包絡線定理・レポート

山岸　敦

2014/6/7

1　はじめに

（例えば価格のような）パラメーターを含む関数 f についての最適化を考えよう。このとき、そのパラメーターに対して最適化された変数をもとの代入するとパラメーターだけを変数とする関数（価値関数と呼ぶ）ができる。f とこの関数の関係として包絡線定理 (envelope theorem) があり、これは経済学の様々な分野で応用されている…のだが、このような言葉の説明だけ読んだところで実感できないだろう。ここでは、python によって描かれた包絡線定理のグラフをもちいつつ包絡線定理を解説し、その後で描画に用いた python プログラムについて解説する。

2　包絡線定理

さて、いきなり抽象的な関数に対しての議論は難しいかもしれないのでここでは関数形を特定しよう。

変数 t とパラメーター x を持つ関数 f

$$f(x,t) = 2tx - t^2$$

を想定しよう。

これを、変数 t の二次式と見て平方完成すれば

$$f(x,t) = -(t-x)^2 + x^2 \tag{1}$$

を得る。この関数は $t = x$ で最大値 x^2 をとることが (1) の式の形から容易にわかるであろう。この最大値を x の関数とみなし、$V(x) = x^2$ ともとの $f(x,t)$ の関係を考えよう。

$f(x,t)$ の値は、その最大値を取った $V(x)$ の値を上回ることはない。t=x の時だけ $V(x)$ と $f(x,t)$ は同じ値をとり、それ以外の時は $f(x,t)$ が下回る。ここで $f(x,t)$ は x について 1 次関数なので、以上の条件を考えると、x を横軸にとってプロットすれば $f(x,y)$ は $V(x)$ に $x = t$ の点で接する接線になる。

具体例を出そう。たとえば、$t = 2$ なら

$$f(x,t) = 2 \times\ 2x - 2^2 = 4x - 4 \tag{2}$$

となるが、これは確かに x^2 の $x = 2$ における接線になっている。この規則（t の値がなんであれ、関数 f は常に $x = t$ で x^2 の接線になる）を頭に入れておけば、$f(x, t)$ は t の値をどういじっても決して x^2 の上を通らないことがわかる。そこで、t の値をいろいろとって、沢山の接線を書いてみよう。

図 1 は、t の値を変えて $f(x, t)$ を横軸に x を取った平面にプロットしたものである。なんとなく、x^2 のあたりを通過していないことがわかるだろう。図 2 はより多くの t の値に対して $f(x, t)$ を同様にプロットしたもので、本数を増やすにつれ x^2 が浮かび上がってくるのがわかるだろう。このように、$f(x, t)$ の変数 t の値を変えたグラフを何本も描くことで、最適値がどうパラメーターによって変わるかを示すグラフ $V(x)$ が描ける。こういう曲線 $V(x)$ を直線群 f の包絡線、と呼ぶ。

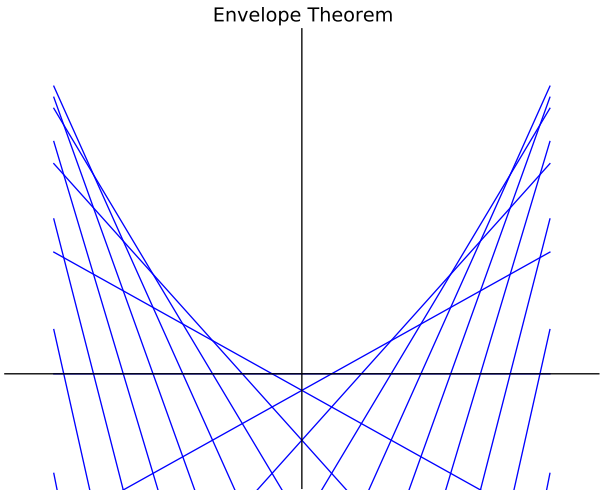

図 1: 接線の本数が少なめの状態

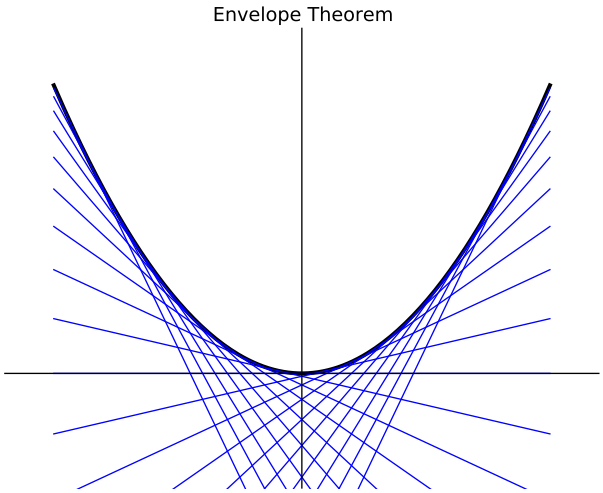

図 2: 多くの接線を引いた状態

さて、$V(x)$ の、x における接線の傾きは、

$$.\frac{dV(x)}{dx} = \frac{d}{dx}f(x,t(x)) \tag{3}$$

です。関数 f に t=x を代入してから、x で微分しろと言っているわけである。ところでこの点においては、接線たる $f(x,t)$ の傾きと (3) の値は一致しているはずだ。つまり

$$\left.\frac{\partial f(x,t)}{\partial x}\right|_{t=x} \tag{4}$$

と (3) の値は一致する（図参照)。(4) は「x で微分してから t に x を代入する」という意味だから、先程と代入と微分の順序が逆転している。一般には両者は一致しないのだが、変数に最適値を代入して作る包絡線の場合には一致する、という寸法である。これを包絡線定理、と呼ぶ。これをまとめておこう。

包絡線定理
$\frac{dV(x)}{dx} = \frac{d}{dx}f(x,t(x)) = \left.\frac{\partial f(x,t)}{\partial x}\right|_{t=x}$

3 Python プログラム

さて、ここからは描画につかった python プログラムのコードを解説しよう。

```
from __future__ import division
import matplotlib.pyplot as plt
import numpy as np

# parameter a = P_INCREMENT * P_COUNT_MIN, P_INCREMENT * (P_COUNT_MIN + 1),
#               ..., P_INCREMENT * P_COUNT_MAX
P_COUNT_MAX = 9
P_COUNT_MIN = -P_COUNT_MAX
P_INCREMENT = 0.5
#P_INCREMENT = 1.2

#choose 0.5 if you want a smooth curve.

# As a,we're gonna define x.
x_max=5
x_min = -x_max
x_increment= 0.05
x_ticks = 2*x_max/x_increment
```

```
def f(x):
return x**2  #sample

def slope(f,a): #approximating the slope of a tangent line
return (f(a+0.0001)-f(a))/0.0001


def subplots():
    "Custom subplots with axes throught the origin"
    fig, ax = plt.subplots()

    # Set the axes through the origin
    for spine in ['left', 'bottom']:
        ax.spines[spine].set_position('zero')
    for spine in ['right', 'top']:
        ax.spines[spine].set_color('none')

    return fig, ax


fig, ax = subplots()  # Call the local version, not plt.subplots()
ax.tick_params(which="both",bottom="off",top="off",left="off",right="off",
labelbottom="off",labeltop="off",labelleft="off",labelright="off")
ax.set_ylim(-10,30) # Moving x-axis downward
x = np.linspace(x_min,x_max,x_ticks)

y = f(x)
ax.plot(x, y, 'k-', linewidth=3)
#this is not necessary if you wanna draw a rough curve


for i in range(P_ COUNT_MIN, P_COUNT_MAX+1):
tan =
x*slope(f,P_INCREMENT*i)+f(P_INCREMENT*i)-slope(f,P_INCREMENT*i)*(P_INCREMENT*i)
ax.plot(x,tan,"b-",linewidth=1) #drawing tangent lines


ax.set_title("Envelope Theorem")
plt.savefig("envelope0.pdf",bbox_inches="tight",pad_inches=0)
plt.show()
```

まず最初に断っておくが、このプログラムは描画目的であって、多少包絡線定理の趣旨とは違った方法で作図している。「最初に x^2 を定義し、それに接線を書く」という方針で作図しているが、包絡線定理に素直に従えば「$f(x,t)$ を t の値を変えて沢山プロットし、x^2 が浮かび上がらせる」という方針になる。その意味で、このプログラムはいわば天下り的に作図していることをご了承願いたい。では、各部分の解説に入ろう。

まず matplotlib と numpy から必要なものを import し、その後で変数の設定パートに入っている。 `P_COUNT_MAX*P_INCREMENT` は、接点の x 座標の最大値を設定している。その符号を変えたものを最小値とし、そこから `P_INCREMENT` ごとに等間隔で x 座標の値をだんだん増やして対応する接線を書いている。

`x_max` はグラフを書く範囲（x の定義域）の最大値を指定している。プログラム上での描画処理の関係上、グラフ上のすべての点を計算し、それを結んでグラフを書くことはできない（有理数は無限にあるのだから…）。だから、`x_increments` でどれくらいの間隔でグラフ上の点を計算して結ぶかを指定している（イメージとしてはグラフのなめらかさ度合い、とでも言うべきか）。`x_ticks` は描画につかう x の数である。

さて、Section2 の $V(x)$ は x^2 だったから、ここでもそれを定義した。次の slope 関数は、グラフの接線を求める関数で、要するに「微分」を近似的にやっている関数である。さらに、subplot 関数を定義する。これは、グラフを書く際の「キャンバス」に少々細工を加える事ができる関数である。初期設定の `plt.subplot` ではウインドウの縁に軸が生成されてしまうのだが、この関数は左、下の縁をウインドウの中心にもっていき、右と上の縁を無色にして見えなくすることでウィンドウの中心に原点を設定している。さらに、`tick_params` 関数と `set_ylim` 関数で、それぞれ軸のデザイン変更、グラフの縮尺調整を行った。

さて、これでグラフを描く舞台と変数が整ったのでいよいよ描画に移る。`linspace` 関数で x に `x_min` から `x_increment` 刻みの値を付与する。そして $y = x^2$ をプロットする。ただし、図 1 ではこのプロットは省いてある。

つぎの for 文は複雑に見えるが、要は以前設定した変数たちを使って、さまざまな接点において接線を書いているだけにすぎない。接線の傾きを導出するときに前もって定義した `slope` 関数を活用している。

あとはグラフにタイトルをつけ、pdf として保存させ、実際にグラフを表示する指示をだしてプログラム終了である。

参考文献

[1] 尾山大輔・安田洋祐「経済学で出る包絡線定理」『経済セミナー』2011 年 10・11 月号.

[2] Thomas J. Sargent and John Stachurski ”Quantitative Economics” http://quant-econ.net/

[3] 辻真吾「Python スタートブック」　技術評論社