第三次实验

1 实验一

1.1 题目

实验1 随机变量X_1, X_2服从标准正态分布，相关系数为r，现采样到样本点(1,2)。试利用最大似然估计法估计r。

解 带参数的联合密度函数

$$f(x_1, x_2; r) = \frac{1}{2\pi\sqrt{\begin{vmatrix} 1 & r \\ r & 1 \end{vmatrix}}} e^{-\frac{1}{2}(x_1 \quad x_2)\begin{pmatrix} 1 & r \\ r & 1 \end{pmatrix}^{-1}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}}, \begin{pmatrix} 1 & r \\ r & 1 \end{pmatrix}^{-1} = \frac{1}{1-r^2}\begin{pmatrix} 1 & -r \\ -r & 1 \end{pmatrix}$$

$$f(x_1, x_2; r) = \frac{1}{2\pi\sqrt{1-r^2}} e^{-\frac{1}{2(1-r^2)}(x_1^2 - 2rx_1x_2 + x_2^2)}$$

在出现样本点(1,2)下的似然函数为:

$$f(r) = \frac{1}{2\pi\sqrt{1-r^2}} e^{-\frac{5-4r}{2(1-r^2)}}$$

通过调用数学优化库命令，最大化似然函数，可求出r。

具体代码为:

```
import numpy as np
from scipy.optimize import minimize

# 定义对数似然函数，增加 epsilon 以避免除零错误
def log_likelihood(r, x1, x2):
    epsilon = 1e-10 # 一个非常小的数，防止除零错误
    n = len(x1)
    term1 = -n * np.log(2 * np.pi * np.sqrt(1 - r**2 + epsilon))
    term2 = -1 / (2 * (1 - r**2 + epsilon)) * np.sum(x1**2 - 2 * r * x1 * x2 + x2**2)
    return -(term1 + term2)

# 样本点
x1, x2 = 1, 2

# 使用优化函数来最大化对数似然函数
result = minimize(log_likelihood, 0, args=(np.array([x1]), np.array([x2])), bounds=[(-1,
1)])
r_estimate = result.x[0]

r_estimate
```

1.2 实验结果

$$r = 0.6388969120173674$$

2 实验二

2.1 题目

- 实验2 给定测试函数（假设每一次评估是昂贵的）

$$X(t_1, t_2, ..., t_{10}) = \sum_{i=1}^{10} (t_i^2 - 10\cos(2\pi t_i) + 10)$$

其中$-20 \le t_1, t_2, ..., t_{10} \le 20$

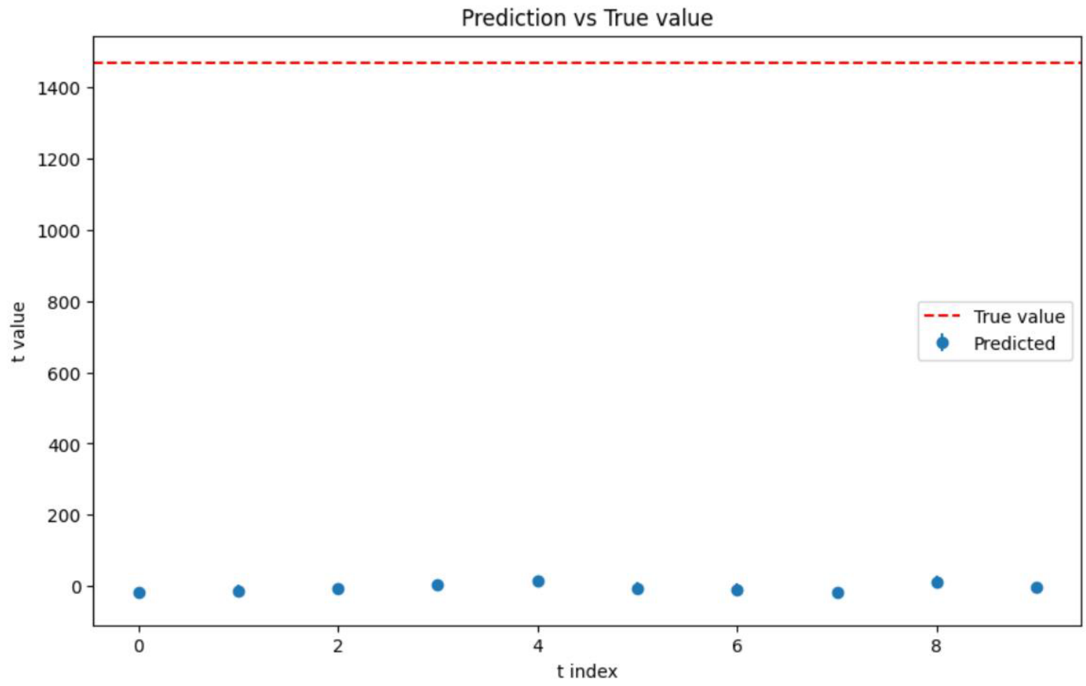
- **建模**：在定义域范围内随机产生200个$\vec{t}$，并计算昂贵函数值，用这200个数据建立高斯模型。
- **预测实验**：随机产生一个$\vec{t}_0$，用高斯模型进行预测均值与方差。
- **验证预测误差**：计算精确值$X(\vec{t}_0)$与预测均值之差的平方，并看看它与预测方差是否一致。

2.2 具体代码

```
import numpy as np from sklearn.gaussian_process
import GaussianProcessRegressor from sklearn.gaussian_process.kernels
import RBF, ConstantKernel as C
import matplotlib.pyplot as plt

# 定义响应函数
def response_function(t):
    return np.sum(t**2 - 10 * np.cos(2 * np.pi * t) + 10)

# 生成数据
np.random.seed(0)
T = np.random.uniform(-20, 20, (200, 10))
X = np.array([response_function(t) for t in T])

# 构建高斯过程模型
kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10)
gp.fit(T, X)

# 随机生成一个新的 t0
t0 = np.random.uniform(-20, 20, 10)
X_t0_true = response_function(t0)

# 用高斯过程模型进行预测
X_t0_pred, sigma = gp.predict(t0.reshape(1, -1), return_std=True)

# 计算误差
error = (X_t0_true - X_t0_pred)**2

# 打印结果
print("True value:", X_t0_true) 42
print("Predicted value:", X_t0_pred)
print("Prediction error:", error)

# 可视化
plt.figure(figsize=(10, 6))
plt.errorbar(np.arange(10), t0, yerr=sigma, fmt='o', label='Predicted')
plt.axhline(X_t0_true, color='r', linestyle='--', label='True value')
plt.xlabel('t index')
plt.ylabel('t value')
plt.title('Prediction vs True value')
plt.legend()
plt.show()
```

2.3 实验结果

```
True value: 1469.403757894978
Predicted value: [1210.04661358]
Prediction error: [67266.12830608]
```



第四次实验

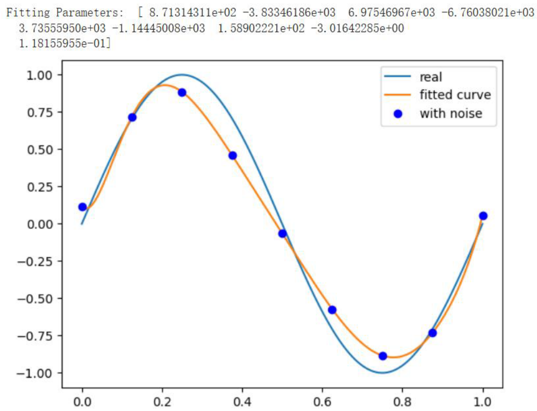
1 书本实验 1

1.1 具体代码

```
import numpy as np
import scipy as sp
import pylab as pl from scipy.optimize
import leastsq #引入最小二乘函数

#多项式次数
n=9

#目标函数
def real_func(x):
    #目标函数:sin(2*pi*x)
    return np.sin(2 * np.pi * x)

#定义多项式函数，用多项式去拟合数据:
def fit_func(p, x):
    f = np.poly1d(p)
    return f(x)

#定义残差函数，残差函数值为多项式拟合结果与真实值的差值:
def residuals_func(p,Y,x):
    ret= fit_func(p,x)-Y
    return ret

x=np.linspace(0,1,9) # 随机选择 9 个点作为 x
```

```python
x_points =np.linspace(0,1,1000) # 画图时需要的连续点
y0 = real_func(x) # 目标所数
y1 = [np.random.normal(0,0.1)+y for y in y0] # 在目标函数上添加符合正态分布 噪声后的函数
p_init = np.random.randn(n) # 随机初始化多项式参数

# 调用 scipy.optimize 中的 leastsq 函数，通过最小化误差的平方和来寻找最佳的 匹配函数
#func 是一个残差函数，x0 是计算的初始参数值，把残差函数中除了初始化以外的参 数打包到 args 中

plsq= leastsq(func=residuals_func, x0=p_init, args=(y1, x))
print('Fitting Parameters: ',plsq[0])#输出拟合参数
pl.plot(x_points, real_func(x_points), label='real')
pl.plot(x_points, fit_func(plsq[0], x_points), label='fitted curve')
pl.plot(x,y1,'bo',label='with noise')
pl.legend()
pl.show()
```

1.2 实验结果



1.3 书本实验

1.4 具体代码

```python
#y[i] 样本点对应的输出
x=[(1,0.,3),(1,1.,3),(1,2.,3),(1,3.,2),(1,4.,4)]
y=[95.364,97.217205,75.195834,60.105519,49.342380]

#迭代阀值，当两次迭代损失函数之差小于该阀值时停止迭代
epsilon =0.0001

#学习率
alpha =0.01

diff =[0,0]
max_itor = 1000
error1 = 0
error0 = 0
cnt = 0
m = len(x)

#初始化参数
theta0 =0
theta1 =0
theta2 =0
while True:
```

```python
    cnt += 1
    # 参数迭代计算
    for i in range(m):
        #拟合数为 y= theta0 * x[0]+ theta1 * x[1] +theta2 * x[2]
        #计算残差，即拟合函数值-真实值
        diff[0]=(theta0* x[i][0] + theta1 * x[i][1] + theta2 * x[i][2])-y[i]
        # 梯度 = diff[0]* x[i][j]。根据步长*梯度更新参数
        theta0 -= alpha * diff[0]* x[i][0]
        theta1 -= alpha * diff[0]* x[i][1]
        theta2 -= alpha * diff[0]* x[i][2]

    # 计算损失函数
    error1=0
    for lp in range(len(x)):
        error1 +=(y[lp]-(theta0* x[lp][0]+ theta1 * x[lp][1] + theta2 * x[lp][2]))**2/2

    #若当两次迭代损失函数之差小于该阀值时停止迭代，跳出循环
    if abs(error1-error0)< epsilon:
        break
    else:
        error0 = error1

print('theta0 : %f, theta1 : %f, theta2 : %f, error1 : %f'% (theta0,theta1,theta2,error1))
print('Done: theta0 : %f, theta1 : %f, theta2 : %f'% (theta0, theta1,theta2))
print('迭代次数:%d'%cnt)
```

1.5 实验结果

```
Done: theta0 : 97.710840, theta1 : -13.224430, theta2 : 1.344721
迭代次数:2595
theta0 : 97.711549, theta1 : -13.224421, theta2 : 1.344496, error1 : 58.733276
Done: theta0 : 97.711549, theta1 : -13.224421, theta2 : 1.344496
迭代次数:2596
theta0 : 97.712257, theta1 : -13.224413, theta2 : 1.344271, error1 : 58.733172
Done: theta0 : 97.712257, theta1 : -13.224413, theta2 : 1.344271
迭代次数:2597
theta0 : 97.712963, theta1 : -13.224405, theta2 : 1.344047, error1 : 58.733069
Done: theta0 : 97.712963, theta1 : -13.224405, theta2 : 1.344047
迭代次数:2598
theta0 : 97.713668, theta1 : -13.224396, theta2 : 1.343823, error1 : 58.732967
Done: theta0 : 97.713668, theta1 : -13.224396, theta2 : 1.343823
迭代次数:2599
theta0 : 97.714371, theta1 : -13.224388, theta2 : 1.343600, error1 : 58.732865
Done: theta0 : 97.714371, theta1 : -13.224388, theta2 : 1.343600
迭代次数:2600
theta0 : 97.715073, theta1 : -13.224380, theta2 : 1.343377, error1 : 58.732763
Done: theta0 : 97.715073, theta1 : -13.224380, theta2 : 1.343377
迭代次数:2601
theta0 : 97.715773, theta1 : -13.224372, theta2 : 1.343155, error1 : 58.732661
Done: theta0 : 97.715773, theta1 : -13.224372, theta2 : 1.343155
迭代次数:2602
theta0 : 97.716472, theta1 : -13.224363, theta2 : 1.342933, error1 : 58.732560
Done: theta0 : 97.716472, theta1 : -13.224363, theta2 : 1.342933
迭代次数:2603
theta0 : 97.717169, theta1 : -13.224355, theta2 : 1.342712, error1 : 58.732459
Done: theta0 : 97.717169, theta1 : -13.224355, theta2 : 1.342712
迭代次数:2604
theta0 : 97.717864, theta1 : -13.224347, theta2 : 1.342491, error1 : 58.732358
Done: theta0 : 97.717864, theta1 : -13.224347, theta2 : 1.342491
迭代次数:2605
theta0 : 97.718558, theta1 : -13.224339, theta2 : 1.342271, error1 : 58.732258
Done: theta0 : 97.718558, theta1 : -13.224339, theta2 : 1.342271
迭代次数:2606
theta0 : 97.719251, theta1 : -13.224330, theta2 : 1.342051, error1 : 58.732157
Done: theta0 : 97.719251, theta1 : -13.224330, theta2 : 1.342051
迭代次数:2607
```

2 用凸优化以及数学优化工具库方法求解问题

2.1 优化问题 1

$$\min f(\vec{x}) = 2x_1^{\ 2} + x_2^{\ 2} + x_3^{\ 2}$$

$$s.t.\begin{cases} x_1^{\ 2} + x_2^{\ 2} - 4 \le 0 \\ 5x_1 - 4x_2 - 8 = 0 \\ x_1, x_2, x_3 \ge 0 \end{cases}$$

2.1.1 数学优化工具库求解

```
import numpy as np from scipy.optimize
import minimize

# 定义目标函数
def objective(x):
    return 2*x[0]**2 + x[1]**2 + x[2]**2

# 定义约束条件
cons = (
    {'type': 'ineq', 'fun': lambda, x: -(x[0]**2 + x[1]**2 - 4)},
    {'type': 'eq', 'fun': lambda x: 5*x[0] - 4*x[1] - 8},
    {'type': 'ineq', 'fun': lambda x: x[0]},
    {'type': 'ineq', 'fun': lambda x: x[1]},
    {'type': 'ineq', 'fun': lambda x: x[2]} )

# 初始猜测
x0 = np.array([1, 1, 1])
# 求解
result = minimize(objective, x0, constraints=cons)
print(result)
```

2.1.2 实验结果分析

由实验结果可知，优化成功终止，说明找到了解的最优解，最优目标函数值为 5.12， 最优解为 $[1.6, 6.661e^{-15}, 1.554e^{-15}]$。由于$6.661e^{-15}$ 和 $1.554e^{-15}$非常接近于零，可以认为 x_1 和 x_2 都是零；优化过程共进行了 2 次迭代；目标函数在最优解处的梯度为 $[6.4, 0, 0]$；目标函数被调用了 8 次；梯度函数被调用了 2 次。

```
message: Optimization terminated successfully
success: True
 status: 0
    fun: 5.120000000000035
      x: [ 1.600e+00  6.661e-15  1.554e-15]
    nit: 2
    jac: [ 6.400e+00  0.000e+00  0.000e+00]
   nfev: 8
   njev: 2
```

2.1.3 用凸优化工具库 cvxpy 求解

```
 #用凸优化工具库求解
import cvxpy as cp

# 定义变量
x1 = cp.Variable()
x2 = cp.Variable()
x3 = cp.Variable()

# 定义目标函数
objective = cp.Minimize(2*x1**2 + x2**2 + x3**2)
```

```python
# 定义约束条件
constraints = [
    x1**2 + x2**2 <= 4,
    5*x1 - 4*x2 == 8,
    x1 >= 0, x2 >= 0,
    x3 >= 0
]

# 定义问题
prob = cp.Problem(objective, constraints)

# 求解
result = prob.solve()
print(f"Optimal value: {result}")
print(f"x1: {x1.value}, x2: {x2.value}, x3: {x3.value}")
```

2.1.4 实验结果及分析

从输出结果来看，最优值为 5.119999975530055，表示在给定的约束条件下， 目标函数 $2x_{12}+x_{22}+x_{32}$ 的最小值。

变量值：$x_1 \approx 1.6$，满足约束条件 $5x_1-4x_2=8$ 和 $x_{12}+x_{22} \leq 4$。$x_2 \approx 0$ 由于 x_2 非负且接近于 0，满足约束条件。$x_3 \approx 0$ 由于 x_3 非负且接近于 0， 满足约束条件。并且输出结果与使用数学优化工具库求出的结果一致。

2.2 优化问题 2

$$\min f(\vec{x}) = 2x_1^2 - x_2^2 - x_3^2$$

$$s.t. \begin{cases} x_1^2 + x_2^2 - 4 \leq 0 \\ 5x_1 - 4x_2 - 8 = 0 \\ x_1, x_2, x_3 \geq 0 \end{cases}$$

2.2.1 用数学优化工具库求解

```python
import numpy as np from scipy.optimize
import minimize

# 定义目标函数
def objective(x):
    return x[0]**2 - x[1]**2 - x[2]**2

# 定义约束条件
cons = (
    {'type': 'ineq', 'fun': lambda, x: -(x[0]**2 + x[1]**2 - 4)},
    {'type': 'eq', 'fun': lambda, x: 5*x[0] - 4*x[1] - 8},
    {'type': 'ineq', 'fun': lambda, x: x[0]},
    {'type': 'ineq', 'fun': lambda, x: x[1]},
    {'type': 'ineq', 'fun': lambda, x: x[2]} )

# 初始猜测
x0 = np.array([1, 1, 1])

# 求解
result = minimize(objective, x0, constraints=cons)
print(result)
```

2.2.2 实验结果

```
message: Inequality constraints incompatible
success: False
 status: 4
    fun: -6.742189191122648e+79
      x: [ 9.461e+14  1.183e+15  8.211e+39]
    nit: 16
    jac: [ 0.000e+00  0.000e+00 -1.642e+40]
   nfev: 60
   njev: 15
```

从输出结果来看，优化过程没有成功，主要原因是约束条件不兼容。具体分析如下：

- `message: Inequality constraints incompatible`这个信息表明不等式约束条件之间存在冲突，导致优化问题无法找到可行解。
- `success: False` 这个标志表明优化过程没有成功。
- `status: 4` 状态码 4 表示优化过程由于约束条件不兼容而失败。
- `fun: -6.742189191122648e+79`目标函数的值非常大，表明优化过程可能在尝试找到解的过程中遇到了数值问题。
- `x: [9.461e+14, 1.183e+15, 8.211e+39]` 变量的值非常大，进一步表明优化过程可能在尝试找到解的过程中遇到了数值问题。
- `nit: 16` 优化过程进行了 16 次迭代。
- `jac: [0.000e+00, 0.000e+00, -1.642e+40]` 目标函数在解处的梯度值。
- `nfev: 60` 目标函数被评估了 60 次。
- `njev: 15` 目标函数的梯度被评估了 15 次。

2.2.3 用凸优化 cvxpy 工具库求解

```python
#用凸优化工具求解
import cvxpy as cp

# 定义变量
x1 = cp.Variable()
x2 = cp.Variable()
x3 = cp.Variable()

# 定义目标函数
objective = cp.Minimize(2 * cp.square(x1) - cp.square(x2) - cp.square(x3))

# 定义约束条件
constraints = [
    cp.square(x1) + cp.square(x2) <= 4,
    5 * x1 - 4 * x2 == 8,
    x1 >= 0,
    x2 >= 0,
    x3 >= 0
]

# 定义问题
prob = cp.Problem(objective, constraints)

# 求解
```

```python
result = prob.solve()
print(f"Optimal value: {result}")
print(f"x1: {x1.value}, x2: {x2.value}, x3: {x3.value}")
```

实验结果

出现了报错，但这很正常，因为这道题中的函数不是凸函数，不满足凸优化条件。

```
---------------------------------------------------------------------------
DCPError                                  Traceback (most recent call last)
<ipython-input-4-b84d60b96038> in <cell line: 25>()
     23
     24 # 求解
---> 25 result = prob.solve()
     26 print(f"Optimal value: {result}")
     27 print(f"x1: {x1.value}, x2: {x2.value}, x3: {x3.value}")

5 frames
/usr/local/lib/python3.10/dist-packages/cvxpy/reductions/solvers/solving_chain.py in _reductions_for_problem_class(problem, candidates, gp, solver_opts)
    113                 append += ("\nHowever, the problem does follow DQCP rules. "
    114                            "Consider calling solve() with `qcp=True`.")
--> 115             raise DCPError(
    116                 "Problem does not follow DCP rules. Specifically:\n" + append)
    117         elif gp and not problem.is_dgp():

DCPError: Problem does not follow DCP rules. Specifically:
The objective is not DCP. Its following subexpressions are not:
2.0 @ power(var135, 2.0) + -power(var136, 2.0) + -power(var137, 2.0)
```

2.3 优化问题 3

$$\min f(\vec{x}) = \|\vec{x}\|_1 = |x_1| + |x_2| + |x_3|$$

$$s.t. \begin{cases} {x_1}^2 + {x_2}^2 + {x_3}^2 \le 2 \\ x_1 + x_2 + x_3 = 1 \\ -1 \le x_1, x_2, x_3 \le 1 \end{cases}$$

2.3.1 用数学优化工具库求解

```python
from scipy.optimize import minimize
import numpy as np

# 定义目标函数
def objective(x):
    return np.sum(np.abs(x))

# 定义约束条件
def constraint1(x):
    return x[0] + x[1] + x[2] - 1

def constraint2(x):
    return 2 - (x[0]**2 + x[1]**2 + x[2]**2)

# 定义变量的界限
bounds = [(-1, 1), (-1, 1), (-1, 1)]

# 初始猜测
x0 = [0, 0, 0]

# 定义约束
constraints = [
    {'type': 'eq', 'fun': constraint1},
    {'type': 'ineq', 'fun': constraint2}]
```

```python
# 求解问题
solution = minimize(objective, x0, method='SLSQP', bounds=bounds, constraints=constraints)

# 输出结果
print("Optimal value:", solution.fun)
print("Optimal x1:", solution.x[0])
print("Optimal x2:", solution.x[1])
print("Optimal x3:", solution.x[2])
```

2.3.2 实验结果

通过使用 `scipy.optimize` 库中的 `minimize` 函数，我们对一个带有约 束条件的优化问题进行了求解。 最优目标值接近于 1，这表明在满足约束条件的情况下，绝对值之和的最小 值为 1。 最优解中的每个变量值都接近于 1/3，这符合约束条件 $x_1 + x_2 + x_3 = 1$。

```
Optimal value: 1.0000000000000004
Optimal x1: 0.3333333333333335
Optimal x2: 0.3333333333333334
Optimal x3: 0.3333333333333334
```

2.3.3 用凸优化 cvxpy`工具库求解

```python
import cvxpy as cp

# 定义变量
x1 = cp.Variable()
x2 = cp.Variable()
x3 = cp.Variable()

# 定义目标函数
objective = cp.Minimize(cp.norm1(cp.hstack([x1, x2, x3])))

# 定义约束条件
constraints = [
    x1**2 + x2**2 + x3**2 <= 2,
    x1 + x2 + x3 == 1,
    52 -1 <= x1,
    x1 <= 1,
    -1 <= x2,
    x2 <= 1,
    -1 <= x3,
    x3 <= 1
]

# 定义问题
problem = cp.Problem(objective, constraints)

# 求解问题
problem.solve()

# 输出结果
print("Optimal value:", problem.value)
print("Optimal x1:", x1.value)
print("Optimal x2:", x2.value)
print("Optimal x3:", x3.value)
```

2.3.4 实验结果及分析

通过凸优化工具库 CVXPY，我们成功地求解了这个优化问题。结果表明，在满足所有约束条件的情况下，x_1, x_2 和 x_3 的最优值使得目标函数（L_1 范 数）最小化，并且这些值都接近于 1/3。

这与用数学优化工具库求解结果一致。

```
Optimal value: 0.9999999999999937
Optimal x1: 0.3333333333174739
Optimal x2: 0.3333333333108717
Optimal x3: 0.33333333335143267
```

2.4 优化问题

2.4.1 用数学优化工具库求解

```python
from scipy.optimize import minimize
import numpy as np

# 定义目标函数
def objective(x):
    return np.max(np.abs(x))

# 定义约束条件
def constraint1(x):
    return 2 - np.sum(x**2)

def constraint2(x):
    return np.sum(x) - 1

# 定义边界
bounds = [(-1, 1), (-1, 1), (-1, 1)]

# 定义初始猜测
x0 = [0, 0, 0]

# 定义约束
constraints = [
    {'type': 'ineq', 'fun': constraint1},
    {'type': 'eq', 'fun': constraint2}
]
# 求解问题
solution = minimize(objective, x0, method='SLSQP', bounds=bounds, constraints=constraints)
# 输出结果
print("Optimal value:", solution.fun)
print("Optimal x:", solution.x)
```

2.4.2 实验结果

通过 `scipy.optimize.minimize` 求解的结果显示，最优解 x 为 $[0.33333333, 0.33333333, 0.33333333]$，此时目标函数的值 为 0.3333333333333335，并且满足所有的约束条件。这个结果是合理的，因为 在满足约束条件的情况下，目标函数的最小值确实是 0.3333333333333335。

2.4.3 用凸优化 cvxpy 工具库求解

```
import cvxpy as cp

# 定义变量
x1 = cp.Variable()
x2 = cp.Variable()
x3 = cp.Variable()

# 定义目标函数
objective = cp.Minimize(cp.norm_inf(cp.hstack([x1, x2, x3])))

# 定义约束条件
constraints = [
    x1**2 + x2**2 + x3**2 <= 2,
    x1 + x2 + x3 == 1,
    x1 >= -1,
    x1 <= 1,
    x2 >= -1,
    x2 <= 1,
    x3 >= -1,
    x3 <= 1
]

# 定义问题
prob = cp.Problem(objective, constraints)

# 求解
result = prob.solve()
print(f"Optimal value: {result}")
print(f"x1: {x1.value}, x2: {x2.value}, x3: {x3.value}")
```

通过凸优化工具库 `CVXPY` 求解该问题，得到了最优解 $x_1 = x_2 = x_3 = 0.33333333333333309$，并且最优值为 0.33333333333333309。这个结果满足所有给定的约束条件，并且使目标函数（无穷范数）最小化。与用数学优化工具库求解的结果一致。