8   hoge08     biiru             nuclear  
9     gene9   hoge09   nihonshu          nuclear  
10    gene10   hoge10   agen1             membrane  
11    gene11   hoge11   iyaaaa            endoplasmic  
> |
```



行列の要素へのアクセス

行列dataの要素へのアクセスは[行, 列].

①humeiは、読み込み元ファイルのannotation.txt中では7行×4列目だが、
②1行目をヘッダー行としているので③
6行×4列目とする必要がある。利用例は、ファイル読み込み時に「x行×y列目に不具合がある」のようなエラーが出た時のトラブルシューティングなど

```
R Console
[1] 11 4
> data
  gene_name accession description subcellular_location
1 gene1      hoge01  plasma_mem      nuclear
2 gene2      hoge02    hohinu          membrane
3 gene3      hoge03    agriblio        endoplasmic
4 gene4      hoge04    genesis         endoplasmic
5 gene5      hoge05      kamo            membrane
6 gene6      hoge06    netteba
7 gene7      hoge07    tebasaki
8 gene8      hoge08      biiru
9 gene9      hoge09    nihonshu
10 gene10     hoge10     agene1
11 gene11     hoge11     iyaaaa
> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
> |
```

	A	B	C	D
1	gene_name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic



Tips: 上下左右の矢印キー

①上矢印キーを押すと、②直前に打ったコマンドが表示される。最初から全部打ち直すのではなく、上下左右の矢印キーを有効に利用し最小限の労力で打つべし!

```
R Console
[1] 1
> data
  geneName accession description subcellular_location
1   gene1     hoge01  plasma_mem          nuclear
2   gene2     hoge02    hohinu             membrane
3   gene3     hoge03    agribio            endoplasmic
4   gene4     hoge04    genesis            endoplasmic
5   gene5     hoge05      kamo               membrane
6   gene6     hoge06    netteba            humei
7   gene7     hoge07    tebasaki           nuclear
8   gene8     hoge08      biiru              nuclear
9   gene9     hoge09    nihonshu
10  gene10    hoge10    agene1
11  gene11    hoge11    iyaaaa

> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
> data[6, 4]
```



行列の要素へのアクセス

行列dataの要素へのアクセスは[行, 列]。
①2行目の情報のみ抽出。読み込み時にhead=TRUEとしていたので、②ヘッダ行がついている

```
R Console  
1 gene1 hoge01 plasma_mem  
2 gene2 hoge02 hohinu  
3 gene3 hoge03 agribio  
4 gene4 hoge04 genesis  
5 gene5 hoge05 kamo  
6 gene6 hoge06 netteba  
7 gene7 hoge07 tebasaki  
8 gene8 hoge08 biiru  
9 gene9 hoge09 nihonshu  
10 gene10 hoge10 agene1  
11 gene11 hoge11 iyaaaa  
> data[6, 4]  
[1] humei  
Levels: endoplasmic humei membrane nuclear  
> data[2, ]  
  gene name accession description subcellular location  
2 gene2 hoge02 hohinu membrane  
> |
```

	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic



行列の要素へのアクセス

```
R Console  
5 gene5 hoge05 kamo  
6 gene6 hoge06 netteba  
7 gene7 hoge07 tebasaki  
8 gene8 hoge08 biiru  
9 gene9 hoge09 nihonshu  
10 gene10 hoge10 agene1  
11 gene11 hoge11 iyaaaa  
> data[6, 4]  
[1] humei  
Levels: endoplasmic humei membrane nuclear  
> data[2, ]  
 gene name accession description subcellular_lo  
2 gene2 ① hoge02 hohinu me  
> data[, 2]  
[1] hoge01 hoge02 hoge03 hoge04 hoge05 hoge06  
[8] hoge08 hoge09 hoge10 hoge11  
11 Levels: hoge01 hoge02 hoge03 hoge04 hoge05 ... hoge11  
> |
```

	A	B	C	D
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

行列の要素へのアクセス

行列dataの要素へのアクセスは[行, 列]。① param列目の情報のみ抽出。② paramには1という数値が代入されていたのでこうなる

```
R Console
11 gene11 hoge11 iyaaaa end
> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
> data[2, ]
 gene1 gene2 gene3 gene4 gene5 gene6
2 gene2 hoge02 hohinu
> data[, 2]
[1] hoge01 hoge02 hoge03 hoge04 hoge05 hoge06
[8] hoge08 hoge09 hoge10 hoge11
11 Levels: hoge01 hoge02 hoge03 hoge04 hoge05
> data[, param]
[1] gene1 gene2 gene3 gene4 gene5 gene6
[8] gene8 gene9 gene10 gene11
11 Levels: gene1 gene10 gene11 gene2 gene3 gene4
> param
[1] 1
> |
```

	A	B	C	D
1	gene1	hoge01	plasma_mem	nuclear
2	gene2	hoge02	hohinu	membrane
3	gene3	hoge03	agribio	endoplasmic
4	gene4	hoge04	genesis	endoplasmic
5	gene5	hoge05	kamo	membrane
6	gene6	hoge06	netteba	humei
7	gene7	hoge07	tebasaki	nuclear
8	gene8	hoge08	biiru	nuclear
9	gene9	hoge09	nihonshu	nuclear
10	gene10	hoge10	agene1	membrane
11	gene11	hoge11	iyaaaa	endoplasmic

```
in_f1 <- "annotation.txt"
in_f2 <- "genelist1.txt"
out_f <- "hoge1.txt"
param <- 1
```



Tips: 関数とオプション

行列dataの最初の数行を表示したい場合は、head関数を利用。①n=3というオプションを利用すると最初の3行分のみ表示。関数ごとに様々なオプションを利用可能

R Console

```
> head(data)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem             nuclear
2  gene2   hoge02   hohinu                membrane
3  gene3   hoge03   agribio                endoplasmic
4  gene4   hoge04   genesis                endoplasmic
5  gene5   hoge05   kamo                   membrane
6  gene6   hoge06   netteba                 humei

> head(data, n=3)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem             nuclear
2  gene2   hoge02   hohinu                membrane
3  gene3   hoge03   agribio                endoplasmic

> head(data, n=1)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem             nuclear

> |
```

Tips: タブ補完

列番号を指定する以外にも特定の列を表示するやり方がある。head=TRUEで入力ファイルを読み込むと、列の名前を利用することができる。①subcellular_location列の情報を抽出したい場合は、②「data\$su」くらいまで打ち込んでから③Tabキーを押すと、

```
R Console
> head(data, n=3)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem      nuclear
2  gene2   hoge02   hohinu          membrane
3  gene3   hoge03   agriblio        endoplasmic
> head(data, n=1)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem      nuclear
> data[, 4]
[1] nuclear      membrane      endoplasmic  endoplasmic
[5] membrane     humei         nuclear       nuclear
[9] nuclear      membrane      endoplasmic
Levels: endoplasmic humei membrane nuclear
> data$su|
```



annotation.txt

	A	B	C	①
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

Tips: タブ補完

列名中のsuからはじまる文字列を補完して表示してくれる。「Tabキーを用いた補完機能」という意味で「タブ補完」という。このテクニックはLinuxでも利用可能

```
R Console
> head(data, n=3)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem      nuclear
2  gene2   hoge02    hohinu          membrane
3  gene3   hoge03    agriblio        endoplasmic
> head(data, n=1)
  gene accession description subcellular_location
1  gene1   hoge01  plasma_mem      nuclear
> data[, 4]
[1] nuclear      membrane      endoplasmic  endoplasmic
[5] membrane     humei         nuclear      nuclear
[9] nuclear      membrane      endoplasmic
Levels: endoplasmic humei membrane nuclear
> data$subcellular_location
[1] nuclear      membrane      endoplasmic  endoplasmic
[5] membrane     humei         nuclear      nuclear
[9] nuclear      membrane      endoplasmic
Levels: endoplasmic humei membrane nuclear
> |
```

annotation.txt

	A	B	C	①
1	gene name	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humai
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

Tips: table関数

①hogeの中身は②subcellular_location列の情報からなる。③table関数は、ベクトル中の要素ごとの出現回数を返す。「NGSデータ中の特定のリードの出現回数」や、「アノテーションファイル中の染色体ごとの遺伝子数」など、様々な局面で利用可能

```

R R
> hoge <- data$subcellular_location
> hoge
[1] nuclear      membrane      endoplasmic  endoplasmic
[5] membrane      humei         nuclear       nuclear
[9] nuclear      membrane      endoplasmic
Levels: endoplasmic humei membrane nuclear
> table(hoge)
hoge
endoplasmic      humei      membrane      nuclear
              3              1              3              4
> table(data$subcellular_location)

endoplasmic      humei      membrane      nuclear
              3              1              3              4
> |

```

annotation.txt

	A	B	C	D
1	gene name	accession	description	subcellular location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

sort関数と併用することで全体像を俯瞰可能。例えば①nuclearに局在する遺伝子数が最も多く4個であった、などが簡単にわかる

Tips: ソート

```
R Console
> sort(table(hoge))
hoge
      humei endoplasmic  membrane  nuclear
      1         3         3         4
> sort(table(hoge), decreasing=T)
hoge
      nuclear endoplasmic  membrane  humei
      4         3         3         1
> hoge2 <- table(hoge)
> sort(hoge2, decreasing=T)
hoge
      nuclear endoplasmic  membrane  humei
      4         3         3         1
> |
```



	A	B	C	D
1	gene name	accession	description	subcellular location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

Tips: is.element関数

is.element関数は、hogeベクトルに対して、“nuclear”の文字が存在する場所をTRUE、存在しない場所をFALSEとして返す。①as.character関数は、文字列ベクトルとして取り扱いたい場合に利用

```
R Console
> hoge
[1] nuclear      membrane      endoplasmic  endoplasmic
[5] membrane     humei         nuclear      nuclear
[9] nuclear      membrane      endoplasmic
Levels: endoplasmic humei membrane nuclear
> is.element(hoge, "nuclear")
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
[10] FALSE FALSE
①
> as.character(hoge)
[1] "nuclear"      "membrane"     "endoplasmic"
[4] "endoplasmic" "membrane"     "humei"
[7] "nuclear"      "nuclear"      "nuclear"
[10] "membrane"     "endoplasmic"
> is.element(as.character(hoge), "nuclear")
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE
[10] FALSE FALSE
> |
```

	A	B	C	D
1	gene name	accession	description	subcellular location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic

Tips: “二重クォーテーション”

二重クォーテーションが自動で変更されるエディタは非推奨です。日本語の二重クォーテーションもだめです。Microsoft WordやPDFファイル中のコードのコピペ時によくハマります

R Console

```
> is.element(hoge, "nuclear")
[1] TRUE FALSE FALSE FALSE FALSE FALSE
[7] TRUE TRUE TRUE FALSE FALSE
> is.element(hoge, "nuclear")
> is.element(hoge, "nuclear")
> is.element(hoge, "nuclear")
[1] TRUE FALSE FALSE FALSE FALSE FALSE
[7] TRUE TRUE TRUE FALSE FALSE
> |
```

目的は、数万～数百万行からなるファイルを読み込んで特定のキーワードを含む行のみ取り出すテクニックの習得。①is.elementやas.characterはここで利用

目的をおさらい

1. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、リストファイル(genelist.txt)中のものが含まれる行全体を出力したい場合:

```

in_f1 <- "annotation.txt"
in_f2 <- "genelist.txt"
out_f <- "hogel.txt"
param <- 1

#入力ファイルの読み込み
data <- read.table(in_f1, header=TRUE, sep="\t", quote="")
keywords <- readLines(in_f2)
dim(data)

#本番
obj <- is.element(as.character(data[,param]), keywords)
out <- data[obj,]
dim(out)

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

```

#入力ファイル名を指定してin_f1に格納(アノテーションファイル)
 #入力ファイル名を指定してin_f2に格納(リストファイル)
 #出力ファイル名を指定してout_fに格納
 #アノテーションファイル中の検索したい列番号を指定
 #in_f1で指定したファイルの読み込み
 #in_f2で指定したファイルの読み込み
 #オブジェクトdataの行数と列数を表示
 ①件を満たすかどうかを判定した結果
 #objがTRUEとなる行のみ抽出した結果をoutに格納
 #オブジェクトoutの行数と列数を表示
 #outの中身を指定したファイルに保存

①論理値ベクトルobjを用いて、②TRUEの要素に対応する行を抽出

コード内部の説明

#本番

```
obj <- is.element(as.character(data[,param]), keywords)#条件を満たすかどうかを判定した
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示
```

入力2:リストファイル
(genelist1.txt)

	A
1	gene1
2	gene7
3	gene9

```
R Console
> data[, param]
[1] gene1 gene2 gene3 gene4 gene5 gene6 gene7
[8] gene8 gene9 gene10 gene11
11 Levels: gene1 gene10 gene11 gene2 gene3 gene4 ... gene9
> keywords
[1] "gene1" "gene7" "gene9"
> is.element(as.character(data[,param]), keywords)
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE
[10] FALSE FALSE
> obj <- is.element(as.character(data[,param]), keywords)
> obj
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE
[10] FALSE FALSE
①
②
> data[obj,]
  genename accession description subcellular_location
1  gene1     hoge01  plasma_mem             nuclear
7  gene7     hoge07    tebasaki             nuclear
9  gene9     hoge09    nihonshu             nuclear
> |
```

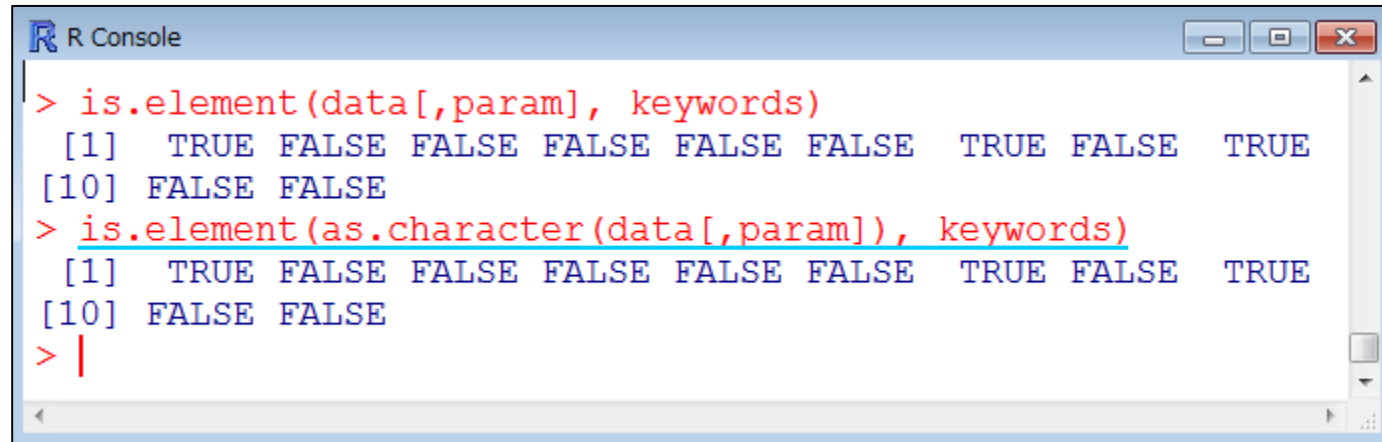
Tips: 昔話

コード作成当時は`as.character`関数を用いてデータの型を文字列ベクトルに揃えていた。少なくとも現在(R ver. 3.1.3以降)は、この関数がなくても大丈夫なようだ。同じ関数でもバージョンによって挙動が異なる(バージョンの違いの一例)

#本番

```
obj <- is.element(as.character(data[,param]), keywords)
out <- data[obj,]
dim(out)
```

#objがTRUEとなる行の抽出した結果とoutの行数
#オブジェクトoutの行数と列数を表示



```
R Console
> is.element(data[,param], keywords)
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE
[10] FALSE FALSE
> is.element(as.character(data[,param]), keywords)
[1] TRUE FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE
[10] FALSE FALSE
> |
```

1. 目的のタブ区切りテキストファイル([annotation.txt](#))中の第1列目をキーとして、リストファイル([genelist1.txt](#))中のものが含まれる行全体を出力したい場合:

```
in_f1 <- "annotation.txt" #入力ファイル名を指定してin_f1に格納(アノテーションファイル)
in_f2 <- "genelist1.txt" #入力ファイル名を指定してin_f2に格納(リストファイル)
out_f <- "hoge1.txt" #出力ファイル名を指定してout_fに格納
param <- 1 #アノテーションファイル中の検索したい列番号を指定
```

#入力ファイルの読み込み

```
data <- read.table(in_f1, header=TRUE, sep="\t", quote="") #in_f1で指定したファイルの読み込み
keywords <- readlines(in_f2) #in_f2で指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示
```

#本番

```
obj <- is.element(as.character(data[,param]), keywords) #条件を満たすかどうかを判定した結果をobjに格納
```

out <-

dim(out)

#ファイ

write.

genelist1.txt

	A
1	gene1
2	gene7
3	gene9

12. 目的のタブ区切りテキストファイル([annotation.txt](#))中の第1列目をキーとして、param2で指定した文字列が含まれる行全体を出力したい場合:

```
in_f <- "annotation.txt" #入力ファイル名を指定してin_fに格納(アノテーションファイル)
out_f <- "hoge12.txt" #出力ファイル名を指定してout_fに格納
param1 <- 1 #アノテーションファイル中の検索したい列番号を指定
param2 <- c("gene1", "gene7", "gene9") #検索したい文字列を指定
```

#入力ファイルの読み込み

```
data <- read.table(in_f, header=TRUE, sep="\t", quote="") #in_f1で指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示
```

#本番

```
obj <- is.element(as.character(data[,param1]), param2) #条件を満たすかどうかを判定した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示
```

#ファイルに保存

```
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F) #outの中身を指定したファイル名で保存
```

12. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキーとして、param2で指定

ヘッダー行が①ある場合(例題12)

```

in_f <- "annotation.txt" #入力ファイル名を指定してin_fに格納(アノテーションファイル)
out_f <- "hoge12.txt" #出力ファイル名を指定してout_fに格納
param1 <- 1 #アノテーションファイル中の検索したい列番号を指定
param2 <- c("gene1", "gene7", "gene9") #検索したい文字列を指定

#入力ファイルの読み込み
data <- read.table(in_f, header=TRUE, sep="\t", quote="") #in_fで指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示

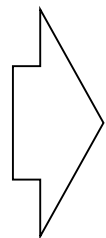
#本番
obj <- is.element(as.character(data[,param1]), param2) #条件を満たすかどうかを判定した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F) #outの中身を指定したファイル名で保存
    
```



入力: annotation.txt

	A	B	C	D
1	genename	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08	biiru	nuclear
10	gene9	hoge09	nihonshu	nuclear
11	gene10	hoge10	agene1	membrane
12	gene11	hoge11	iyaaaa	endoplasmic



出力: hoge12.txt

	A	B	C	D
1	genename	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene7	hoge07	tebasaki	nuclear
4	gene9	hoge09	nihonshu	nuclear

13. 目的のタブ区切りテキストファイル(annotation2.txt)中の第1列目をキーとして、param2

入力ファイル中にヘッダー行がない場合の読み込み例です。

ヘッダー行が②ない場合(例題13)。③
出力時にヘッダー部分を表示させない

```
in_f <- "annotation2.txt" #入力ファイル名を指定してin_fに格納(ファイル名)
out_f <- "hoge13.txt" #出力ファイル名を指定してout_fに格納
param1 <- 1 #アノテーションファイル中の検索したい列番号を指定
param2 <- c("gene1", "gene7", "gene9") #検索したい文字列を指定

#入力ファイルの読み込み
data <- read.table(in_f, header=F, sep="\t", quote="") #in_fで指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示

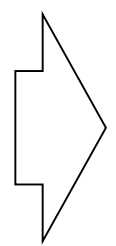
#本番
obj <- is.element(as.character(data[,param1]), param2) #条件を満たすかどうかを判定した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示

#ファイルに保存
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F, col.names=F) #outの中身を指定したファイル名で保存
```



入力: annotation2.txt

	A	B	C	D
1	gene1	hoge01	plasma_mem	nuclear
2	gene2	hoge02	hohinu	membrane
3	gene3	hoge03	agribio	endoplasmic
4	gene4	hoge04	genesis	endoplasmic
5	gene5	hoge05	kamo	membrane
6	gene6	hoge06	netteba	humei
7	gene7	hoge07	tebasaki	nuclear
8	gene8	hoge08	biiru	nuclear
9	gene9	hoge09	nihonshu	nuclear
10	gene10	hoge10	agene1	membrane
11	gene11	hoge11	iyaaaa	endoplasmic



出力: hoge13.txt

	A	B	C	D
1	gene1	hoge01	plasma_mem	nuclear
2	gene7	hoge07	tebasaki	nuclear
3	gene9	hoge09	nihonshu	nuclear

12. 目的のタブ区切りテキストファイル(annotation.txt)中の第1列目をキー

```
in_f <- "annotation.txt" #入力ファイル名を
out_f <- "hoge12.txt" #出力ファイル名を
param1 <- 1 #アンテーションフ
param2 <- c("gene1", "gene7", "gene9") #検索したい文字列
```

#入力ファイルの読み込み

```
data <- read.table(in_f, header=TRUE, sep="\t", quote="") #in_fで指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示
```



#本番

```
obj <- is.element(as.character(data[,param1]), param2) #条件を満たすかどうかを判定した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示
```

#ファイルに保存

```
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F) #outの中身を指定したファイル名で保存
```

ヘッダ一行が①ある場合(例題12)と②ない場合(例題13)の主な違い。③ヘッダ一行がない場合は出力ファイルにもヘッダ一行は必要ないのでそう明記しているだけ

13. 目的のタブ区切りテキストファイル(annotation2.txt)中の第1列目をキーとして、param2で指定した文字列が含まれる行全体を出力したい場合:

入力ファイル中にヘッダ一行がない場合の読み込み例です。

```
in_f <- "annotation2.txt" #入力ファイル名を指定してin_fに格納(アンテーションファイル)
out_f <- "hoge13.txt" #出力ファイル名を指定してout_fに格納
param1 <- 1 #アンテーションファイル中の検索したい列番号を指定
param2 <- c("gene1", "gene7", "gene9") #検索したい文字列を指定
```

#入力ファイルの読み込み

```
data <- read.table(in_f, header=F, sep="\t", quote="") #in_fで指定したファイルの読み込み
dim(data) #オブジェクトdataの行数と列数を表示
```



#本番

```
obj <- is.element(as.character(data[,param1]), param2) #条件を満たすかどうかを判定した結果をobjに格納
out <- data[obj,] #objがTRUEとなる行のみ抽出した結果をoutに格納
dim(out) #オブジェクトoutの行数と列数を表示
```

#ファイルに保存

```
write.table(out, out_f, sep="\t", append=F, quote=F, row.names=F, col.names=F) #outの中身を指定したファイル名で保存
```



Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

FASTA形式

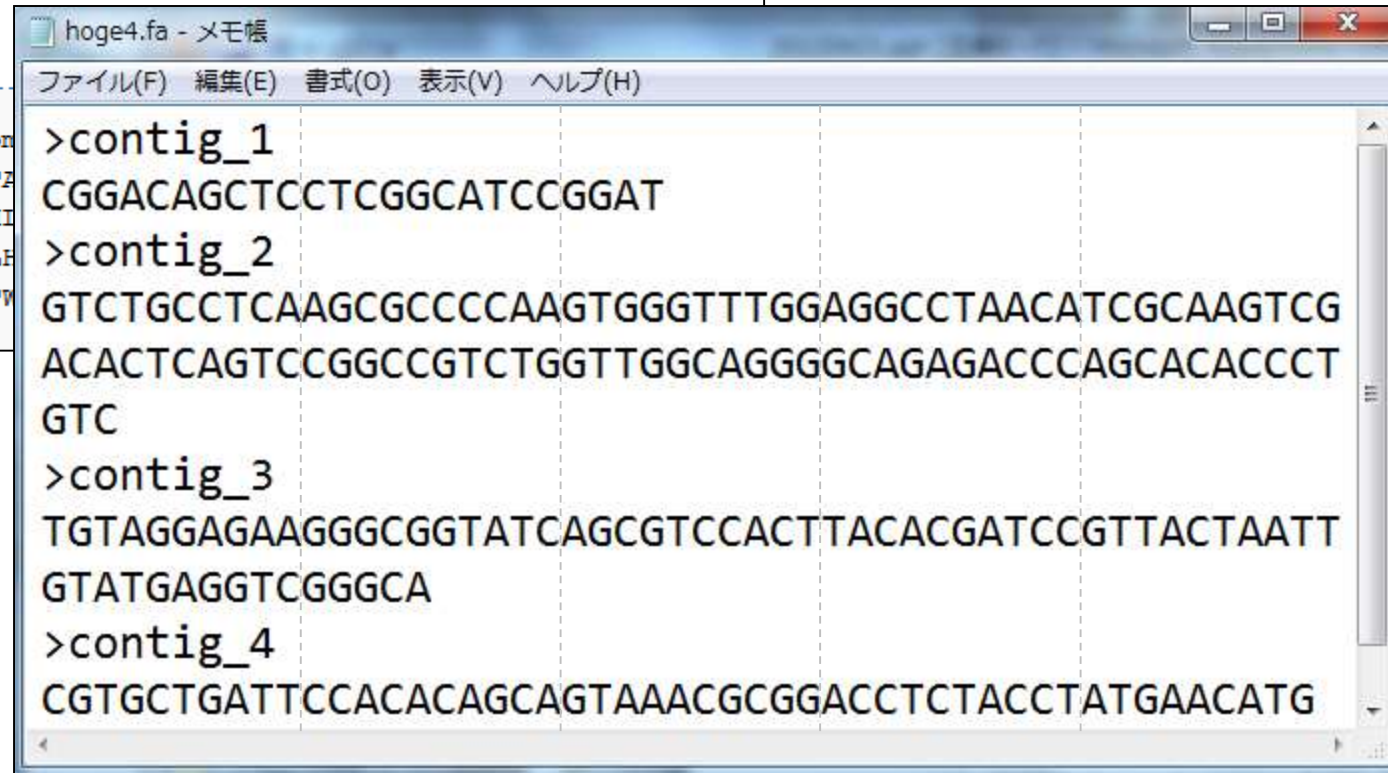
Rでmulti-FASTAファイルを読み込んで自在に解析できます。ゲノム配列解析≒FASTA形式ファイルの解析。ここでは全体像を完全に把握すべくhoge4.faファイル(ダウンロードは後程)を仮想ゲノム配列ファイルとして取り扱う

FASTAフォーマット [編集]

FASTAでは、シーケンスデータの記述形式としてFASTAフォーマットという形式を使う。FASTAフォーマットはブレンテキストである。1つのシーケンスのデータは、">"で始まる1行のヘッダ行と、2行目以降の実際のシーケンス文字列で構成される。ヘッダ行では、">"の次にシーケンスデータを識別するための文字列を記述し、続けてそのシーケンスデータを説明する文字列を記述する(両方とも省略してよい)。ヘッダ行の">"と識別文字列の間にスペースを入れてはいけない。FASTAフォーマットの全ての行は、80文字未満とすることが推奨される。">"で始まる別の行が出現すると、そこでシーケンスデータが区切れ、別のシーケンスデータが始まる。

FASTA ファイルフォーマットの例を示す。

```
>gi|5524211|gb|AAD44166.1| cytochrom  
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLI  
EWIWWGGSVDKATLNRFFAFHFILPFTMVALAGVHI  
LLILILLLLLLLALLSPDMLGDPDNHMPADPLNTPLH  
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTV  
IENY
```



```
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
```

①実際のゲノム配列はここからも取得可能。Rで染色体ごとの配列長やGC含量の計算ができる

ゲノム配列

(Rで)塩基配列解析

(last modified 2017/04/11, since 2010)

このウェブページはRと必要
(Windows2015/2015/04/03)

- ・ [イントロ](#) | [一般](#) | [Tips](#) | [任意の拡張子でファイルを保存](#) (last modified 2013/09/26)
- ・ [イントロ](#) | [一般](#) | [Tips](#) | [拡張子は同じで任意の文字を追加して保存](#) (last modified 2013/09/26)
- ・ [イントロ](#) | [一般](#) | [配列取得](#) | [ゲノム配列](#) | [公共DBから](#) (last modified 2017/04/11) **NEW**
- ・ [イントロ](#) | [一般](#) | [配列取得](#) | [ゲノム配列](#) | [BSgenome](#) (last modified 2015/04/22)
- ・ [イントロ](#) | [一般](#) | [配列取得](#) | [ゲノム配列](#) | [公共DBから](#) (last modified 2017/04/11) **NEW**



What's new?

- ・ [Galaxyのウ](#)
- ・ [日本乳酸菌](#)
- ・ [「書籍」日本含む後ろxi](#)
- ・ [門田から](#)

[イントロ](#) | [一般](#) | [配列取得](#) | [ゲノム配列](#) | [公共DBから](#) **NEW**

- ・ [UCSCの Sequence and Annotation Downloads](#) (Tyner et al., *Nucleic Acids Res.*, 2017)
 - [ヒト; Human \(H.sapiens\)](#)
 - [ラット; Rat \(R.norvegicus\)](#)
 - [ネコ; Cat \(F.catus\)](#)
 - [ウサギ; Rabbit \(O.cuniculus\)](#)
 - [ニワトリ; Chicken \(G.gallus\)](#)
 - [イヌ; Dog \(C.familiaris\)](#)
 - [ウマ; Horse \(E.caballus\)](#)
 - ...
- ・ [Helix Systems Scientific Databases](#) (アップデートの日付順になっている。RefSeqやESTなど様々なデータベースを一度にみられる)
- ・ イネ: [RAP-DB](#) (Sakai et al., *Plant Cell Physiol.*, 2013)
 - 「[ダウンロード](#)」-「Genome assembliesのところの [Download](#)」。IRGSP-1.0_genome.fasta.gz (116MB程度)の圧縮ファイル。
- ・ シロイヌナズナ: [The Arabidopsis Information Resource \(TAIR\)](#) (Lamesch et al., *Nucleic Acids Res.*, 2012)
 - 「[ダウンロード](#)」-「[Genes](#)」-「[TAIR10 genome release](#)」-「[TAIR10 chromosome files](#)」の [TAIR10 chr_all.fas](#) (120MB程度)
- ・ [Ensembl Genomes](#) (Yates et al., *Nucleic Acids Res.*, 2016)
 - [バクテリア \(Bacteria\)](#)
 - [乳酸菌 \(Lactobacillus casei 12A\)](#)
 - [乳酸菌 \(Lactobacillus casei A2-362\)](#)
 - [乳酸菌 \(Lactobacillus casei BL23\)](#)
 - [菌類 \(Fungi\)](#)
 - [後生動物 \(Metazoa\)](#)

基本情報取得

multi-FASTAファイルを読み込んで、トータルの配列長、染色体数(コンティグ数)、配列長の平均、中央値、最大値、最小値、N50、GC含量を計算した結果を返すコードを実行してみよう

入力: `hoge4.fa`

```
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
GTCTGCCTCAAGCGCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
```

出力: `hoge1.txt`

Total length (bp)	241
Number of contigs	4
Average length	60.25
Median length	57
Max length	103
Min length	24
N50	65
GC content	0.577



基本情報取得

①「... | FASTA形式 | 基本情報を取得」。②
コードの最初のほうに入力ファイルと出力ファイル
を記述するので、コピペ実行結果としてどう
いう名前のファイルが出力されるべきかわかる

- ・ [イントロ](#) | [NGS](#) | [アノテーション情報取得](#) | [TxDb](#) | [GenomicFeatures\(Lawrence 2013\)](#) (last modified 2016/04/22)
- ・ [イントロ](#) | [NGS](#) | [アノテーション情報取得](#) | [TxDb](#) | [GFF/GTF形式ファイルから](#) (last modified 2016/04/22)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [BSgenome](#) | [基本情報を取得](#) (last modified 2016/04/22)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTA形式](#) | [基本情報を取得](#) (last modified 2016/04/22)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTA形式](#) | [description行の記述を整形](#) (last modified 2014/04/05)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [基礎](#) (last modified 2015/07/26)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [応用](#) (last modified 2015/06/18)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [description行の記述を整形](#) (last modified 2014/08/21)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [SAM/BAM形式](#) (last modified 2016/09/14)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [Illuminaの*](#)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [Illuminaの*](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [BAM -> FASTQ](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [FASTQ -> BAM](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [Genbank](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [qseq -> BAM](#)

イントロ | NGS | 読み込み | FASTA形式 | 基本情報を取得

multi-FASTAファイルを読み込んで、Total lengthやaverage lengthなどの各種情報取得を行うためのやり方を示します。例題6以降は、ヒトやマウスレベルの巨大ファイルを取り扱うためのコードです。具体的には、塩基数を整数(integer)ではなく実数(real number)として取り扱うためのas.numeric関数を追加しています。
「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:

```
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(基本情報取得)
Total_len <- sum(width(fasta)) #配列の「トータルの長さ」を取得
Number_of_contigs <- length(fasta) #「配列数」を取得
Average_len <- mean(width(fasta)) #配列の「平均長」を取得
Median_len <- median(width(fasta)) #配列の「中央値」を取得
Max_len <- max(width(fasta)) #配列の長さの「最大値」を取得
```

ダウンロードと確認

①例題の入力ファイル(hoge4.fa)をダウンロード。②R上で作業ディレクトリの確認。③作業ディレクトリに解析したい入力ファイルがあることを確認。他のファイルがあってもよい

multi-FASTAファイルを読み込んで、Total lengthやaverage lengthなどの各種情報取得を行うためのやり方を示します。例題6以降は、ヒトやマウスレベルの巨大ファイルを取り扱うためのコードです。具体的には、塩基数を整数(integer)ではなく実数(real number)として取り扱うためのas.numeric関数を追加しています。

「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ー。

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:

```
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(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)
N50 <- sorted[obj][1]
```

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge4.fa"
> |
```

コピー

①一連のコマンド群をコピーして②R Console画面上でペースト。ブラウザがInternet Explorerの場合は、CTRLとALTキーを押しながらコードの枠内で左クリックすると、全選択できます。トリプルクリックでも可

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-F

```
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 <- readDNASTri

#本番(基本情報取得)
Total_len <- sum(wid
Number_of_contigs <-
Average_len <- mean(
Median_len <- median
Max len <- max(width
Min len <- min(width

#本番(N50情報取得)
sorted <- rev(sort(wid
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうか
N50 <- sorted[obj][1] #objがTRUEとなる1番
```

- 切り取り(T)
- コピー(C)** ①
- 貼り付け
- すべて選択(A)
- 印刷(I)...
- 印刷プレビュー(N)...
- Bing でマップ
- Bing で翻訳
- Google で検索
- 電子メール (Windows Live Hotmail)
- すべてのアクセラレータ
- Send to OneNote

このメニューを指定してin_fに格納
を指定してout_fに格納
読み込み
指定したファイルの読み込み
「最大値」を取得
「平均長」を取得
「中央値」を取得
「最小値」を取得

R Console

```
> getwd()
[1] "C:/Users/
> list.files()
[1] "hoge4.fa"
>
>
> |
```

- コピー Ctrl+C
- ペースト** ② Ctrl+V
- コマンドのペースト
- コピー&ペースト Ctrl+X
- ウインドウの消去 Ctrl+L
- 全て選択
- バッファに出力 Ctrl+W

コピー後に①list.files()で、②出力ファイルとして指定したファイル名の③hoge1.txtが作成されていることを確認

実行結果

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:

```

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(in_f, format="fasta")

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)
N50 <- sorted[obj][1]

```

```

#入力ファイル名を指定してin_fに格納
#出力ファイル名を指定してout_fに格納

#パッケージの読み込み

#配列の「
#「配列数
#配列の「
#配列の「
#配列の「
#配列の長

#長さ情報
#条件を満た
#objがTRUE

```

```

R Console
> tmp <- rbind(tmp, c("N50", N50))
> tmp <- rbind(tmp, c("GC content", GC_content$
> write.table(tmp, out_f, sep="\t", append=F, $
> list.files()
[1] "hoge1.txt" "hoge4.fa"
> tmp
      [,1]      [,2]
[1,] "Total length (bp)" "241"
[2,] "Number of contigs" "4"
[3,] "Average length" "60.25"
[4,] "Median length" "57"
[5,] "Max length" "103"
[6,] "Min length" "24"
[7,] "N50" "65"
[8,] "GC content" "0.576763485477178"
> |

```



実行結果

①の出力ファイルをテキストエディタやExcelで眺めてもよいが、②オブジェクトtmpの中身を出力しているだけなので、③R上で眺めている

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:

```
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を  
  
#本番(GC含量情報取得)  
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh  
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGIに格納(2015年9月12日以前  
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12  
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を計算してCGIに格納(2015年9月12日  
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総数を計算してACGTに格納(2015年9月12日  
GC_content <- sum(CG)/sum(ACGT) #トータルCGI
```

#ファイルに保存

```
tmp <- NULL  
tmp <- rbind(tmp, c("Total length (bp)", Total_l  
tmp <- rbind(tmp, c("Number of contigs", Number_  
tmp <- rbind(tmp, c("Average length", Average_le  
tmp <- rbind(tmp, c("Median length", Median_len  
tmp <- rbind(tmp, c("Max length", Max_len))  
tmp <- rbind(tmp, c("Min length", Min_len))  
tmp <- rbind(tmp, c("N50", N50))  
tmp <- rbind(tmp, c("GC content", GC_content))  
write.table(tmp, out_f, sep="\t", append=F, quot
```

```
R Console  
> tmp <- rbind(tmp, c("N50", N50))  
> tmp <- rbind(tmp, c("GC content", GC_content$  
> write.table(tmp, out_f, sep="\t", append=F, $  
> list.files()  
[1] "hoge1.txt" "hoge4.fa"  
> tmp  
[,1] [,2]  
[1,] "Total length (bp)" "241"  
[2,] "Number of contigs" "4"  
[3,] "Average length" "60.25"  
[4,] "Median length" "57"  
[5,] "Max length" "103"  
[6,] "Min length" "24"  
[7,] "N50" "65"  
[8,] "GC content" "0.576763485477178"  
> |
```

① contig_1が最短、contig_2が最長。② N50の値は65 bpであり、③ contig_3の長さと同じ

入出力の関係確認

入力: hoge4.fa

ID	Length
contig_1	24
contig_2	103
contig_3	65
contig_4	49

```
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

Total length (bp)	241
Number of contigs	4
Average length	60.25
Median length	57
Max length	103
Min length	24
N50	65
GC content	0.577

averageだと外れ値の影響を受けやすく、medianだと短いコンティグが多くを占める場合に不都合らしい

N50

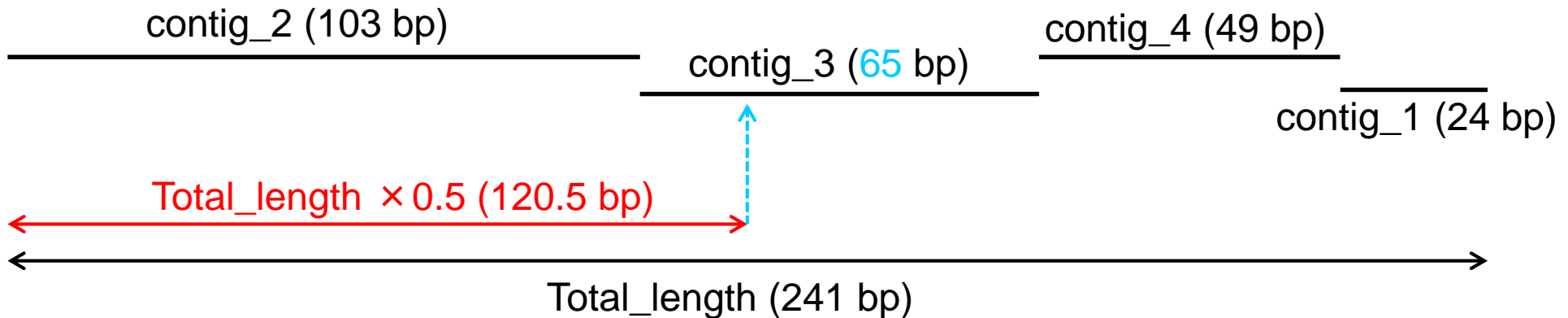
■ アセンブル結果の評価基準の1つ

- 長いコンティグから足していってTotal_lengthの50%に達したときのコンティグの長さ
- 一般に数値が大きいほどよい

ID	Length
contig_1	24
contig_2	103
contig_3	65
contig_4	49

出力: `hoge1.txt`

Total length (bp)	241
Number of contigs	4
Average length	60.25
Median length	57
Max length	103
Min length	24
N50	65
GC content	0.577



課題1

左記のコードをhoge4.faの代わりにhoge6.faを入力として実行し、1. 全配列長(配列長の総和)、2. N50の値、および3. GC含量を示せ

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:

```

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(in_f)

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len/2)
N50 <- sorted[obj][1]

```

```

>contig_1
NAGACAGCTCAACGGCA
>contig_2
GTCTGCCTCAAGCGAAACAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGNNGTCTGGTTGGCAGGGGCAGANNCCAGCACACCAA
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACCATCCCTTACTAATT
GTATGAGGTCNNGCA
>contig_4
CGTGCTGATANAACACAGCAGTAAACG
>contig_5
AGTGCTGATTCCACACAGCAGTAAACG
>contig_6
CACGTTGCATATT
>contig_7
AACGTTGCAG
>contig_8
AANCGTTNGCAGNANACCTTG

```

hoge6.fa

講義日程 (平成29年度)

- 平成29年04月17日 (PC使用)
講師: 嶋田 透
講師: 門田幸二
[バイオインフォマティクス基礎知識](#)
[講義資料PDF\(Win版; 完全版\)](#)
[講義資料PDF\(Mac版; Rの説明部分のみ\)](#)
- 平成29年04月18日 (PC使用)
講師: 門田幸二
[講義資料PDF](#)
[\(Rで\)塩基配列解析](#)
[hoge6.fa \(課題用\)](#)
- 平成29年04月24日 (PC使用)
講師: 嶋田 透
講師: 門田幸二
- 平成29年05月01日 (PC使用)
講師: 勝間 進



課題1

①作業ディレクトリの確認、②解析したいファイル(hoge6.fa)の存在確認、③推奨エディタの起動

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:

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

#入力ファイル名を指定してin_fに格納
#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
```

```
#本番(基本情報取得)
```

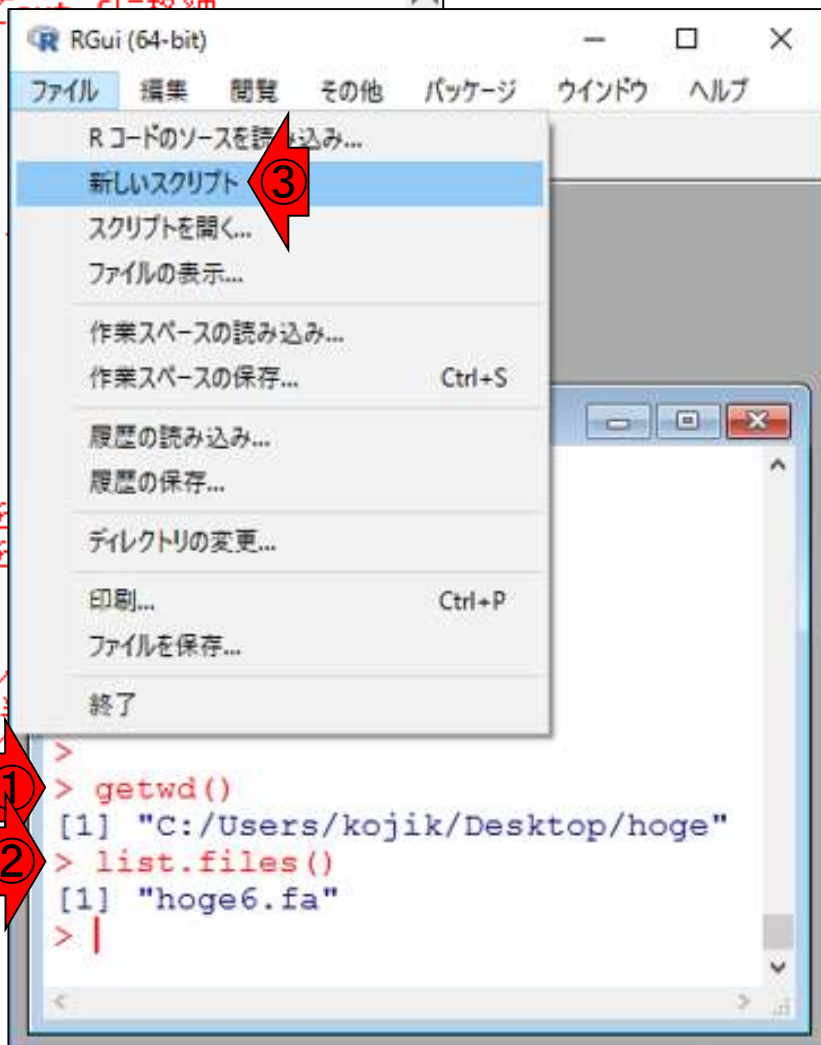
```
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))
```

#配列の「トータルの長さ」
#「配列数」を取得
#配列の「平均長」を取得
#配列の「中央値」を取得
#配列の長さの「最大値」を
#配列の長さの「最小値」を

```
#本番(N50情報取得)
```

```
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうかを
N50 <- sorted[obj][1]
```

#長さ情報を降順にソートし
#条件を満たすかどうかを
#objがTRUEとなる1番最初



課題1

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:

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

#必要なパッケージをロー
library(Biostrings)

#入力ファイルの読み込み
fasta <- readDNASTring

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= 0.5 * Total_len)
N50 <- sorted[obj][1]
```

RGui (64-bit) window showing the R Console and a script editor. The R Console displays the output of `getwd()` and `list.files()`. The script editor shows the R code with a red arrow pointing to the file name `hoge4.fa` in the `in_f` variable, which is annotated with a circled '1'.

```
> getwd()
[1] "C:/Users/kojik/Desktop/hoge"
> list.files()
[1] "hoge6.fa"
> |
```

```
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(in_f, format="fasta")#in_fで指定

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
```

課題1

変更後のコードを全選択して①コピーし、R Console画面上で②ペースト。全選択後に③を押すやり方でもよい

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:

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

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

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

#本番(基本情報取得)

```
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))
```

#本番(N50情報取得)

```
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) <= Total_len)[1]
N50 <- sorted[obj][1]
```

#入力ファイル名を指定してin_fに格納

RGui (64-bit) 画面のスクリーンショット。R Consoleには、`> getwd()` の結果として `[1] "C:/Users/kojik/Desktop/hoge"` と `> list.files()` の結果として `[1] "hoge6.fa"` が表示されている。RGuiのメニューバーには「ファイル」が赤い矢印③で指されている。また、RGuiのコードエディタには、課題1のコードが貼り付けられており、そのコードが青い背景で全選択されている。この状態で「コピー」が赤い矢印①で指されている。R Consoleには、赤い矢印②が指されている。

コード内部の説明

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:

```
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(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)
N50 <- sorted[obj][1]
```

```
R Console
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge4.fa"
> in_f <- "hoge4.fa" #入力ファイル名$
> out_f <- "hoge1.txt" #出力ファイル名$
>
> #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$
> |
```


コード内部の説明

①入力ファイル情報を格納したものが(DNAStringSetという形式で保持されている) fastaオブジェクト。widthの位置にあるのがコンティグごとの配列長情報。配列長情報は②width(fasta)で数値ベクトルとして抽出可能

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mul

```
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(in_f, format="fasta")#in_fで指定し

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)
N50 <- sorted[obj][1]
```

#入力ファイル名を指定し
#出力ファイル名を指定し
#パッケージの読み込み
#配列の「トータル長さ」を取得
#「配列数」を取得
#配列の「平均長さ」を取得
#配列の「中央値」を取得

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

```
R Console
> fasta <- readDNAStringSet(in_f, format="fasta")#in_fで$
> fasta
A DNAStringSet instance of length 4
width seq names
[1] 24 CGGACAGCTCCTCGGCATCCGGAT contig_1
[2] 103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
[3] 65 TGTAGGAGAAGG...TGAGGTCGGGCA contig_3
[4] 49 CGTGCTGATTCC...CCTATGAACATG contig_4

② > width(fasta)
[1] 24 103 65 49
> sum(width(fasta))
[1] 241
> |
```

コード内部の説明

width(fasta)に①sum関数を適用すれば、トータルの配列長(配列長の総和)になる。そのものずばりを②の部分で利用

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:

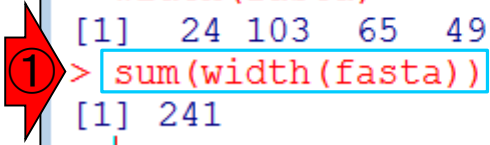
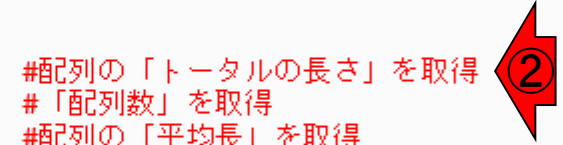
```
in_f <- "hoge4.fa" #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.txt" #出力ファイル名を指定してout_fに格納

#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(基本情報取得)
Total_len <- sum(width(fasta)) #配列の「トータルの長さ」を取得
Number_of_contigs <- length(fasta) #「配列数」を取得
Average_len <- mean(width(fasta)) #配列の「平均長」を取得
Median_len <- median(width(fasta)) #配列の「中央値」を取得
Max_len <- max(width(fasta)) #
Min_len <- min(width(fasta)) #

#本番(N50情報取得)
sorted <- rev(sort(width(fasta))) #
obj <- (cumsum(sorted) >= Total_len*0.5) #
N50 <- sorted[obj][1] #
```



```
R Console
> fasta <- readDNAStringSet(in_f, format="fasta")#in_fで$
> fasta
A DNAStringSet instance of length 4
  width seq names
[1] 24 CGGACAGCTCCTCGGCATCCGGAT contig_1
[2] 103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
[3] 65 TGTAGGAGAAGG...TGAGGTCGGGCA contig_3
[4] 49 CGTGCTGATTCC...CCTATGAACATG contig_4
> width(fasta)
[1] 24 103 65 49
> sum(width(fasta))
[1] 241
> |
```

①length関数は要素数を返す。この場合、fastaオブジェクトの要素数(つまりコンティグ数)を返す

コード内部の説明

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:

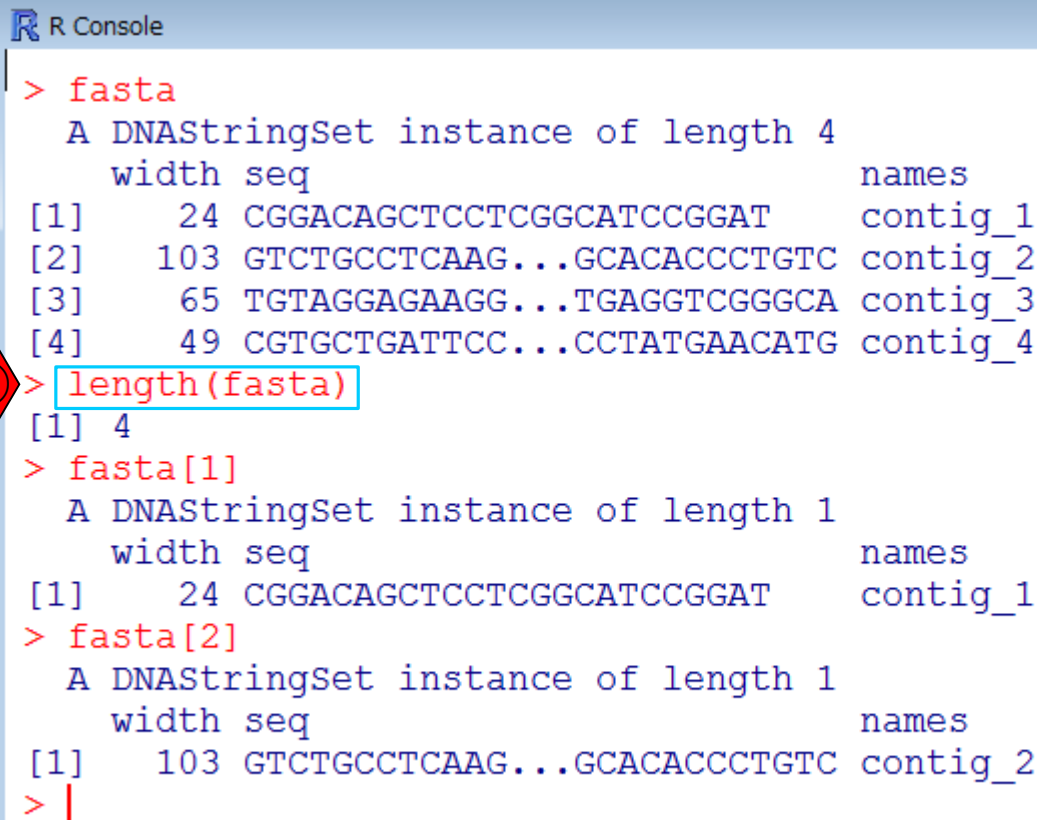
```
in_f <- "hoge4.fa" #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.txt" #出力ファイル名を指定してout_fに格納

#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="")

#本番(基本情報取得)
Total_len <- sum(width(fasta))
Number_of_contigs <- length(fasta)
Average_len <- mean(width(fasta))
Median_len <- median(width(fasta))
Max_len <- max(width(fasta))
Min_len <- min(width(fasta))

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))
obj <- (cumsum(sorted) >= Total_len*0.5)
N50 <- sorted[obj][1]
```



```
R Console
> fasta
A DNAStringSet instance of length 4
  width seq          names
[1]    24 CGGACAGCTCCTCGGCATCCGGAT contig_1
[2]   103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
[3]    65 TGTAGGAGAAGG...TGAGGTCGGGCA contig_3
[4]    49 CGTGCTGATTCC...CCTATGAACATG contig_4
> length(fasta)
[1] 4
> fasta[1]
A DNAStringSet instance of length 1
  width seq          names
[1]    24 CGGACAGCTCCTCGGCATCCGGAT contig_1
> fasta[2]
A DNAStringSet instance of length 1
  width seq          names
[1]   103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
> |
```

Tips: 条件判定

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:

```
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)

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

#本番(基本情報取得)
Total_len <- sum(width(fasta)) #
Number_of_contigs <- length(fasta) #
Average_len <- mean(width(fasta)) #
Median_len <- median(width(fasta)) #
Max_len <- max(width(fasta)) #
Min_len <- min(width(fasta)) #

#本番(N50情報取得)
sorted <- rev(sort(width(fasta))) #
obj <- (cumsum(sorted) >= Total_len*0.5) #
N50 <- sorted[obj][1]
```

```
R Console
> fasta
A DNAStringSet instance of length 4
  width seq          names
[1]   24 CGGACAGCTCCTCGGCATCCGGAT contig_1
[2]  103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
[3]   65 TGTAGGAGAAGG...TGAGGTCGGGCA contig_3
[4]   49 CGTGCTGATTCC...CCTATGAACATG contig_4
> width(fasta) > 100
[1] FALSE TRUE FALSE FALSE
> width(fasta) == 65
[1] FALSE FALSE TRUE FALSE
> width(fasta) >= 50
[1] FALSE TRUE TRUE FALSE
> obj <- width(fasta) >= 50
> fasta[obj]
A DNAStringSet instance of length 2
  width seq          names
[1]  103 GTCTGCCTCAAG...GCACACCCTGTC contig_2
[2]   65 TGTAGGAGAAGG...TGAGGTCGGGCA contig_3
> |
```



①サブセット抽出テクニックはここで使っている。コードの中身が分かると応用範囲が飛躍的に増大。一定以上のスキルをもつバイオインフォマティシャンは、例題を探すよりも自分で作るヒトのほうが多いかも...

Tips: 条件判定

- 前処理 | フィルタリング | [ACGT以外の character "-" を N に変換](#) (last modified 2013/06/18)
- 前処理 | フィルタリング | [ACGT以外の文字数が閾値以下の配列を抽出](#) (last modified 2013/06/18)
- 前処理 | フィルタリング | [重複のない配列セットを作成](#) (last modified 2013/06/18)
- 前処理 | フィルタリング | [指定した長さ以上の配列を抽出](#) (last modified 2014/02/07)
- 前処理 | フィルタリング | [任意のリード\(サブセット\)を抽出](#) (last modified 2014/08/21)
- 前処理 | フィルタリング | [指定した長さの範囲の配列を抽出](#) (last modified 2015/02/06)
- 前処理 | フィルタリング | [任意のIDを抽出](#) (last modified 2015/02/06)
- 前処理 | フィルタリング | [IlluminaのGFF/GTFを抽出](#) (last modified 2015/02/06)
- 前処理 | フィルタリング | [組合せ | A](#)
- 前処理 | トリミング | [ポリA配列除去](#)
- 前処理 | トリミング | [アダプター配列](#)
- 前処理 | トリミング | [アダプター配列](#)
- 前処理 | トリミング | [アダプター配列](#)
- 前処理 | トリミング | [アダプター配列](#)
- 前処理 | トリミング | [指定した末端](#)
- [アセンブル | について](#) (last modified 2015/02/06)
- [アセンブル | ゲノム用](#) (last modified 2015/02/06)

前処理 | フィルタリング | 指定した長さ以上の配列を抽出

FASTA形式やFASTQ形式ファイルを入力として、指定した配列長以上の配列を抽出するやり方を示します。「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ー。

1. multi-FASTAファイル(hoge4.fa)の場合:

[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ファイルです。

```

in_f <- "hoge4.fa"           #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.fasta"       #出力ファイル名を指定してout_fに格納
param <- 50                 #配列長の閾値を指定

#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
fasta                                     #確認してるだけです

#本番
obj <- as.logical(width(fasta) >= param)#条件を満たすかどうかを判定した結果をobjに格納
fasta <- fasta[obj]         #objがTRUEとなる要素のみ抽出した結果をfastaに格納
fasta                       #確認してるだけです

#ファイルに保存
writeXStringSet(fasta, file=out_f, format="fasta", width=50)#fastaの中身を指定したファイル名で
    
```

Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

任意の領域の切り出し

①subseq関数を用いて、任意の領域の配列を切り出すことができます。②例題1は、3-9塩基目の配列を切り出すやり方です

- ・ イントロ | 一般 | [ランダムな塩基配列を生成](#) (last modified 2014/06/16)
- ・ イントロ | 一般 | [任意の長さの可能な全ての塩基配列を作成](#) (last modified 2015/02/19)
- ・ イントロ | 一般 | [任意の位置の塩基を置換](#) (last modified 2013/09/12)
- ・ イントロ | 一般 | [指定した範囲の配列を取得](#) (last modified 2015/04/06) **NEW**
- ・ イントロ | 一般 | [指定したID\(染色体やdescription\)の配列を取得](#) (last modified 2014/03/10)
- ・ イントロ | 一般 | [翻訳配列\(translate\)を取得\(基礎\)](#)
- ・ イントロ | 一般 | [翻訳配列\(translate\)を取得\(応用\)](#)
- ・ イントロ | 一般 | [相補鎖\(complement\)を取得](#) (last modified 2014/03/10)
- ・ イントロ | 一般 | [逆相補鎖\(reverse complement\)を取得](#) (last modified 2014/03/10)
- ・ イントロ | 一般 | [逆鎖\(reverse\)を取得](#) (last modified 2014/03/10)
- ・ イントロ | 一般 | [2連続塩基の出現頻度情報を取得](#) (last modified 2014/03/10)
- ・ イントロ | 一般 | [3連続塩基の出現頻度情報を取得](#) (last modified 2014/03/10)

イントロ | 一般 | 指定した範囲の配列を取得 **NEW**

single-FASTA形式やmulti-FASTA形式ファイルから様々な部分配列を取得するやり方を示します。この項目は、「この染色体の、ここから、ここまで」という指定の仕方になります。例えば入力ファイルがヒトゲノムだった場合に、chr3の20000から500000の座標の配列取得を行いたい場合などに利用します。したがって、chr4とchr8の配列のみ抽出といったやり方には対応していませんのでご注意ください。また、ファイルダウンロード時に、*.fastaという拡張子が*.txtに勝手に変更されることがありますのでご注意ください。

「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ー。

1. (single-)FASTA形式ファイル(sample1.fasta)の場合:

任意の範囲(始点が3, 終点が9)の配列を抽出するやり方です。

```
in_f <- "sample1.fasta" #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.fasta" #出力ファイル名を指定してout_fに格納
param <- c(3, 9) #抽出したい範囲の始点と終点を指定

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

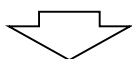
#入力ファイルの読み込み
fasta <- readDNASTringSet(in_f, format="fasta")#in_fで指定したファイルの読み込み
fasta #確認してるだけです

#本番
fasta <- subseq(fasta, param[1], param[2])#paramで指定した始点と終点の範囲の配列を抽出
fasta #確認してるだけです

#ファイルに保存
writeXStringSet(fasta, file=out_f, format="fasta", width=50)#fastaの中身を指定したフ
```

入力: sample1.fasta

```
>kadota
AGTGACGGTCTT
```



出力: hoge1.fasta

```
>kadota
TGACGGT
```

コピペ

①入力ファイル読み込み直後は12 bp。
 ②出力ファイルのfastaオブジェクトは、
 部分配列の抽出後なので当然7bp

1. (single-)FASTA形式ファイル(sample1.fasta)の場合:

任意の範囲 (始点が3, 終点が9)の配列を抽出するやり方です。

```

in_f <- "sample1.fasta" #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.fasta" #出力ファイル名を指定してout_fに格納
param <- c(3, 9) #抽出したい範囲

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

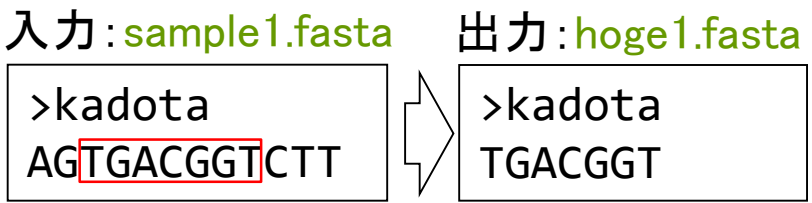
#入力ファイルの読み込み
fasta <- readDNAStringSet(in_f, format="fasta") #in_fに格納
fasta #確認してるか

#本番
fasta <- subseq(fasta, param[1], param[2]) #paramで抽出範囲を指定
fasta #確認してるか

#ファイルに保存
writeXStringSet(fasta, file=out_f, format="fasta")
  
```

```

R Console
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")
> fasta #確認$
A DNAStringSet instance of length 1
width seq names $
[1] 12 AGTGACGGTCTT kadota
>
> #本番
> fasta <- subseq(fasta, param[1], param[2]) #paramで抽出範囲を指定
> fasta #確認$
A DNAStringSet instance of length 1
width seq names $
[1] 7 TGACGGT kadota
>
> #ファイルに保存
> writeXStringSet(fasta, file=out_f, format="fasta")
> fasta
A DNAStringSet instance of length 1
width seq names
[1] 7 TGACGGT kadota
> |
  
```



Tips: 関数のオプション

1. (single-)FASTA形式ファイル(sample1.fasta)の場合:

任意の範囲 (始点が3, 終点が9)の配列を抽出するやり方です。

```

in_f <- "sample1.fasta"      #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.fasta"      #出力ファイル名を指定してout_fに格納
param <- c(3, 9)           #抽出したい範囲の始点と終点を指定

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

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

#本番
fasta <- subseq(fasta, param[1], param[2])
fasta                        #確認して$

#ファイルに保存
writeXStringSet(fasta, file=out_f, format="fasta")
    
```

```

R Console
> #入力ファイルの読み込み
> fasta <- readDNAStringSet(in_f, format="fasta")#$
> fasta                                #確認して$
A DNAStringSet instance of length 1
width seq          names
[1]      12 AGTGACGGTCTT      kadota
> subseq(fasta, param[1], param[2])
A DNAStringSet instance of length 1
width seq          names
[1]       7 TGACGGT          kadota
① > subseq(fasta, 3, 9)
A DNAStringSet instance of length 1
width seq          names
[1]       7 TGACGGT          kadota
② > subseq(fasta, start=3, end=9)
A DNAStringSet instance of length 1
width seq          names
[1]       7 TGACGGT          kadota
> |
    
```

入力: sample1.fasta

出力: hoge1.fasta

```
>kadota
AGTGACGGTCTT
```

```
>kadota
TGACGGT
```

Tips: 関数のオプション

- ①原因既知状態でエラーを出す。
- ②「3番目の位置から5塩基分抽出」というwidthオプションを利用

1. (single-)FASTA形式ファイル(sample1.fasta)の場合:

任意の範囲 (始点が3, 終点が9)の配列を抽出するやり方です。

```
in_f <- "sample1.fasta" #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.fasta" #出力ファイル名を指定してout_fに格納
param <- c(3, 9) #抽出したい範囲の始点と終点を指定

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

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

#本番
fasta <- subseq(fasta, param[1], param[2]) #param[1]とparam[2]を指定
fasta #確認して表示

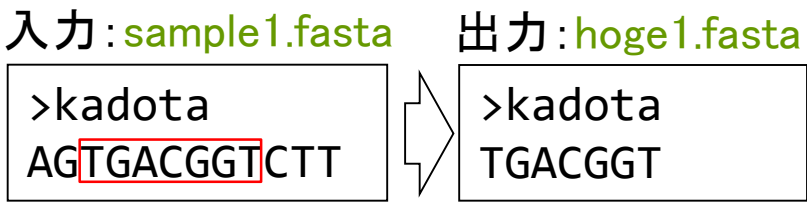
#ファイルに保存
writeXStringSet(fasta, file=out_f, format="fasta")
```

```
> subseq(fasta, start=3, end=12)
A DNASTringSet instance of length 1
width seq names
[1] 10 TGACGGTCTT kadota

① > subseq(fasta, start=3, end=13)
以下にエラー .Call2("solve_user_SEW", refwidths, $
solving row 1: 'allow.nonarrowing' is FALSE and$

② > subseq(fasta, start=3, width=5)
A DNASTringSet instance of length 1
width seq names
[1] 5 TGACG kadota

> |
```



Tips: 関数の使用法

①「?関数名」で使用法を記したウェブページが開く。ページの下の方に、大抵の場合使用例が掲載されている。使用法既知の関数のマニュアルをいくつか読んで慣れておく

```
R Console
> ?subseq
starting httpd help server ... done
> |
```

XVector-class {IRanges} R Documentation

XVector objects

Description

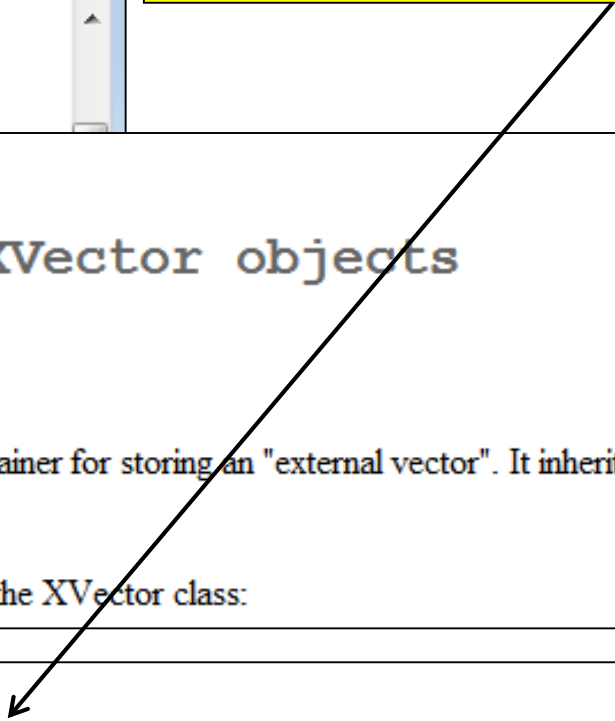
The XVector virtual class is a general container for storing an "external vector". It inherits from the [Vector](#), which has a very rich interface.

The following classes derive directly from the XVector class:

```
subseq(x4, start=10)
subseq(x4, start=-10)
subseq(x4, start=-20, end=-10)
subseq(x4, start=10, width=5)
subseq(x4, end=10, width=5)
subseq(x4, end=10, width=0)

x3[length(x3):1]
x3[length(x3):1, drop=FALSE]
```

[Package *IRanges* version 1.12.6 [Index](#)]



任意の領域の切り出し

②例題4。入力がmulti-FASTAファイル (hoge4.fa) で、リストファイル (list_sub2.txt) で指定した複数領域を切り出したい場合

- ・ イントロ | 一般 | [ランダムな塩基配列を生成](#) (last modified 2014/06/16)
- ・ イントロ | 一般 | [任意の長さの可能な全ての塩基配列を作成](#) (last modified 2015/02/19)
- ・ イントロ | 一般 | [任意の位置の塩基を置換](#) (last modified 2013/09/12)
- ・ イントロ | 一般 | [指定した範囲の配列を取得](#) (last modified 2015/04/06) **NEW**
- ・ イントロ | 一般 | [指定したID\(染色体やdescription\)の配列を取得](#) (last modified 2014/03/10)

イントロ | 一般 | 指定した範囲の配列を取得 **NEW**

② 4. 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: 目的の accession番号が複数ある場合に対応したものです。予め用意しておいた「1列目: accession, 2列目: start位置, 3列目: end位置」からなるリストファイル (list_sub2.txt) を読み込ませて、目的の配列の multi-FASTA ファイルをゲットするやり方です。

1. (single-)FASTA形式ファイル

任意の範囲 (始点が3, 終点が9)

```
in_f <- "sample1.fasta"
out_f <- "hoge1.fasta"
param <- c(3, 9)
```

```
#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 <- readDNASTringS
fasta
```

```
in_f1 <- "hoge4.fa"
in_f2 <- "list_sub2.txt"
out_f <- "hoge4.fasta"
```

#入力ファイル名を指定してin_f1に格納(multi-FASTA)
#入力ファイル名を指定してin_f2に格納(リストファイル)
#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_f1, format="fasta")#in_f1で指定したファイルの読み込み
posi <- read.table(in_f2)
fasta
```

#in_f2で指定したファイルの読み込み
#確認してるだけです

```
#本番
hoge <- NULL
for(i in 1:nrow(posi)){
  obj <- names(fasta) == posi[i,1]
  hoge <- append(hoge, subseq(fasta[obj], start=posi[i,2], end=posi[i,3]))#subseq
}
fasta <- hoge
fasta
```

#最終的に得る結果を格納するためのプレースホルダ
#length(posi)回だけループを回す
#条件を満たすかどうかを判定した結果をobjに格納
#hogeの中身をfastaに格納
#確認してるだけです

```
#ファイルに保存
```

Contents

- 行列形式ファイルの解析基礎(アノテーションファイルを例に)
 - 例題をテンプレートとして任意の解析を行う基本手順
 - 入力ファイルの最後の改行の有無
 - ありがちなミスとエラーメッセージ
 - コード内部の説明(行列演算の基礎)
- multi-FASTAファイルからの各種情報抽出
 - 基本情報取得(コンティグ数、配列長、N50、GC含量)
 - 任意の領域の切り出し
 - GC含量計算部分の説明

GC含量計算部分の説明

- ・ [イントロ](#) | [NGS](#) | [アノテーション情報取得](#) | [TxDb](#) | [GenomicFeatures\(Lawrence 2013\)](#) (last modified 2015/02/19)推奨
- ・ [イントロ](#) | [NGS](#) | [アノテーション情報取得](#) | [TxDb](#) | [GFF/GTF形式ファイルから](#) (last modified 2016/02/09)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [BSgenome](#) | [基本情報を取得](#) (last modified 2015/09/12)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTA形式](#) | [基本情報を取得](#) (last modified 2015/09/12)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTA形式](#) | [description行の記述を整形](#) (last modified 2014/04/05)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [基礎](#)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [応用](#)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [FASTQ形式](#) | [descri](#)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [Illuminaの* seq.txt](#)
- ・ [イントロ](#) | [NGS](#) | [読み込み](#) | [Illuminaの* qseq.txt](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [について](#) (last m
- ・ [イントロ](#) | [ファイル形式の変換](#) | [BAM -> BED](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [FASTQ -> FAS](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [Genbank -> FA](#)
- ・ [イントロ](#) | [ファイル形式の変換](#) | [qseq -> FASTA](#)

イントロ | NGS | 読み込み | FASTA形式 | 基本情報を取得

multi-FASTAファイルを読み込んで、Total lengthやaverage lengthなどの各種情報取得を行うためのやり方を示します。例題6以降は、ヒトやマウスレベルの巨大ファイルを取り扱うためのコードです。具体的には、塩基数を整数(integer)ではなく実数(real number)として取り扱うためのas.numeric関数を追加しています。「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ー。

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:

```

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(基本情報取得)
Total_len <- sum(width(fasta)) #配列の「トータルの長さ」を取得
Number_of_contigs <- length(fasta) #「配列数」を取得
Average_len <- mean(width(fasta)) #配列の「平均長」を取得
Median_len <- median(width(fasta)) #配列の「中央値」を取得
Max_len <- max(width(fasta)) #配列の長さの「最大値」を取得
Min_len <- min(width(fasta)) #配列の長さの「最小値」を取得

#本番(N50情報取得)
sorted <- rev(sort(width(fasta))) #長さ情報を降順にソートした結果をsortedに格納
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうかを判定した結果をobjに格納
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を
    
```

GC含量計算部分の説明

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:

```

obj <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を
#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGIに格納(2015年9月12日以前
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTIに格納(2015年9月12
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を計算してCGIに格納(2015年9月12日)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総数を計算してACGTIに格納(2015年9月12日)
GC_content <- sum(CG)/sum(ACGT) #トータルGC含量

```

```

#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_l
tmp <- rbind(tmp, c("Number of contigs", Number_
tmp <- rbind(tmp, c("Average length", Average_le
tmp <- rbind(tmp, c("Median length", Median_len)
tmp <- rbind(tmp, c("Max length", Max_len))
tmp <- rbind(tmp, c("Min length", Min_len))
tmp <- rbind(tmp, c("N50", N50))
tmp <- rbind(tmp, c("GC content", GC_content))
write.table(tmp, out_f, sep="\t", append=F, quot

```

```

R Console
> tmp <- rbind(tmp, c("N50", N50))
> tmp <- rbind(tmp, c("GC content", GC_content$
> write.table(tmp, out_f, sep="\t", append=F, $
> list.files()
[1] "hoge1.txt" "hoge4.fa"
> tmp
      [,1]      [,2]
[1,] "Total length (bp)" "241"
[2,] "Number of contigs" "4"
[3,] "Average length"    "60.25"
[4,] "Median length"     "57"
[5,] "Max length"        "103"
[6,] "Min length"        "24"
[7,] "N50"               "65"
[8,] "GC content"        "0.576763485477178"
> |

```

GC含量計算部分の説明

①行頭に#がついている箇所はコメントアウトされている箇所に相当する。実際に消すのではなく、一時的に機能しないようにしている。②が機能していないコメント部分だったことを思い出せば納得できるはず

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

```
sorted[obj][1] #objがTRUEとなる1番最初の要素のみ

#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,...の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得

#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
tmp <- rbind(tmp, c("Average length", Average_len))
tmp <- rbind(tmp, c("Median length", Median_len))
tmp <- rbind(tmp, c("Max length", Max_len))
tmp <- rbind(tmp, c("Min length", Min_len))
tmp <- rbind(tmp, c("N50", N50))
tmp <- rbind(tmp, c("GC content", GC_content))
write.table(tmp, out_f, sep="\t", append=F, quote=F, row.names=F) #tmpの中身を指定し
```

```
R Console
> 1+1
[1] 2
> #1+1
> |
```


GC含量計算部分の説明

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:

```
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

#本番(基本情報取得)
Total_len <- sum(width(fasta)) #配列の「トータル」
Number_of_contigs <- length(fasta) #「配列数」を取得
Average_len <- mean(width(fasta)) #配列の「平均長さ」
Median_len <- median(width(fasta)) #配列の「中央値」
Max_len <- max(width(fasta)) #配列の長さの「最大値」
Min_len <- min(width(fasta)) #配列の長さの「最小値」

#本番(N50情報取得)
sorted <- rev(sort(width(fasta))) #長さ情報を降順にソート
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすか
N50 <- sorted[obj][1] #objがTRUEとなる最初の要素
```

```
R Console
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge4.fa"
> in_f <- "hoge4.fa" #入力フ$
> out_f <- "hoge1.txt" #出力フ$
>
> #必要なパッケージをロード
> library(Biostrings) #パッケ$
>
> #入力ファイルの読み込み
> fasta <- readDNASTringSet(in_f, format="fasta"$
> fasta
A DNASTringSet instance of length 4
width seq names
[1] 24 CGGACAGC...TCCGGAT contig_1
[2] 103 GTCTGCCT...CCCTGTC contig_2
[3] 65 TGTAGGAG...TCGGGCA contig_3
[4] 49 CGTGCTGA...GAACATG contig_4
> |
```



①alphabetFrequency関数は、塩基ごとの出現回数を返す

GC含量計算部分の説明

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:

```

obj <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を
#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGIに格納(2015年9月12日以前)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTIに格納(2015年9月12日以前)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGIに格納(2015年9月12日以前)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTIに格納(2015年9月12日以前)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得

```

```

#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
tmp <- rbind(tmp, c("Average length", Average_len))
tmp <- rbind(tmp, c("Median length", Median_len))
tmp <- rbind(tmp, c("Max length", Max_len))
tmp <- rbind(tmp, c("Min length", Min_len))
tmp <- rbind(tmp, c("N50", N50))
tmp <- rbind(tmp, c("GC content", GC_content))
write.table(tmp, out_f, sep="\t", append=F, quote=F)

```

```

R Console
> fasta
A DNASTringSet instance of length 4
width seq names
[1] 24 CGGACAGC...TCCGGAT contig_1
[2] 103 GTCTGCCT...CCCTGTC contig_2
[3] 65 TGTAGGAG...TCGGGCA contig_3
[4] 49 CGTGCTGA...GAACATG contig_4
> hoge <- alphabetFrequency(fasta)
> hoge
      A C G T M R W S Y K V H D B N - + .
[1,]  4  9  7  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[2,] 20 34 31 18  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[3,] 16 13 20 16  0  0  0  0  0  0  0  0  0  0  0  0  0  0
[4,] 14 15 10 10  0  0  0  0  0  0  0  0  0  0  0  0  0  0
> |

```

DNA配列上の①Mは「A or C」、②Rは「A or G」などというルールがあるようです

GC含量計算部分の説明

Nucleic Acid Code ⇅	Meaning ⇅	Mnemonic ⇅
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
U	U	Uracil
R	A or G	puRine
Y	C, T or U	pYrimidines
K	G, T or U	bases which are K etones
M	A or C	bases with a Mino groups
S	C or G	S trong interaction
W	A, T or U	W eak interaction
B	not A (i.e. C, G, T or U)	B comes after A
D	not C (i.e. A, G, T or U)	D comes after C
H	not G (i.e., A, C, T or U)	H comes after G
V	neither T nor U (i.e. A, C or G)	V comes after U
N	A C G T U	N ucleic acid
X	masked	
-	gap of indeterminate length	

```

R> fasta <- readLines("example.fasta")
R> alphabetFrequency(fasta)
#> AStringSet instance of length 4
#> width seq          names
#> 24 CGGACAGC...TCCGGAT contig_1
#> 103 GTCTGCCT...CCCTGTC contig_2
#> 65 TGTAGGAG...TCGGGCA contig_3
#> 49 CGTGCTGA...GAACATG contig_4
R> alphabetFrequency(fasta)
#> A C G T M R W S Y K V H D B N - + .
#> 4 9 7 4 0 0 0 0 0 0 0 0 0 0 0 0 0
#> 20 34 31 18 0 0 0 0 0 0 0 0 0 0 0 0 0
#> 16 13 20 16 0 0 0 0 0 0 0 0 0 0 0 0 0
#> 14 15 10 10 0 0 0 0 0 0 0 0 0 0 0 0 0
    
```

http://en.wikipedia.org/wiki/FASTA_format

GC含量計算部分の説明

①dim関数は行列の行数と列数を返す。alphabetFrequency関数出力結果は、4行×18列からなることが分かる。②最初の4列分のみ抽出するやり方。キーボードの上下キーを上手に利用して最小限の労力でキータイプ(あるいはコピー)すべし!

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(hc

```
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出

#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量
```

#ファイルに保存

```
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
tmp <- rbind(tmp, c("Average length", Average_len))
tmp <- rbind(tmp, c("Median length", Median_len))
tmp <- rbind(tmp, c("Max length", Max_len))
tmp <- rbind(tmp, c("Min length", Min_len))
tmp <- rbind(tmp, c("N50", N50))
tmp <- rbind(tmp, c("GC content", GC_content))
write.table(tmp, out_f, sep="\t", append=F, quote=F,
```

```
R Console
> hoge <- alphabetFrequency(fasta)
> hoge
      A C G T M R W S Y K V H D B N - + .
[1,]  4 9 7 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[2,] 20 34 31 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[3,] 16 13 20 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[4,] 14 15 10 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> dim(hoge)
[1]  4 18
> dim(alphabetFrequency(fasta))
[1]  4 18
> hoge[,1:4]
      A C G T
[1,]  4 9 7 4
[2,] 20 34 31 18
[3,] 16 13 20 16
[4,] 14 15 10 10
> |
```



GC含量計算部分の説明

任意のサブセットを取得可能。
①2:3やc(1,4)などをうまく利用

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:

```
obj <- (samtools) / ...  
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を  
  
#本番(GC含量情報取得)  
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh  
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納(2015年9月12日以前)  
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)  
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日以前)  
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)  
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得  
  
#ファイルに保存  
tmp <- NULL  
tmp <- rbind(tmp, c("Total length (bp)", Total_len))  
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))  
tmp <- rbind(tmp, c("Average length", Average_len))  
tmp <- rbind(tmp, c("Median length", Median_len))  
tmp <- rbind(tmp, c("Max length", Max_len))  
tmp <- rbind(tmp, c("Min length", Min_len))  
tmp <- rbind(tmp, c("N50", N50))  
tmp <- rbind(tmp, c("GC content", GC_content))  
write.table(tmp, out_f, sep="\t", append=F, quote=F, row.names=F) #tmpの中身を指定し
```

```
R Console  
① > hoge[,2:3]  
      C G  
[1,]  9 7  
[2,] 34 31  
[3,] 13 20  
[4,] 15 10  
> hoge[,c(1,4)]  
      A T  
[1,]  4 4  
[2,] 20 18  
[3,] 16 16  
[4,] 14 10  
> 2:3  
[1] 2 3  
> c(1,4)  
[1] 1 4  
> |
```

GC含量計算部分の説

黒丸中の数値はcontig_1中のAの数が4個、赤丸中の数値は、contig_4中のTの数が10個であるということ。①rowSums関数は行ごとの和を返す。②の記述法でも同じ結果となる

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

```
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素のみ抽出した結果を

#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得
```

#ファイルに保存

```
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
write(tmp, "hoge4.fa")
```

```
hoge4.fa - メモ帳
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
```

```
R Console
> hoge[,1:4]
      A C G T
[1,]  4 9 7 4
[2,] 20 34 31 18
[3,] 16 13 20 16
[4,] 14 15 10 10
> rowSums(hoge[,1:4])
[1] 24 103 65 49
> age <- hoge[,1:4]
> rowSums(age)
[1] 24 103 65 49
> apply(age, 1, sum)
[1] 24 103 65 49
> |
```

GC含量計算部分の説

①rowSums関数の入力として、ACGTのみのカウント数を与えているが、その結果(返り値)は、配列中にNなどを含まない場合は実質的に配列ごとの配列長と同じ。Nを含むhoge6.faでやってみると違いがわかる。尚、①applyという関数でもrowSumsと同じ結果が得られる

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

```

obj <- (sample(26, 10000, replace=TRUE))
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素

#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGIに格納(2015年9月12日以前
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGIに格納(2015年9月12日以前
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得

#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
write(tmp, "hoge4.fa")

```



```

>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG

```

```

R Console
> hoge[,1:4]
      A C G T
[1,]  4 9 7 4
[2,] 20 34 31 18
[3,] 16 13 20 16
[4,] 14 15 10 10
> rowSums(hoge[,1:4])
[1] 24 103 65 49
> age <- hoge[,1:4]
> rowSums(age)
[1] 24 103 65 49
> apply(age, 1, sum)
[1] 24 103 65 49
> |

```



GC含量計算部分の説

①オブジェクトCG中には、配列ごとのCとGのカウント数が格納されている。オブジェクトACGT中には、配列ごとのA, C, G, Tのカウント数が格納されている。例えば49塩基からなるcontig_4中に、②ACGTのいずれかの塩基が49個、③CGの数は25個あることを意味する。④sum関数は、ベクトルの要素の和を返す

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

```

obj <- readLines("hoge4.fa")
N50 <- sorted[obj][1] #objがTRUEとなる1番最初の要素

#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとに格納
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得

```

```

#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
write(tmp, "hoge4.fa", append=TRUE)

```

```

>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCAAGTGGGTTTGGAGGCCTAA
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGAC
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATC
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCT

```

```

R Console
> hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとに格納
> #CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納
> #ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日)
> CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日)
> ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日)
> GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得
> cd
エラー: オブジェクト 'cd' がありません
> cg
エラー: オブジェクト 'cg' がありません
> CG
[1] 16 65 33 25
> ACGT
[1] 24 103 65 49
> sum(CG)
[1] 139
> sum(ACGT)
[1] 241
> |

```


GC含量計算部分の記述

①ここではsum関数を用いて配列全体の総和でGC含量計算をしているが、②CG/ACGTとやると、配列ごとのGC含量を得られる。例えば、contig_1はCGの数が16個で、ACGTの数が24個。それゆえGC含量は $16/24 = 0.666667$

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

```

obj <- sorted[obj][1] #objがTRUEとなる1番最初の
#本番(GC含量情報取得)
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を配列ごとにカウントした結果をh
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum) #C,Gの総数を計算してCGに格納(2015年9月12日以前)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum) #A,C,G,Tの総数を計算してACGTに格納(2015年9月12日以前)
GC_content <- sum(CG)/sum(ACGT) #トータルのGC含量の情報を取得

```

#ファイルに保存

```

tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <- rbind(tmp, c("Number of contigs", Number_of_contigs))
write(tmp, "hoge4.fa", as.is=TRUE)

```

```

>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG

```

```

R Console
> sum(CG) / sum(ACGT)
[1] 0.5767635
> CG/ACGT
[1] 0.6666667 0.6310680 0.5076923
[4] 0.5102041
> |

```

配列ごとのGC含量計算

①sum関数を用いずに「CG/ACGT」とやって、配列ごとのGC含量を得るための項目。
②記述内容がほぼ同じことが分かる

正規化 | サンプル間 | 3群間 | 複製あり | [TMM\(Robinson 2010\)](#) (last modified 2015/03/30)

解析 | 一般 | [アラインメント](#) | [について](#) (last modified 2015/12/16)

解析 | 一般 | [アラインメント](#) | [ペアワイズ](#) | 基礎1 | [Biostrings](#) (last modified 2016/12/29)

解析 | 一般 | [アラインメント](#) | [ペアワイズ](#) | 基礎2 | [Biostrings](#) (last modified 2016/12/29)

解析 | 一般 | [アラインメント](#) | [ペアワイズ](#)

解析 | 一般 | [アラインメント](#) | [マルチプル](#)

解析 | 一般 | [アラインメント](#) | [マルチプル](#)

解析 | 一般 | [パターンマッチング](#) (last m

解析 | 一般 | [GC含量\(GC contents\)](#)

解析 | 一般 | [Sequence logos\(Schneid](#)

解析 | 一般 | [上流配列解析](#) | [LDSS\(Ya](#)

解析 | 一般 | [上流配列解析](#) | [Relative A](#)

解析 | 基礎 | [k-mer](#) | [ゲノムサイズ推定](#)

解析 | 基礎 | [平均-分散プロット](#) | [について](#)

解析 | 基礎 | [平均-分散プロット](#) | [Techn](#)

解析 | 基礎 | [平均-分散プロット](#) | [Biolog](#)

解析 | [新規転写物同定\(ゲノム配列を利](#)

解析 | [発現量推定\(トランスクリプトーム](#)

解析 | [発現量推定\(ゲノム配列を利用\)](#)

解析 | [クラスタリング](#) | [について](#) (last m

解析 | [クラスタリング](#) | [サンプル間](#) | [hcl](#)

解析 | [クラスタリング](#) | [サンプル間](#) | [TC](#)

解析 | 一般 | GC含量 (GC contents)

multi-FASTA形式ファイルやBSgenomeパッケージを読み込んで配列ごとのGC含量 (GC contents)を出力するやり方を示します。出力ファイルは、「description」「CGの総数」「ACGTの総数」「配列長」「%GC含量」としています。尚、%GC含量は「CGの総数/ACGTの総数」で計算しています。

「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ー。

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:

```
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
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の数を各配列ごとにカウントした結果をhogeに格納
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を計算してCGIに格納(2015年9月12日以前の記述)
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総数を計算してACGTIに格納(2015年9月12日以前の記述)
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を計算してCGIに格納(2015年9月12日以降の記述)
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総数を計算してACGTIに格納(2015年9月12日以降の記述)
GC_content <- CG/ACGT*100 #%GC含量を計算してGC_contentに格納

#ファイルに保存
tmp <- cbind(names(fasta), CG, ACGT, width(fasta), GC_content)#保存したい情報をtmpに格納
colnames(tmp) <- c("description", "CG", "ACGT", "Length", "%GC_contents")#列名を付与
write.table(tmp, out_f, sep="\t", append=F, quote=F, row.names=F, col.names=T)#tmpの中身を指定し
```

配列ごとのGC含量計算

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:

```

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
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の$
#CG <- rowSums(hoge[,2:3])      #C,Gの総数を$
#ACGT <- rowSums(hoge[,1:4])    #A,C,G,Tの総$
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を$
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総$
GC_content <- CG/ACGT*100      #%GC含量を計$

#ファイルに保存
tmp <- cbind(names(fasta), CG, ACGT, width(fasta), GC_content)
colnames(tmp) <- c("description", "CG", "ACGT", "Length", "%GC_contents")
write.table(tmp, out_f, sep="\t", append=F, quote=F)
    
```

```

R Console
> #本番
> hoge <- alphabetFrequency(fasta) #A,C,G,T,..の$
> CG <- rowSums(hoge[,2:3])      #C,Gの総数を$
> ACGT <- rowSums(hoge[,1:4])   #A,C,G,Tの総$
> GC_content <- CG/ACGT*100    #%GC含量を計$
>
> #ファイルに保存
> tmp <- cbind(names(fasta), CG, ACGT, width(fasta), GC_content)
> colnames(tmp) <- c("description", "CG", "ACGT", "Length", "%GC_contents")
> write.table(tmp, out_f, sep="\t", append=F, quote=F)
> tmp
      description CG  ACGT Length %GC_contents
[1,] "contig_1"  "16" "24"  "24"  "66.6666666666667"
[2,] "contig_2"  "65" "103" "103" "63.1067961165049"
[3,] "contig_3"  "33" "65"  "65"  "50.7692307692308"
[4,] "contig_4"  "25" "49"  "49"  "51.0204081632653"
> |
    
```

配列ごとのGC含量計算

①ACGT列は4種類の塩基のみの出現数、
 ②Length列は配列長情報を表す。配列長は、ACGT以外の全てを含むので、その差分($Length - ACGT$)がNなどのACGT以外の塩基のトータルの出現回数ということになる。 $Length \geq ACGT$ という関係

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

```

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
hoge <- alphabetFrequency(fasta) #A,C,G,T,..の$
#CG <- rowSums(hoge[,2:3]) #C,Gの総数を$
#ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総$
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を計$
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総$
GC_content <- CG/ACGT*100 #GC含量を計$

#ファイルに保存
tmp <- cbind(names(fasta), CG, ACGT, width(fasta), G$)
colnames(tmp) <- c("description", "CG", "ACGT", "Length")
write.table(tmp, out_f, sep="\t", append=F, quote=F, as.is=T)

```

```

R Console
> #本番
> hoge <- alphabetFrequency(fasta) #A,C,G,T,..の$
> CG <- rowSums(hoge[,2:3]) #C,Gの総数を$
> ACGT <- rowSums(hoge[,1:4]) #A,C,G,Tの総$
> GC_content <- CG/ACGT*100 #GC含量を計$
>
> #ファイルに保存
> tmp <- cbind(names(fasta), CG, ACGT, width(fasta), G$)
> colnames(tmp) <- c("description", "CG", "ACGT", "Length")
> write.table(tmp, out_f, sep="\t", append=F, quote=F, as.is=T)
> tmp

```

	description	CG	ACGT	Length	%GC_contents
[1,]	"contig_1"	"16"	"24"	"24"	"66.66666666666667"
[2,]	"contig_2"	"65"	"103"	"103"	"63.1067961165049"
[3,]	"contig_3"	"33"	"65"	"65"	"50.7692307692308"
[4,]	"contig_4"	"25"	"49"	"49"	"51.0204081632653"

課題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multi-FASTAファイル([hoge4.fa](#))の場合:

```
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(in_f, from

#本番
hoge <- alphabetFrequency(fasta)
#CG <- rowSums(hoge[,2:3])
#ACGT <- rowSums(hoge[,1:4])
CG <- apply(as.matrix(hoge[,2:3]),
ACGT <- apply(as.matrix(hoge[,1:4])
GC_content <- CG/ACGT*100

#ファイルに保存
tmp <- cbind(names(fasta), CG, ACGT
colnames(tmp) <- c("description", "
write.table(tmp, out_f, sep="\t", a
```

```
>contig_1
NAGACAGCTCAACGGCA
>contig_2
GTCTGCCTCAAGCGAAACAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGNNGTCTGGTTGGCAGGGGCAGANNCCAGCACACCAA
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCNNGCA
>contig_4
CGTGCTGATANAACACAGCAGTAAACGCGGACCTCTACCTATGAACANG
>contig_5
AGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATG
>contig_6
CACGTTGCATATT
>contig_7
AACGTTGCAG
>contig_8
AANCGTTNGCAGNANACCTTG
```

hoge6.fa