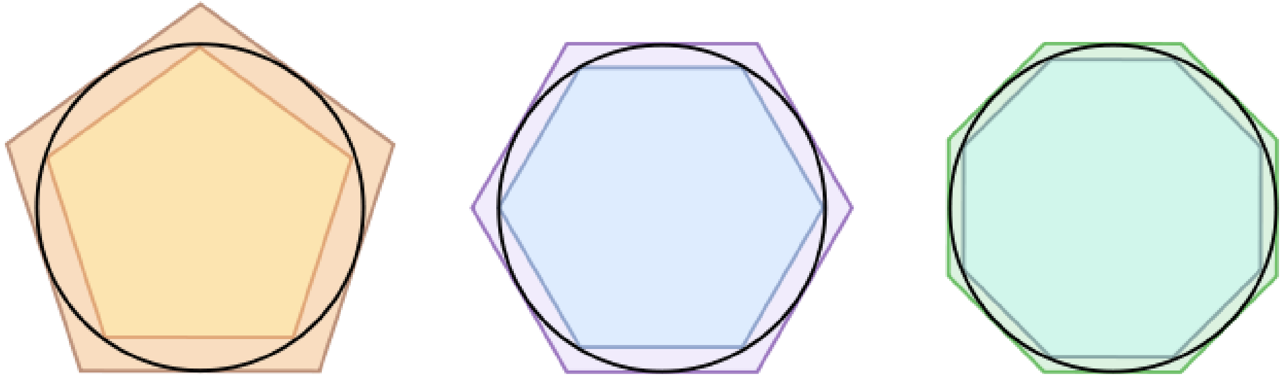
BDMI-课程编号-01510243

大数据与机器智能

如何近似计算圆周率π?

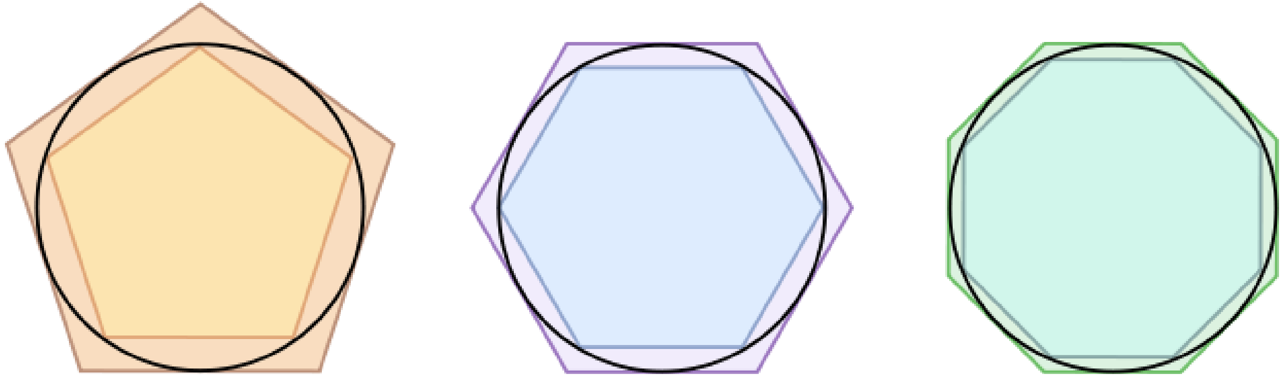
如何近似计算圆周率π?

如何近似计算圆周率π?



π可以透过计算圆的外切多边形及内接多边形周长来估算

如何近似计算圆周率π?



π可以透过计算圆的外切多边形及内接多边形周长来估算

祖冲之在公元480年利用割圆术计算12,288形的边长，得到 $\pi \approx \dfrac{355}{113}$ （现在称为密率），其数值为3.141592920，小数点后的前六位数都是正确值。在之后的八百年内，这都是准确度最高的π估计值。为纪念祖冲之对圆周率发展的贡献，日本数学家三上义夫将这一推算值命名为“祖冲之圆周率”，简称“祖率”。

如何近似计算圆周率π?

$$\arctan(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1} = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots$$

反正切泰勒级数

如何近似计算圆周率π?

$$\arctan(x) = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1} = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \cdots$$

反正切泰勒级数

当x=1时， arctanx = π/4

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

arctan1

如何近似计算圆周率π?

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

arctan1

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + - \dots$$

如何近似计算圆周率π?

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + - \ldots$$

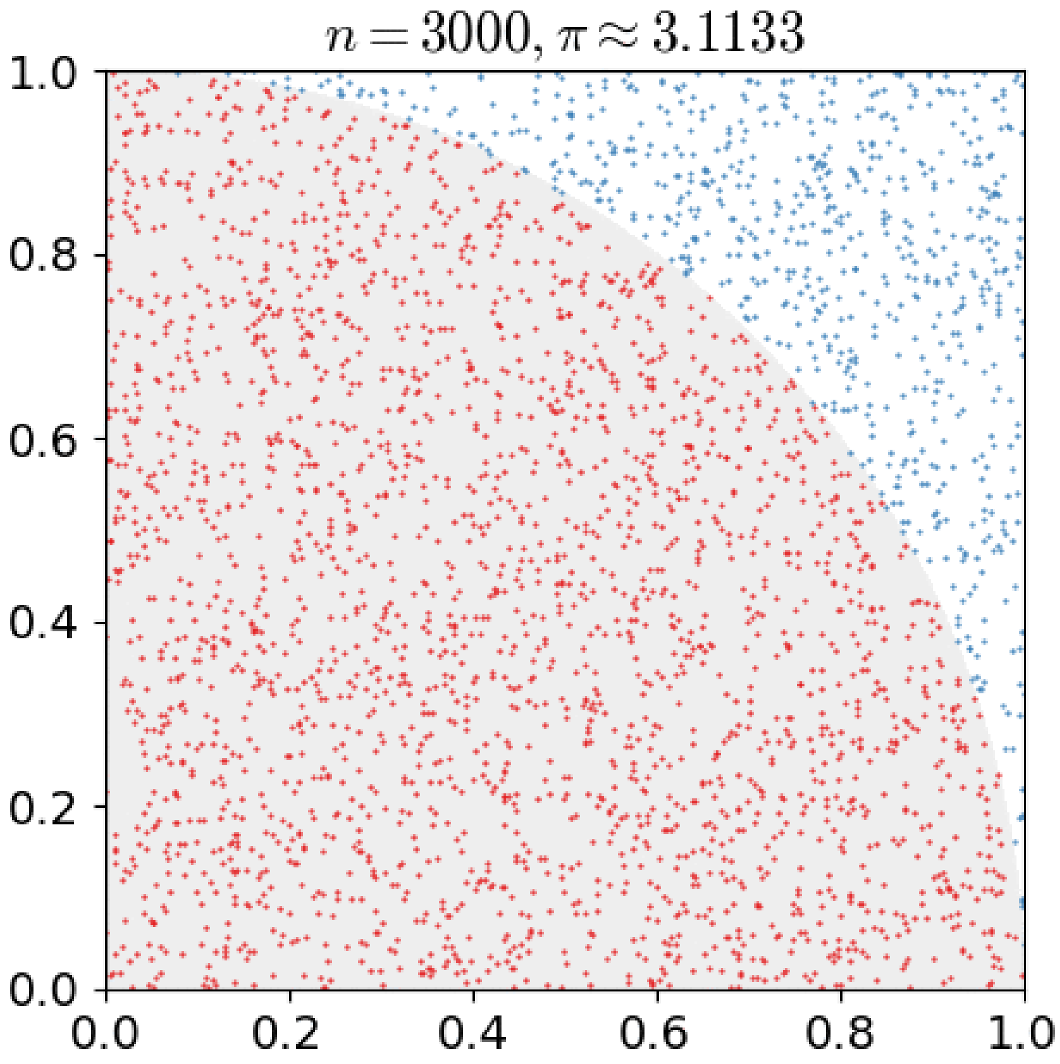
```python
def TaylorPi(k):
    sum,odd = 0,True
    for i in range(1,k):
        sum += 1/(2*i-1) if odd==True else -1/(2*i-1)
        odd = not odd
    print("Taylor PI:",sum*4)
```

如何近似计算圆周率π?

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + - \ldots$$

```haskell
module TaylorPi (main) where
series xs = foldl step 0 xs
    where step acc x
            | (odd x) && (odd(truncate(fromIntegral(x)/2)))
            = acc - 1 / fromIntegral(x)
            | (odd x) && (odd(truncate(fromIntegral(x)/2)+1))
            = acc + 1 / fromIntegral(x)
            | otherwise
            = acc
series_length = 100000
main = print(series([0..series_length]) * 4)
```

如何近似计算圆周率π?



$$\frac{S_1}{S_2} = \frac{\pi/4}{1} = \frac{\pi}{4} \approx \frac{N_1}{N_2}$$

$$\therefore \pi \approx \frac{4N_1}{N_2}$$

当N_2=30000时，π的估计值与真实值只相差0.07%

如何近似计算圆周率π?

```python
import random

if __name__ == '__main__':
    N2 = 30000
    N1 = 0.
    for i in range(N2):
        x = random.random()
        y = random.random()
        if x*x+y*y<=1:
            N1+=1
    print("PI:",4*N1/N2)
```

$$\frac{S_1}{S_2} = \frac{\pi/4}{1} = \frac{\pi}{4} \approx \frac{N_1}{N_2}$$

$$\therefore \pi \approx \frac{4N_1}{N_2}$$

当N_2=30000时，π的估计值与真实值只相差0.07%

如何近似计算圆周率π?

$$\frac{S_1}{S_2} = \frac{\pi/4}{1} = \frac{\pi}{4} \approx \frac{N_1}{N_2} \qquad \therefore \pi \approx \frac{4N_1}{N_2}$$

当N_2=30000时，π的估计值与真实值只相差0.07%

```haskell
module MonPi (main) where
import System.Random
random_times = 1000000
xcoor = take random_times $ randoms (mkStdGen 100) :: [Double]
ycoor = take random_times $ randoms (mkStdGen 101) :: [Double]
myzip :: [Double] -> [Double] -> [(Double, Double)]
myzip xs [] = []
myzip [] ys = []
myzip (x:xs) (y:ys) = (x, y) : myzip xs ys
xycoor = myzip xcoor ycoor
filtered_xycoor = filter (\s -> (fst s)^2 + (snd s)^2 < 1) xycoor
main = print(fromIntegral(length filtered_xycoor) / fromIntegral(random_times) * 4.0)
```

如何近似计算圆周率π?

Chudnovsky公式

$$\pi = \frac{426880\sqrt{10005}}{\sum_{k=0}^{\infty} \frac{(6k)!(13591409+545140134k)}{(3k)!(k!)^3(-640320)^{3k}}}$$

这个公式可以做到每计算一项得出15位有效数字！1994年，人们利用这个公式，得到了圆周率小数点后40.44亿位。

如何近似计算圆周率π?

Chudnovsky公式

```
def Ch_cal(k):
    uper_value = math.factorial(6*k)*(13591409+545140134*k)
    lower_value = math.factorial(3*k)*math.pow(math.factorial(k),3)*math.pow((-640320),3*k)
    return uper_value/lower_value

def Chudnovsky(number):
    uper_value = 426880*math.sqrt(10005)
    lower_sum = 0.
    for k in range(number):
        lower_sum+=Ch_cal(k)
    print("Chudnovsky PI:",uper_value/lower_sum)
```

如何近似计算圆周率π?

Chudnovsky公式

```haskell
module ChudnovskyPi (main) where

factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)


series xs = foldl step 0 xs
    where step acc x = acc + (fromIntegral(factorial(6*x)) *
    fromIntegral(13591409 + (545140134 * x)))/fromIntegral(factorial(x+x
    +x))  / (fromIntegral(factorial(x)))^3 / (-640320)^fromIntegral(3*x)


series_length  = 10
main = print(426880.0 * sqrt(10005) / series([0..series_length]))
```

如何近似计算圆周率π?

迭代算法：

1. 设置初始值：

$$a_0 = 1 \qquad b_0 = \frac{1}{\sqrt{2}} \qquad t_0 = \frac{1}{4} \qquad p_0 = 1.$$

2. 反复执行以下步骤直到a_n与b_n之间的误差到达所需精度：

$$a_{n+1} = \frac{a_n + b_n}{2},$$
$$b_{n+1} = \sqrt{a_n b_n},$$
$$t_{n+1} = t_n - p_n (a_n - a_{n+1})^2,$$
$$p_{n+1} = 2p_n.$$

3. 则π的近似值为：

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

如何近似计算圆周率π?

迭代算法：

```python
def Iterative_cal(number):
    a_now = 1.
    b_now = 1./math.sqrt(2)
    t_now = .25
    p_now = 1.
    for i in range(number):
        a = (a_now+b_now)/2
        b = math.sqrt(a_now*b_now)
        t = t_now-p_now*math.pow((a_now-a),2)
        p = 2*p_now

        a_now = a
        b_now = b
        t_now = t
        p_now = p
    print("Iterative PI:",math.pow(a_now+b_now,2)/(4*t_now))
```

如何近似计算圆周率π?

迭代算法:

```haskell
module IterativePi (main) where

iter_times = 25
fib a b c d = a:b:c:d:fib ((a+b)/2) (sqrt(a*b)) (c-d*(a-(a+b)/2)^2) (2*d)
x = iter_times * 4
series = take x (fib 1.0 (1/sqrt(2.0)) 0.25 1.0)
s1 = series !! (x-4)
s2 = series !! (x-3)
s3 = series !! (x-2)
s4 = series !! (x-1)
main = putStrLn (show (((s1+s2)^2)/(4*s3)))
```