数値計算

d7 補間と数値積分 5/31

チーム名：π息

36021334 坂内崚真
36021348 中村聡太
36021372 八木幹太
36021302 兼松颯太

課題

補間法

次の4点

```
maple
x y
0 1
1 2
2 3
3 1
```

を通る多項式を逆行列で求めよ.

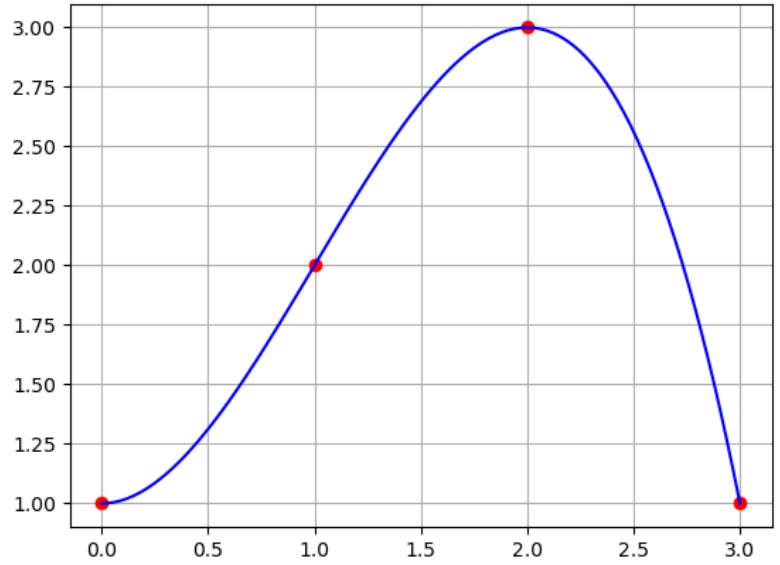
ラグランジュ補間式

In [12]:
```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt

# もとの点
x = np.array([0,1,2,3])
y = np.array([1,2,3,1])
for i in range(0,4):
    plt.plot(x[i],y[i],'o',color='r')


# Lagrange補間
f = interpolate.lagrange(x,y)
print(f)
x = np.linspace(0,3, 100)
y = f(x)
plt.plot(x, y, color = 'b')
```



```
plt.grid()
plt.show()
```

```
   3     2
-0.5 x + 1.5 x + 2.776e-16 x + 1
```



逆行列で求める

In [17]:
```
from pprint import pprint

x = np.array([0,1,2,3])
y = np.array([1,2,3,1])
n=4
A=np.zeros((n,n))
b=np.zeros(n)
for i in range(0,n):
    for j in range(0, n):
        A[i,j]+=x[i]**j
        b[i]=y[i]
pprint(A)
pprint(b)
```

```
array([[ 1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  2.,  4.,  8.],
       [ 1.,  3.,  9., 27.]])
array([1., 2., 3., 1.])
```

In [18]:
```
import scipy.linalg as linalg

inv_A = linalg.inv(A)
pprint(inv_A)
np.dot(inv_A,b)
```

```
array([[ 1.      , -0.      ,  0.      , -0.      ],
       [-1.83333333,  3.      , -1.5     ,  0.33333333],
       [ 1.      , -2.5     ,  2.      , -0.5     ],
       [-0.16666667,  0.5     , -0.5     ,  0.16666667]])
```

Out[18]:
```
array([ 1. ,  0. ,  1.5, -0.5])
```

補間式と逆行列で求めたが、概ね同じ係数となった。結果、多項式は
$F(x) = -0.5x^3 + 1.5x^2 + 1$となる。

対数関数のニュートンの差分商補間

2を底とする対数関数(Mapleでは$\log[2](x)$)の$F(9.2) = 2.219203$をニュートンの差分商補間を用いて求める．ニュートンの内挿公式は，

$$F(x) = F(x_0) + (x-x_0)f_1\lfloor x_0, x_1\rfloor + (x-x_0)(x-x_1)f_2\lfloor x_0, x_1, x_2\rfloor + \cdots + \prod_{i=0}^{n-1}(x-x_i)\, f_n\lfloor x_0, x_1, \cdots, x_n\rfloor$$

である．ここで$f_i\lfloor\ \rfloor$は次のような関数を意味していて，

$$\begin{aligned} f_1\lfloor x_0, x_1\rfloor &= \frac{y_1-y_0}{x_1-x_0} \\ f_2\lfloor x_0, x_1, x_2\rfloor &= \frac{f_1\lfloor x_1,x_2\rfloor - f_1\lfloor x_0,x_1\rfloor}{x_2-x_0} \\ &\vdots \\ f_n\lfloor x_0, x_1, \cdots, x_n\rfloor &= \frac{f_{n-1}\lfloor x_1,x_2\cdots,x_n\rfloor - f_{n-1}\lfloor x_0,x_1,\cdots,x_{n-1}\rfloor}{x_n-x_0} \end{aligned}$$

差分商と呼ばれる．$x_k = 8.0, 9.0, 10.0, 11.0$をそれぞれ選ぶと，差分商補間のそれぞれの項は以下の通りとなる．

k	x_k	$y_k = F_0(x_k)$	$f_1\lfloor x_k, x_{k+1}\rfloor$	$f_2\lfloor x_k, x_{k+1}, x_{k+2}\rfloor$	$f_3\lfloor x_k, x_{k+1}, x_{k+2}, x_{k+3}\rfloor$
0	8.0	2.079442			
			0.117783		
1	9.0	2.197225		$[XXX]$	
			0.105360		0.0003955000
2	10.0	2.302585		-0.0050250	
			0.095310		
3	11.0	2.397895			

それぞれの項は，例えば，

$$f_2\lfloor x_1, x_2, x_3\rfloor = \frac{0.095310 - 0.105360}{11.0 - 9.0} = -0.0050250$$

で求められる．ニュートンの差分商の一次多項式の値はx=9.2で

$F(x) = F_0(8.0) + (x-x_0)f_1\lfloor x_1, x_0\rfloor = 2.079442 + 0.117783(9.2-8.0) = 2.220782$

となる．

差分商補間の表中の開いている箇所[XXX]を埋めよ．

ニュートンの二次多項式

$$F(x) = F(x_0) + (x-x_0)f_1\lfloor x_0, x_1\rfloor + (x-x_0)(x-x_1)f_2\lfloor x_0, x_1, x_2\rfloor$$

の値を求めよ．

ニュートンの三次多項式の値を求めよ．

ただし，ここでは有効数字7桁程度はとるように．(E.クライツィグ著「数値解析」(培風館,2003), p.31, 例4改)

それぞれ定義して計算を行うと

In [6]:
```
x_0=8.0
x_2=10.0
f1_01=0.117783
f1_12=0.105360
f2_012=(f1_12-f1_01)/(x_2-x_0)
print(f2_012)
```

```
-0.006211500000000002
```

XXXは

$$f_2\lfloor x_0, x_1, x_2\rfloor = \frac{0.105360 - 0.117783}{10.0 - 8.0} = -0.0062115$$

ニュートンの二次多項式、三次多項式も同様に定義して

In [9]:
```
F_0=2.079442
x_1=9.0
x=9.2
F_a=F_0+(x-x_0)*(f1_01)+(x-x_0)*(x-x_1)*(f2_012)
print(F_a)
```

```
2.2192908399999998
```

二次多項式では

$$F(x) = F(x_0) + (x-x_0)f_1\lfloor x_0, x_1\rfloor + (x-x_0)(x-x_1)f_2\lfloor x_0, x_1, x_2\rfloor = 2.2192908$$

In [10]:
```
f3_0123=0.0003955
F_b=F_0+(x-x_0)*(f1_01)+(x-x_0)*(x-x_1)*(f2_012)+(x-x_0)*(x-x_1)*(x-x_2)*(f3_0123)
print(F_b)
```

```
2.2192149039999998
```

三次多項式は

$$F(x) = F(x_0) + (x-x_0)f_1\lfloor x_0, x_1\rfloor + (x-x_0)(x-x_1)f_2\lfloor x_0, x_1, x_2\rfloor + (x-x_0)(x-x_1)(x-x_2)f_3\lfloor x_0, x_1, x_2, x_3\rfloor = 2.2192149$$

となった。

二次多項式と三次多項式では５桁目あたりから違いが見られる。

数値積分(I)

次の関数

$$f(x) = \frac{4}{1+x^2}$$

を$x = 0..1$で数値積分する.

1. Nを2,4,8,...256ととり，N個の等間隔な区間にわけて中点法と台形則で求めよ．(15)
2. 小数点以下10桁まで求めた値3.141592654との差をdXとする．dXと分割数Nとを両対数プロット(loglogplot)して比較せよ(10) (2008年度期末試験)

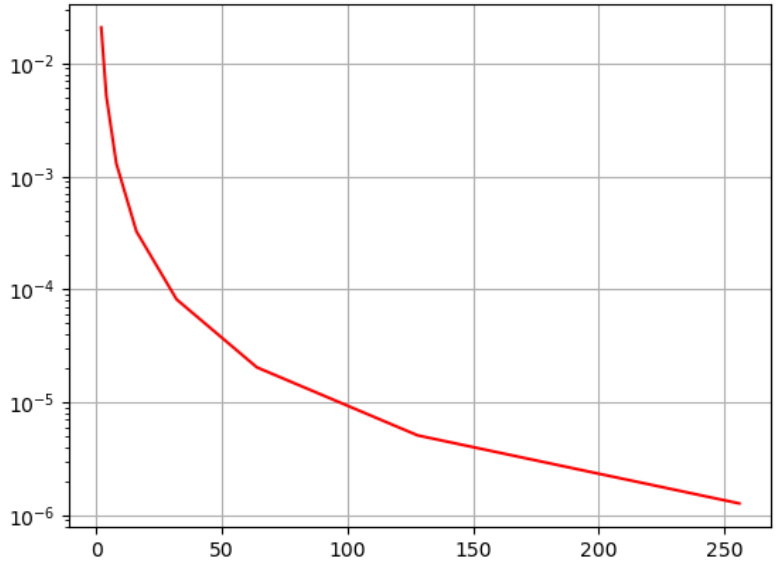
中点法

In [21]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np

def func(x):
    return 4.0/(1.0+x**2)

def mid(N):
    x0, xn = 0.0, 1.0


    h = (xn-x0)/N
    S = 0.0
    for i in range(0, N):
        xi = x0 + (i+0.5)*h
        dS = h * func(xi)
        S = S + dS
    return S

xm, ym = [], []
for i in range(1,9):
    xm.append(2**i)
    ym.append(abs(mid(2**i)-np.pi))

    print("N=",2**i)
    print("value:",mid (2**i)," error:", abs(mid(2**i)-np.pi))
```

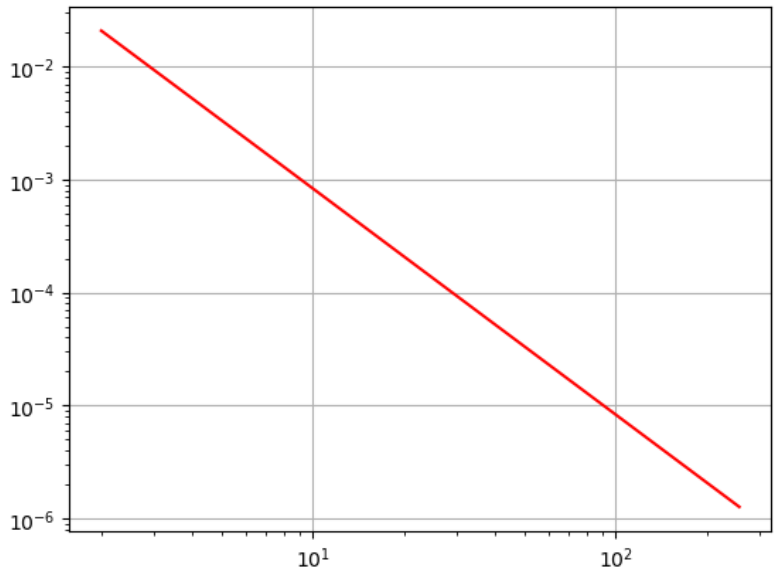


```
N= 2
value: 3.1623529411764704  error: 0.02076028758667725
N= 4
value: 3.1468005183939427  error: 0.005207864804149587
N= 8
value: 3.142894729591689  error: 0.0013020760018958022
N= 16
value: 3.14191817430856  error: 0.0003255207187669029
N= 32
value: 3.1416740337963374  error: 8.138020654424594e-05
N= 64
value: 3.1416129986418473  error: 2.0345052054171475e-05
N= 128
value: 3.1415977398528145  error: 5.086263021425452e-06
N= 256
value: 3.1415939251555467  error: 1.2715657535800062e-06
```

In [22]:
```
plt.plot(xm, ym, color = 'r')
plt.yscale('log')

plt.grid()
plt.show()
```



In [23]:
```
plt.plot(xm, ym, color = 'r')
plt.yscale('log')
plt.xscale('log')

plt.grid()
plt.show()
```



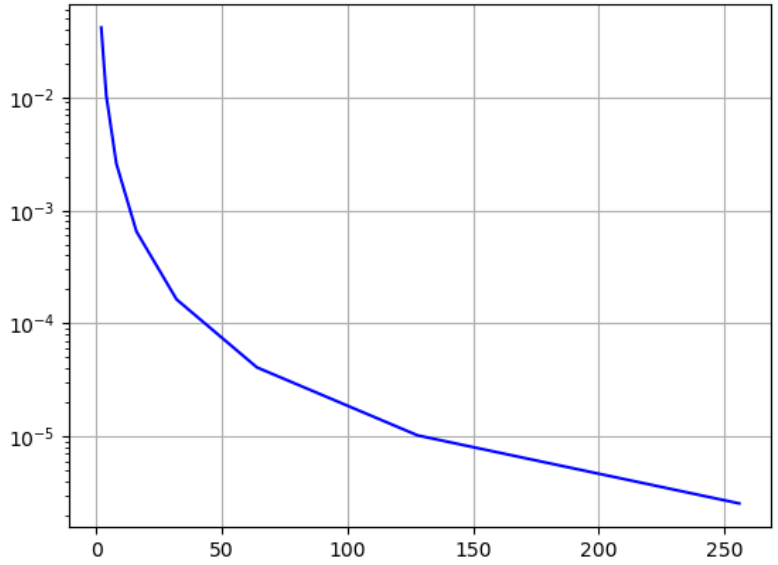
台形則

In [24]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np

def func(x):
    return 4.0/(1.0+x**2)

def tra(N):
    x0, xn = 0.0, 1.0

    h = (xn-x0)/N
    S = func(x0)/2.0
    for i in range(1, N):
        xi = x0 + i*h
        dS = func(xi)
        S = S + dS
    S = S + func(xn)/2.0
    return h*S

xt, yt = [], []
for i in range(1, 9):
    xt.append(2**i)
    yt.append(abs(tra(2**i)-np.pi))

    print("N=", 2**i)
    print("value:", tra(2**i), " error:", abs(tra(2**i)-np.pi))
```

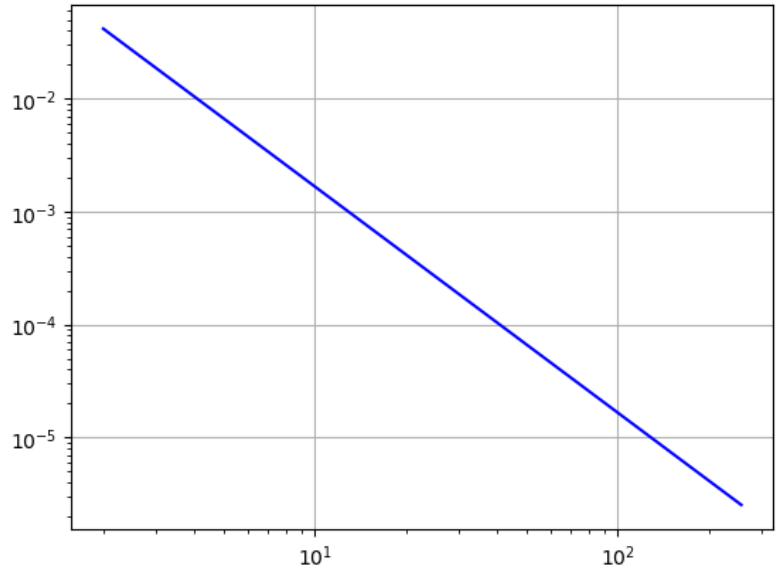


```
N= 2
value: 3.1  error: 0.04159265358979303
N= 4
value: 3.1311764705882354  error: 0.010416183001557666
N= 8
value: 3.138988494491089  error: 0.0026041590987042618
N= 16
value: 3.140941612041389  error: 0.0006510415484042298
N= 32
value: 3.141429893174975  error: 0.0001627604148182193 5
N= 64
value: 3.141551963485654  error: 4.0690104138985106e-05
N= 128
value: 3.141582481063752  error: 1.0172526041074548e-05
N= 256
value: 3.1415901104582815  error: 2.5431315116009046e-06
```

In [25]:
```
plt.plot(xt, yt, color = 'b')
plt.yscale('log')

plt.grid()
plt.show()
```

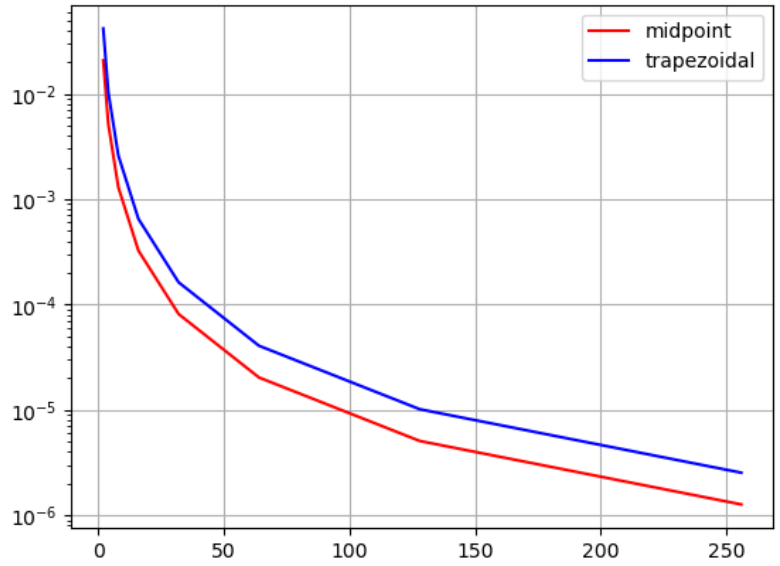


In [26]:
```
plt.plot(xt, yt, color = 'b')
plt.yscale('log')
plt.xscale('log')
plt.grid()
plt.show()
```

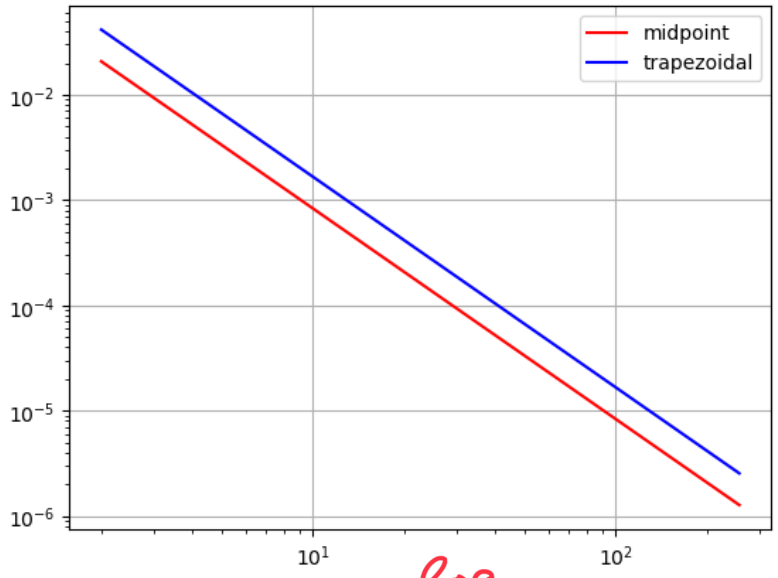


これらを比較すると

In [27]:
```
plt.plot(xm, ym, color = 'r',label="midpoint")
plt.plot(xt, yt, color = 'b',label="trapezoidal")
plt.yscale('log')
plt.legend()
plt.grid()
plt.show()
```



In [28]:
```
plt.plot(xm, ym, color = 'r',label="midpoint")
plt.plot(xt, yt, color = 'b',label="trapezoidal")
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



結果から中点法の方が台形則よりも誤差が小さいことがわかる。数値を見ても中点法の誤差は台形則の誤差の1/2程度で、中点法の精度が高いことがわかる。ただ指数的に誤差に違いが出るほどではなく、ほとんど並行な傾きをとっている。

台形則は曲線を台形で近似し、中点法は曲線を直線で近似することで誤差が生まれる。曲線が滑らかであれば直線で近似する中点法が有利であり、今回も誤差が台形則よりも小さくなったが、関数の変化が激しいところや不連続なところでは直線での近似が難しいため、台形則を用いた方が良い場合も考えられる。

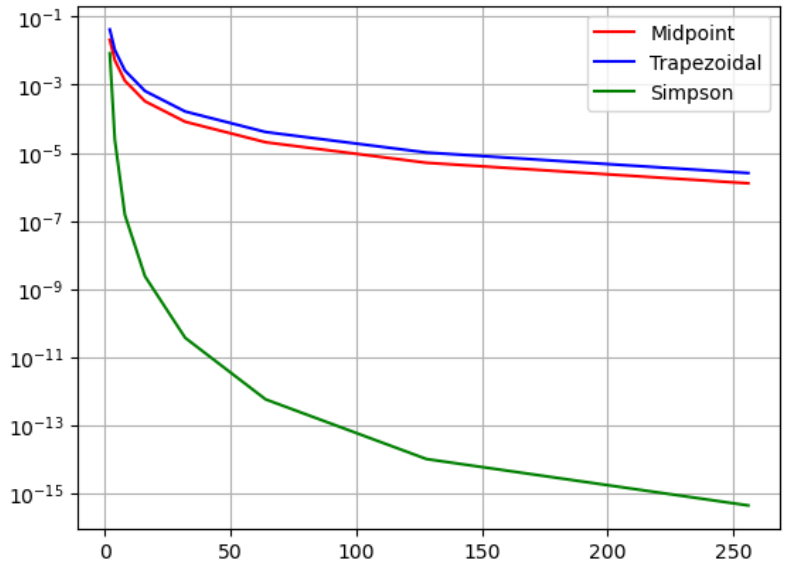
追加：シンプソン則との比較

In [35]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np

def func(x):
    return 4.0/(1.0+x**2)

def sim(N):
    x0, xn = 0.0, 1.0
    M = int(N/2)
    h = (xn-x0)/N
    Seven, Sodd = 0.0, 0.0
    for i in range(1, 2*M, 2):
```

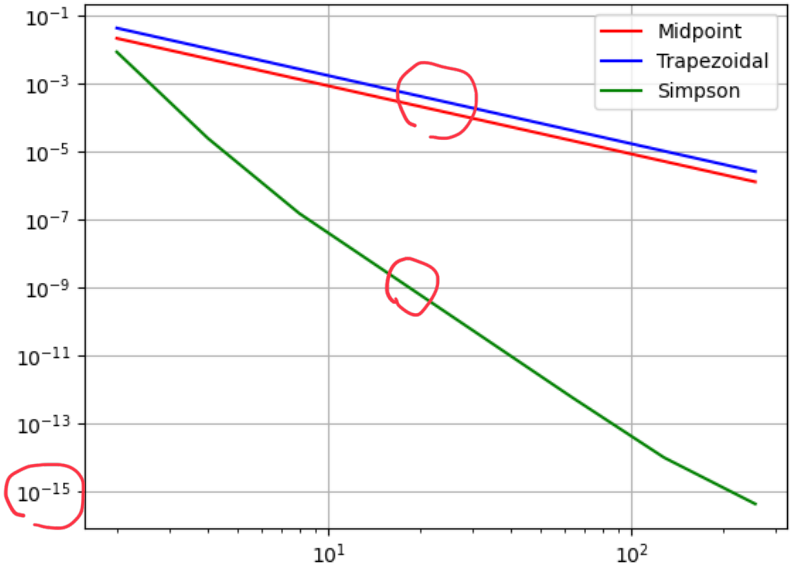
```
        xi = x0 + i*h
        Sodd += func(xi)
    for i in range(2, 2*M, 2):
        xi = x0 + i*h
        Seven += func(xi)

    return h*(func(x0)+4*Sodd+2*Seven+func(xn))/3

xs, ys = [], []
for i in range(1, 9):
    N = 2**i
    xs.append(N)
    ys.append(abs(sim(N)-np.pi))

    print("N=", N)
    print("value:", sim(N), " error:", abs(sim(N)-np.pi))
```

```
N= 2
value: 3.1333333333333333  error: 0.008259320256459812
N= 4
value: 3.14156862745098  error: 2.4026138813137976e-05
N= 8
value: 3.141592502458707  error: 1.5113108631226169e-07
N= 16
value: 3.1415926512248222  error: 2.3649708857931273e-09
N= 32
value: 3.1415926535528365  error: 3.695665995451236e-11
N= 64
value: 3.1415926535892162  error: 5.768718835952313e-13
N= 128
value: 3.141592653589783  error: 1.021405182655144e-14
N= 256
value: 3.1415926535897936  error: 4.440892098500626e-16
```

In [30]:
```
plt.plot(xm, ym, color = 'r',label="Midpoint")
plt.plot(xt, yt, color = 'b',label="Trapezoidal")
plt.plot(xs, ys, color = 'g',label="Simpson")
plt.yscale('log')
plt.legend()
plt.grid()
plt.show()
```

In [32]:
```
plt.plot(xm, ym, color = 'r',label="Midpoint")
plt.plot(xt, yt, color = 'b',label="Trapezoidal")
plt.plot(xs, ys, color = 'g',label="Simpson")
plt.yscale('log')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



結果からシンプソン則は台形則や中点法よりも誤差が小さく、かなり精度が高いことが窺える。両対数でプロットしているグラフを見ても、台形則と中点法の傾きがほとんど並行にあるのに対して、シンプソン則では傾きが違っていることがわかるほどである。シンプソン則は二次関数で近似をおこなっているため、中点法と同じく不連続なものや関数の値が大きく変化するものに対しては扱いづらいが、スムーズな関数であれば中点法よりも高い精度を得ることができると考えられる。