第4回
代数方程式，超越方程式 with Octave

2021.06.21（Mon.）

菅沼 雅徳，川越 吉晃

Comment Screen

本日の講義内容

- 代数方程式および超越方程式をOctaveで解く
- Octaveでの関数の定義方法について
- Octaveでのシンボリック計算の方法について

今回の演習課題

次の方程式の全ての解を求めるプログラム

- $(\sin x)^2 \exp\left(-\frac{x}{2}\right) - 0.1 = 0, \qquad (x \geq 0)$

Why 代数方程式，超越方程式？

- **n次代数方程式は，「n次多項式$= 0$」**という形の方程式のこと
  - $a_n x^n + \cdots + ax + a_0 = 0$

- 上記の多項式方程式を満たさない方程式のことを**超越方程式**と呼ぶ
  - aが超越数 ≡ どんな有理数係数多項式$f(x)$を用いても$f(a) \neq 0$
  - 三角関数，指数関数，対数関数などを含むもの
  - $(\sin x)^2 - x + 0.1 = 0$

- 多くの場面で代数・超越方程式が登場するため，効率的に解けると嬉しい
  - 前回扱った固有値を求めるときにも，固有多項式が登場する
  - $\det(A - \lambda I) = 0$

代数方程式の解き方

- 方程式をシンボリック変数で定義
- solve関数で解く

```
# main.m
# x^2 − 5x + 6 = 0
syms x
eqn = x^2 - 5*x + 6 == 0;
sol_x = solve(eqn)
```

```
≫ main

sol_x = (sym 2×1 matrix)

[2]
[ ]
[3]
```

シンボリック計算

- **記号（数式）のまま演算が可能なため，方程式を立てて解くことが可能**

```
# x^2 − 5x + 6 = 0
syms x
eqn = x^2 - 5*x + 6 == 0;
sol_x = solve(eqn)
```

- **丸め誤差がないため，高い精度で演算可能**
  - 例 $\left(1-\frac{8}{9}\right)*9-1$

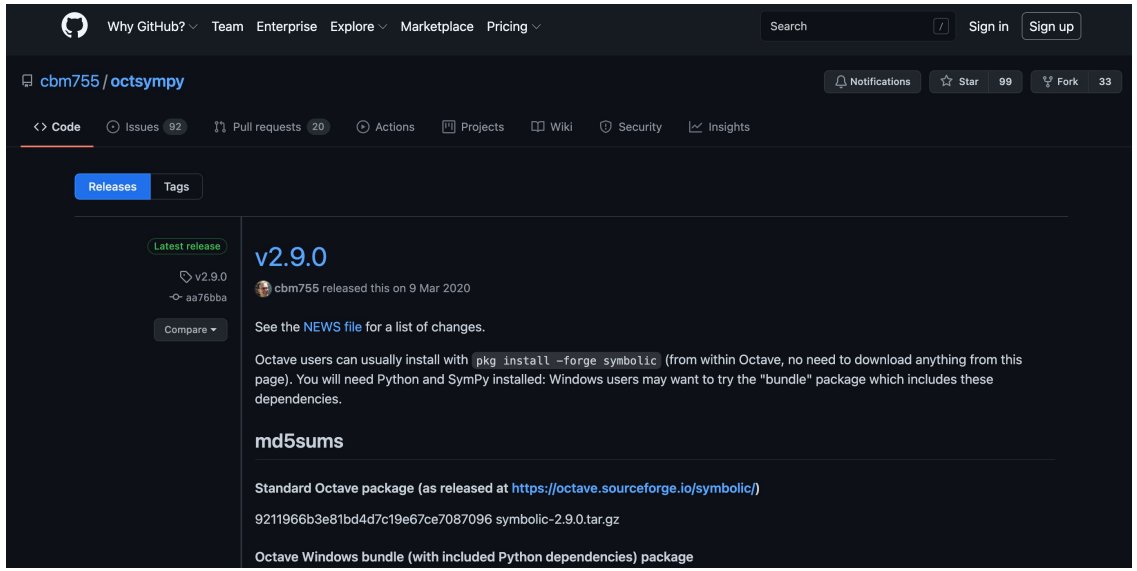
```
# 通常の演算
>> (1-8/9)*9-1
ans = 4.4409e-16
```

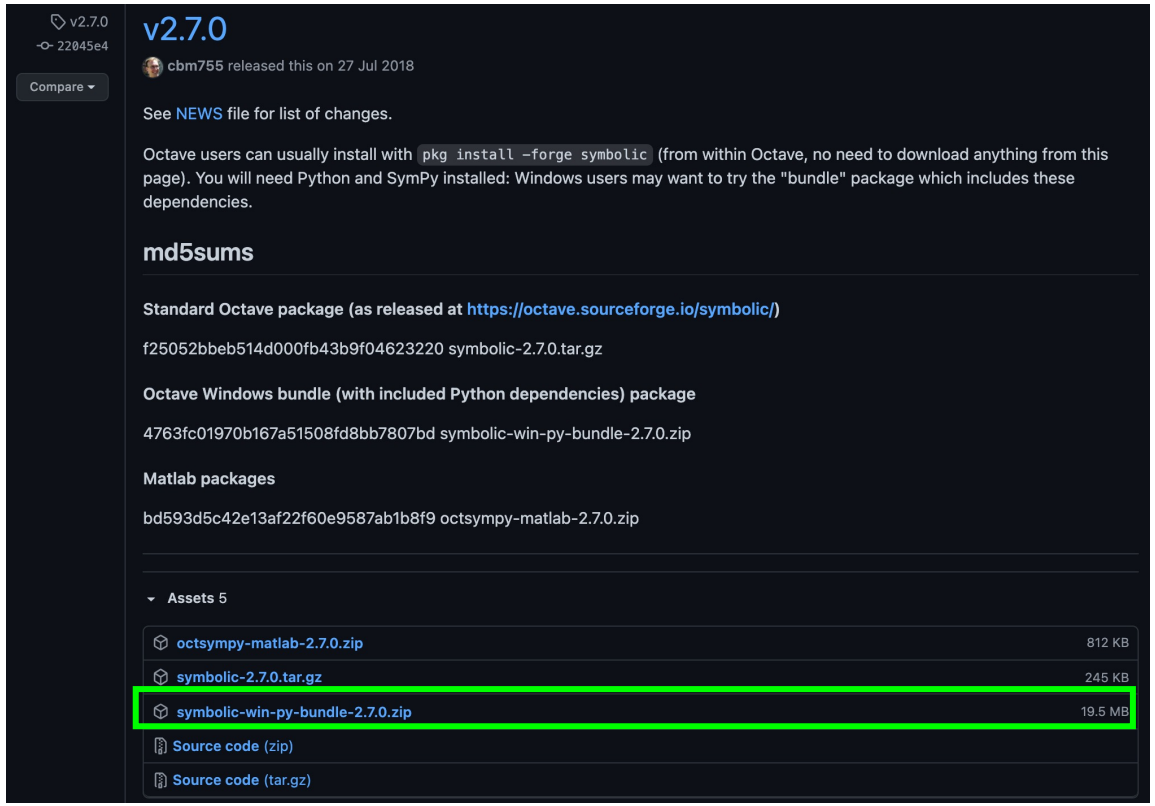
```
# シンボリック演算
>> (1-sym(8)/sym(9))*9-1
ans = (sym) 0
```

シンボリック演算を利用するには，Symbolic packageのインストールが必要（次ページで説明）

Symbolic packageのインストール方法

https://github.com/cbm755/octsympy/releases



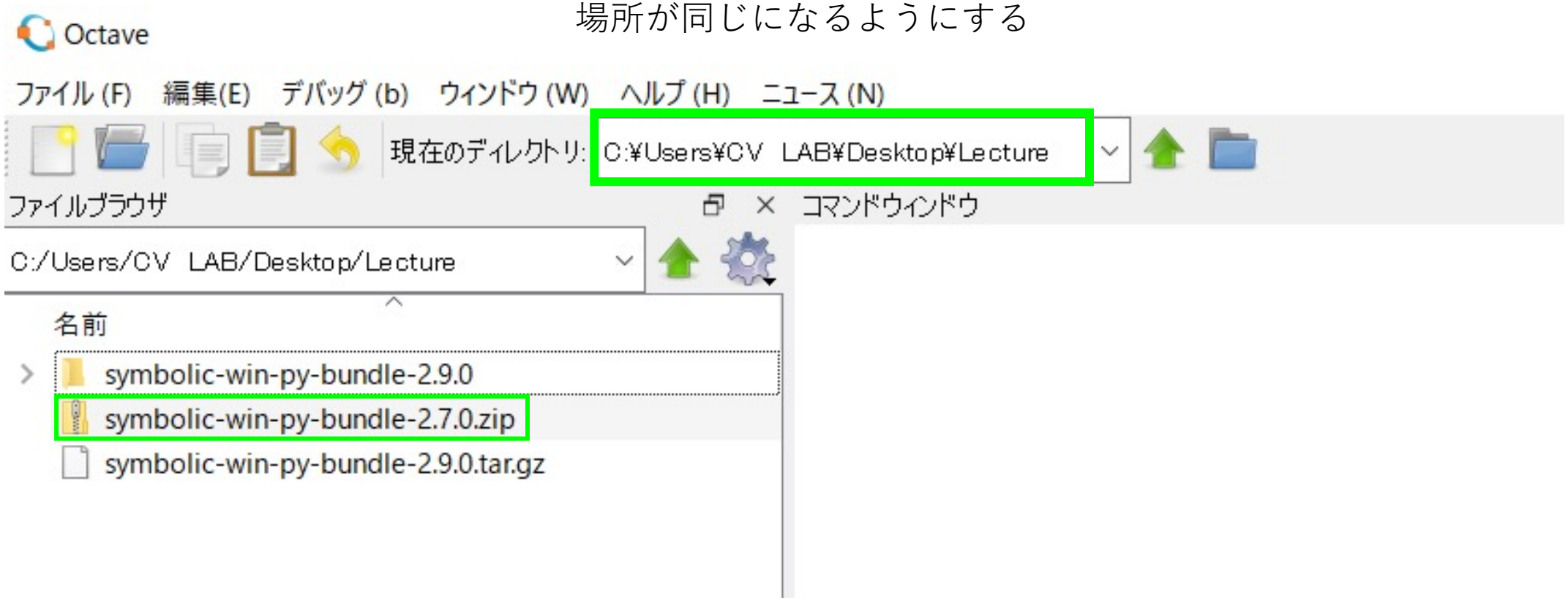


- 上記のURLへアクセス
- 下の方へスクロールしていき，v2.7.0から「**symbolic-win-py-bundle-2.7.0.zip**」をDL

Symbolic packageのインストール方法

DLしたzipファイルを普段Octaveを実行しているディレクトリへ移動

ここのパスと「**symbolic-win-py-bundle-2.7.0.zip**」が置いてある場所が同じになるようにする

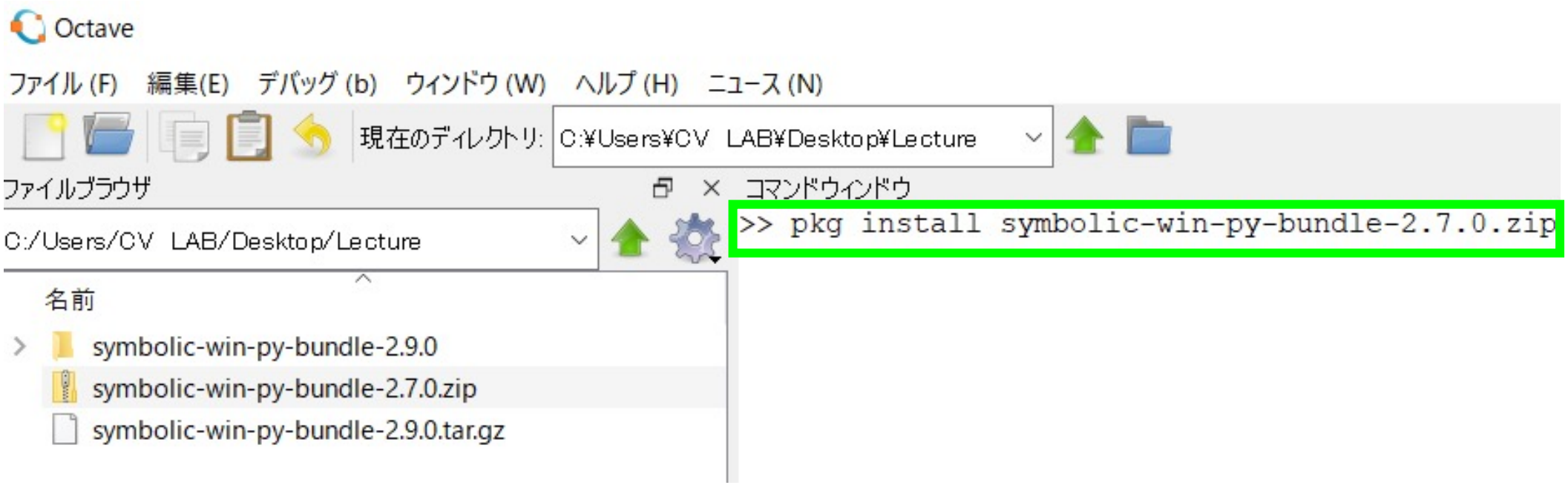


Symbolic packageのインストール方法

コマンドウィンドウで

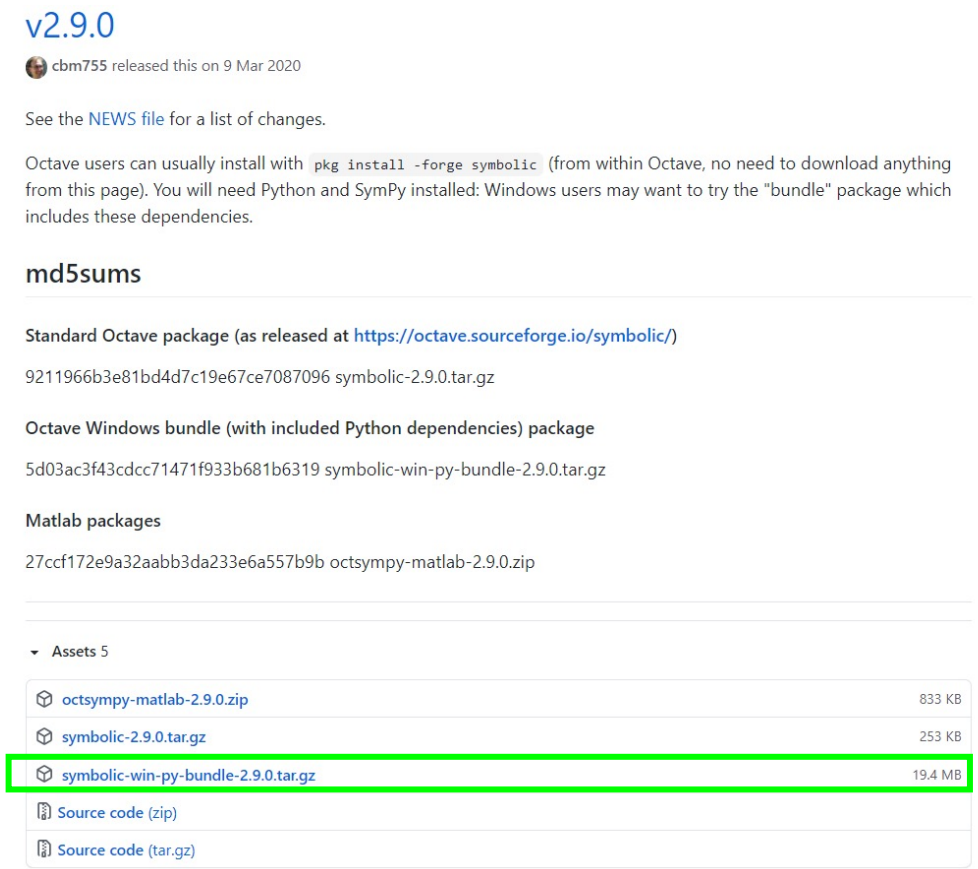
pkg install symbolic-win-py-bundle-2.7.0.zip

を実行



（補足）Symbolic package（v2.9）インストール方法

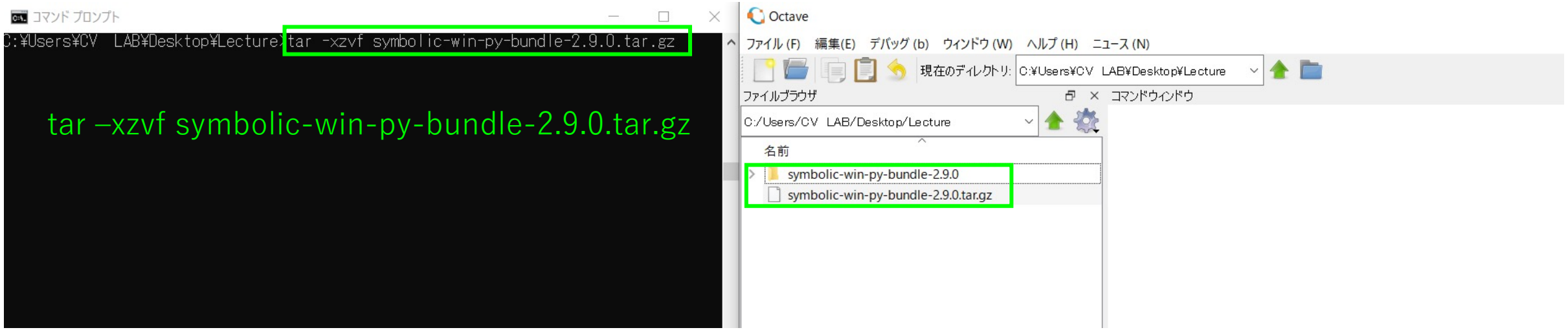
- https://github.com/cbm755/octsympy/releases　へアクセス
- 「symbolic-win-py-bundle-2.9.0.tar.gz」をダウンロード
- DLしたファイルを普段Octaveを実行しているディレクトリへ移動



（補足）Symbolic package（v2.9）インストール方法

コマンドプロンプトでファイルを解凍

- tar –xzvf symbolic-win-py-bundle-2.9.0.tar.gz
- 「symbolic-win-py-bundle-2.9.0」という名前のフォルダが生成される



（補足）Symbolic packageのインストール方法（Mac）

1. コマンドウィンドウで「pkg install –forge symbolic」を実行
2. Anaconda or miniforgeをインストール
   - Intel CPU版の場合：
     - https://www.anaconda.com/products/individual からPython 3.8・64-Bit Graphical InstallerをDL
     - ターミナルで「conda install sympy」を実行
   - M1 chipの場合：
     - 下記サイトへ行き，**Miniforge3-MacOSX-arm64** をダウンロード
     - https://github.com/conda-forge/miniforge
     - ターミナルで「bash Miniforge3-MacOSX-arm64.sh」を実行し，miniforgeをインストール
     - ターミナルで「arch -x86_64 zsh」を実行
     - ターミナルで「conda install sympy」もしくは「pip3 install sympy」を実行
3. Octaveのコマンドウィンドウで
   「setenv PYTHON /usr/local/anaconda3/bin/python」or
   「setenv PYTHON /usr/local/bin/python3」を実行後，
   「pkg load symbolic」を実行（pythonへのパスは which python3で検索．上記は例．）

超越方程式の解き方 with Octave

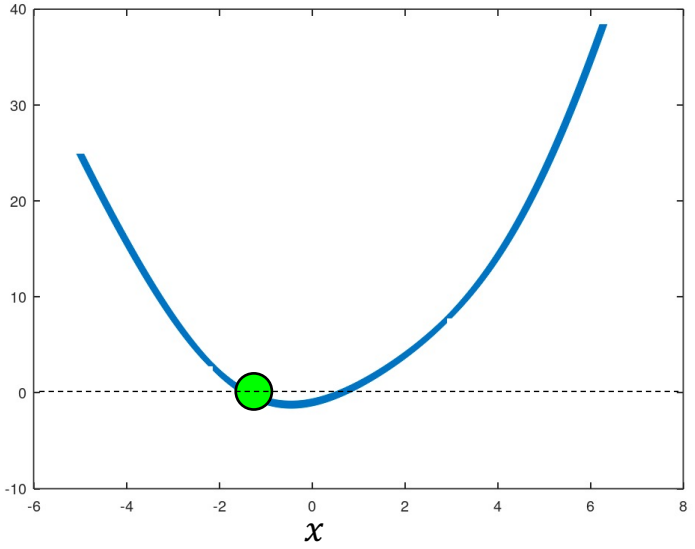
- ユーザ定義関数もしくは無名関数定義によって方程式を定義
- fsolve関数で解く
  - 信頼領域法による非線形関数最適化

例）ユーザ定義関数による超越方程式の解き方

```
# myfunc.m
function y = myfunc(x)
  y = x^2 + sin(x) - 1;
end
```

```
# main.m（初期値 x_0 = −2.0）
x_sol = fsolve(@myfunc, -2.0)
```

$x^2 + \sin x - 1 = 0$



実行結果

```
>> main
x_sol = -1.4096
```

ユーザ定義関数

- 関数名と同じファイル名のスクリプトを用意（.m）
- 下記の例だと，関数名＝スクリプト名＝ **myfunc**

```
# myfunc.m
function y = myfunc(x)
  y = x^2 + sin(x) - 1;
end
```

function 出力変数 = 関数名(入力変数)
**　出力変数 = 関数の定義;**
end

別のスクリプトから定義した関数を呼べる

```
# main.m
y = myfunc(2.0)
```

$f(x) = x^2 + \sin x - 1$ に$x = 2.0$ を代入した結果

```
>> main
y = 3.9093
```

無名関数定義

別のスクリプトをわざわざ用意したくない場合の定義方法

関数名 = @(変数) 関数定義;

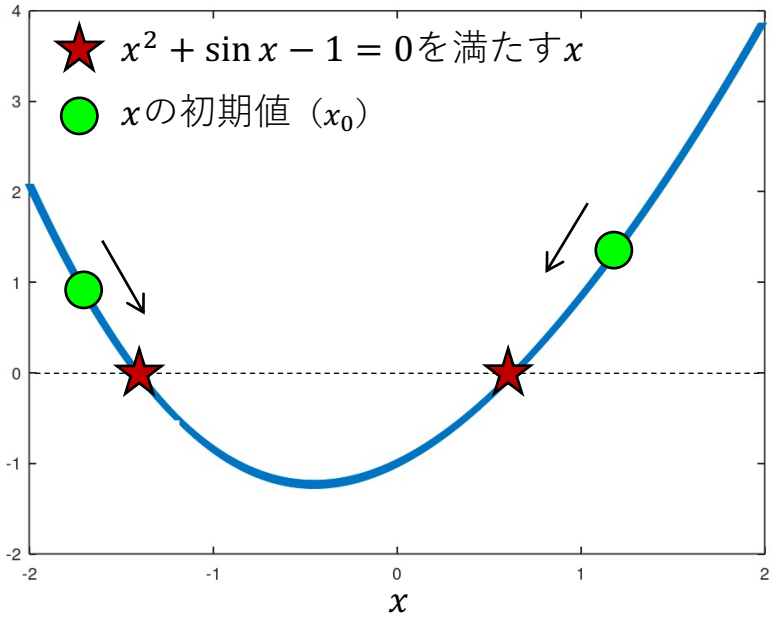
```
# main.m
# x^2 + sin x - 1 を定義
myfunc_a = @(x) x^2+sin(x)-1;
y = myfunc_a(2.0)

# 同じスクリプト内から定義した関数を呼べる
```

```
# f(x) = x^2 + sin x - 1 にx = 2.0 を代入した結果
>> main
y = 3.9093
```

超越方程式の解き方（無名関数の例で説明）

fsolve関数

- 解 ＝ fsolve(対象の方程式，初期値)
- 超越方程式は解析的に解けないため，反復法に基づくアルゴリズムによって数値的に解く
- 初期値$\boldsymbol{x}_o$を繰り返し更新することで解を求める
- $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta\boldsymbol{x}$




```
# main.m
# x^2 + sin x - 1 = 0 を解く
myfunc_a = @(x) x^2+sin(x)-1;
x_sol = fsolve(myfunc_a, -2.0)
```

```
# 実行結果
>> main
x_sol = -1.4096
```

fsolve関数＝信頼領域法

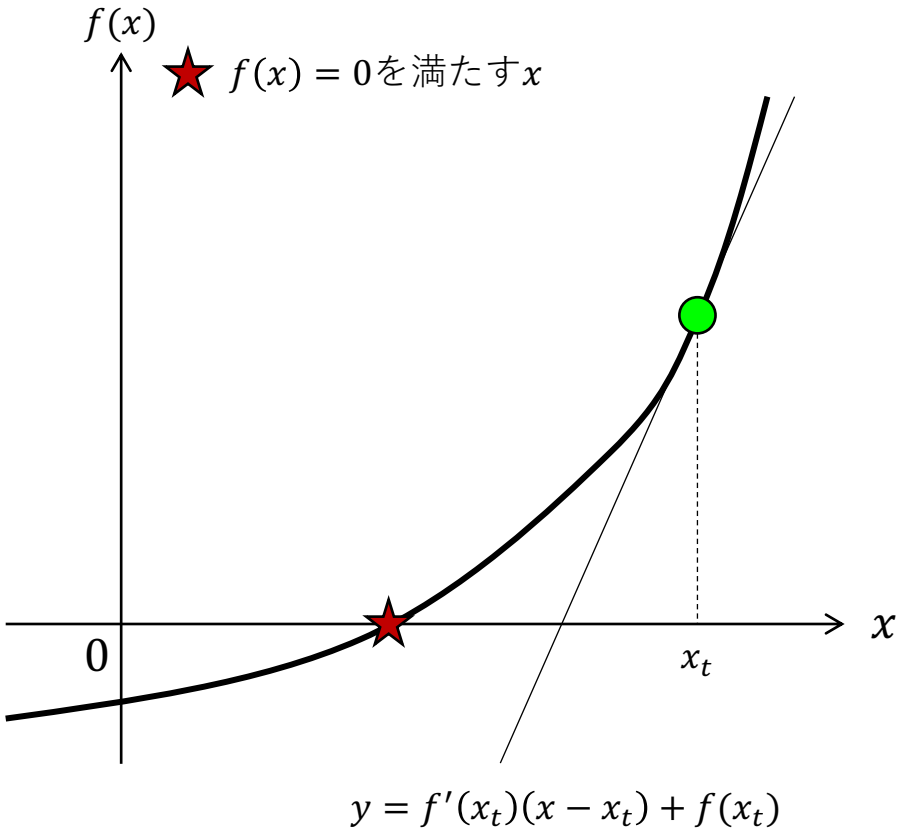
信頼領域法 ≈ 対象の近似2次関数が信頼出来る範囲内で直線探索（ニュートン法など）

ニュートン法

- ある点x_tでの導関数$f'(x_t)$が与えられるとき，
  $(x_t, f(x_t))$ を通り，傾きが$f'(x_t)$の直線は，
  $y = f'(x_t)(x - x_t) + f(x_t)$

- これを$f(x)$の近似式とみなすと，
  $f(x) = 0$の近似解は$y = 0$となるxの値となるから，
  $x = x_t - \frac{f(x_t)}{f'(x_t)}$

- これをx_{t+1}として，繰り返すことで解が求まる



fsolve関数＝信頼領域法

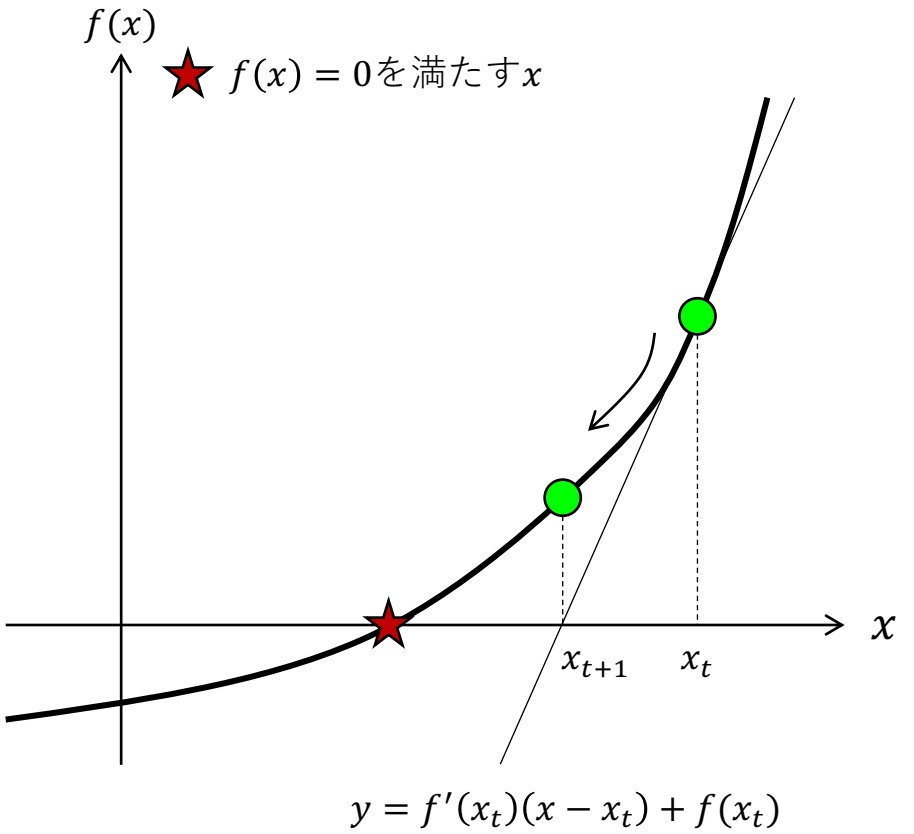
信頼領域法 ≈ 対象の近似2次関数が信頼出来る範囲内で直線探索（ニュートン法など）

ニュートン法

- ある点x_tでの導関数$f'(x_t)$が与えられるとき，
  $(x_t, f(x_t))$ を通り，傾きが$f'(x_t)$の直線は，
  $y = f'(x_t)(x - x_t) + f(x_t)$

- これを$f(x)$の近似式とみなすと，
  $f(x) = 0$の近似解は$y = 0$となるxの値となるから，
  $x = x_t - \frac{f(x_t)}{f'(x_t)}$

- これをx_{t+1}として，繰り返すことで解が求まる



fsolve関数＝信頼領域法

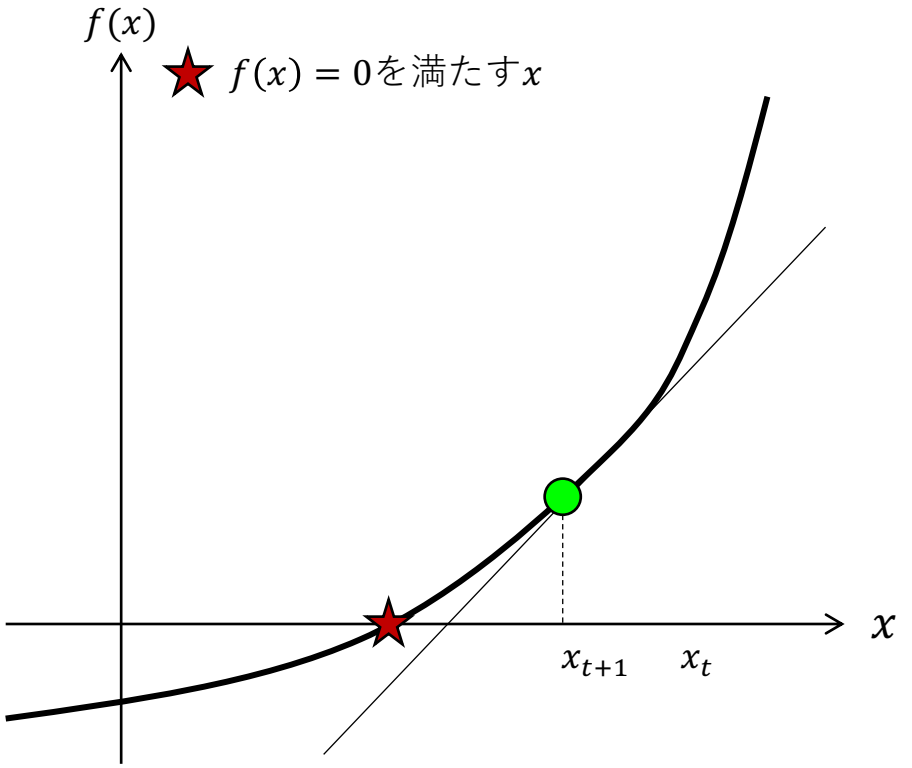
信頼領域法 ≈ 対象の近似2次関数が信頼出来る範囲内で直線探索（ニュートン法など）

ニュートン法

- ある点x_tでの導関数$f'(x_t)$が与えられるとき，
  $(x_t, f(x_t))$ を通り，傾きが$f'(x_t)$の直線は，
  $y = f'(x_t)(x - x_t) + f(x_t)$

- これを$f(x)$の近似式とみなすと，
  $f(x) = 0$の近似解は$y = 0$となるxの値となるから，
  $x = x_t - \frac{f(x_t)}{f'(x_t)}$

- これをx_{t+1}として，繰り返すことで解が求まる



fsolve関数＝信頼領域法

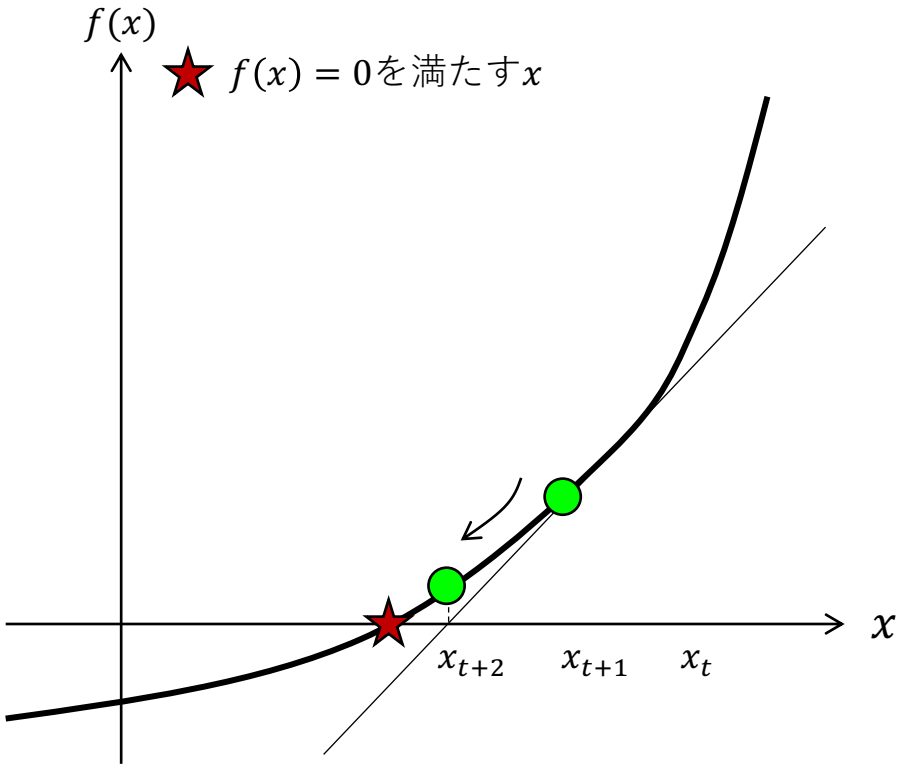
信頼領域法 ≈ 対象の近似2次関数が信頼出来る範囲内で直線探索（ニュートン法など）

ニュートン法

- ある点x_tでの導関数$f'(x_t)$が与えられるとき，
  $(x_t, f(x_t))$ を通り，傾きが$f'(x_t)$の直線は，
  $y = f'(x_t)(x - x_t) + f(x_t)$

- これを$f(x)$の近似式とみなすと，
  $f(x) = 0$の近似解は$y = 0$となるxの値となるから，
  $x = x_t - \frac{f(x_t)}{f'(x_t)}$

- これをx_{t+1}として，繰り返すことで解が求まる



（補足）因数分解

- シンボリック演算を利用することで因数分解も可能
- factor関数

$\#\ f = x^2 - 5x + 6$ の因数分解

```
>> syms x
>> f = x^2 - 5*x + 6;
>> factor(f)
ans = (sym) (x - 3)·(x - 2)
```

$\#\ f = x^3y - 3x^3 - 4x^2y + 12x^2 - 3xy + 9x + 18y - 54$ の因数分解

```
>> syms x y
>> f=x^3*y-3*x^3-4*x^2*y+12*x^2-3*x*y+9*x+18*y-54;
>> factor(f)
ans = (sym)

                2
 (x - 3) ·(x + 2)·(y - 3)
```

（補足）微分

- シンボリック演算を利用することで微分も可能
- diff関数

$f = x^2$ の微分

```
>> diff(x^2)
ans = (sym) 2·x
```

$f = x^2 + \sin x - 1$ の微分

```
>> diff(x^2+sin(x)-1)
ans = (sym) 2·x + cos(x)
```

演習課題

次の方程式の全ての解を求めるプログラムを書いて，解とともに提出してください

- $(\sin x)^2 \exp\left(-\frac{x}{2}\right) - 0.1 = 0, \quad (x \geq 0)$

- 超越方程式なのでfsolve関数を使って求める
- 初期値依存があるため，まずは上記関数を表示してから，初期値のあたりをつける

```
# 0 ≤ x ≤ 10 の範囲で上記関数をプロットするプログラム
x=0:0.01:10;
y=sin(x).^2.*exp(-x/2) - 0.1;
y0=zeros(1,length(x));
plot(x,y,x,y0)
```