Trzecie zadanie z kolokwium RPiS - sprawozdanie

Kacper Bajkiewicz

8 maja 2020

1 Słowem wstępu

Zadanie polega na numerycznym wyznaczeniu dystrybuanty rozkładu t-studenta z n stopniami swobody dla dowolnego dodatniego $x \in \mathbb{R}$. Gęstość rozkładu dana jest wzorem:

$$f(x) = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(n/2)\sqrt{n\pi}}(1+\frac{x^2}{n})^{-\frac{n+1}{2}} \tag{1}$$

Znakiem tego dystrybuanta jawi się pod postacią:

$$F(x) = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(n/2)\sqrt{n\pi}}\int_{-\infty}^{x}(1+\frac{x^2}{n})^{-\frac{n+1}{2}}dx \tag{2}$$

W moim rozwiązaniu użyję metody Romberga, która opisana została w zadaniu *kolokwialnym* nr 1, wobec czego w tym sprawozdaniu nie będę się skupiać na jej przedstawieniu od strony teoretycznej.

2 Potrzebne przekształcenia i uproszczenia

Najpierw pokażemy parzystość funkcji gęstości, zatem:

$$f(x) = (1+\frac{x^2}{n})^{-\frac{n+1}{2}} = (1+\frac{(-x)^2}{n})^{-\frac{n+1}{2}} = f(-x) \tag{3}$$

Zatem gęstość jest parzysta (bo to dziwne coś z funkcją Γ nie wpływa na parzystość). Wobec tego dla dystrybuanty zachodzi $F(x) = 1 - F(-x)$ (paralelnie jak w przypadku zadania pierwszego). Znakiem tego $2 \times F(0) = 1$, czyli: $F(0) = \frac{1}{2}$.
Pokażmy jeszcze jeszcze parę rzeczy. Wiadomo, że stopnie swobody należą do liczb naturalnych, więc sprawa z funkcją Γ trochę się ułatwia. Zachodzą następujące własności dla $k \in \mathbb{N}_+$:

$$\Gamma(k) = (k-1)! \tag{4}$$

$$\Gamma(k+\frac{1}{2}) = \frac{(2k-1)!!\sqrt{\pi}}{2^n} \tag{5}$$

Rozpatrzmy dwa przypadki:

2.1 Nieparzysty stopień swobody

Wtedy n jest nieparzyste, zaś $n+1$ parzyste ($\frac{n+1}{2} \in \mathbb{N}_+$). Wobec tego zachodzi:

$$\Gamma(\frac{n+1}{2}) \overset{(4)}{=} (\frac{n+1}{2}-1)! = (\frac{n-1}{2})! \tag{6}$$

Ale $\frac{n}{2}$ jest nieparzyste, czyli $\frac{n}{2} \notin \mathbb{Z}$:

$$\Gamma(\frac{n}{2}) = \Gamma(\frac{n-1}{2}+\frac{1}{2}) \overset{(5)}{=} \frac{(2\frac{n-1}{2}-1)!!\sqrt{\pi}}{2^{(n-1)/2}} = \frac{(n-2)!!\sqrt{\pi}}{2^{(n-1)/2}}, n > 1 \tag{7}$$

Zaznaczam, że działa to tylko dla $n > 1$, wiadomo, że $\Gamma(1) = 1$.

2.2 Patrzysty stopień swobody

Wtedy n jest parzyste, czyli $\frac{n}{2} \in \mathbb{N}_+$, wobec tego zachodzi:

$$\Gamma(\frac{n}{2}) \stackrel{(4)}{=} (\frac{n}{2} - 1)! = (\frac{n-2}{2})! \tag{8}$$

A dla $\frac{n+1}{2}$ otrzymujemy:

$$\Gamma(\frac{n+1}{2}) = \Gamma(\frac{n}{2} + \frac{1}{2}) \stackrel{(5)}{=} \frac{(2\frac{n}{2} - 1)!!\sqrt{\pi}}{2^{\frac{n}{2}}} = \frac{(n-1)!!\sqrt{\pi}}{2^{\frac{n}{2}}} \tag{9}$$

Nie rozpatrujemy przypadku, kiedy stopień swobody jest równy 0, bo dla punktu $x = 0$ funkcja Gamma nie przyjmuje wartości rzeczywistej.

2.3 Podsumowanie tej części

Końcowy wzór na dystrybuantę wygląda dokładnie tak:

$$F(x) = \frac{1}{2} + \frac{\Gamma(\frac{n+1}{2})}{\Gamma(n/2)\sqrt{n\pi}} \int_0^x (1 + \frac{x^2}{n})^{-(n+1)/2} dx \tag{10}$$

Więc zostaje mi rozwiązać dwa minizadania - znaleźć numerycznie całkę i iloraz (na lewo od całki w powyższym równaniu). Końcowy wynik będzie wtedy na wyciągniecie ręki, wystarczy przecież połączyć te dwa osiągnięcia!

3 Pora na liczenie!

3.1 Funkcje Gamma

Postanowiłem napisać funkcję, która dla zadanego $n \in \mathbb{N}_+$ wyliczy wartość całego ilorazu, bazując na równaniach od (6) do (10). Kiedy stopień swobody to 1, to całość ewaluuję na podstawie powszechnie znanych własności.

```
def calculate_iloraz(n):
    def podwojna_silnia(k):
        if (k == 1) or (k == 0):
            return 1
        return k*podwojna_silnia(k-2)
    def pojedyncza_silnia(k):
        if k == 0:
            return 1
        return k*pojedyncza_silnia(k-1)
    if n == 1: #no to wtedy Γ(1/2) = sqrt(π) a Γ(1) = 1, sqrt(1) = 1
        return 1/pi #sprowadza nam sie do 1/sqrt(π)*sqrt(π) = 1 / π
    if n % 2 == 0:
        gora = podwojna_silnia(n-1)*sqrt(pi)/2**(n/2)
        dol = pojedyncza_silnia((n-2)/2)
    if n % 2 == 1:
        gora = pojedyncza_silnia((n-1)/2)
        dol = podwojna_silnia(n-2)*sqrt(pi)/(2**((n-1)/2))
    return gora/(dol*sqrt(n*pi))
```

3.2 Całka wyznaczana numerycznie

Warto przypomnieć, jak działa algorytm wykorzystujący metodę Romberga.

```
def romberg_approx(measur_err, function, a, b):
    def calculate_romberg(n, m):
        if m != 0:
            return romberg_tab[n][m-1] + ((romberg_tab[n][m-1] - romberg_tab[n-1][m-1]) / ((4 ** m) - 1))
        else:
            temp_sum = 0
            for i in range(1, 2**(n-1)+1):
                temp_sum += function(a + (2*i - 1)*h)
            return (romberg_tab[n-1][0] / 2) + h * temp_sum
    romberg_tab = []
    h = b - a
    romberg_tab.append([h*(function(a) + function(b)) / 2]) #dodajemy pierwszy wyraz R(0,0)
    k = 1
    while True:
        h = h / 2
        romberg_tab.append([]) #dodajemy kolejny wiersz naszemu Rombergowi R(k, ...)
        romberg_tab[k].append(calculate_romberg(k, 0)) #dodajemy wyraz R(k, 0)
        for i in range(1, k+1):
            romberg_tab[k].append(calculate_romberg(k, i))
            if abs(romberg_tab[k][i] - romberg_tab[k-1][i-1]) < measur_err:
                return romberg_tab[k][i]
        k += 1
```

Program przerywa działanie kiedy spełniony jest warunek $|R(n,k)-R(n-1,k-1)| < measurerr$, w tym przypadku będę liczył dla błędu równego 0,0001. Obliczenie $\int_0^x (1+\frac{x^2}{n})^{-(n+1)/2} dx$ wygląda tak (wynik zaokrąglam do 6ciu miejsc po przecinku):

```
def calculate_tstudent_cdf(x, n):
    def pom(x):
        return (1 + (x*x)/n)**(-(n+1)/2)
    iloraz = calculate_iloraz(n)
    return round(1/2 + iloraz*romberg_approx(0.0001,pom,0,x), 6)
```

Na koniec dołączę jeszcze trochę testów:

```
Wartość dystrybuanty dla stopnia swobody 1 i x = 1 to 0.75
Wartość dystrybuanty dla stopnia swobody 3 i x = 0.5 to 0.674276
Wartość dystrybuanty dla stopnia swobody 2 i x = 0.2 to 0.570014
Wartość dystrybuanty dla stopnia swobody 9 i x = 1 to 0.828282
Wartość dystrybuanty dla stopnia swobody 6 i x = 0.125 to 0.547697
```

Kacper Bajkiewicz
nr indeksu 314438