第 15 回　数値計算アルゴリズム（ニュートン法）

【ニュートン法】

ニュートン法とは、非線形方程式 f(x)=0 の解を反復法の数値計算によって求める手法であり、収束の早いアルゴリズムとして有名である。適当な初期値 x_0 から始めて、漸化式

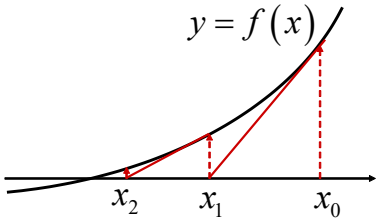
$$x_{k+1} = x_k - f(x_k) / f'(x_k)$$

の収束先から非線形方程式の解を求める。すなわち、y = f(x) のグラフが x 軸と交差するときの x の値を近似的に求めるアルゴリズムである。まず、y = f(x) 上の点 (x_0、$f(x_0)$) を考える。この点における y = f(x) の接線の方程式は、

$$y - f(x_0) = f'(x_0)(x - x_0)$$

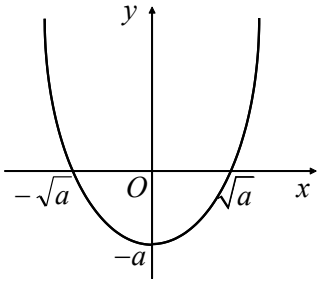
で与えられる（高校数学を思い出して下さい）。

この接線が x 軸と交わる点の x 座標を求めてみましょう。はい、やってみて下さい。



こうして得られた x 座標の値を x_1 と置く。そして同じように、y = f(x) 上の点 (x_1、$f(x_1)$) における接線の方程式を考え、この接線が x 軸と交わる点の x 座標を求め、これを x_2 と置く。こうして順次繰り返していき、$|x_k - x_{k-1}|$ が十分に小さくなった時点で計算を終了する。そのときの x_k の値を近似解とする。

それでは実際に、ニュートン法を用いて整数 a の平方根 $\sqrt{a}$ を求めるプログラムを考える。$\sqrt{a}$ を求めるためには、$y = x^2 - a$ が x 軸と交わる点を求めれば良い。したがって、$f(x) = x^2 - a$ を上記の漸化式に代入し、x_k を求める式を作る。あとはニュートン法に従い、$|x_k - x_{k-1}|$ が十分に小さくなった時点で得られた x_k の値を、a の平方根の近似解とする。



【★課題】

ニュートン法により平方根を求めるプログラムを????に式を入れることによって完成させて、ToyoNet-ACE に提出して下さい。

```
a, x = map(int,input().split())
error = 1e-15
prev_x = x + 1
while (abs(x - prev_x) > error):
    prev_x = x
    x = ????
    print(x)
```

入力：2 10
出力：（最後の行が）1.414213562373095

平方根を求めたい数値は、a に代入される。平方根の近似解を求めるために用いる初期値は x に代入される。前ページの説明では、x_0 に対応する。ニュートン法の漸化式において、x_k と x_{k+1} をともに x とする式を使って、x の値を更新する。つまり、while ループの 1 回目では x_0 から x_1 が計算され、2 回目では x_1 から x_2 が計算される。この計算をする前に、prev_x=x によって x の値を prev_x に保存しておくことで、x - prev_x によって誤差を計算できるようにする。誤差の絶対値 abs(x - prev_x)が許容誤差 error よりも小さくなったところで計算が終了するようにループをまわす。ここで、prev_x の初期値は x+1 としておくことで、必ず最初の 1 回はループを通るようにしている。上の式で a と x_0 に当たる数を入力に指定してプログラムを実行する。

√2 の値を計算してみましょう。正確に計算されているかどうかは import math; print (math.sqrt(2)) とすれば確認できる。なお、この課題はニュートン法による平方根の計算を試みるというものなので、math.sqrt 関数など、他の方法を使った場合は不正解となる。動作確認のためには、√4 のように整数となる計算を試すのも良い。

√2 の計算ができても、√3 の計算ができなければ不正解である。実際に、そのような解答を提出する学生がいるので、√2 以外の値でも、確かめてみること。

ニュートン法についてのいくつかの注釈

(1) 複数の解がある場合には、初期値の選び方によってどの解に収束するのかが変わる。このプログラムの場合は、初期値が正の値なのか負の値なのかによって収束先が決まる。
(2) 解がある場合でも、初期値によっては必ず収束するとは限らない。
(3) n 次元ニュートン法によって n 変数非線形連立方程式を解く手法がある。

【おわりに】

これで授業は終了です。この課題の採点が終わると、課題の平均点が計算され、ガイダンスで示したように課題の平均点から成績が評価されます。課題の平均点と成績評価については、ToyoNet-ACE の成績でこの課題の採点とほぼ同時に見ることができるようになります。

この授業では、ほぼ完成しているプログラムの穴埋めをするという課題をしてきました。プログラミングの考え方について要領がつかめたでしょうか。

プログラミングを習得するためには、半完成プログラムの穴埋めだけではなく、自らプログラムを作ってみる、という経験が重要です。興味を持った学生は、プログラミングの本やネットの記事を参考に、自分で何か作ってみましょう。この授業が足がかりとなり、勉強しやすくなっているはずです。

ちょっとした計算をするときや、定型作業をするときに、この作業は無駄なのでもっと効率化できないだろうか、と考えることはないでしょうか。そのときに、必ずしもプログラミングをしなくても、既存のサービスを使う、作業の流れを工夫する、といったことによって、効率化できる可能性があります。この授業でのプログラミング経験が、そういった作業のアルゴリズムを考える訓練につながっていれば幸いです。