通过牛顿迭代法计算平方根:

$$x_1 = \frac{x_0 + \frac{x}{x_0}}{2}$$
$$x_0 = x_1$$

In [ ]:
```
import math

def mysqrt(x):
    epsilon = 1e-6
    x0 = 1
    x1 = 1
    #默认输入求解结果大于零
    while True:
        x0 = (x0 + x / x0) / 2
        if abs(x0 - x1) < epsilon:
            break
        x1 = x0
    return x0

a = 5
out1 = math.sqrt(a)
out2 = mysqrt(a)

print("The result via Python's Math is :" + str(out1))
print("The result via Newton's Method is:" + str(out2))
```

```
The result via Python's Math is :2.23606797749979
The result via Newton's Method is:2.236067977499978
```

通过割圆术计算PI的值: 其思想是将圆等分为n份后在不断做二分逼近,最终在数值上做出近似的结果。 代码实现上大抵分为:DP算法,递归算法等

In [ ]:
```
r = 1  # Global Variable


#递归算法效率太低 没有编写

#动态规划算法
def calcuPI(n:int):
    i = 1
    edge_list = list()
    edge_list.append(1)
    while i <= n:
        x_n1 = edge_list[i-1]
        x_n = math.sqrt(x_n1**2 / 4 + (1 - math.sqrt(1 - x_n1**2 / 4))**2)
        edge_list.append(x_n)
        i = i+1
    x = edge_list[n-1]
    return 6 * (2**(n-1)) * math.sqrt(1 - x**2 / 4) * x / 2


out1 = calcuPI(7)

print("The PI calculated by 6*xe+7 polygon is:" + str(out1))
print("The division of myPI and math.pi is:" + str(out1 / math.pi))
```

```
The PI calculated by 6*xe+7 polygon is:3.1414524722854624
The division of myPI and math.pi is:0.9999553789049734
```

In [ ]:
```
n1 = int(math.log2(96 / 6))
n2 = int(math.log2(192 / 6))

omiga = 1 / 3

myPI = calcuPI(n2) + omiga * (calcuPI(n2) - calcuPI(n1))

print("The result of the loosen calcu is:" + str(myPI))
print("The standardModule of Python is:" + str(math.pi))
```

```
The result of the loosen calcu is:3.141590732968744
The standardModule of Python is:3.141592653589793
```

课本 P49 ex17

设$f(x) = \frac{1}{1+x^2}$,在$-5 \leq x \leq 5$上取n = 10,按等距节点求分段线性插值函数$I_n(x)$,计算各节点间中点处的$I_n(x)$与$f(x)$的值和误差。

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np


def linearInterpolation(x1, x2, y1, y2, x):
    return y1 + (y2 - y1) * (x - x1) / (x2 - x1)


def calcuInterpolation(x_in,y_in, x):
    for i in range(len(x_in) - 1):
        if x >= x_in[i] and x < x_in[i + 1]:
            return linearInterpolation(x_in[i], x_in[i + 1], y_in[i], y_in[i + 1], x)




def targetFunction(x):
    return 1 / (1 + x**2)

x0 = np.linspace(-5,5,11)

y0 = targetFunction(x0)

x = np.linspace(-5,5,10000)

y = targetFunction(x)

y_interpolation = list(map(lambda t:calcuInterpolation(x0,y0,t),x))

x_redis = [(x0[i] + x0[i+1]) / 2 for i in range(len(x0) - 1)]
y_redis = [targetFunction(x_redis[i]) - calcuInterpolation(x0,y0,x_redis[i]) for i in range(len(x_redis))]

print("中点数值残差和为:" + str(sum(y_redis)))

plt.scatter(x0,y0,color = "red")
plt.plot(x,y)
plt.plot(x,y_interpolation)
plt.scatter(x_redis,y_redis,color = "orange")

plt.legend(["ScatterOfX&Y","targetFuction","linearInterpolation","Redisual"])
plt.show()
```

```
中点数值残差和为:-0.019800869649696504
```