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/

全てPC使用予定です

講義予定

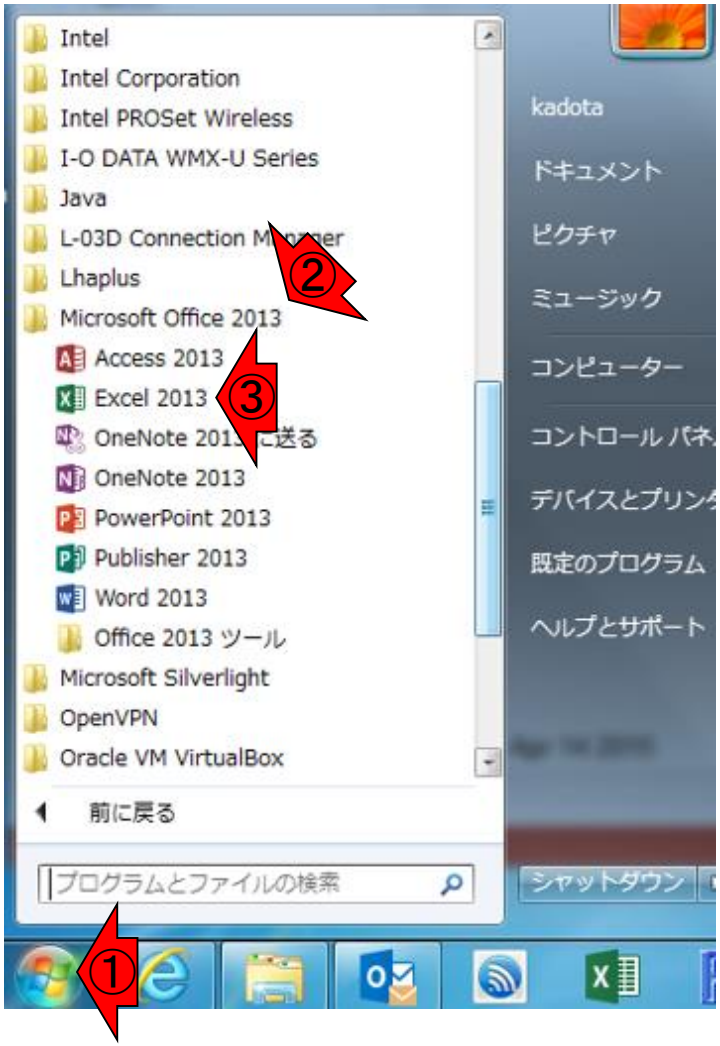
- 4月11日月曜日（17:15–20:30）PC使用
  - 嶋田透：ゲノムからの遺伝子予測
  - 門田幸二：バイオインフォマティクス基礎知識、Rのイントロダクション
- 4月18日月曜日（17:15–20:30）PC使用
  - 門田幸二：Rで塩基配列解析1、multi-FASTAファイルの各種解析
- 4月25日月曜日（17:15–20:30）PC使用
  - 嶋田透：ゲノムアノテーション、遺伝子の機能推定、RNA-seqなどによる発現解析、比較ゲノム解析
  - 門田幸二：Rで塩基配列解析2、Rパッケージ、k-mer解析の基礎
- 5月02日月曜日（17:15–19:00頃）PC使用
  - 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

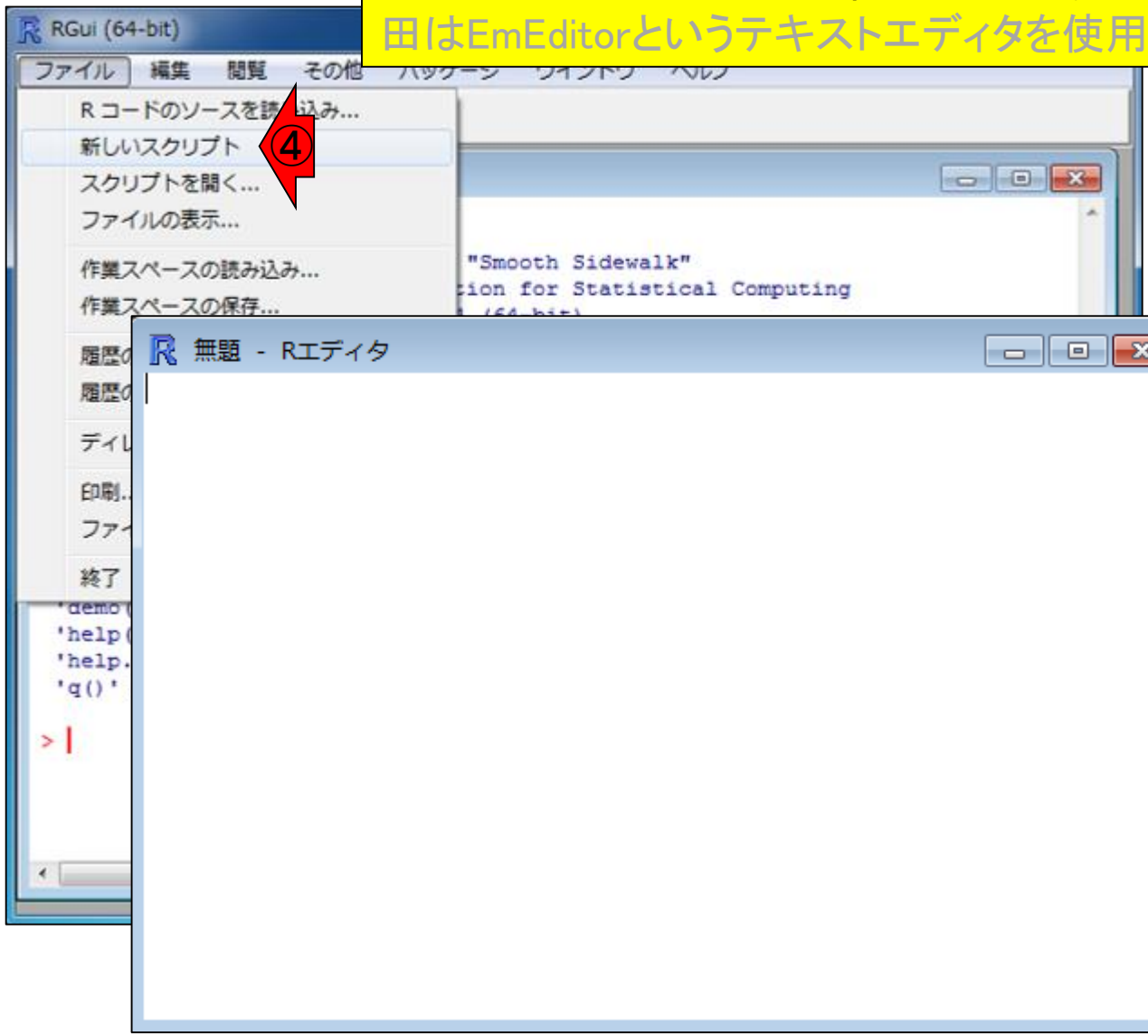
Contents

- 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hoge5.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
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・
勝間 進 (東大・農・生産・
門田幸二 (東大・農・アグ

参考図書

講義日程 (平成28年度)

1. 平成28年04月11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
2. 平成28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 ①
   ② hoge5.fa (課題用)
3. 平成28年04月25日 (PC使用)
   講師：嶋田 透
   講師：門田幸二
4. 平成28年05月02日 (PC使用)
   講師：勝間 進

(Rで)塩基配列解析

～NGS、RNA-seq、ゲノム、トランスクリプトーム、正規化、発現変動、統計、モデル、バイオインフォマティクス～
(last modified 2016/04/08, since 2011)

What's new?

- このウェブページはインストール||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)
- **平成28年度ハンズオン講習会が7/19-8/4の日程で開催されます。申込み期間は4/4-5/31です。追加申込みやキャンセル待ちの受付は絶対に行いませんので受講希望者はお気をつけください。「担当部分は主にこれまでの受講生を対象としています(リクエストを多く反映させています)。「昨年度との違い、対応関係、想定受講者、および予習事項」については こちらのPDF (約2MB)をご覧になり、必要に応じてつまみ食いしてください。**(2016/04/05) NEW

- QuasRでBowtieのマッピングを行う場合に、(内部的にはbowtie1が動いているため)リード長が1本でも1,024 bpを超えたものがあればコケマス(1024 bplはセーフで1025 bplはアウト)のでご注意ください(高橋 広夫 氏提供情報)。(2016/04/06) NEW
- 日本乳酸菌学会誌のNGS関連連載の 第6回ウェブ資料を更新しました。具体的には、W20-2のスライド256の解説内容を変更しました。(2016/03/29) NEW
- RNA-Seq実験ハンドブック(鈴木穣 編)が刊行されます。(2016/03/22) NEW
- 「書籍|トランスクリプトーム解析|2.3.1 RNA-seqデータ(FASTQファイル)」のgetFASTQinfoやgetFASTQfile関数実行部分でsra_conという引数を追加しました。が、おそらくデータを取りに行っているENAのURL (http://www.ebi.ac.uk/ena/data/view/reports/sra/fastq_files/)がリンク切れになっているようでうまく情報の取得ができなくなっているようです m(_ _)mもちろん私のせいではありませんw。ご指摘深謝m(_ _)m(2016/03/17) NEW

- はじめに (last modified 2015/03/31)
- 参考資料 (講義、講習会、本など) (last modified 2016/02/29)
- 過去のお知らせ (last modified 2016/04/05) NEW
- 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)

(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 (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)

① ←

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

多くの生物種についてEnsembl (Flicek et al., Nucleic Acids Res., 2014)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。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、ど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 (Flicek et al., Nucleic Acids Res., 2014)
  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

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

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

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	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

入力2:リストファイル(genelist1.txt)

	A
1	gene1
2	gene7
3	gene9

出力:hoge1.txt

hoge1.txt

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

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

① ②

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にあり、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_f1に格納(アノテーショ
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
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
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
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
```

解析基礎2

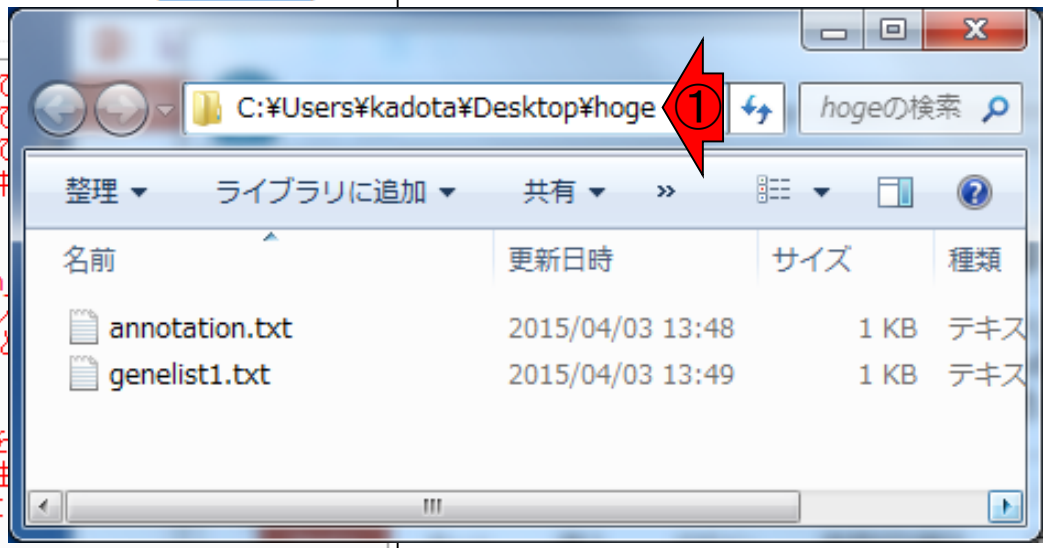
①作業ディレクトリは「デスクトップ – hoge」。hogeフォルダ中にannotation.txtとgenelist1.txtが存在するという前提。②貸与PCはkadotaではなくiu

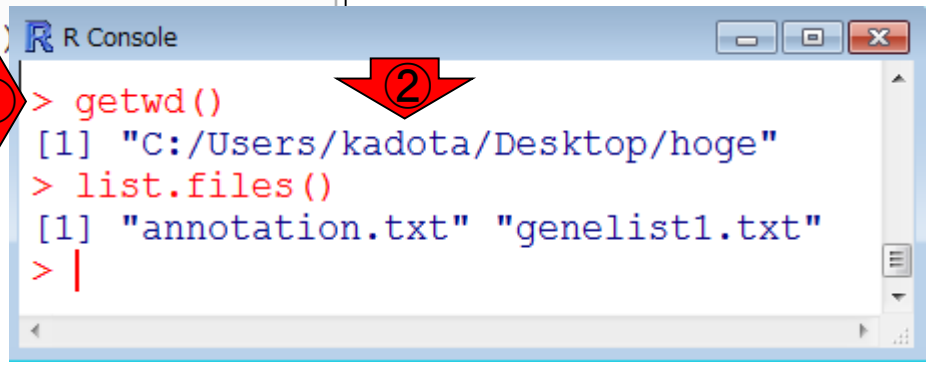
イントロ | 一般 | 任意のキーワードを含む行を抽出(基礎)

例えばタブ区切りテキストファイルが手元にあり、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                               #アノ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)
```





基本はコピペ

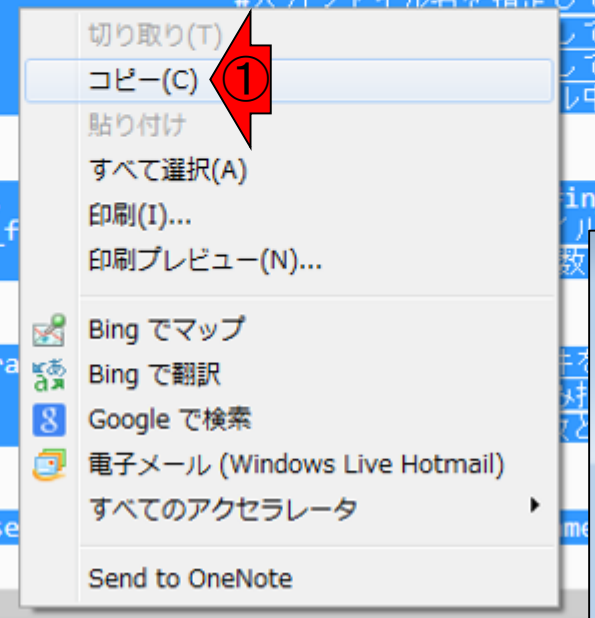
①一連のコマンド群をコピーして②R Console画面上でペースト。ブラウザがInternet Explorerの場合は、CTRLとALTキーを押しながらコードの枠内で左クリックすると、全選択できます。トリプルクリックでも全選択可能

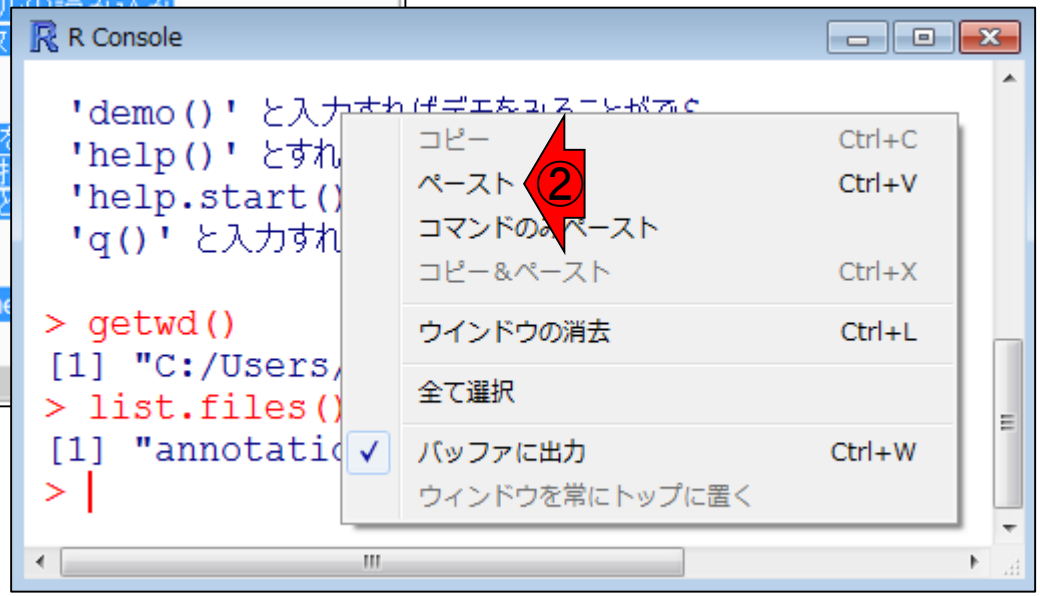
イントロ | 一般 | 任意のキーワードを含む行を抽

例えばタブ区切りテキストファイルが手元にあり、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_f1に格納(アノテーショ
in_f2 <- "genelist1.txt"                     してin_f2に格納(リストファイ
out_f <- "hoge1.txt"                         してout_fに格納
param <- 1                                   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,                    in_f1で指定したファイルの読
keywords <- readLines(in_f
dim(data)

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

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





実行結果

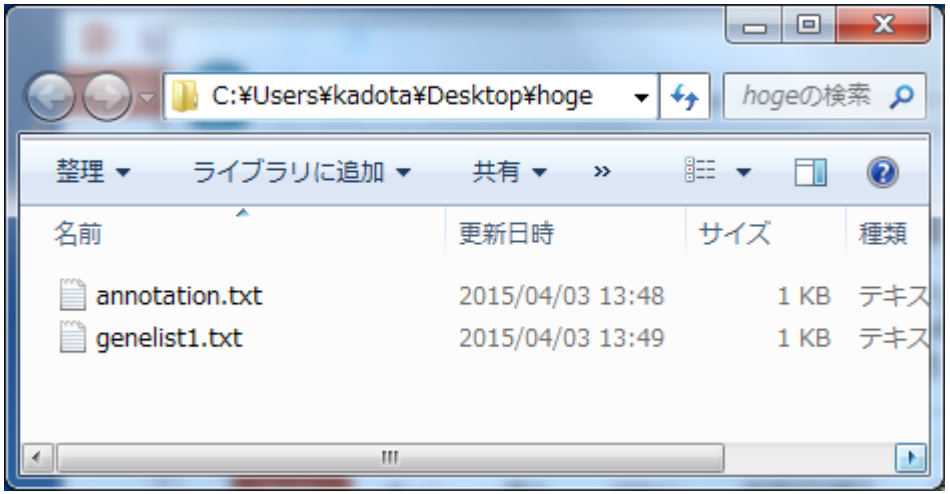
①コピペ実行後にlist.files()。②出力ファイル名として指定したhoge1.txtが生成されているのがわかる。「list.files()で表示される結果」と「実行後のhogeフォルダの中身」は当然同じ

R Console

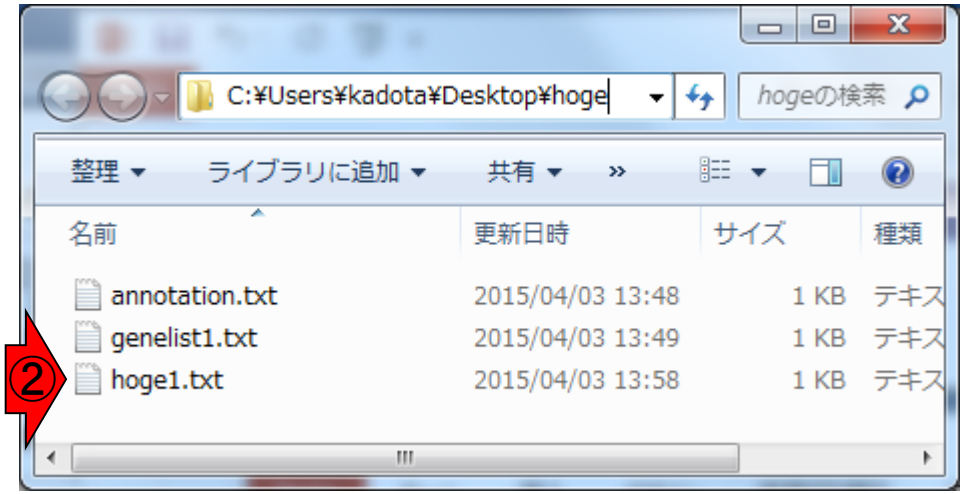
```
> #本番
> obj <- is.element(as.character(data[,pa$
> out <- data[obj,]                      $
> dim(out)                               $
[1] 3 4
>
> #ファイルに保存
> write.table(①t, out_f, sep="\t", appen$
> list.files()
[1] "annotation.txt" "genelist1.txt"
[3] "hoge1.txt"
> |
```

②

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

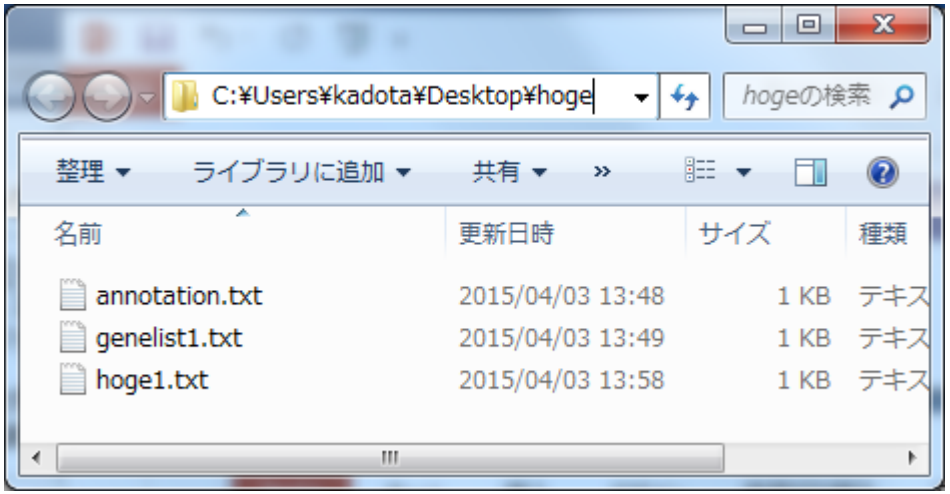
```
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F, quote=F$
> list.files()
[1] "annotation.txt" "genelist1.txt"  "hoge1.txt"
> out
  genename accession description subcellular_location
1    gene1    hoge01  plasma_mem                nuclear
7    gene7    hoge07    tebasaki                nuclear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

①

hoge1.txt

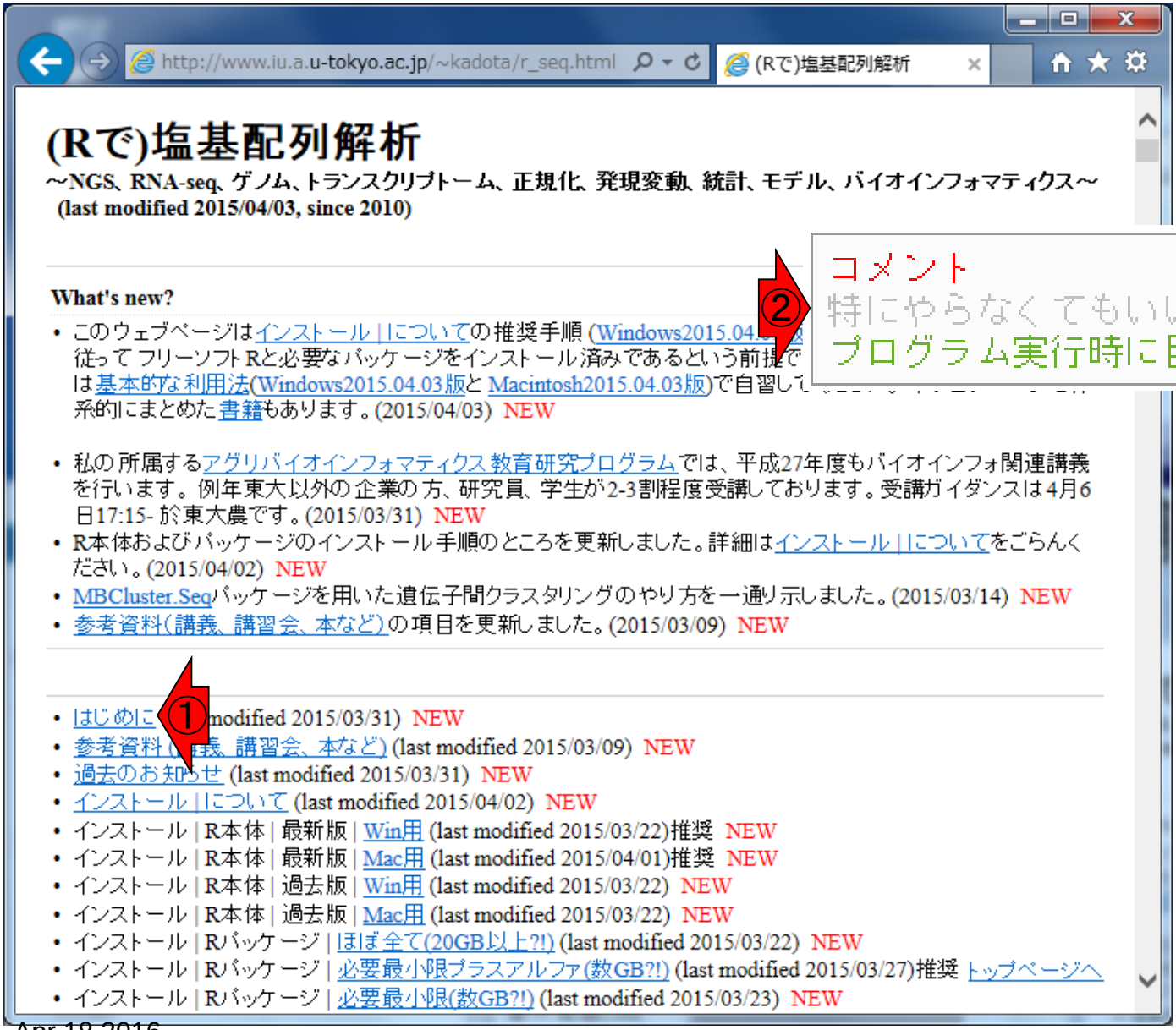
	A	B	C	D
1	genename	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene7	hoge07	tebasaki	nuclear
4	gene9	hoge09	nihonshu	nuclear

実行後のhogeフォルダ



色の説明

Rコード中の色の使い分けについて説明します
①はじめに、のところに②の情報があります



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(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

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)             #in_f2で指定したファイ
dim(data)                                #オブジェクトdataの行

#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 <- 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

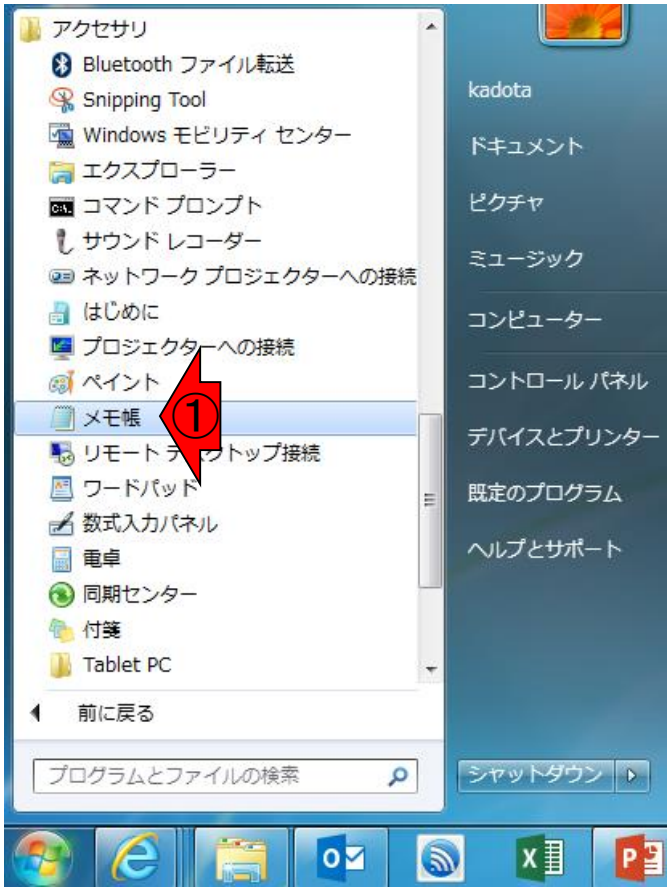
	A	B	C	D
1	genename	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08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

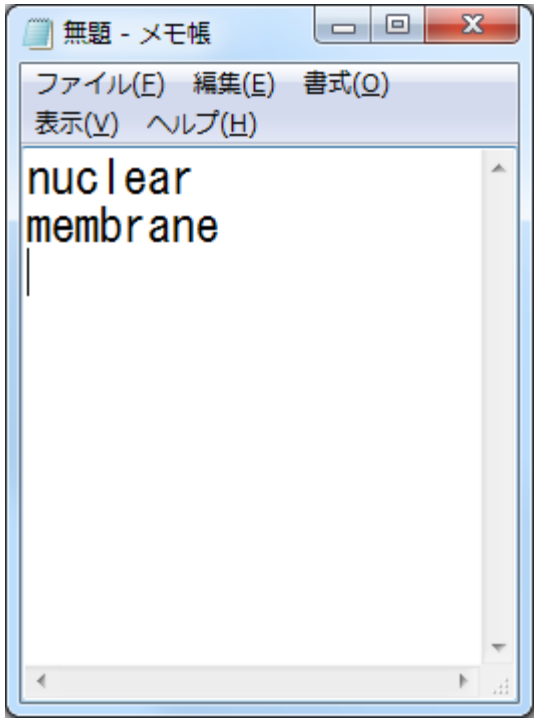
	A	B	C	D
1	genename	accession	description	subcellular_location
2	gene1	hoge01	plasma_mem	nuclear
3	gene2	hoge02	hohinu	membrane
4	gene3	hoge03	agribio	endoplasmic
5	gene4	hoge04	genesis	endoplasmic
6	gene5	hoge05	kamo	membrane
7	gene6	hoge06	netteba	humei
8	gene7	hoge07	tebasaki	nuclear
9	gene8	hoge08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

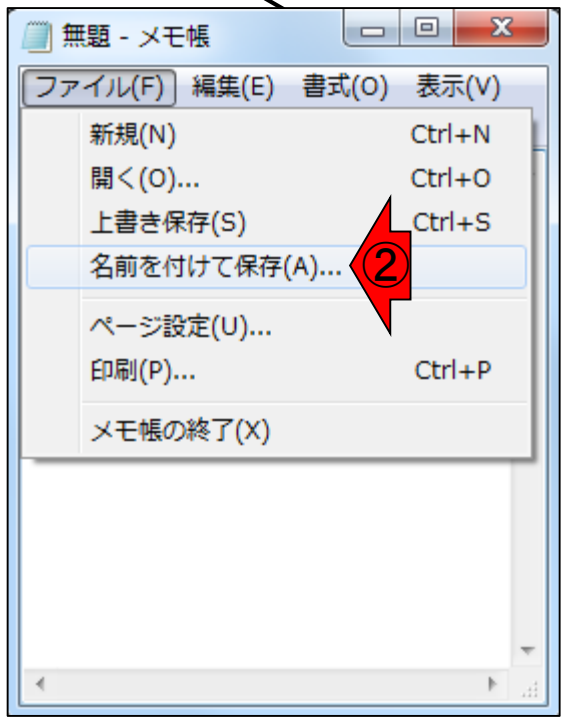
解答例

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

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ペ



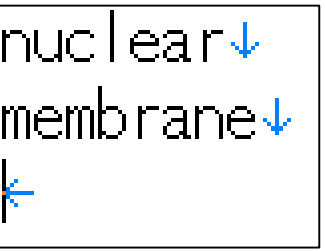




一連の作業手順を記述したスクリプトを1つのファイルとして保存することをお勧め

解答例

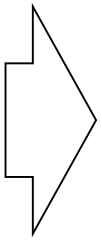
1. 目的のキーワードリストを含むファイルを作成し(例:list.txt)
2. 該当箇所を変更し、Rコンソール画面上でコピペ



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 <- "hoge1.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)

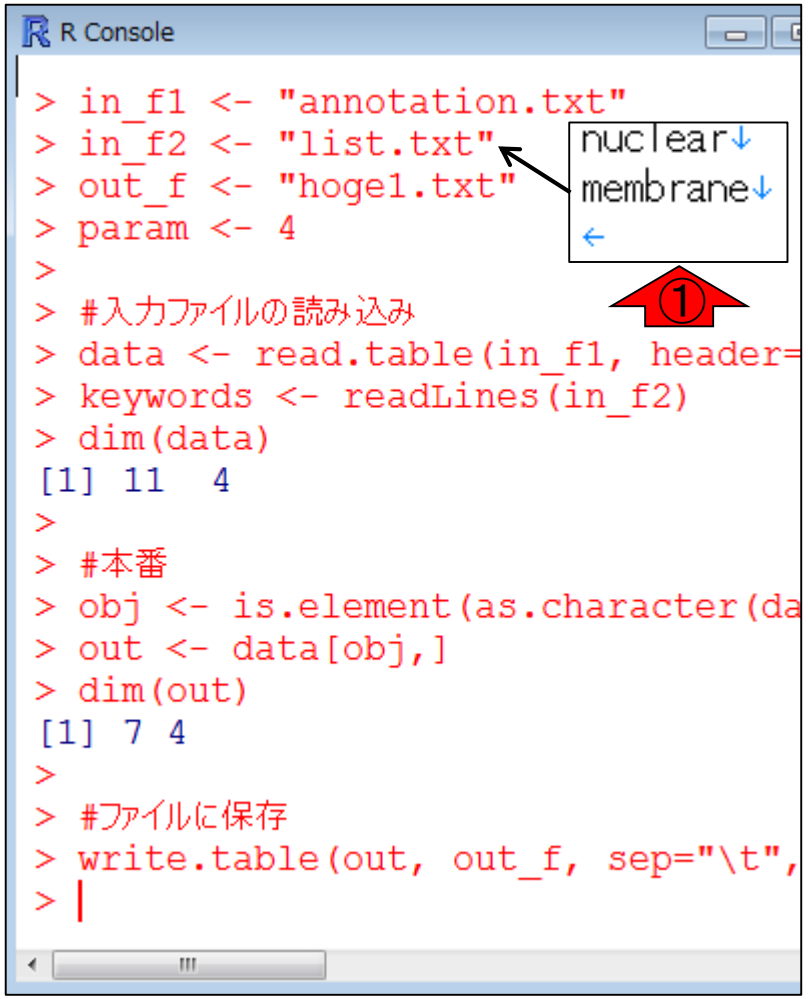
```
in_f1 <- "annotation.txt"
in_f2 <- "list.txt"
out_f <- "hoge1.txt"
param <- 4

#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="¥
```

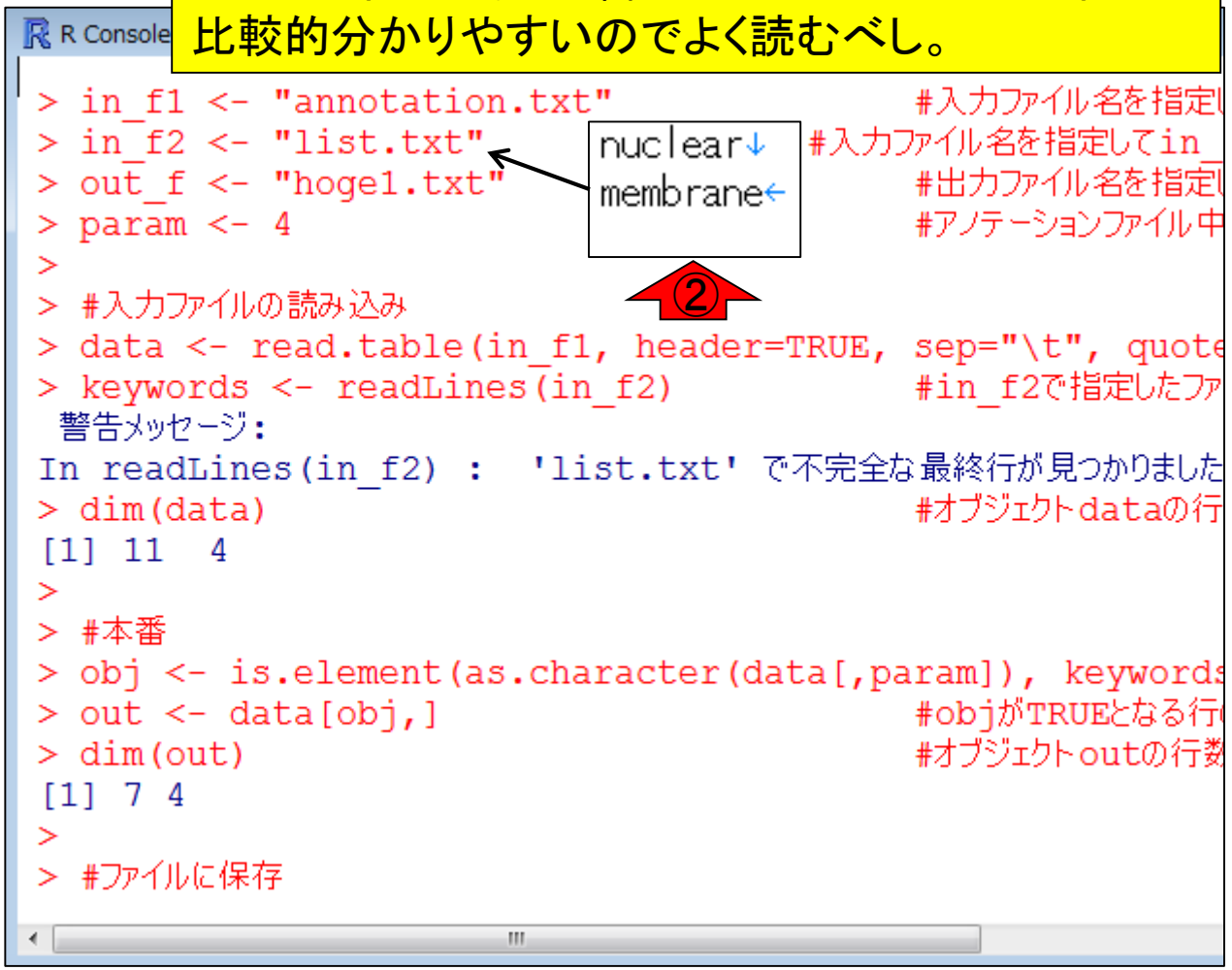
Contents

- 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よい。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。



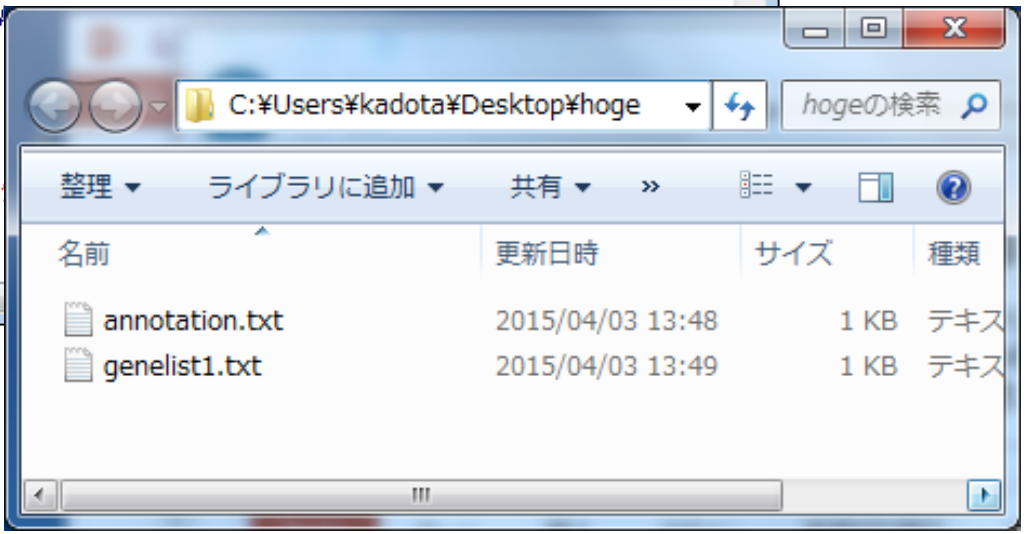


ありがちなミス1

作業ディレクトリの変更を忘れているため、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"                    #入力ファイル名を指定してin_f1に格納($
> in_f2 <- "genelist1.txt"                     #入力ファイル名を指定してin_f2に格納($
> 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$
>
> #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
 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
```

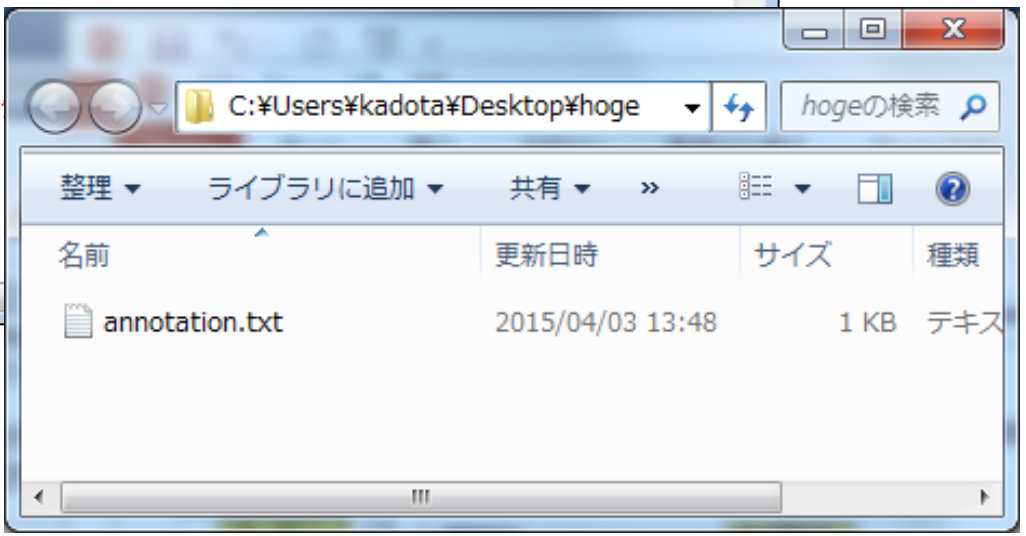


ありがちなミス2

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、それ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"                    #入力ファイル名を指定してin_f1に格納($
> in_f2 <- "genelist1.txt"                     #入力ファイル名を指定してin_f2に格納($
> 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$
>
> #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
```

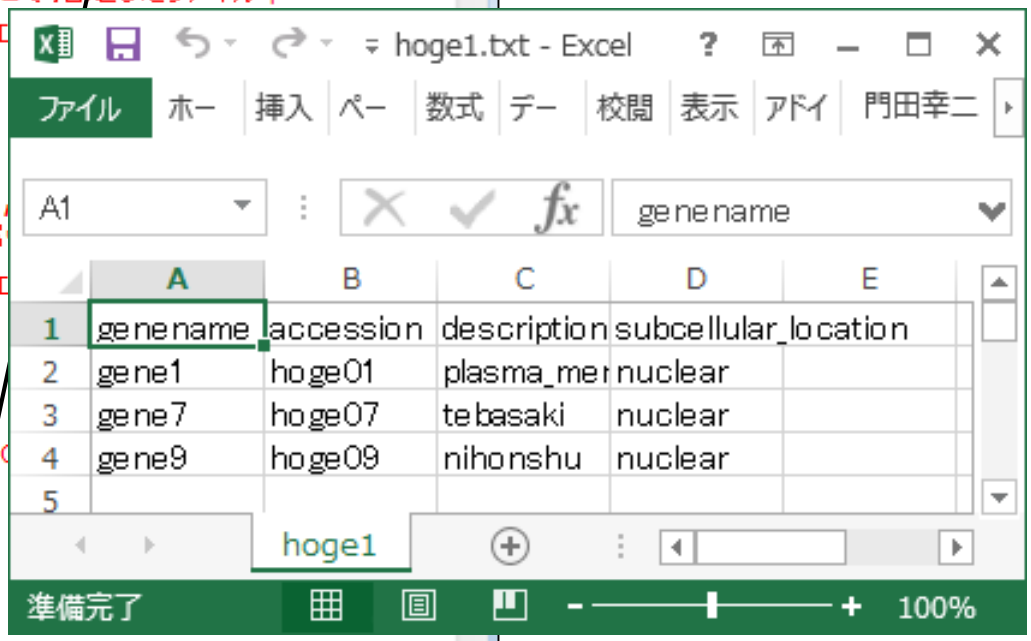


ありがちなミス3

出力予定のファイル名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、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。

R Console

```
> #入力ファイルの読み込み
> data <- read.table(in_f1, header=TRUE, sep="\t", quote="")#in_f$
> 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]),
> out <- data[obj,]                                         #objが
> dim(out)                                                  #オブジェ
[1] 3 4
>
> #ファイルに保存
> write.table(out, out_f, sep="\t", append=F,
 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
>
```

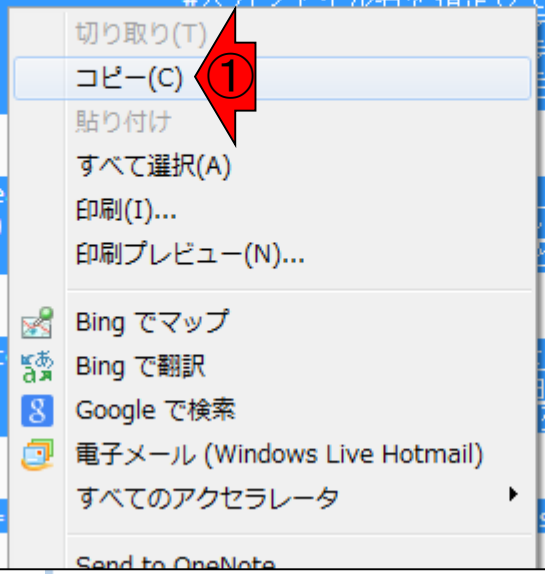


ありがちなミス4

①実行スクリプトをコピーする際、最後の行のところで改行を含ませずにR 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。コピペ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、リス
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"
out_f <- "hoge1.txt"
param <- 1

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

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

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




```
$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


$end=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

コードの中身を説明します。
黒枠部分を再度コピペ。

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にあり、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"
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, heade
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 <- "hoge1.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

①in_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ー部分
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

```
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
keywords <- readLines(in_f2)
dim(data)
```

```
#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
```

①
②
③
④

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

行列data

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。

R Console

②

```
> 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           endo
5     gene5    hoge05        kamo              m
6     gene6    hoge06     netteba
7     gene7    hoge07    tebasaki
8     gene8    hoge08       biiru
9     gene9    hoge09    nihonshu
10   gene10    hoge10      agene1              m
11   gene11    hoge11      iyaaaa           endo
>
```

③

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

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"                    #入力ファイル名を指定してin_f1に格納
in_f2 <- "genelist1.txt"                     #入力ファイル名を指定してin_f2に格納
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

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

②

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      agene1               membrane
11   gene11     hoge11      iyaaaa           endoplasmic
>
```

①

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③6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
   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
8     gene8    hoge08       biiru
9     gene9    hoge09    nihonshu
10   gene10    hoge10      agene1
11   gene11    hoge11      iyaaaa           endo
> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
>
```

③

annotation.txt

	A	B	C	D
② 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
① 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

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

上矢印キーを押すと、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!**

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ヘッダー行がついている

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, ]
  genename accession description subcellular_location
2    gene2    hoge02      hohinu              membrane
>
```

①

②

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

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列目の情報のみ抽出。

R Console

```
5      gene5     hoge05        kamo              r
6      gene6     hoge06      netteba
7      gene7     hoge07     tebasaki
8      gene8     hoge08        biiru
9      gene9     hoge09     nihonshu
10    gene10     hoge10       agene1              r
11    gene11     hoge11       iyaaaa          endo
> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
> data[2, ]
  genename accession description subcellular_lo
2    gene2     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
>
```

①

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

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]。①param列目の情報のみ抽出。②param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          endo
> data[6, 4]
[1] humei
Levels: endoplasmic humei membrane nuclear
> data[2, ]
  genename accession description subcellular_lo
2    gene2    hoge02      hohinu             me
> 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 gene...
> param
[1] 1
>
```

①

②

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

②

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

参考

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

①

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Tabキーを押す。

R Console

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

②

annotation.txt

	A	B	C	①
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

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)
  genename accession description subcellular_location
1    gene1    hoge01  plasma_mem              nuclear
2    gene2    hoge02      hohinu             membrane
3    gene3    hoge03     agribio           endoplasmic
> head(data, n=1)
  genename 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 endopl
 [5] membrane    humei       nuclear     nuclea
 [9] nuclear     membrane    endoplasmic
Levels: endoplasmic humei membrane nuclear
>
```

②

annotation.txt

	A	B	C	① 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

参考

Tips：table関数

table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 Console

```
> 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	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

参考

Tips：ソート

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。

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
>
```

①

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

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

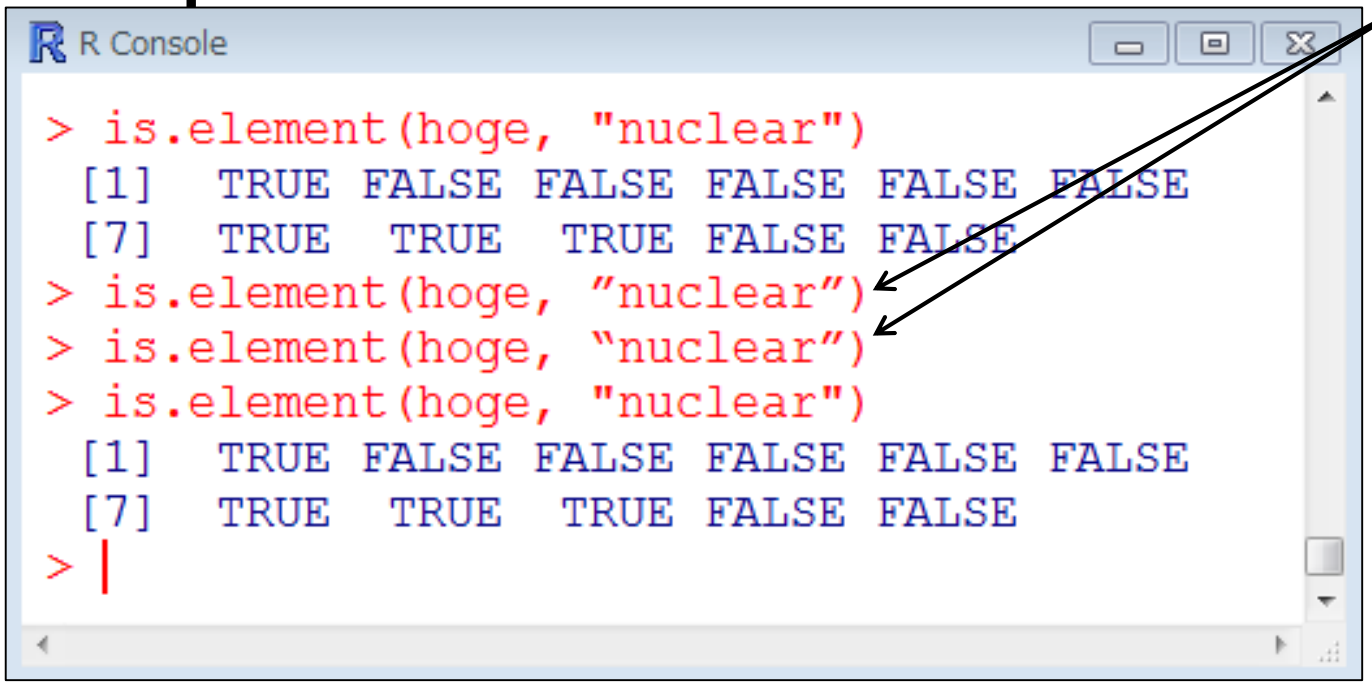
```
> hoge
 [1] nuclear     membrane    endoplasmic endoplasmic
 [5] membrane    humei       nuclear     nuclear
 [9] nuclear     membrane    endoplasmic
Levels: endoplasmic humei membrane nuclear
> is.element(hoge, "nuclear")
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[10] FALSE FALSE
> 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
>
```

①

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

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。

目的をおさらい

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

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
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
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)①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
```

コード内部の説明

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

```
#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
> data[obj,]
  genename accession description subcellular_location
1    gene1    hoge01  plasma_mem              nuclear
7    gene7    hoge07    tebasaki              nuclear
9    gene9    hoge09    nihonshu              nuclear
>
```

①

②

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]),
out <- data[obj,]                              #objがTRUEとなる行の抽出の元結果をoutに格納
dim(out)                                       #オブジェクトoutの行数と列数を表示
```

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
>
```

例題1と12は手順が異なるだけで実質的に同じ

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(ou

#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
```

①

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(アノ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
```

② ③

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

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

```
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                             #アノテーションフ
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
```

①

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ー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(アノ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
```

②

③

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

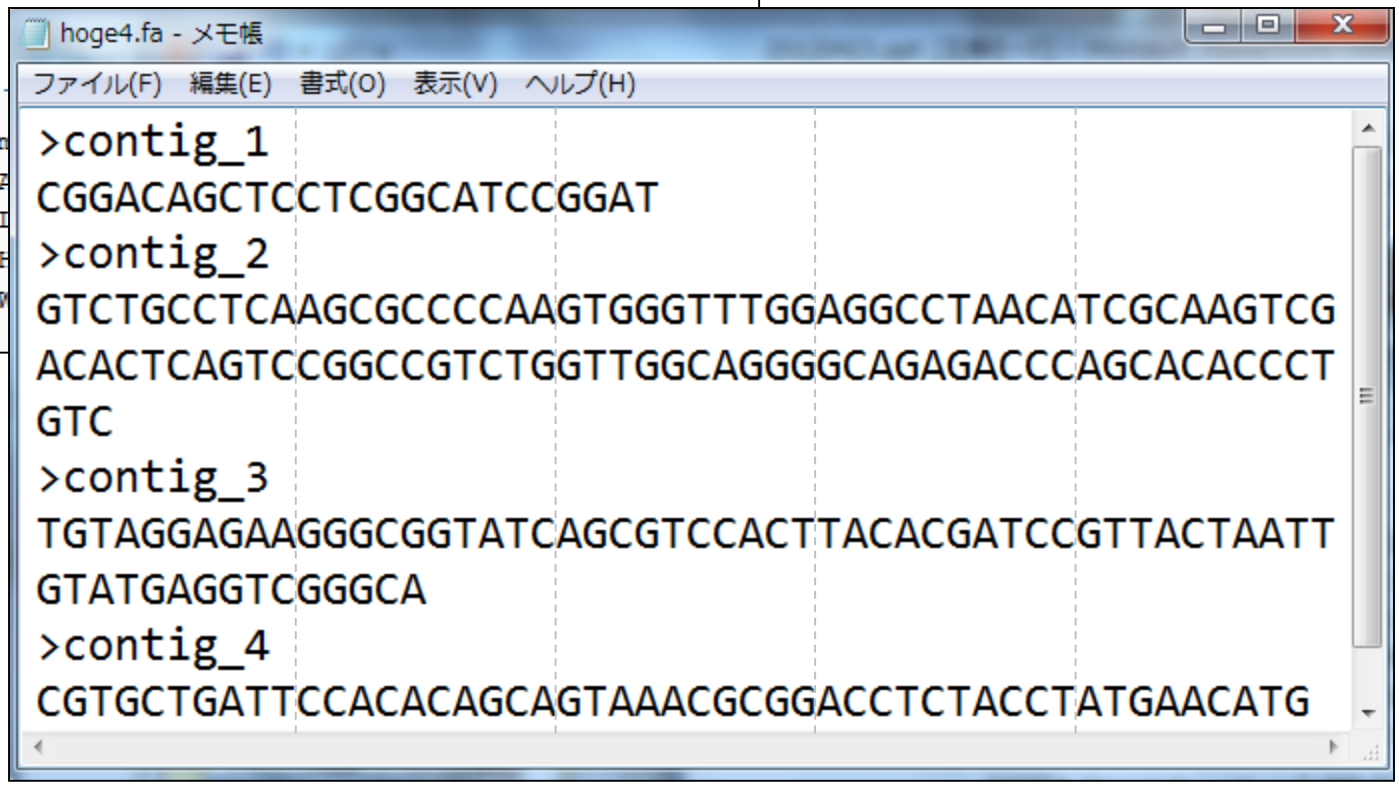
Rでmulti-FASTAファイルを読み込んで自在に解析できます。ゲノム配列解析≒FASTA形式ファイルの解析。ここでは全体像を完全に把握すべくhoge4.faファイル（ダウンロードは後述）を仮想ゲノム配列ファイルとして取り扱う

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フォーマットはプレーンテキストである。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られ、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| cytochron
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATA
EWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHI
LLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLH
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTW
IENY
```



ゲノム配列

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

(Rで)塩基配列解析

～NGS、RNA-seq、ゲノム、トランスクリプトーム、正規化、発現変動、統計、モデル、バイオインフォマティクス～

(last modified 2015/04/04, since 2010)

What's new?

- このウェブ
  トRと必要な
  (Windows2
  (2015/04/03
- 私の所属す
  例年東大以
  (2015/03/31
- R本体およ
  (2015/04/02
- MBCluster.

- (削除予定)イントロ | 一般 | 任意の長さの連続塩基の出現頻度情報を取得 (last modified 2015/02/19)
- 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 2014/05/28)
- 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 | プロモーター配列 | 公共DBから (last modified 2014/04/02)
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般
- イントロ | 一般
- イントロ | NGS
- イントロ | NGS
- イントロ | NGS
- イントロ | NGS
- イントロ | NGS
- イントロ | NGS

① (1)

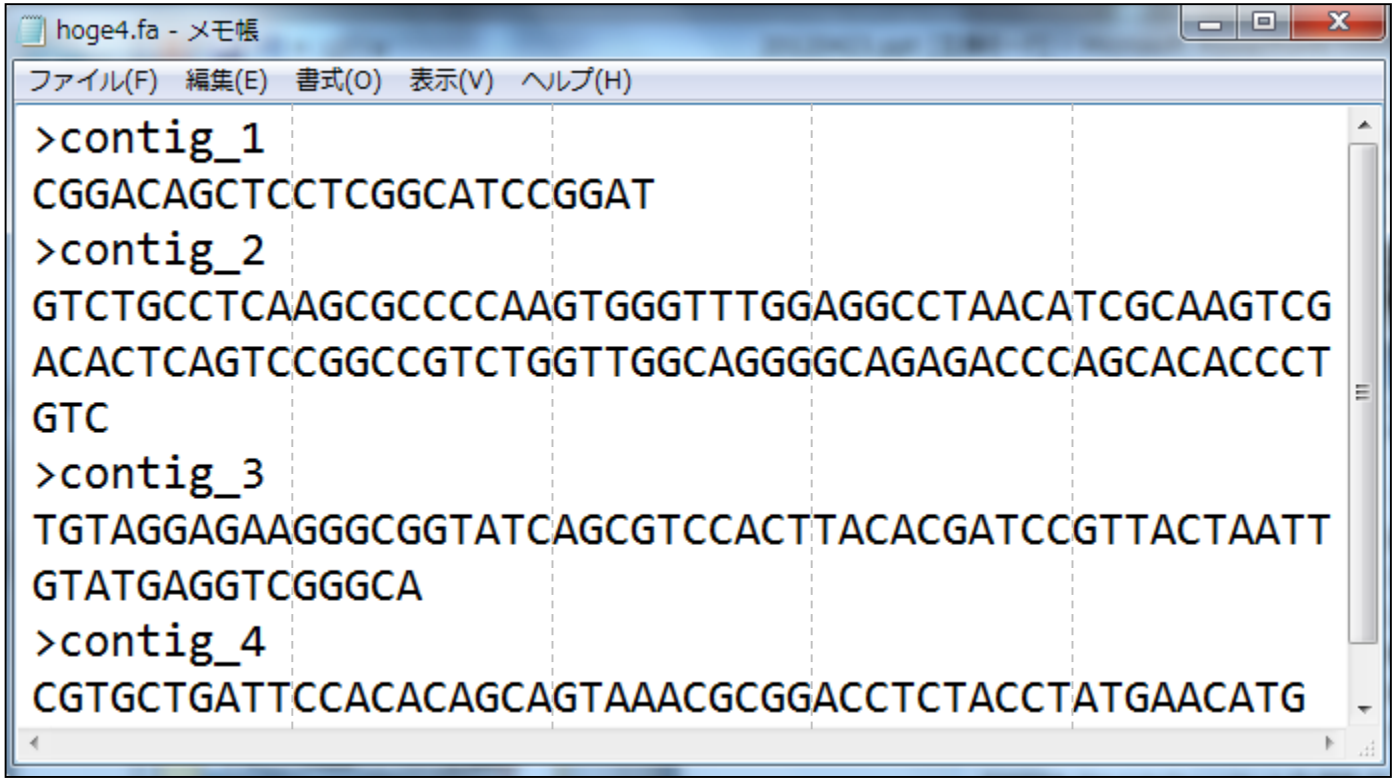
イントロ | 一般 | 配列取得 | ゲノム配列 | 公共DBから

- UCSCの Sequence and Annotation Downloads (Karolchik et al., Nucleic Acids Res., 2014)
  - 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 (Flicek et al., Nucleic Acids Res., 2014)
  - 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



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、コピペで実行した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) (
- 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 mo
- イントロ | NGS | 読み込み | BSgenome | 基本情報を取得 (la odified 2015/09/12)
- イントロ | NGS | 読み込み | FASTA形式 | 基本情報を取得 ① modified 2015/09/12)
- イントロ | NGS | 読み込み | FASTA形式 | description行の記 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 | Illumina(1.* seq.txt (last modified 2013/06/13)
- イントロ | NGS | 読み込み | Illu
- イントロ | ファイル形式の変換
- イントロ | ファイル形式の変換
- イントロ | ファイル形式の変換
- イントロ | ファイル形式の変換
- イントロ | ファイル形式の変換

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。
「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:

②

```
in_f <- "hoge4.fa"                          #入力ファイル名を指定してin_fに格納
out_f <- "hoge1.txt"                        #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))              #コンティグの「トータルの長さ」を取得
Number_of_contigs <- length(fasta)          #「コンティグ数」を取得
Average_len <- mean(width(fasta))           #コンティグの「平均長」を取得
Median_len <- median(width(fasta))          #コンティグの「中央値」を取得
Max_len <- max(width(fasta))                #コンティグの長さの「最大値」を取得
Min_len <- min(width(fasta))                #コンティグの長さの「最小値」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（その他のファイルがあってもよい）

イントロ | NGS | 読み込み | FASTA形式 | 基本情報

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。
「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:

①

```
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))        #コンティグの「トータルの長さ」を取得
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
N50 <- sorted[obj][1]
```

R Console

②
```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
```

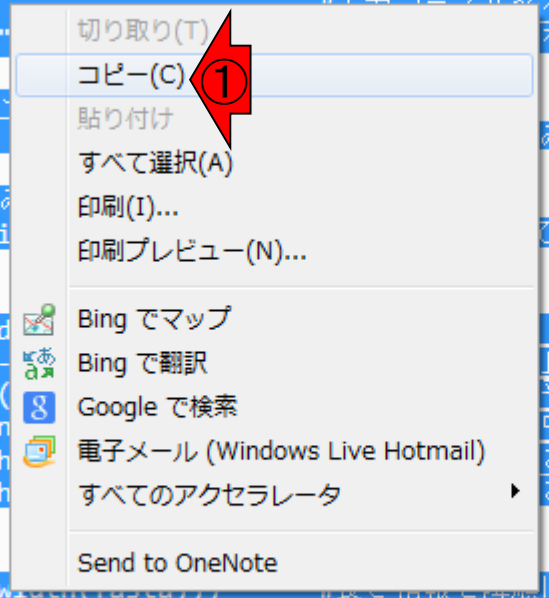
③
```
> list.files()
[1] "hoge4.fa"
>
```

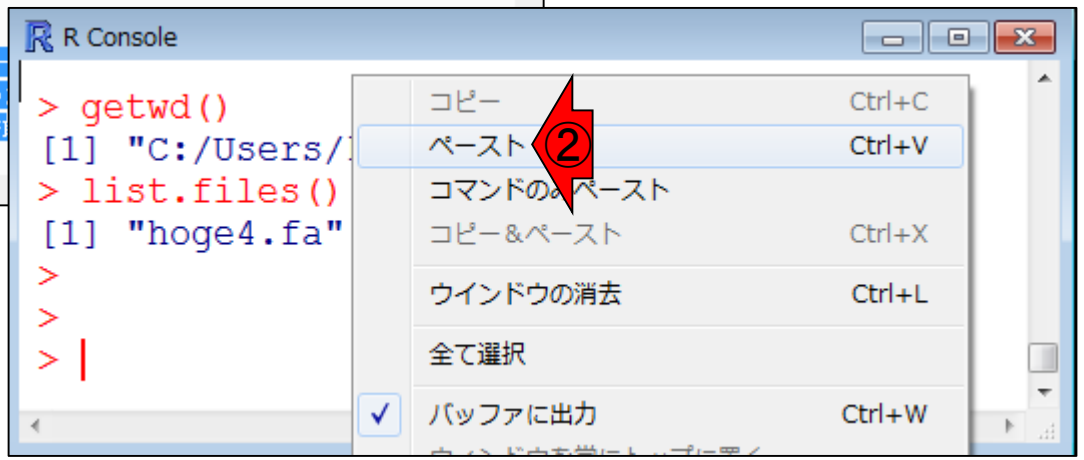
コピペ

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"                                    "]ホコーイルタを指定してin fに格納
out_f <- "hoge1.txt"                                  を指定してout_fに格納

#必要なパッケージをロ
library(Biostrings)                                   み込み

#入力ファイルの読み込
fasta <- readDNAStri                                  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(wid                                  トータルの長さ」を取得
Number_of_contigs <-                                  」を取得
Average_len <- mean(                                  平均長」を取得
Median_len <- median                                  中央値」を取得
Max_len <- max(width                                  さの「最大値」を取得
Min_len <- min(width                                  さの「最小値」を取得

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





実行結果

コピペ後に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

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")#

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

#本番(N50情報取得)
sorted <- rev(sort(width(fasta)))     #長さ情報
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満
N50 <- sorted[obj][1]                 #objがTRU
```

②

①

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"
>
```

③

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:

```
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(長い配
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N50に格

#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hogeに格
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
GC_content <- sum(CG)/sum(ACGT)         #トータル

#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
```

②

③

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
     [,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"
>
```

入出力の関係確認

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

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

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

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

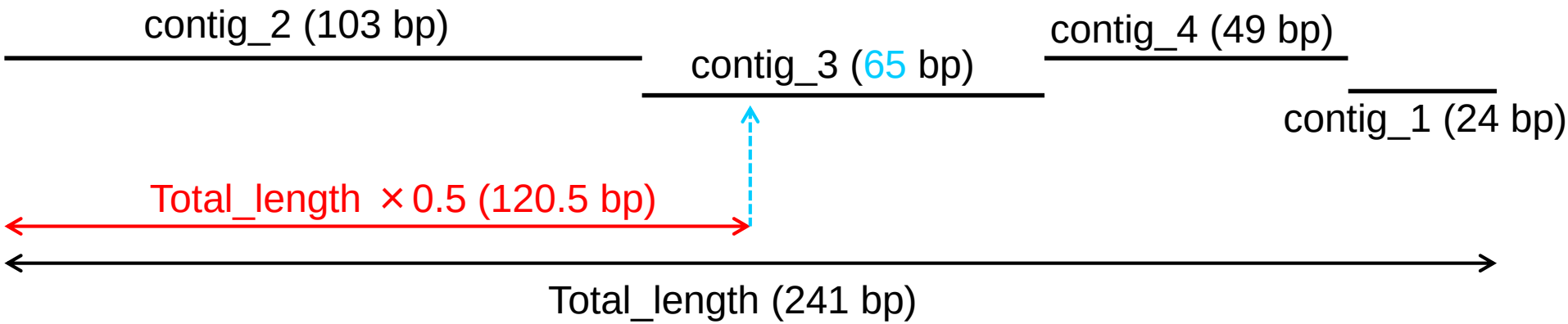
N50

- 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hoge5.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_

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

#本番(N50情報取得)
sorted <- rev(sort(width(fast
obj <- (cumsum(sorted) >= Tot
N50 <- sorted[obj][1]
```

hoge5.fa

```
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAAC                          CATG
>contig_5
CGTGCTGATTCCACACAGCAGTAAAC                          CATG
>contig_6
AACGTTGCA
>contig_7
AACGTTGCAG
>contig_8
AANNCGTTNGCAGNNN
```

講義日程（平成28年度）

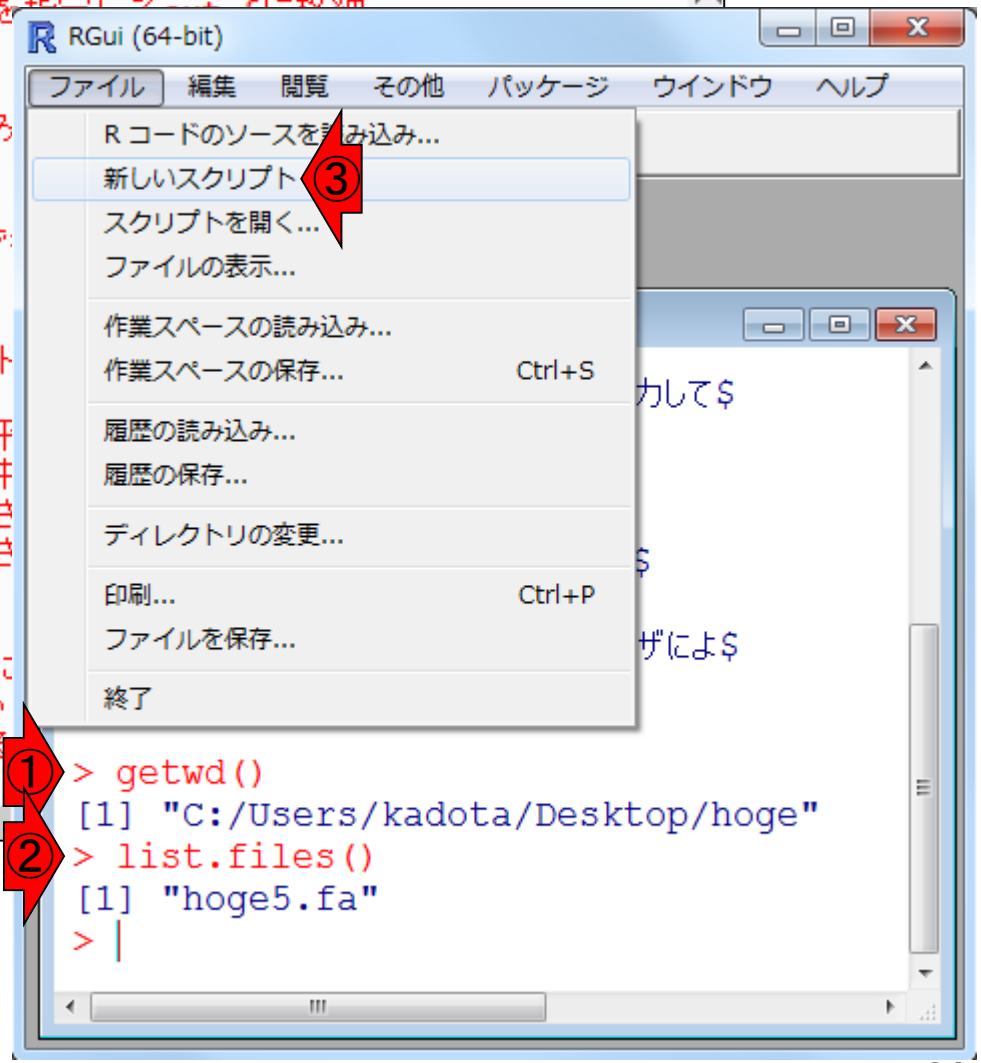
1. 平成28年04月11日（PC使用）
   講師：嶋田 透
   講師：門田幸二
   バイオインフォマティクス基礎知識
   講義資料PDF(Win版；完全版)
   講義資料PDF(Mac版；Rの説明部分のみ)
2. 平成28年04月18日（PC使用）
   講師：門田幸二
   講義資料PDF
   (Rで)塩基配列解析
   hoge5.fa（課題用）
3. 平成28年04月25日（PC使用）
   講師：嶋田 透
   講師：門田幸二
4. 平成28年05月02日（PC使用）
   講師：勝間 進

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(hoge5.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"                          #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

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

#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))              #コンティグの「ト
Number_of_contigs <- length(fasta)          #「コンティグ数」
Average_len <- mean(width(fasta))           #コンティグの「平
Median_len <- median(width(fasta))          #コンティグの「中
Max_len <- max(width(fasta))                #コンティグの長さ
Min_len <- min(width(fasta))                #コンティグの長さ

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



課題1

テンプレートコードをコピペし、
①入力ファイル名部分を変更

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.t

#必要なパッケージを
library(Biostring

#入力ファイルの読み
fasta <- readDNAS

#本番(基本情報取得)
Total_len <- sum(
Number_of_contigs
Average_len <- me
Median_len <- med
Max_len <- max(wi
Min_len <- min(wi

#本番(N50情報取得)
sorted <- rev(sor
obj <- (cumsum(so
N50 <- sorted[obj
```

RGui (64-bit)

ファイル 編集 パッケージ ウインドウ ヘルプ

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge5.fa"
>
```

①

無題 - Rエディタ

```
in_f <- "hoge5.fa"                          #入力ファイル名を指定
out_f <- "hoge1.txt"                        #出力ファイル名を指定

#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

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

コピペ

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.t

#必要なパッケージを
library(Biostring

#入力ファイルの読み
fasta <- readDNAS

#本番(基本情報取得)
Total_len <- sum(
Number_of_contigs
Average_len <- me
Median_len <- med
Max_len <- max(wi
Min_len <- min(wi

#本番(N50情報取得)
sorted <- rev(sor
obj <- (cumsum(so
N50 <- sorted[obj
```

RGui (64-bit)

ファイル 編集 パッケージ ウインドウ ヘルプ

③

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge5.fa"
>
```

②

無題 - Rエディタ

```
in_f <- "hoge5.fa"                    #入力ファイル名を指定
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 <- readDNAStringSe

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

カーソル行または選択中の R コードを実行 Ctrl+R
やり直し Ctrl+Z
カット Ctrl+X
コピー Ctrl+C
ペースト Ctrl+V
消去
全て選択 Ctrl+A

①

課題1の実行結果

コピペ後に①出力ファイル内容に相当するtmpオブジェクトを表示

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.t

#必要なパッケージを
library(Biostring

#入力ファイルの読み
fasta <- readDNAS

#本番(基本情報取得)
Total_len <- sum(
Number_of_contigs
Average_len <- me
Median_len <- med
Max_len <- max(wi
Min_len <- min(wi

#本番(N50情報取得)
sorted <- rev(sor
obj <- (cumsum(so
N50 <- sorted[obj
```

RGui (64-bit)

ファイル 編集 閲覧 その他 パッケージ ウインドウ ヘルプ

R Console

```
> tmp <- rbind(tmp, c("N50", N50))
> tmp <- rbind(tmp, c("GC content", GC_conte$
> write.table(tmp, out_f, sep="\t", append=F$
> tmp
      [,1]                  [,2]
[1,] "Total length (bp)" "325"
[2,] "Number of contigs" "8"
[3,] "Average length"    "40.625"
[4,] "Median length"     "36.5"
[5,] "Max length"        "103"
[6,] "Min length"        "9"
[7,] "N50"               "65"
[8,] "GC content"        "0.557993730407524"
>
```

①

①

```
#入力ファイル名を指定
#出力ファイル名を指定

#パッケージの読み込み

rmat="fasta")#in_fで指

#配列の「トータルの長さ
#「配列数」を取得
Average_len <- mean(width(fasta))   #配列の「平均長」を取
Median_len <- median(width(fasta))  #配列の「中央値」を取
```

コード内部の説明

コードの中身を説明します。
①黒枠部分を再度コピペ。

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))              #
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]                       #
```

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mu

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

#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

#本番(基本情報取得)
Total_len <- sum(width(fasta))        #コンティグの「トータ
Number_of_contigs <- length(fasta)    #「コンティグ数」を取
Average_len <- mean(width(fasta))     #コンティグの「平均長
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]
```

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

R Console

①

```
> 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))              #コンティグの「トータルの長さ」を取得
Number_of_contigs <- length(fasta)          #「コンティグ数」を取得
Average_len <- mean(width(fasta))           #コンティグの「平均長」を取得
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]                       #
```

②

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
>
```

①

コード内部の説明

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

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.
N50 <- sorted[obj][1]
```

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

①50 bp以上のコンティグからなるサブセットの抽出ができそうだ!

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

• 前処理 | フィルタリング | 指定した長さ以上の配列を抽出　参考

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 2013/0
- 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 2015/02/26)
- 前処理 | フィルタリング | 任意のID
- 前処理 | フィルタリング | Illuminaの
- 前処理 | フィルタリング | GFF/GTF
- 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 modifie
- アセンブル | ゲノム用 (last modifie

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…

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ペ。

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 ① modified 2015/04/06) NEW
- イントロ | 一般 | 指定したID(染色体やdescript 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
- イントロ | 一般 | 逆相補鎖(reverse complement)を
- イントロ | 一般 | 逆鎖(reverse)を取得 (last modifie
- イントロ | 一般 | 2連続塩基の 出現頻度情報を取
- イントロ | 一般 | 3連続塩基の 出現頻度情報を取

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ペ。

②

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だが、②出力ファイルのfastaオブジェクトは、当然サブセット抽出後なので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
param <- c(3, 9)                           #抽出したい範

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

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

#本番
fasta <- subseq(fasta, param[1], param[2])#paramで
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$
  A DNAStringSet instance of length 1
    width seq                                  names       $
[1]    12 AGTGACGGTCTT                         kadota
>
> #本番
> fasta <- subseq(fasta, param[1], param[2])#pa$
> 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="fa$
> fasta
  A DNAStringSet instance of length 1
    width seq                                  names
[1]     7 TGACGGT                              kadota
>
```

①

②

②

入力：sample1.fasta

```
>kadota
AGTGACGGTCTT
```

出力：hoge1.fasta

```
>kadota
TGACGGT
```

Tips：関数のオプション

subseq関数実行時に、①数値を直接指定してもいいし、②オプション名を明記してもよい

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

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

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

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

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
> subseq(fasta, 3, 9)                          ①
  A DNAStringSet instance of length 1
    width seq                    names
[1]     7 TGACGGT                kadota
> subseq(fasta, start=3, end=9)                ②
  A DNAStringSet instance of length 1
    width seq                    names
[1]     7 TGACGGT                kadota
>
```

入力：sample1.fasta

```
>kadota
AGTGACGGTCTT
```

出力：hoge1.fasta

```
>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

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

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

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

R Console

```
> subseq(fasta, start=3, end=12)
  A DNAStringSet instance of length 1
    width seq                    names
[1]    10 TGACGGTCTT             kadota
> subseq(fasta, start=3, end=13) ①
 以下にエラー .Call2("solve_user_SEW", refwidths, $
  solving row 1: 'allow.nonnarrowing' is FALSE and$
> subseq(fasta, start=3, width=5) ②
  A DNAStringSet instance of length 1
    width seq                    names
[1]     5 TGACG                  kadota
>
```

入力：sample1.fasta

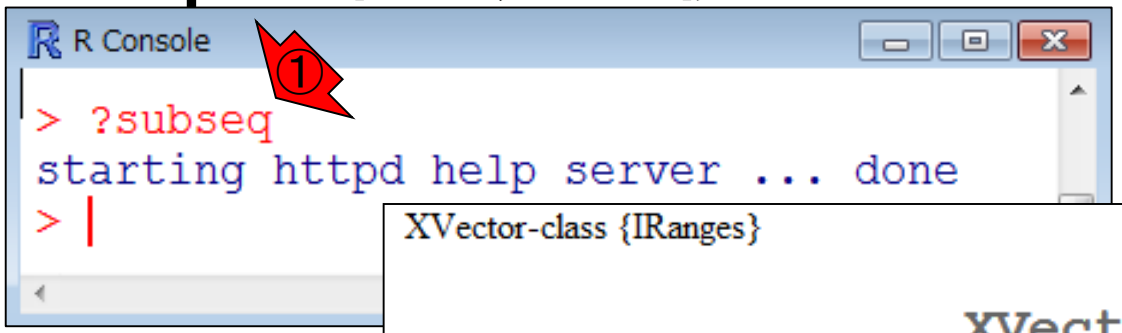
```
>kadota
AGTGACGGTCTT
```

出力：hoge1.fasta

```
>kadota
TGACGGT
```

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。ページの下のほうに、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



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)

①

イントロ | 一般 | 指定した範囲の配列を取得 NEW

②

single-FASTA形式やmulti-F
は、「この染色体の、ここから、
場合に、chr3の20000から500000
chr8の配列のみ抽出といったやり
に、*.fastaという拡張子が*.txtに
「ファイル」-「ディレクトリの変更」

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
```

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。

```
in_f1 <- "hoge4.fa"                          #入力ファイル名を指定してin_f1に格納(multi-FAS
in_f2 <- "list_sub2.txt"                     #入力ファイル名を指定してin_f2に格納(リストファ
out_f <- "hoge4.fasta"                       #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)                    #in_f2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
hoge <- NULL                                 #最終的に得る結果を格納するためのプレースホルダ
for(i in 1:nrow(posi)){                      #length(posi)回だけループを回す
    obj <- names(fasta) == posi[i,1]         #条件を満たすかどうかを判定した結果をobjに格納
    hoge <- append(hoge, subseq(fasta[obj], start=posi[i,2], end=posi[i,3]))#subse
}
fasta <- hoge                                #hogeの中身をfastaに格納
fasta                                        #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

②右のサイドバーを下に移動させるとGC含量計算部分を見られる

①

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。
「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))              #コンティグの「トータルの長さ」を取得
Number_of_contigs <- length(fasta)          #「コンティグ数」を取得
Average_len <- mean(width(fasta))           #コンティグの「平均長」を取得
Median_len <- median(width(fasta))          #コンティグの「中央値」を取得
Max_len <- max(width(fasta))                #コンティグの長さの「最大値」を取得
Min_len <- min(width(fasta))                #コンティグの長さの「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(長い
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N50に
```

②

GC含量計算部分の説

①fastaオブジェクトを入力として、②配列全体のGC含量(57.68%)を得る部分を説明。2015年9月12日に黒枠部分の記述内容を若干変更しているが、入力ファイルがsingle-FASTAでない限りうまくいくので、ここでは昔の記述内容で解説する

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

```
sorted <- rev(sort(width(fasta)))       #長さ情報を降順にソートした
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
```

①

```
#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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,T
GC_content <- sum(CG)/sum(ACGT)         #トータル
```

```
#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"
>
```

②

GC含量計算部分の説明

黒枠部分を再度コピペしたのち、①fastaオブジェクトの中身を表示させたところ

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))              #コンティグの「
Number_of_contigs <- length(fasta)          #「コンティグ数」
Average_len <- mean(width(fasta))           #コンティグの「
Median_len <- median(width(fasta))          #コンティグの「
Max_len <- max(width(fasta))                #コンティグの長
Min_len <- min(width(fasta))                #コンティグの長

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

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$
>
> #必要なﾊﾟｯｹｰｼﾞをﾛｰﾄﾞ
> library(Biostrings)                                   #ﾊﾟｯｹ$
>
> #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
>
```

①

GC含量計算部分の説明

①alphabetFrequency関数は、塩基ごとの出現回数を返す

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(hoge4.fa)の場合:

```
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

#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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,Tの総数を計算してACGTに格納
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_co
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
```

①

R Console

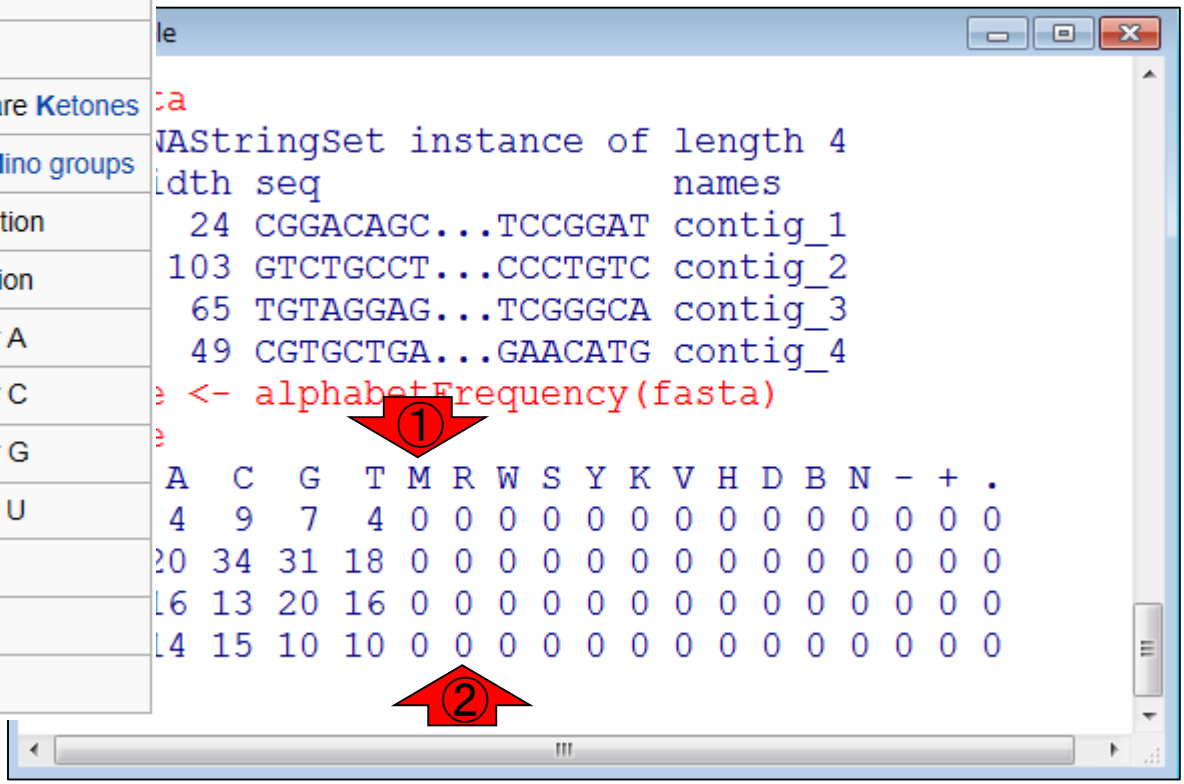
```
> fasta
  A DNAStringSet instance of length 4
    width seq                 names
[1]    24 CGGACAGC...TCCGGAT contig_1
[2]   103 GTCTGCCT...CCCTGTC contig_2
[3]    65 TGTAGGAG...TCGGGCA contig_3
[4]    49 CGTGCTGA...GAACATG contig_4
> hoge <- alphabetFrequency(fasta)
> hoge
      A  C  G  T M R 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
>
```

GC含量計算部分の説明

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

Nucleic Acid Code	Meaning	Mnemonic
A	A	Adenine
C	C	Cytosine
G	G	Guanine
T	T	Thymine
U	U	Uracil
R	A or G ②	puRine
Y	C, T or U	pYrimidines
K	G, T or U	bases which are Ketones
M	A or C ①	bases with aMino groups
S	C or G	Strong interaction
W	A, T or U	Weak 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	Nucleic acid
X	masked	
-	gap of indeterminate length	

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（あるいはコピペ）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(h

```
sorted <- rev(sort(width(fasta)))       #長さ情報を降順にソートした結果をsc
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうかを判定した結
N50 <- 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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,Tの総数
GC_content <- sum(CG)/sum(ACGT)         #トータルのGC含

#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_co
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,
```

②

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
>
```

①

②

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:

```
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

#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
ACGT <- rowSums(hoge[,1:4])            #A,C,G,Tの総数を計算してACGTに格納
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
```

①

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含量計算部分の説明

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。

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

```
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

#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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,Tの総数を計算してACGTに格納
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))
```

①

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
```

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
>
```

①

GC含量計算部分の説

rowSums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。①applyという関数でもrowSumsと同じ結果を得られる。Nを含むhoge5.faでやってみると違いがわかる

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

```
sorted <- rev(sort(width(fasta)))       #長さ情報を降順にソートした結
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうかを判定し
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
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 <
tmp <
tmp <
tmp <
tmp <
tmp <
tmp <
write
```

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
```

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
>
```

①

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

```
sorted <- rev(sort(width(fasta)))       #長さ情報を降順にソートした結
obj <- (cumsum(sorted) >= Total_len*0.5)#条件を満たすかどうかを判定
N50 <- sorted[obj][1]                   #objがTRUEとなる1番最初の要
```

```
#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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,Tの総数を計算してACG
GC_content <- sum(CG)/sum(ACGT)         #トータルのGC含量の情報を取
```

①

```
#ファイルに保存
tmp <- NULL
tmp <- rbind(tmp, c("Total length (bp)", Total_len))
tmp <
tmp <
tmp <
tmp <
tmp <
tmp <
tmp <
write
```

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
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACA
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAG
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCG
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTAT
```

①

②

③

④

R Console

```
> hoge <- alphabetFrequency(fasta)        $
> CG <- rowSums(hoge[,2:3])               $
> ACGT <- rowSums(hoge[,1:4])             $
> GC_content <- sum(CG)/sum(ACGT)         $
> 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含量計算部分の説

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 = 0.6666667。

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

```
sorted <- rev(sort(width(fasta)))       #長さ情報を降順にソートし
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

#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
ACGT <- rowSums(hoge[,1:4])             #A,C,G,Tの総数を計算してACGTに格納
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))
write
```

①

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
```

R Console

①

```
> sum(CG)/sum(ACGT)
[1] 0.5767635
```

②

```
> 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であることが分かる。

- 正規化 | サンプル間 | 3群間 | 複製あり | iDEGES/edgeR(Sun 2013) (last modified 2015/03/30)推奨 NE
- 正規化 | サンプル間 | 3群間 | 複製あり | TMM(Robinson 2010) (last modified 2015/03/30) NEW
- 解析 | 一般 | アラインメント(ペアワイズ; 基礎1) (last modified 2010/6/8)
- 解析 | 一般 | アラインメント(ペアワイズ; 基礎2) (last modified 2010/6/8)
- 解析 | 一般 | アラインメント(ペアワイズ
- 解析 | 一般 | パターンマッチング (last
- 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 | Techn
- 解析 | 基礎 | 平均-分散プロット | Biolo
- 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 | 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 | クラスタリング | 遺伝子間(基礎)
- 解析 | クラスタリング | 遺伝子間(応用)
- 解析 | シミュレーションカウントデータ |
- 解析 | シミュレーションカウントデータ |
- 解析 | シミュレーションカウントデータ |
- 解析 | シミュレーションカウントデータ |

①

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
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CG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ACGT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CG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ACGTに格納(2015年9月12日以降の
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
```

②

配列ごとのGC含量計算

①出力ファイル(hoge1.txt)中の②1番右側の列が**配列ごとのGC含量**

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
#CG <- rowSums(hoge[,2:3])            #C,Gの
#ACGT <- rowSums(hoge[,1:4])          #A,C,G
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#
GC_content <- CG/ACGT*100             #%GC含

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

①

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", "Len$
> 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"
>
```

②

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 ≧ 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
out_f <- "hoge1.txt"                        #出力ファイル名を指定してout_fに格

#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
#CG <- rowSums(hoge[,2:3])                  #C,Gの
#ACGT <- rowSums(hoge[,1:4])                #A,C,G
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#
GC_content <- CG/ACGT*100                   #%GC含

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

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", "Len$
> 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"
>
```

①

②

hoge5.faの配列ごとのGC含量を示せ

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, for

#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
```

hoge5.fa

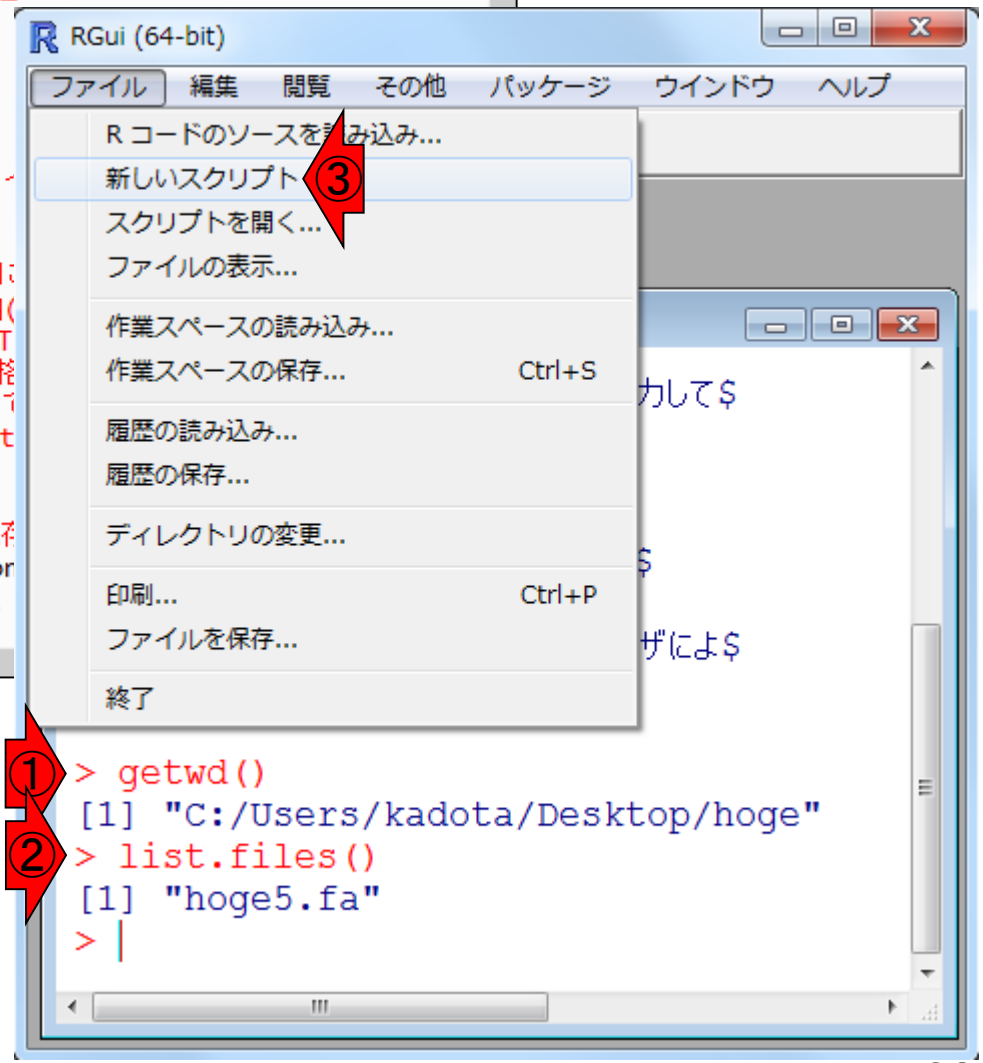
```
>contig_1
CGGACAGCTCCTCGGCATCCGGAT
>contig_2
GTCTGCCTCAAGCGCCCCAAGTGGGTTTGGAGGCCTAACATCGCAAGTCG
ACACTCAGTCCGGCCGTCTGGTTGGCAGGGGCAGAGACCCAGCACACCCT
GTC
>contig_3
TGTAGGAGAAGGGCGGTATCAGCGTCCACTTACACGATCCGTTACTAATT
GTATGAGGTCGGGCA
>contig_4
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
>contig_5
CGTGCTGATTCCACACAGCAGTAAACGCGGACCTCTACCTATGAACATG
>contig_6
AACGTTGCA
>contig_7
AACGTTGCAG
>contig_8
AANNCGTTNGCAGNNN
```

課題2

①作業ディレクトリの確認、②解析したいファイル(hoge5.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"                          #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

#本番
hoge <- alphabetFrequency(fasta)            #A,C,G,T,..の数を各配列ごとに
#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
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,Gの総数を計算してCGに格
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#A,C,G,Tの総数を計算して
GC_content <- CG/ACGT*100                   #%GC含量を計算してGC_content

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



課題2

テンプレートコードをコピペし、
①入力ファイル名部分を変更

1. イントロ | 一般 | ランダムな塩基配列を作成の4.を

```
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

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

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

RGui (64-bit)

ファイル 編集 パッケージ ウインドウ ヘルプ

R Console

```
> getwd()
[1] "C:/Users/kadota/Desktop/hoge"
> list.files()
[1] "hoge5.fa"
>
```

①

無題 - Rエディタ

```
in_f <- "hoge5.fa"                          #入力ファイル名を指
out_f <- "hoge1.txt"                        #出力ファイル名を指

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

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

#本番
hoge <- alphabetFrequency(fasta)            #A,C,G,T,..の数
#CG <- rowSums(hoge[,2:3])                  #C,Gの総数を計算
#ACGT <- rowSums(hoge[,1:4])                #A,C,G,Tの総数を
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
GC_content <- CG/ACGT*100                   #%GC含量を計算して
```

課題2の実行結果

①tmpオブジェクトの中身。出力ファイル(hoge1.txt)中の②1番右側の列が配列ごとのGC含量です。③contig_8中にACGT以外の文字が6個あることがわかる

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:

```
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,
#CG <- rowSums(hoge[,2:3])                  #C,Gの総数
#ACGT <- rowSums(hoge[,1:4])                #A,C,G,T
CG <- apply(as.matrix(hoge[,2:3]), 1, sum)#C,G
ACGT <- apply(as.matrix(hoge[,1:4]), 1, sum)#
GC_content <- CG/ACGT*100                   #%GC含

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

①

R Console

```
> tmp <- cbind(names(fasta), CG, ACGT, width(fasta),$
> colnames(tmp) <- c("description", "CG", "ACGT", "L$
> write.table(tmp, out_f, sep="\t", append=F, quote=$
> 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"
[5,] "contig_5"  "25" "49"  "49"   "51.0204081632653"
[6,] "contig_6"  "4"  "9"   "9"    "44.4444444444444"
[7,] "contig_7"  "5"  "10"  "10"   "50"
[8,] "contig_8"  "5"  "10"  "16"   "50"
>
```

②

③