Newtonova metoda

Newtonova metoda: Praktická ukázka

Tomáš Kalvoda, 2014

Formulace problému

V této demonstraci použijeme Newtonovu metodu na řešení následujícího problému. Pro zadané $y \in \langle 0, \pi \rangle$ hledejme řešení rovnice

$$x - \sin(x) = y, \quad x \in \langle 0, \pi \rangle$$

Nejprve si rozmysleme, že zadání úlohy je korektní. Označme $f(x) = x - \sin(x)$ s definičním oborem $x \in \langle 0, \pi \rangle$. Pro derivaci této funkce platí $f'(x) = 1 - \cos(x)$. Proto ihned zjišťujeme, že f je rostoucí a tedy i prostá funkce na intervalu $\langle 0, \pi \rangle$. Obraz tohoto intervalu je opět interval $\langle f(0), f(\pi) \rangle = \langle 0, \pi \rangle$. Jinak řečeno, pro každé $y \in \langle 0, \pi \rangle$ existuje právě jedno $x \in \langle 0, \pi \rangle$ splňující $f(x) = y$, tj. $x = f^{-1}(y)$. Toto řešení ale nelze vyjádřit jako elementární funkci. K jeho výpočtu se musíme uchýlit k přibližným numerickým metodám.

Tuto inverzi je potřeba znát pokud chceme určit středový úhel $\alpha \in \langle 0, \pi \rangle$ kruhové výseče na základě znalosti poloměru kruhu R a obsahu příslušné kruhové úseče A. Pro α pak platí $\alpha = f^{-1}(2A/R^2)$. Povšimněte si, že skutečně $2A/R^2 \in \langle 0, \pi \rangle$.

Řešení problému

K řešení použijeme Newtonovu metodu. Rekurentní předpis pro posloupnost aproximující řešení $F(x) = f(x) - y = 0$ pro zadané $y \in \langle 0, \pi \rangle$ je

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)} = x_n - \frac{x_n - \sin(x_n) - y}{1 - \cos(x_n)}, \quad n = 0, 1, 2, \ldots$$

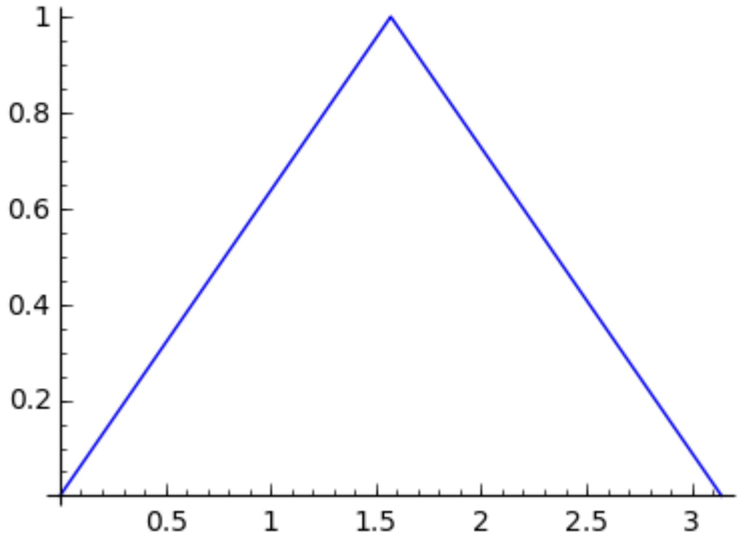
Zbývá vhodně zvolit nultý člen posloupnosti, první tip na řešení. Možností může být více, my jako první nástřel použijeme zjednodušené úlohy

$$x - S(x) = y. \quad \text{resp. } x - y = S(x),$$

kde jsme funkci sin nahradili po částech lineární funkcí $S(x)$ procházející body $(0,0)$, $(\pi/2,1)$ a $(\pi,0)$.

```
def S(x):
    if 0 <= x <= pi/2:
        return 2/pi*x
    elif pi/2 < x < pi:
        return -2/pi*(x - pi)

plot(S, (0,pi), figsize=4)
```



Řešením této úlohy, jak snadno zjistíme, je

```
def approx(y):
    if y < 0.2:
        return (6*y)^(1/3)
    elif y <= pi/2-1:
        return y/(1-2/pi)
    else:
        return (y+2)/(1+2/pi)
```

U nuly jsme ovšem použili řešení úlohy, kde jsme místo funkce sin použili její třetí Taylorův polynom v bodě nula. Tato volba dává daleko lepší výsledky pro malé vstupy (v nule je nula derivace!).

Implementace

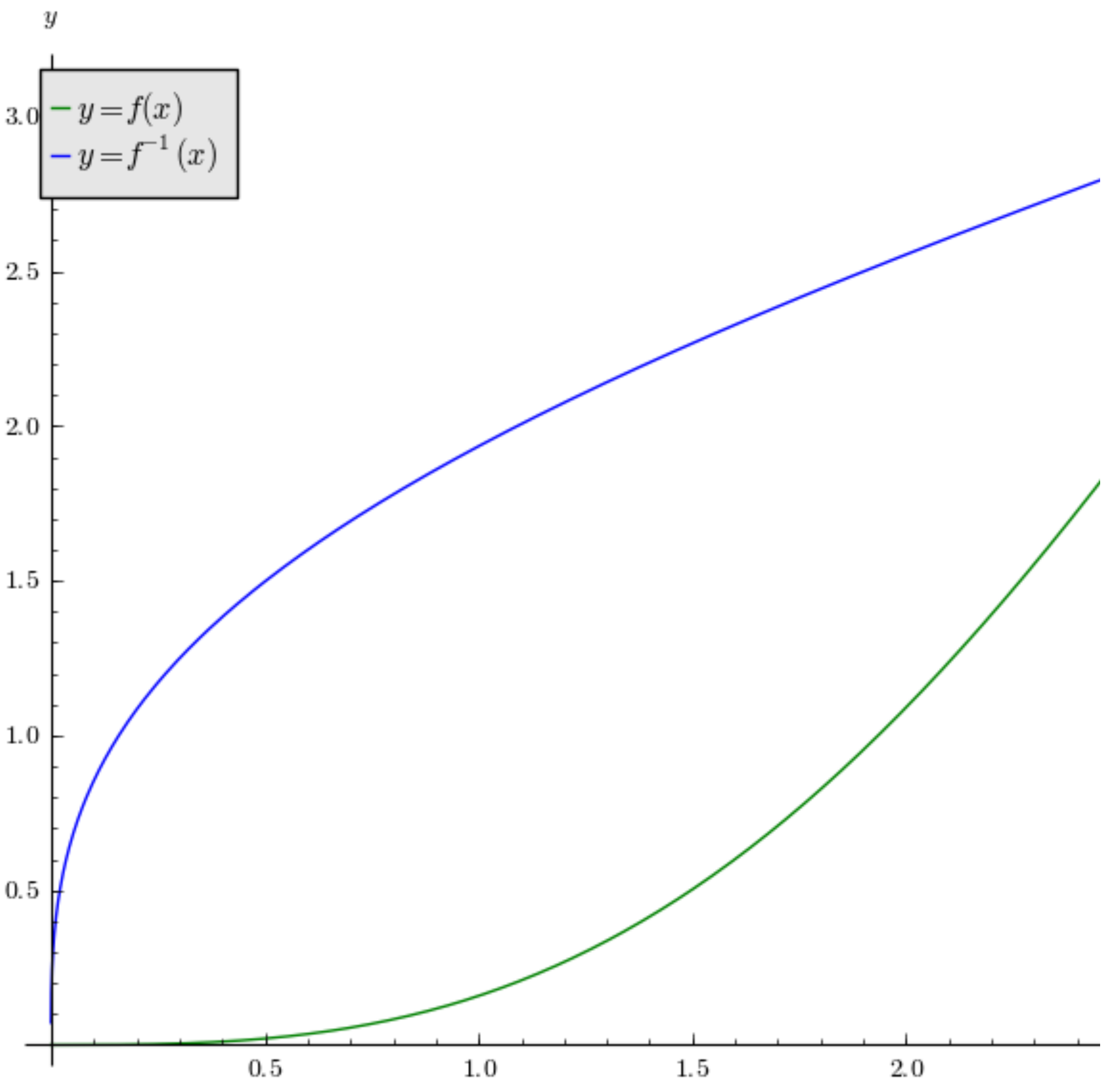
Nyní stačí dát všechny ingredience dohromady.

```
f(x) = x-sin(x)
```

```
def invf(x, eps):
    x0 = approx(x)
    x1 = x0 - (x0 - sin(x0) - x)/(1 - cos(x0))
    while abs(x0 - x1) > eps:
        x1, x0 = x1 - (x1 - sin(x1) - x)/(1 - cos(x1)), x1
    return x1
```

Graf funkce f a její inverze f^{-1} napočtené způsobem uvedeným výše.

```
%timeit
show(plot(lambda x: f(x), (0,pi), adaptive_recursion=8,
plot_points=500, color='green', legend_label='$y=f(x)$',
axes_labels=['$x$','$y$'], tick_formatter='latex') +
plot(lambda x: invf(x,0.001), (0,pi), adaptive_recursion=8,
plot_points=500, color='blue', legend_label='$y=f^{-1}(x)$'))
```




```
CPU time: 46.95 s,  Wall time: 47.00 s
```

Kompilovaná verze

Výše uvedená verze funkce **invf**, napsaná přímo v Pythonu, není příliš vhodná. Zejména pokud budeme tuto funkci volat často. Všimněte si, že i jenom vykreslení jejího grafu zabralo takřka minutu. Přesně pro tento případ Sage nabízí Cython. Syntaxí je tento jazyk prakticky totožný s Pythonem, navíc ale můžeme proměnné typovat, volat funkce C a kompilovat kód přímo z Notebooku.

V následující buňce nejprve importujeme potřebné matematické funkce a pak definujeme naší kompilovanou verzi funkce **invf**. Všimněte si, že kód je prakticky copy-past verze výše uvedeného kódu.

```
%cython

from libc.math cimport sin, cos, cbrt, M_PI

def cinvf(double x, double eps):
    cdef double x0
    if x < 0.2:
        x0 = cbrt(6*x)
    elif x <= M_PI/2-1:
        x0 = x/(1-2/M_PI)
    else:
        x0 = (x+2)/(1+2/M_PI)
    cdef double x1 = x0 - (x0 - sin(x0) - x)/(1 - cos(x0))
    while abs(x0 - x1) > eps:
        x1, x0 = x1 - (x1 - sin(x1) - x)/(1 - cos(x1)), x1
    return x1
```

__home_kal...0_code_sage57_spyx.c   __home_kal...ode_sage57_s

Porovnejme nyní obě funkce, konkrétně čas jejich běhu. K tomu lze použít šikovnou funkci **timeit**. Aby měření mělo smysl, provedeme vždy 1000 volání obou funkcí.

```
# původní verze
timeit('invf(1.15, 0.01)', number=1000)
```

    1000 loops, best of 3: 10 ms per loop

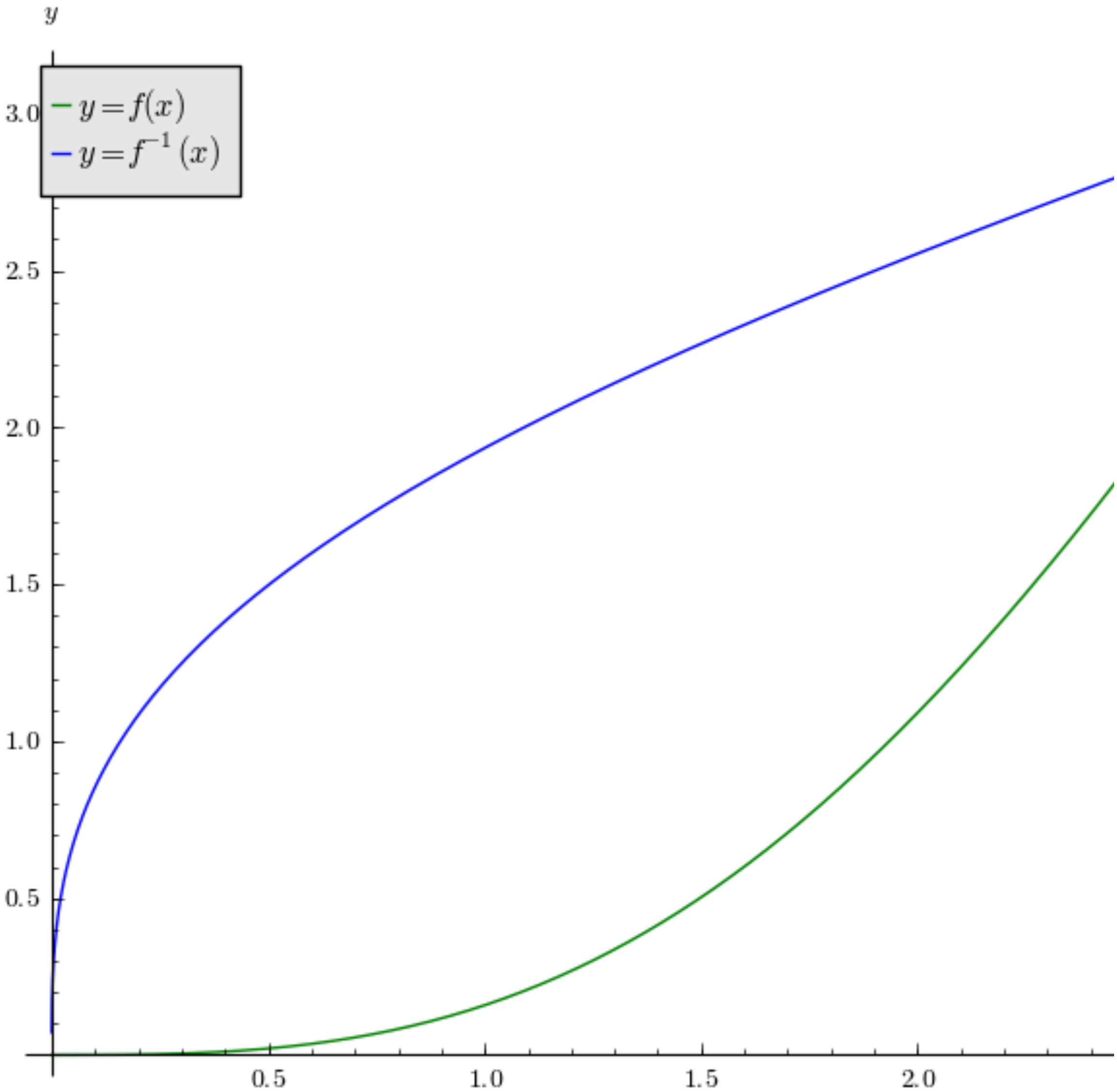
```
# kompilovaná verze
timeit('cinvf(1.15, 0.01)', number=1000)
```

    1000 loops, best of 3: 8.01 µs per loop

Vidíme, že rozdíl v době výpočtu je propastný. Očividný je tento fakt také v době potřebné na vykreslení grafu. Nyní graf dostaneme prakticky okamžitě.

```
%timeit
show(plot(lambda x: f(x), (0,pi), adaptive_recursion=8,
```

```
plot_points=500, color='green', legend_label='$y=f(x)$',
axes_labels=['$x$','$y$'], tick_formatter='latex') +
plot(lambda x: cinvf(x,0.001), (0,pi), adaptive_recursion=8,
plot_points=500, color='blue', legend_label='$y=f^{-1}(x)$'))
```



CPU time: 1.10 s, Wall time: 1.11 s