Pythonによる数値シミュレーション入門
―放物運動を予測しよう―

第１部：放物運動の理論

１　ニュートン力学

われわれの身のまわりで見られる物体の運動は、通常は古典力学（ニュートン力学）によって説明される。ニュートン力学は以下の３つの法則からなる。

ニュートン力学の第１法則（慣性の法則）:
外部から力を加えられない限り、静止している物体は静止し続け、運動している物体は等速直線運動を続ける。

ニュートン力学の第２法則（運動方程式）:
物体が力を受けると力と同じ方向に加速度が生じる。加速度の大きさは力の大きさに比例し、物体の質量に反比例する。

ニュートン力学の第３法則（作用反作用の法則）:
一方の物体が他方の物体に及ぼす力と、その物体が他方の物体から受ける力は、向きが反対で大きさが等しい。

以上の３つの法則のうち、ニュートン力学の第２法則は、運動方程式によって次のように記述される。

$$ma = F$$

ただし、mは質量、aは加速度、Fは力である。ここで、物体の位置x、速度u、加速度aの間は次のような関係がある。まず、位置xの時間微分が速度uであって、

$$u = \frac{dx}{dt}$$

が成り立つ。また、速度uの時間微分が加速度aであって、

$$a = \frac{du}{dt} = \frac{d^2x}{dt^2}$$

が成り立つ。したがって、運動方程式を

$$m\frac{d^2x}{dt^2} = F$$

と書くこともできる。

２　放物運動の基礎

$x-z$ 平面内で重力による運動を考える。物体にはたらく水平方向の力はゼロだから、

$$F_x = 0$$

である。また、鉛直方向にはたらく力は重力であって、鉛直下向き方向にはたらくから、

$$F_z = -mg$$

である。したがって、x 方向（水平方向）の運動方程式は、

$$m\frac{d^2x}{dt^2} = 0 \qquad ①$$

と書け、また、z 方向（鉛直方向）の運動方程式は、

$$m\frac{d^2z}{dt^2} = -mg \qquad ②$$

と書ける。

まず、水平方向の運動方程式を解く。①より、

$$\frac{d^2x}{dt^2} = \frac{d}{dt}\left(\frac{dx}{dt}\right) = 0$$

である。両辺を t について積分すると、

$$\frac{dx}{dt} = C \quad （C \text{ は積分定数}）$$

となる。初期条件として $t=0$ で $\frac{dx}{dt} = u_0$ とすると、$C = u_0$ となるので、

$$\frac{dx}{dt} = u_0$$

である。さらに、両辺を t について積分すると、

$$x = u_0 t + C' \quad （C' \text{ は積分定数}）$$

となる。初期条件として $t=0$ で $x=0$ とすると、$C'=0$ となるので、

$$x = u_0 t \qquad ③$$

が得られる。

次に鉛直方向の運動方程式を解く。②より、

$$\frac{d^2z}{dt^2} = \frac{d}{dt}\left(\frac{dz}{dt}\right) = -g$$

両辺を t について積分すると、

$$\frac{dz}{dt} = -gt + C \quad （C \text{ は積分定数}）$$

となる。初期条件として$t = 0$で$\frac{dz}{dt} = w_0$とすると、$C = w_0$となるので、

$$\frac{dz}{dt} = w_0 - gt$$

である。さらに、両辺をtについて積分すると、

$$z = w_0 t - \frac{1}{2} gt^2 + C' \quad （C' \text{ は積分定数}）$$

となる。初期条件として$t = 0$で$z = 0$とすると、$C' = 0$となるので、

$$z = w_0 t - \frac{1}{2} gt^2 \qquad ④$$

が得られる。

３　放物運動の性質

（１）滞空時間を求める：

滞空時間とは、投げ出してから着地するまでの時間のことである。したがって、投げ出された時刻と着地した時刻との差を求めればよい。④で$z = 0$とすると、

$$w_0 t - \frac{1}{2} gt^2 = 0$$

となる。二次方程式として解くと、

$$t = 0,\ \frac{2w_0}{g}$$

が得られる。２つの解の差が滞空時間Tであるから、

$$T = \frac{2w_0}{g} \qquad ⑤$$

となる。初速度の大きさをV、仰角をθとすれば、$w_0 = V \sin\theta$だから、

$$T = \frac{2V \sin\theta}{g}$$

が得られる。

（２）飛距離を求める：

飛距離とは、投げ出した時刻と着地した時刻における物体の位置の差のことである。③より、$t = 0$のとき、$x = 0$であり、$t = \frac{2w_0}{g}$のとき、$x = \frac{2u_0 w_0}{g}$であることがわかる。したがって、飛距離Lは、

$$L = \frac{2u_0 w_0}{g} \qquad ⑥$$

である。初速度の大きさVと仰角θを用いれば、$u_0 = V\cos\theta$、$w_0 = V\sin\theta$だから、

$$L = \frac{2V^2 \sin\theta\cos\theta}{g} = \frac{V^2 \sin 2\theta}{g} \qquad (\sin 2\theta = 2\sin\theta\cos\theta)$$

と書ける。

（３）軌跡を求める：

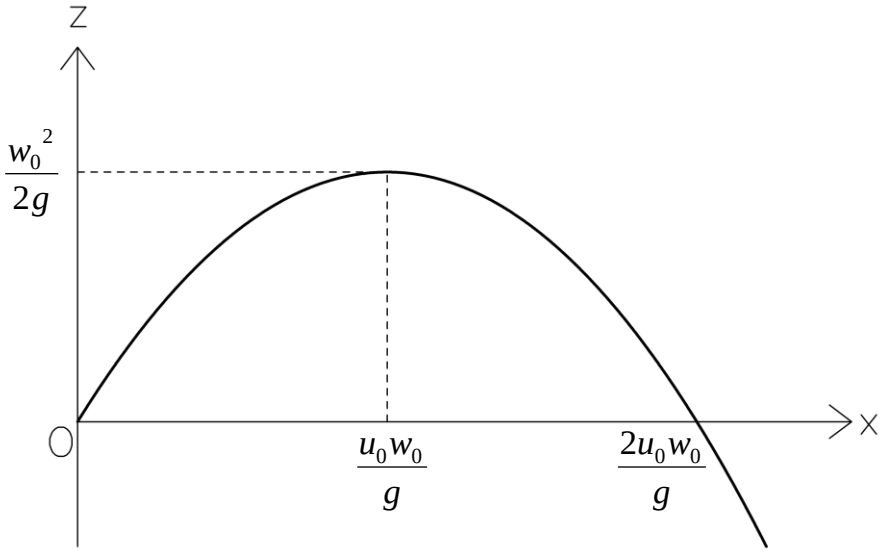
③より、

$$t = \frac{x}{u_0} \qquad ③'$$

③’を④に代入すると、

$$z = \frac{w_0}{u_0}x - \frac{1}{2}\frac{g}{{u_0}^2}x^2 = -\frac{1}{2}\frac{g}{{u_0}^2}\left(x - \frac{u_0 w_0}{g}\right)^2 + \frac{{w_0}^2}{2g} \qquad ⑦$$

が得られる。⑦は二次関数であり、放物運動の軌跡は二次曲線であることがわかる。



放物運動の軌跡

４　数値シミュレーションの方法

数値シミュレーションによって放物運動の軌跡を計算してみよう。速度の定義より、

$$\frac{x^+ - x}{\Delta t} = u$$

$$\frac{z^+ - z}{\Delta t} = w$$

となるので、

$$x^+ = x + u\Delta t$$

$$z^+ = z + w\Delta t$$

である。ただし、x^+、z^+は時間Δtだけ後のx、zの値である。これらの関係を使うと、ある時刻のx、z、u、wの値から時間Δtだけ後のx、zを計算することができる。

同様に、加速度の定義より、

$$\frac{u^+ - u}{\Delta t} = a_x$$

$$\frac{w^+ - w}{\Delta t} = a_z$$

である。$a_x = 0$、$a_z = -g$を代入すると、

$$\frac{u^+ - u}{\Delta t} = 0$$

$$\frac{w^+ - w}{\Delta t} = -g$$

となる。したがって、

$$u^+ = u$$

$$w^+ = w - g\Delta t$$

が得られる。これらの関係を使うと、ある時刻のu、wの値から時間Δtだけ後のu、wを計算することができる。

第２部：放物運動のシミュレーション

プログラミング作業の流れ

まずターミナルを起動する。　（⇒簡単操作マニュアル１）

☞ターミナル（端末）がすべての作業の起点になります。

１．Emacs でプログラムを書く。　（⇒１－②、２－②）

２．実行する。　（⇒１－③）

３．結果を確かめる。　（⇒１－④⑤）

簡単操作マニュアル（Ｗｉｎｄｏｗｓ用）

１．ターミナルの使い方

ターミナルを起動する	デスクトップの「 msys.bat へのショートカット 」を ダブルクリック
①ファイルの一覧を表示する	$ ls ↵
ファイルをコピーする	$ cp *コピー元ファイル名* *コピー先ファイル名* ↵
ファイル名を変更する	$ mv *変更前のファイル名* *変更後のファイル名* ↵
ファイルを消去する	$ rm *ファイル名* ↵
②Ｅｍａｃｓを起動する	$ emacs *ファイル名* & ↵ （または emacs & ↵）
③実行する	$ python *ファイル名* ↵
④ファイルの中身を見る	$ less *ファイル名* ↵　矢印キーで移動、Ｑを押して終了
⑤ｇｎｕｐｌｏｔを起動する	$ gnuplot ↵
ターミナルを終了する	$ exit ↵

２．Ｅｍａｃｓの使い方

①ファイルを開く	コントロールキー＋Ｘ　コントロールキー＋Ｆ →ファイル名を入力
②ファイルを保存する	コントロールキー＋Ｘ　コントロールキー＋Ｓ
※操作の取り消し	コントロールキー＋Ｇ
Ｅｍａｃｓを終了する	コントロールキー＋Ｘ　コントロールキー＋Ｃ

３．ｇｎｕｐｌｏｔの使い方

①グラフをかく	> plot ”*ファイル名*” ↵ 複数の場合：　plot ”*ファイル名*”，　”*ファイル名*”，　… ↵
②折れ線グラフにする	> set　style　data　lines ↵
範囲を指定する	> set　xrange　[*0*:*50*] ↵ > set　yrange　[*0*:*30*] ↵
③再描画する	> replot ↵
※画像ファイルに保存する	画面上で作図したあとで： > set　term　png ↵ > set　output　”*ファイル名.png*” ↵ > replot ↵
ｇｎｕｐｌｏｔを終了する	> quit ↵

簡単操作マニュアル（Ｌｉｎｕｘ用）

１．ターミナルの使い方

ターミナルを起動する	ランチャー（画面左側）の「端末」をクリック
①ファイルの一覧を表示する	> ls
ファイルをコピーする	> cp *コピー元ファイル名* *コピー先ファイル名*
ファイル名を変更する	> mv *変更前のファイル名* *変更後のファイル名*
ファイルを消去する	> rm *ファイル名*
②Ｅｍａｃｓを起動する	> emacs & または emacs *ファイル名* &
③実行する	> python *ファイル名*
④ファイルの中身を見る	> less *ファイル名*　　矢印キーで移動、Ｑを押して終了
⑤ｇｎｕｐｌｏｔを起動する	> gnuplot
ターミナルを終了する	> exit

２．Ｅｍａｃｓの使い方

①ファイルを開く	コントロールキー＋Ｘ　コントロールキー＋Ｆ →ファイル名を入力
②ファイルを保存する	コントロールキー＋Ｘ　コントロールキー＋Ｓ
※操作の取り消し	コントロールキー＋Ｘ　コントロールキー＋Ｇ
Ｅｍａｃｓを終了する	コントロールキー＋Ｘ　コントロールキー＋Ｃ

３．ｇｎｕｐｌｏｔの使い方

①グラフをかく	> plot ”*ファイル名*” 複数の場合：　plot ”*ファイル名*”，　”*ファイル名*”，　…
②折れ線グラフにする	> set style data lines
範囲を指定する	> set xrange [*0*:*50*] > set yrange [*0*:*30*]
③再描画する	> replot
※画像ファイルに保存する	画面上で作図したあとで： > set term png > set output ”*ファイル名.png*” > replot
ｇｎｕｐｌｏｔを終了する	> quit

①滞空時間と飛距離を計算しよう

作業の手順：以下のサンプルプログラムを Emacs で作成しよう。

☞プログラムのファイル名は自由だが、「.py」で終わる必要がある。
　例：prog01.py なら　　⇒　　ターミナル上で　$ emacs　prog01.py　& ↵
☞書き終わったら忘れずに保存する（コントロールキー＋Ｘ　コントロールキー＋Ｓ）。
　　コントロールキーを押しながらＸを押す

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle                   ☞角度はラジアンで表します。

T  =  2.0  *  V  *  math.sin  (theta)  /  g
X  =  V  **  2  *  math.sin  (2.0  *  theta)  /  g      この部分が計算式です。

print  ("T  =  %9.3f,  X  =  %9.3f"  %  (T,  X))   #  結果を出力する
```

※　大文字と小文字は区別する。
※　#以降はコメントであり処理に影響しないので、書かなくてよい。

メモ：

$$T = \frac{2w_0}{g} = \frac{2V\sin\theta}{g}$$

$$X = \frac{2u_0 w_0}{g} = \frac{V^2 \sin 2\theta}{g}$$

実行例：ターミナル上で実行してみよう。

$ ls ↵	ファイルを確認します。
prog01.py	
$ python prog01.py ↵	実行します。
Velocity [m/s]?	
10 ↵	初速度の大きさを入力します。
Angle [deg.]?	
45 ↵	仰角を入力します。
T = 1.443, X = 10.204	結果が出力されます。

メモ：

$V = 10\,[\mathrm{m/s}]$、$\theta = 45^\circ$ ⇒ $T \fallingdotseq 1.4\,[\mathrm{s}]$、$X \fallingdotseq 10\,[\mathrm{m}]$

☞正常に実行できることを確認したら、Emacs を終了してよい
（コントロールキー＋Ｘ　コントロールキー＋Ｃ）。

②初期の状態を計算しよう

作業の手順：①で作成したプログラムをコピーして、Emacs で完成させよう。

☞コピーするときには cp コマンドを使う。
　例：$ cp　prog01.py　prog02.py ↵　　（⇒簡単操作マニュアル１－①）
☞コピーしたファイルを改めて emacs で開く。

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle

x  =  0.0                                          #  初期値を計算する
z  =  0.0
u  =  V  *  math.cos  (theta)
w  =  V  *  math.sin  (theta)

print  ("%9.3f  %9.3f"    %  (x,  z))              #  結果を出力する
```

ここで初期値を計算します。

実行例：ターミナル上で実行してみよう。

```
$ python  prog02.py ↵
Velocity  [m/s]?
10 ↵
Angle  [deg.]?
45 ↵
     0.000        0.000
```

実行します。

初速度の大きさを入力します。

仰角を入力します。

結果が出力されます。

③0.01 秒後の状態を計算しよう

作業の手順：②で作成したプログラムをコピーして、Emacs で完成させよう。

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle

x  =  0.0                                          #  初期値を計算する
z  =  0.0
u  =  V  *  math.cos  (theta)
w  =  V  *  math.sin  (theta)

print  ("%9.3f  %9.3f"    %  (x,  z))             #  結果を出力する

dxdt  =  u                                         #  時間微分を計算する
dzdt  =  w
dudt  =  0.0
dwdt  =  -  g

x  =  x  +  0.01  *  dxdt                          #  次の時刻の値を計算する
z  =  z  +  0.01  *  dzdt
u  =  u  +  0.01  *  dudt
w  =  w  +  0.01  *  dwdt

print  ("%9.3f  %9.3f"    %  (x,  z))             #  結果を出力する
```

ここで 0.01 秒後の値を計算します。

実行例：ターミナル上で実行してみよう。

```
$ python  prog03.py ↵
Velocity  [m/s]?
10 ↵
Angle  [deg.]?
45 ↵
     0.000       0.000
     0.071       0.071
```

実行します。

初速度の大きさを入力します。

仰角を入力します。

結果が出力されます。

④結果をファイルに書き出そう

作業の手順：③で作成したプログラムをコピーして、Emacs で完成させよう。

実行したら出力ファイルの内容を確認しよう。

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle

x  =  0.0                                          #  初期値を計算する
z  =  0.0
u  =  V  *  math.cos  (theta)
w  =  V  *  math.sin  (theta)

f  =  open  ("output.txt",  "w")                   #  ファイルを開く
f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))            #  結果をファイルに出力する

dxdt  =  u                                         #  時間微分を計算する
dzdt  =  w
dudt  =  0.0
dwdt  =  -  g

x  =  x  +  0.01  *  dxdt                          #  次の時刻の値を計算する
z  =  z  +  0.01  *  dzdt
u  =  u  +  0.01  *  dudt
w  =  w  +  0.01  *  dwdt

f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))            #  結果をファイルに出力する

f.close  ()                                        #  ファイルを閉じる
```

実行例：ターミナル上で実行してみよう。

```
$ python  prog04.py ↵
                                          実行します。
Velocity  [m/s]?
10 ↵                                      初速度の大きさを入力します。
Angle  [deg.]?
45 ↵                                      仰角を入力します。
$  ls ↵                                   出力ファイルを確認します。
output.txt  prog01.py  prog02.py  prog03.py  prog04.py
$  less  output.txt ↵                     出力ファイルの中身を見ます。→見終わったらＱで終了。
```

⑤10 秒後まで計算しよう

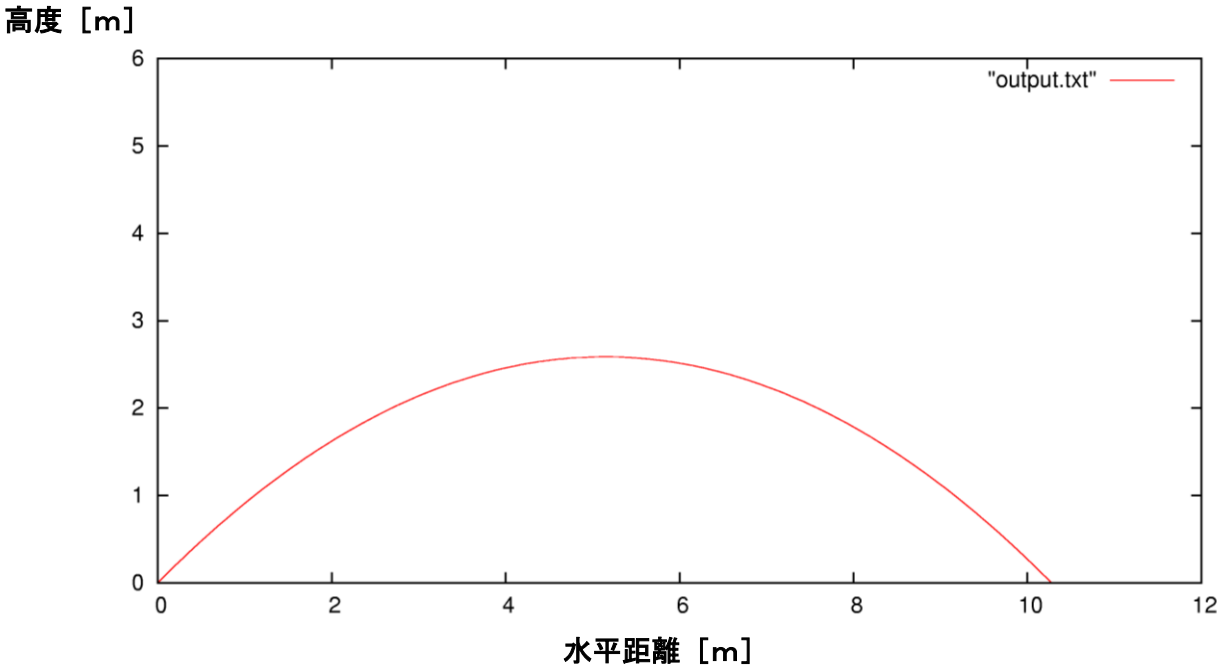
作業の手順：④で作成したプログラムをコピーして、Emacs で完成させよう。
実行したら出力ファイルの内容を gnuplot で作図しよう。

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle

x  =  0.0                                          #  初期値を計算する
z  =  0.0
u  =  V  *  math.cos  (theta)
w  =  V  *  math.sin  (theta)

f  =  open  ("output.txt",  "w")                   #  ファイルを開く
f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))            #  結果をファイルに出力する

i  =  1
while  i  <=  1000:                                #  同じ処理を 1000 回繰り返す

    dxdt  =  u                                     #  時間微分を計算する
    dzdt  =  w
    dudt  =  0.0
    dwdt  =  -  g

    x  =  x  +  0.01  *  dxdt                      #  次の時刻の値を計算する
    z  =  z  +  0.01  *  dzdt
    u  =  u  +  0.01  *  dudt
    w  =  w  +  0.01  *  dwdt

    f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))       #  結果をファイルに出力する

    i  =  i  +  1                                  #  カウントをひとつ進める

f.close  ()                                        #  ファイルを閉じる
```

反復する範囲内では
行頭を４文字空ける

作図例：実行した後、less で結果を確認したら、gnuplot で作図しよう。

$ gnuplot ↵	gnuplot を起動します。
gnuplot> set style data lines ↵	折れ線グラフを指定します。
gnuplot> plot "output.txt" ↵	グラフをかきます。
gnuplot> set xrange [0:12] ↵	範囲を指定します。
gnuplot> set yrange [0:6] ↵	
gnuplot> replot ↵	再描画します。



計算結果の例（初速度の大きさ 10m/s、仰角 45° の場合）

☞結果を確認したら、quit とタイプして、gnuplot を終了する。

⑥初速度の大きさを変えてみよう

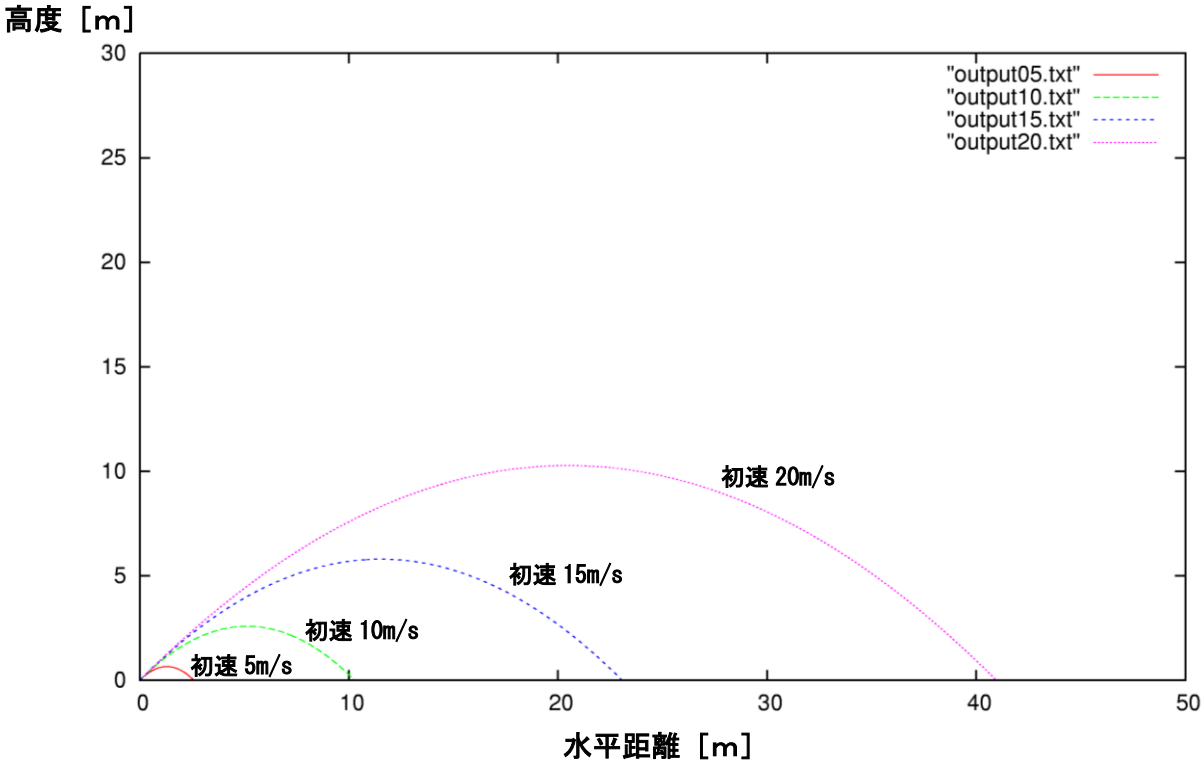
作業の手順：⑤で作成したプログラムで、仰角は一定のまま、
初速度の大きさを変えて計算しよう。
計算ごとに出力ファイル名を変えて保存しよう。
出力ファイルの内容をまとめて gnuplot で作図しよう。

初速度の大きさ 5m/s、仰角 45° で実験 → 出力ファイル名を output05.txt に変更。
初速度の大きさ 10m/s、仰角 45° で実験 → 出力ファイル名を output10.txt に変更。
初速度の大きさ 15m/s、仰角 45° で実験 → 出力ファイル名を output15.txt に変更。
初速度の大きさ 20m/s、仰角 45° で実験 → 出力ファイル名を output20.txt に変更。
☞ファイルの名前を変更するときには mv コマンドを用いる（簡単操作マニュアル参照）。

例： $ python prog05.py ↵
Velocity [m/s]?
5 ↵
Angle [deg.]?
45 ↵
$ mv output.txt output05.txt ↵

→ gnuplot で４つの出力ファイルをまとめて作図（簡単操作マニュアル参照）。

例： $ gnuplot ↵
gnuplot> set style data lines ↵
gnuplot> plot "output05.txt", "output10.txt", … ↵



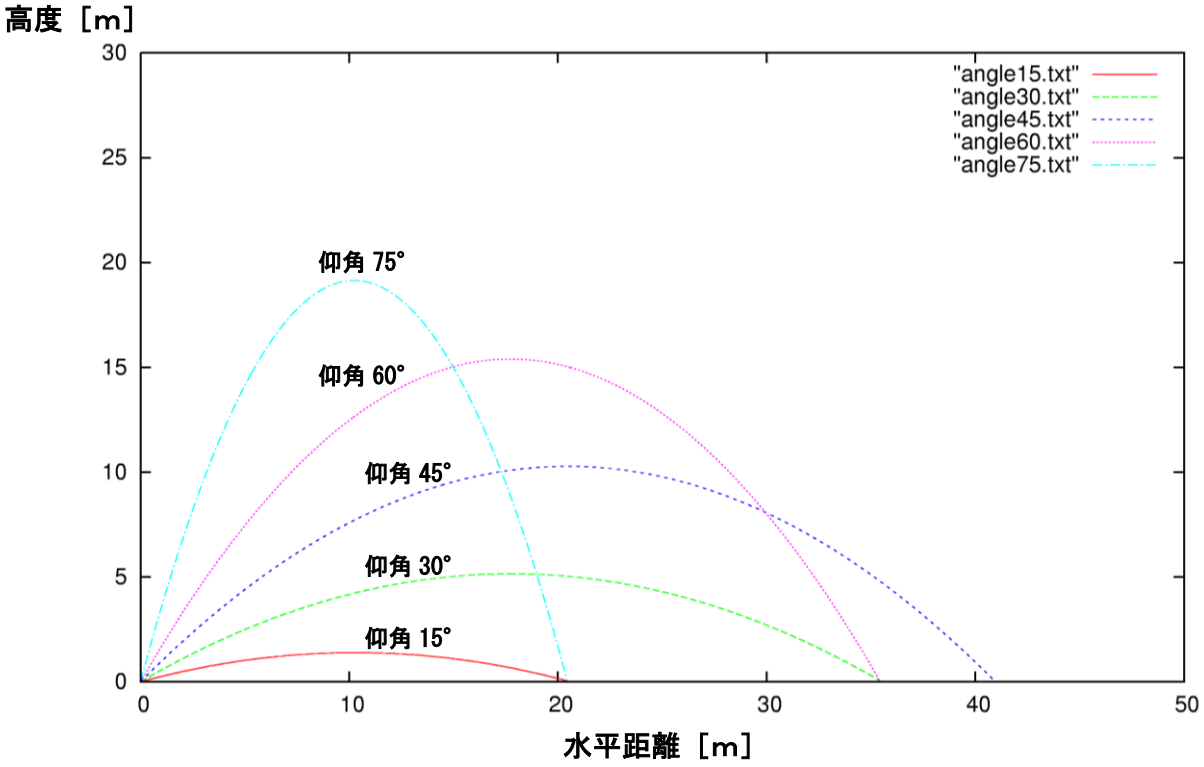
計算結果の例（仰角 45° の場合）

→ 初速度が大きいほど飛距離が長くなる。

⑦仰角を変えてみよう

作業の手順：⑤で作成したプログラムで、初速度の大きさは一定のまま、
仰角を変えて計算しよう。
計算ごとに出力ファイル名を変えて保存しよう。
出力ファイルの内容をまとめて gnuplot で作図しよう。

初速度の大きさ 20m/s、仰角 15° で実験　→　出力ファイル名を angle15.txt に変更。
初速度の大きさ 20m/s、仰角 30° で実験　→　出力ファイル名を angle30.txt に変更。
初速度の大きさ 20m/s、仰角 45° で実験　→　出力ファイル名を angle45.txt に変更。
初速度の大きさ 20m/s、仰角 60° で実験　→　出力ファイル名を angle60.txt に変更。
初速度の大きさ 20m/s、仰角 75° で実験　→　出力ファイル名を angle75.txt に変更。
☞ファイルの名前を変更するときには mv コマンドを用いる（簡単操作マニュアル参照）。
→ gnuplot で５つの出力ファイルをまとめて作図。



計算結果の例（初速度の大きさ 20m/s の場合）

→ 仰角が 45° のとき飛距離が最大になる。

【発展】

⑧空気抵抗を考慮しよう

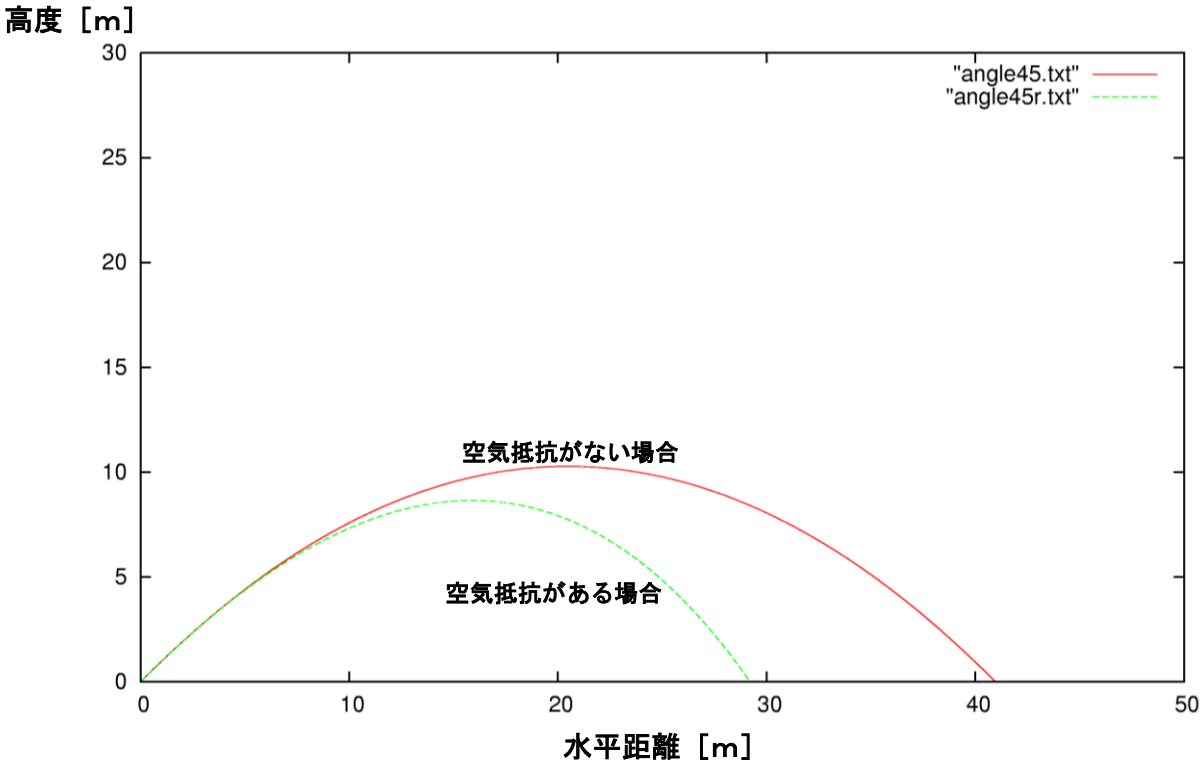
作業の手順：⑤で作成したプログラムをコピーして、Emacs で完成させよう。

初速度の大きさは一定のまま、仰角を変えて計算しよう。

```
import  math

g  =  9.8                                          #  定数を宣言する

print  ("Velocity  [m/s]?")
V  =  float  (input())                             #  初速度の大きさを入力する
print  ("Angle  [deg.]?")
angle  =  float  (input())                         #  仰角を入力する
theta  =  3.14159  /  180.0  *  angle

x  =  0.0                                          #  初期値を計算する
z  =  0.0
u  =  V  *  math.cos  (theta)
w  =  V  *  math.sin  (theta)

f  =  open  ("output.txt",  "w")                   #  ファイルを開く
f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))            #  結果をファイルに出力する

i  =  1
while  i  <=  1000:                                #  同じ処理を 1000 回繰り返す

    dxdt  =  u                                     #  時間微分を計算する
    dzdt  =  w
    dudt  =  -  0.2  *  u
    dwdt  =  -  g  -  0.2  *  w

    x  =  x  +  0.01  *  dxdt                      #  次の時刻の値を計算する
    z  =  z  +  0.01  *  dzdt
    u  =  u  +  0.01  *  dudt
    w  =  w  +  0.01  *  dwdt

    f.write  ("%9.3f  %9.3f¥n"  %  (x,  z))       #  結果をファイルに出力する

    i  =  i  +  1                                  #  カウントをひとつ進める

f.close  ()                                        #  ファイルを閉じる
```

※ボールに対する空気抵抗を考えるときには、空気抵抗の大きさはボールの速さの２乗に比例すると近似する場合が多いですが、ここでは簡単のため、ボールの速さに比例すると仮定しています。

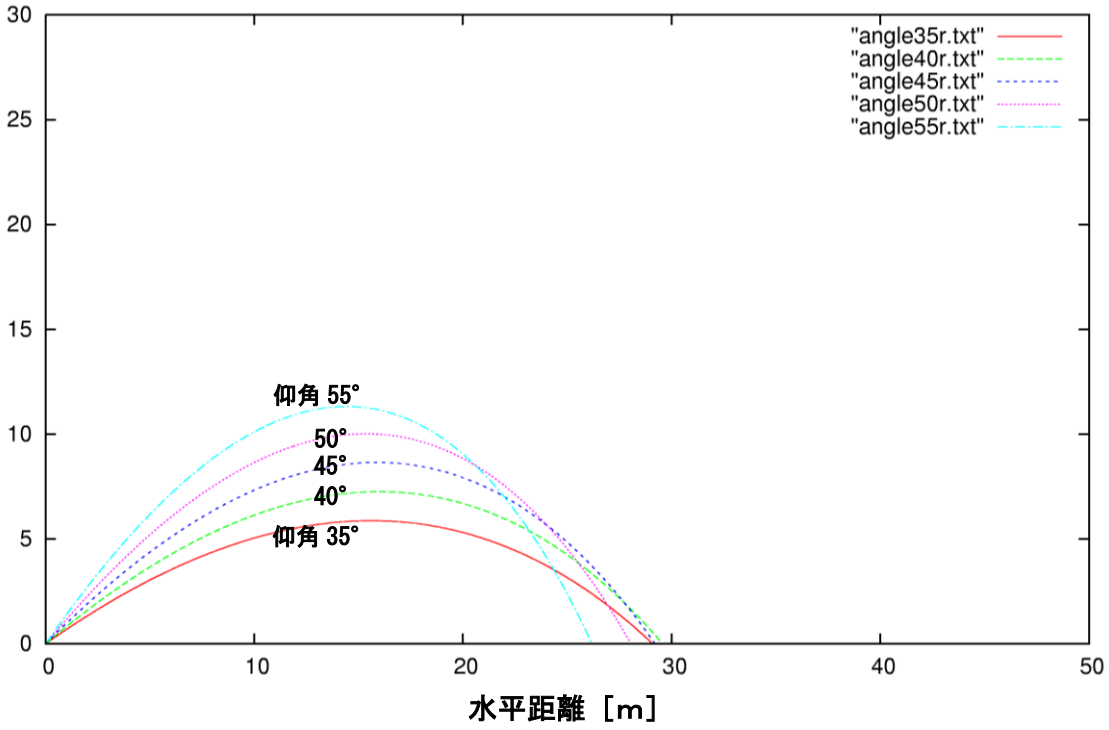
初速度の大きさ 20m/s、仰角 35° で実験　→　出力ファイル名を angle35r.txt に変更。
初速度の大きさ 20m/s、仰角 40° で実験　→　出力ファイル名を angle40r.txt に変更。
初速度の大きさ 20m/s、仰角 45° で実験　→　出力ファイル名を angle45r.txt に変更。
初速度の大きさ 20m/s、仰角 50° で実験　→　出力ファイル名を angle50r.txt に変更。
初速度の大きさ 20m/s、仰角 55° で実験　→　出力ファイル名を angle55r.txt に変更。



計算結果の例（初速度の大きさ 20m/s、仰角 45° の場合）

→ 空気抵抗がある場合のほうが飛距離は短くなる。

高度［m］



計算結果の例（初速度の大きさ 20m/s、空気抵抗がある場合）

→ 飛距離が最大になる仰角は、45° よりも低い。

速さが同じであるならば、
４５° よりも低めに投げたほうが遠くまで飛ぶ！

補遺：文法の解説

【変数の宣言】

Python では、適当な名前の変数に値を代入することによって、自動的に変数が宣言される。整数を代入すれば整数型の変数、実数を代入すれば浮動小数点型の変数（小数点以下を含む実数を表現するための変数）として宣言される。

例： i = 1
　　x = 1.0
　　（実際には左側に空白を入れず行頭から書く。）

変数名の大文字と小文字は区別される。変数名は２文字以上の長さでもよい。プログラムの中では、順序や個数などを表すための整数と、連続的な数量を表すための実数（浮動小数点）は区別される。i = 1 と x = 1.0 は別の数である。

Python では、整数型どうしの四則演算の結果は整数型、浮動小数点型どうし、または整数型と浮動小数点型の四則演算の結果は浮動小数点型になる。ただし、割り算は整数型どうしであっても、演算の結果は浮動小数点型になる（割り切れる場合も含む)。整数型で割り算を実行するときは「 / 」の代わりに「 // 」を使う。この場合、小数点以下は切り捨てになる。

注意：以下、見やすくするために、プログラムの行頭に空白（インデント）を入れているが、実際にプログラムを書くときは、反復や条件分岐などの範囲を示す場合を除き、空白は入れない。

【メッセージを書き出す】

メッセージをターミナルに書き出すためには、print を使う。固定されたメッセージを書き出す場合は、

```
print ("Hello, world!")
```

のようにする。二重引用符で囲むことに注意。

【数値を書き出す】

ターミナルに数値を書き出すためには、

```
print ("i = %9d, x = %9.3f" % (i, x))
```

のようにする。「 %9d 」や「 %9.3f 」は書式指定子とよばれる。この例では、１番目の書式指定子%9d に i の値が、２番目の書式指定子%9.3f に x の値が代入される。「 %9d 」は９桁の整数、「 %9.3f 」は９桁の実数で小数点以下は３桁であることを示している。桁数の指定を省略して、「 %d 」、「 %f 」と書いてもよい。

【数値を読み取る】

ターミナルから数値を読み取るためには、input を使う。たとえば、整数型の変数 i に値を入れる場合は、

```
i = int (input())
```

浮動小数点型の変数 x に値を入れる場合は、

```
x = float (input())
```
のようにする。int や float は入力された文字列を整数型や浮動小数点型に変換するための関数である。

【処理の反復】

同じ処理を反復するためには、次のようにループを作成する。たとえば

```
i = 1
while i <= 1000:
    .....
    .....
    i = i + 1
```
この範囲を４文字インデントする

のようにすれば、４文字インデントされている行の処理が反復される。上の例では、初め変数 i の値は 1 であり、i が 1000 以下であれば同じ処理を繰り返す。１回の処理が終わるごとに変数 i に 1 を加えているので、処理は 1000 回反復することになる。Python では、反復処理を行う範囲をインデントによって示している点に注意する。したがって、単にプログラムを見やすくするという理由でインデントを用いることはできない。

【ファイルに書き出す】

数値をターミナルではなくファイルに書き出すためには、まず open で出力ファイルを開く。

```
f = open ("output.txt", "w")
```
f はファイルオブジェクトとよばれ、開いたファイルを示す目印である。open ではファイル名とモードを指定する。モードは今回の場合"w"であり書き込み可能であることを示している。すでに存在するファイルを指定すると上書きされる。open で開いたファイルに数値を書き出すためには、ファイルオブジェクト f の write メソッドを用いて、

```
f.write ("%9d %9.3f\n" % (x, z))
```
のようにする。基本的には print と同じ使い方である。「ファイルオブジェクト. メソッド」の形になっている点が異なっている。print とは異なり write メソッドでは、行末の改行は自動では付かないので行末に改行を意味する「 \n 」を付ける。ファイルへの書き出しが終わったら、

```
f.close ()
```
でファイルを閉じる。上記の例では、ファイルオブジェクトの名前を f としているが、他の名前であってもよい。

参考：文部科学省による教員研修用教材

高等学校情報科に関する特設ページ
高等学校情報科「情報Ｉ」教員研修用教材
https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1416746.htm
第３章　コンピュータとプログラミング

学習１７　自然現象のモデル化とシミュレーション

（２）物体の放物運動のプログラムによるシミュレーション

```
import math as math                                  # 数値計算ライブラリ
import matplotlib.pyplot as plt                      # グラフ描画ライブラリ

dt = 0.01                                            # 微小時間（時間間隔）
v0 = 30                                              # 初速度
g = 9.8                                              # 重力加速度
x = [0]                                              # 水平位置の初期値は０
y = [0]                                              # 鉛直位置の初期値は０
angle = 45.0 * math.pi / 180.0                       # 投げ上げ角度
vx = [v0*math.cos(angle)]                            # 水平方向の初速度
vy = [v0*math.sin(angle)]                            # 鉛直方向の初速度
for i in range(1000):
    vx.append(vx[i])                                 # 微小時間後の水平方向の速度
    vy.append(vy[i]-g*dt)                            # 微小時間後の鉛直方向の速度
    x.append(x[i]+vx[i]*dt)                          # 微小時間後の水平位置
    y.append(y[i]+(vy[i]+vy[i+1])/2.0*dt)            # 微小時間後の鉛直位置
    if y[i] < 0 :                                    # もし鉛直位置が０を下回ったら
        break                                        # ループ中断
plt.plot(x,y)                                        # 位置の配列をプロット
plt.title("parabolic motion")                        # グラフのタイトル
plt.xlabel("distance")                               # ｘ軸ラベル
plt.ylabel("height")                                 # ｙ軸ラベル
plt.show()
```

【本研修で作成したプログラムの考え方】

```
dudt = 0.0              # 速度の時間微分を計算する
dvdt = - g
dxdt = u                # 位置の時間微分を計算する
dydt = v
u    = u + dudt * dt    # 次の時刻の速度を計算する
v    = v + dvdt * dt
x    = x + dxdt * dt    # 次の時刻の位置を計算する
y    = y + dydt * dt
```

【文部科学省の教材のプログラムの考え方】

```
dudt = 0.0              # 速度の時間微分を計算する
dvdt = - g
uu   = u + dudt * dt    # 次の時刻の速度を計算する
vv   = v + dvdt * dt
dxdt = u                # 位置の時間微分を計算する
dydt = (v + vv) / 2.0
x    = x + dxdt * dt    # 次の時刻の位置を計算する
y    = y + dydt * dt
u    = uu
v    = vv
```

アプリケーションのインストール

※ここでは、WindowsＰＣ上にプログラミング環境を構築する方法を解説します。おもにWindows 10を想定した説明になっていますが、Windows 7やXPでも同様にインストールできます。

以下のインストール作業の解説の内容には十分に注意を払っておりますが、環境によっては指示通りに作業をしても正常にインストールできない場合があります。その場合のサポートには応じられませんので、あらかじめご了承ください。また、アプリケーションのインストールや利用の際に生じたトラブルや損害についても責任を負うことはできませんので、この点も理解のうえご利用ください。

【アカウントの準備】

ＰＣ上での自分のユーザ名（ログインするときの名前）を確認する。ユーザ名が半角英数字でない場合はインストールできない。ユーザアカウントを作成したときには半角英数字でなかったが、後で半角英数字に変更した場合も同様である。ユーザ名が半角英数字でない場合は、半角英数字のユーザ名で新しいアカウントを作成し、そのアカウントでインストールやプログラミングを行なう必要がある。

例：○ hgakugei　　× 学芸花子　　× ｈｇａｋｕｇｅｉ　（←全角英数字）

× 学芸花子からhgakugeiに変更

また、インストール作業では管理者権限が必要である。

【インストール作業】

ＣＤ－ＲＯＭの中のフォルダ「プログラミング環境」に保存されている圧縮フォルダmingw.zipをＰＣにコピーした後、すべて展開して、中身を確認する。

☞mingw.zipを右クリックして「すべて展開」を選択する。

単にダブルクリックしただけでは一時展開なので以下の作業がうまくいかない。

①フォルダMinGWをC:¥にコピーする。

☞画面左下のスタートボタンからWindowsシステムツール、エクスプローラーを選び、ローカルディスク(C:)をクリックすると、C:¥を開くことができる。

②フォルダemacs-24.3をC:¥にコピーする。

③フォルダgnuplotをC:¥にコピーする。

④システム環境変数を設定する：

スタートボタン→Windowsシステムツール→コントロールパネル

→システムとセキュリティ→システム→システムの詳細設定→詳細設定→環境変数

「システム環境変数」の中の「Path」を選択し、「編集」をクリックする。
「新規」をクリックして１行に１項目ずつ、「 C:¥MinGW¥bin 」、「 C:¥emacs-24.3¥bin 」、
「 C:¥gnuplot¥bin 」の合計３項目を追加する。
※Windows 7、XP では、「編集」をクリックした後、「 ; 」で区切りながら、
「 C:¥MinGW¥bin;C:¥emacs-24.3¥bin;C:¥gnuplot¥bin 」を追加する。
☞すでに書かれている内容の末尾にセミコロンを付け加え、その後にかぎ括弧の内容を追記する。

注意：すでに設定されている内容を絶対に変えてはいけない。大文字と小文字の違い、コロンとセミコロンの違いにも注意すること。この部分は特に慎重に行なう必要がある。

以上の設定変更を反映するため、①～③の作業で開いていたウィンドウを閉じて、新たに別のウィンドウを開いて⑤以下の作業を進める。

⑤C:¥MinGW¥msys¥1.0¥msys.bat (msys | Windows バッチファイル)をダブルクリックして実行する。
ターミナルが出てきたら以下のコマンドを実行する。

```
$ exit ↵          ☞exit とタイプして、エンターキーを押す
```

⑥ショートカットを作成する：
C:¥MinGW¥msys¥1.0¥msys.bat へのショートカットと
C:¥MinGW¥msys¥1.0¥home¥*username* へのショートカットを
デスクトップ上に作成する。
☞*username* は(半角英数字の)ユーザ名のことである。

⑦フォルダ files の中のファイル profile と emacs を
C:¥MinGW¥msys¥1.0¥home¥*username*¥の中にコピーする。

⑧ショートカットから msys を開始する。
☞「 msys.bat へのショートカット 」をダブルクリックする。
ターミナルが出てきたら以下のコマンドを実行する。

```
$ mv   profile   .profile ↵          ☞３つめの単語の語頭はピリオドである
$ mv   emacs   .emacs ↵
$ exit ↵
```

以上で基本的なプログラミング環境のインストールは完了である。再び「 msys.bat へのショートカット 」をダブルクリックすればターミナルが起動し、Emacs や gnuplot を利用できるはずである。Python をインストールする場合は、さらに⑨を実行する。

⑨フォルダ python の中の python-3.6.8-amd64.exe をダブルクリックして実行する：
まず以下の２つのオプションにチェックを入れる：

☑ Install launcher for all users (recommended)
☑ Add Python 3.6 to PATH
その後で「 Install Now 」をクリックする。
もし最後の段階で「 Disable path length limit 」というボタンが現れたら、
クリックしてから終了する。

以上で、Python のインストールは完了である。

アンインストールについて
①～③で作成したフォルダを削除し、④で行なった環境変数の変更を元に戻せば、インストール前の状態に戻すことができる。