1　配列を処理するユーザ定義関数の実装について

NumPy は配列に対する処理を実行する便利な関数やメソッドを提供しているが，ユーザ独自の関数の実装が求められることも多い．素朴な方法としては，NumPy の配列の要素 1 つ 1 つに対して施す処理を for などの文で繰り返し実行するというものが挙げられる．ただし for 文による繰り返し処理は実行時間が大きくなることも多く，またプログラムの可読性が低下する原因にもなる．ここでは，ユーザ定義関数の実装において実行時間と可読性の問題を小さくするための方法について紹介する．

■ for, map, vectorize の比較

次のような関数を考える．

$$\sin(x) + \frac{1}{2}\sin(2x) + \frac{1}{3}\sin(3x) + \frac{1}{4}\sin(4x) + \frac{1}{5}\sin(5x) + \frac{1}{6}\sin(6x) + \frac{1}{7}\sin(7x) + \frac{1}{8}\sin(8x)$$

これはノコギリ波（鋸歯状波）の波形を近似的に表現した関数である．この関数を定義域となる配列データに対して実行することを考える．次に示すサンプルプログラム npfunctest01.py は $[-20, 20)$ の範囲の 0.0004 刻みのデータ列 x に対して，上に示した関数を fun として定義して適用するものである．

プログラム：npfunctest01.py

```
# coding: utf-8
# モジュールの読み込み
import numpy as np
import matplotlib.pyplot as plt
import time

x = np.arange(-20,20,0.0004)     # 定義域データの配列
n = len(x)                       # 要素の数
print('要素数：',n)

#################################################################
#   sin(x) + 1/2*sin(2*x) + 1/3*sin(3*x) 求める                 #
#################################################################

# 要素に対する計算を実行する関数
def fun(x):
    return np.sin(x) + 1.0/2.0*np.sin(x*2.0) + \
           1.0/3.0*np.sin(x*3.0) + 1.0/4.0*np.sin(x*4.0) + \
           1.0/5.0*np.sin(x*5.0) + 1.0/6.0*np.sin(x*6.0) + \
           1.0/7.0*np.sin(x*7.0) + 1.0/8.0*np.sin(x*8.0)

#-----  実行時間テスト(1) ----------------------
print('方法1：forによる繰り返し')
t1 = time.time()
ly1 = []
for i in range(n):
    ly1.append( fun(x[i]) )
y1 = np.array(ly1)
t = time.time() - t1
print(t,'秒\n')

plt.plot(x,y1)
plt.show()

#-----  実行時間テスト(2) ----------------------
print('方法2：map関数による方法')
t1 = time.time()
ly2 = map(fun,x)
y2 = np.array(list(ly2))     # この時に計算が実行される
t = time.time() - t1
print(t,'秒\n')

plt.plot(x,y2)
plt.show()

#-----  実行時間テスト(3) ----------------------
```

```
47 print('方法3：np.vectorizeによる方法')
48 t1 = time.time()
49 vfun = np.vectorize(fun)     # 関数が「ベクトル化」される
50 y3 = vfun(x)                 # 計算実行
51 t = time.time() - t1
52 print(t,'秒')
53
54 plt.plot(x,y3)
55 plt.show()
```

解説：

7行目で定義域のデータを生成している．16〜20行目で関数 fun を定義しており，この関数を後の行で定義域の全要素に対して実行する．

25〜28行目では for 文を用いて計算を行い，同様の計算を38〜39行目では map 関数を使用して実行している．49行目では NumPy の vectorize 関数を使用して，関数 fun を vfun に変換している．この結果得られた vfun は NumPy の配列の全要素に対して一度に処理を施し，結果の値を要素として持つ配列を返す．

このプログラムを実行した結果図1のようなグラフが表示される．(3回表示される)

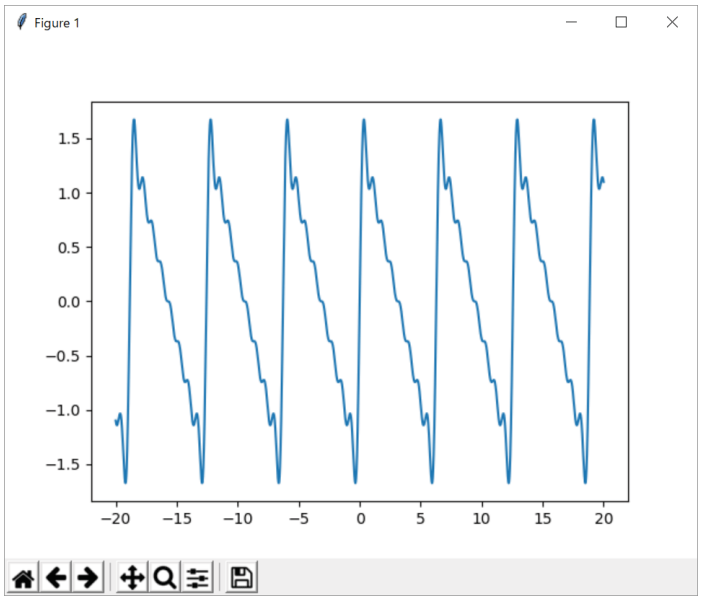

図 1: グラフの表示（3回表示される）

プログラムの実行に伴って次のように標準出力に出力され，for による方法，map による方法，vectorize による各方法での実行時間がわかる．

例. 実行結果の出力例

```
要素数： 100000
方法1：for による繰り返し
2.2479827404022217 秒

方法2：map 関数による方法
1.8539953231811523 秒

方法3：np.vectorize による方法
1.2290003299713135 秒
```

for による実行が最も時間がかかり，map 関数による実行はそれよりも若干早いことがわかる．NumPy の vectorize 関数で**ベクトル化**された関数による実行が最も早い（for と比較して約2倍の速度である）ことがわかる．ただし，計算対象のデータの要素の数や型，実行する関数の定義，さらには計算機環境によって実行時間は変わるので注意が必要である．

2　行列の計算を応用した計算速度の改善の例

多くの場合において NumPy の行列計算の実行速度は非常に早い．計算量が多くなる処理に関しては，行列の計算が応用できるように極力工夫すべきである．先に示したプログラム npfunctest01.py は行列同士の積の計算を応用できる例であり，応用したプログラムを npfunctest02.py に示す．

プログラム：npfunctest02.py

```
# coding: utf-8
# ライブラリの読み込み
import numpy as np
import matplotlib.pyplot as plt
import time

x = np.arange(-20,20,0.0004)     # 定義域データの配列
n = len(x)                       # 要素の数
print('要素数：',n)

#################################################################
#    行列の計算を応用したプログラム                             #
#################################################################
t1 = time.time()         # 時間計測開始
#----- ここから ----------------------------------------------#
# sin(n*x) の配列を n=1～8 について作成して束ねる処理と       #
# 1/n の配列を作成する処理                                     #
#-----------------------------------------------------------#
M = [];      r = []
for n in range(1,9):
    y = np.sin(n*x)      # 各nについてsin(n*x)の配列を作成
    M.append(y)
    r.append(1/n)        # 1/nを作成

R = np.array(r)          # 1/nの配列をNumPyの配列に変換
A = np.vstack(M)         # sin(n*x)の配列を2次元配列に合成
#----- ここまで ----------------------------------------------#

V = np.dot( A.T, R )     # 行列の積を利用して波形合成
t2 = time.time()         # 時間計測終了
print( t2-t1, '秒' )

plt.plot(x,V.T)          # グラフをプロット
plt.show()
```

このプログラムを実行すると図 1 と同じグラフが表示されるが，同一の計算機環境（CPU:Intel Core i7 5500U 2.4GHz）における実行時間が 0.0312 秒であった．これは npfunctest01.py の np.vectorize による方法に比べても 40 倍以上の計算速度である．