地球科学大数据 参考代码

一元线性回归示例

```
# 工作年限与收入之间的散点图
# 导入第三方模块
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# 导入数据集
income = pd.read_csv('Salary_Data.csv')
# 绘制散点图
sns.lmplot(x = 'YearsExperience', y = 'Salary', data = income, ci = None)
# 显示图形
plt.show()


# 简单线性回归模型的参数求解
# 样本量
n = income.shape[0]
# 计算自变量、因变量、自变量平方、自变量与因变量乘积的和
sum_x = income.YearsExperience.sum()
sum_y = income.Salary.sum()
sum_x2 = income.YearsExperience.pow(2).sum()
xy = income.YearsExperience * income.Salary
sum_xy = xy.sum()
# 根据公式计算回归模型的参数
b = (sum_xy-sum_x*sum_y/n)/(sum_x2-sum_x**2/n)
a = income.Salary.mean()-b*income.YearsExperience.mean()
# 打印出计算结果
print('回归参数a的值：',a)
print('回归参数b的值：',b)
```

KMeans示例

```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# 读取iris数据集
iris = pd.read_csv('iris.csv')

# 查看数据集的前几行
iris.head()

# 提取出用于建模的数据集X
X = iris.drop(labels = 'Species', axis = 1)

# 构建Kmeans模型
kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
```

```
# 聚类结果标签
X['cluster'] = kmeans.labels_

# 各类频数统计
X.cluster.value_counts()

# 三个簇的簇中心
centers = kmeans.cluster_centers_

# 绘制聚类效果的散点图
sns.lmplot(x = 'Petal_Length', y = 'Petal_Width', hue = 'cluster', markers = ['^','s','o','x'],
           data = X, fit_reg = False, scatter_kws = {'alpha':0.8}, legend_out = False)
plt.scatter(centers[:,2], centers[:,3], marker = '*', color = 'black', s = 130)
plt.xlabel('花瓣长度')
plt.ylabel('花瓣宽度')

# 图形显示
plt.show()
```

SVM示例

```
# 导入第三方模块
from sklearn import svm
import pandas as pd
from sklearn import model_selection
from sklearn import metrics

# 读取外部数据
letters = pd.read_csv('letterdata.csv')

# 数据前5行
letters.head()

# 将数据拆分为训练集和测试集
predictors = letters.columns[1:]
X_train,X_test,y_train,y_test = model_selection.train_test_split(letters[predictors], letters.letter,
                                                                 test_size = 0.25, random_state = 1234)
# 使用网格搜索法，选择线性可分SVM“类”中的最佳C值
C=[0.05,0.1,0.5,1,2,5]
parameters = {'C':C}
grid_linear_svc = model_selection.GridSearchCV(estimator = svm.LinearSVC(),param_grid =parameters,scoring='accuracy',cv=5,verbose =1)

# 模型在训练数据集上的拟合
grid_linear_svc.fit(X_train,y_train)

# 返回交叉验证后的最佳参数值
grid_linear_svc.best_params_, grid_linear_svc.best_score_
print(grid_linear_svc.best_params_, grid_linear_svc.best_score_)

# 模型在测试集上的预测
```

```
pred_linear_svc = grid_linear_svc.predict(X_test)

# 模型的预测准确率
report = metrics.classification_report(y_test, pred_linear_svc)
print(report)
```

决策树示例

```
# 导入第三方模块
import pandas as pd
# 读入数据
Titanic = pd.read_csv('Titanic.csv')
Titanic.head()

# 删除无意义的变量，并检查剩余自字是否含有缺失值
Titanic.drop(['PassengerId','Name','Ticket','Cabin'], axis = 1, inplace = True)
Titanic.isnull().sum(axis = 0)

# 对Sex分组，用各组乘客的平均年龄填充各组中的缺失年龄
fillna_Titanic = []
for i in Titanic.Sex.unique():
    update = Titanic.loc[Titanic.Sex == i,].fillna(value = {'Age': Titanic.Age[Titanic.Sex == i].mean()})
    fillna_Titanic.append(update)
Titanic = pd.concat(fillna_Titanic)
# 使用Embarked变量的众数填充缺失值
Titanic.fillna(value = {'Embarked':Titanic.Embarked.mode()[0]}, inplace=True)
Titanic.head()

# 将数值型的Pclass转换为类别型，否则无法对其哑变量处理
Titanic.Pclass = Titanic.Pclass.astype('category')
# 哑变量处理
dummy = pd.get_dummies(Titanic[['Sex','Embarked','Pclass']])
# 水平合并Titanic数据集和哑变量的数据集
Titanic = pd.concat([Titanic,dummy], axis = 1)
# 删除原始的Sex、Embarked和Pclass变量
Titanic.drop(['Sex','Embarked','Pclass'], inplace=True, axis = 1)
Titanic.head()



# 导入第三方包
from sklearn import model_selection
# 取出所有自变量名称
predictors = Titanic.columns[1:]
# 将数据集拆分为训练集和测试集，且测试集的比例为25%
X_train, X_test, y_train, y_test = model_selection.train_test_split(Titanic[predictors], Titanic.Survived,
  test_size = 0.25, random_state = 1234)

# 导入第三方模块
from sklearn.model_selection import GridSearchCV
from sklearn import tree
# 预设各参数的不同选项值
max_depth = [2,3,4,5,6]
```

```
min_samples_split = [2,4,6,8]
min_samples_leaf = [2,4,8,10,12]
# 将各参数值以字典形式组织起来
parameters = {'max_depth':max_depth, 'min_samples_split':min_samples_split,
'min_samples_leaf':min_samples_leaf}
# 网格搜索法，测试不同的参数值
grid_dtcateg = GridSearchCV(estimator = tree.DecisionTreeClassifier(),
param_grid = parameters, cv=10)
# 模型拟合
grid_dtcateg.fit(X_train, y_train)
# 返回最佳组合的参数值
grid_dtcateg.best_params_

# 导入第三方模块
from sklearn import metrics
# 构建分类决策树
CART_Class = tree.DecisionTreeClassifier(max_depth=3, min_samples_leaf = 4,
min_samples_split=2)
# 模型拟合
decision_tree = CART_Class.fit(X_train, y_train)
# 模型在测试集上的预测
pred = CART_Class.predict(X_test)
# 模型的准确率
print('模型在测试集的预测准确率：\n',metrics.accuracy_score(y_test, pred))


# 导入第三方包
import matplotlib.pyplot as plt
y_score = CART_Class.predict_proba(X_test)[:,1]
fpr,tpr,threshold = metrics.roc_curve(y_test, y_score)
# 计算AUC的值
roc_auc = metrics.auc(fpr,tpr)

# 绘制面积图
plt.stackplot(fpr, tpr, color='steelblue', alpha = 0.5, edgecolor = 'black')
# 添加边际线
plt.plot(fpr, tpr, color='black', lw = 1)
# 添加对角线
plt.plot([0,1],[0,1], color = 'red', linestyle = '--')
# 添加文本信息
plt.text(0.5,0.3,'ROC curve (area = %0.2f)' % roc_auc)
# 添加x轴与y轴标签
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
# 显示图形
plt.show()
```

LSTM示例

```
import time
import pandas as pd
from keras.datasets.boston_housing import load_data

# x_train,y_train,x_test,y_test,window=load_data('data.csv',50,True)
x_train = pd.read_csv('train.csv', header=None)
x_test = pd.read_csv('test.csv', header=None)
val_x = pd.read_csv('val.csv', header=None)
```

```
x_train = x_train.values
val_x = val_x.values
x_test = x_test.values
x_train, y_train = x_train[:-24, :], x_train[24:, ]
x_test, y_test = x_test[:-24, :], x_test[24:, ]
x_validation, y_validation = val_x[:-24, :], val_x[24:, ]

x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
x_validation =
x_validation.reshape(x_validation.shape[0],x_validation.shape[1], 1)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)

# 构建LSTM神经网络模型进行预测
from keras.models import Sequential
from keras.layers import Bidirectional,LSTM
from keras.layers.core import Dense, Activation,Dropout

model = Sequential() # layer[1,50,50,50,50,1]
model.add(LSTM(input_shape=(None,1), units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(input_shape=(None,50), units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(input_shape=(None,50), units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.add(Activation("linear"))
# model.add(Activation("sigmoid"))
start=time.time()
model.compile(loss='mse', optimizer='rmsprop')
print("compilation Time:",time.time()-start)

len(model.layers)
model.fit(x_train,y_train,batch_size=10,epochs=1,validation_data=
(x_validation, y_validation), verbose=2)
score2=model.evaluate(x_test,y_test)
print(score2)
```

偏微分方程

在以下内容中，我们将应用Brunton et al.，2016的“通过非线性动力系统的稀疏识别从数据中发现控制方程”的方法(https://doi.org/10.1073/pnas.1517384113)以便根据数据估计未知的偏微分方程。该方法利用了一些具有未知控制方程的物理系统的空间和时间密集观测。这种情况在许多涉及流体力学（如海洋学、冰川学）或某种类型的扩散（如热传导）的科学领域很常见。
让我们首先导入一些Python包。在这个练习中，我们只需要使用 `numpy`、`scipy`和`matplotlib`。

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors, cm
from scipy.optimize import minimize
from sklearn.linear_model import Lasso, LassoCV
import sys
import os
```

加载并可视化数据

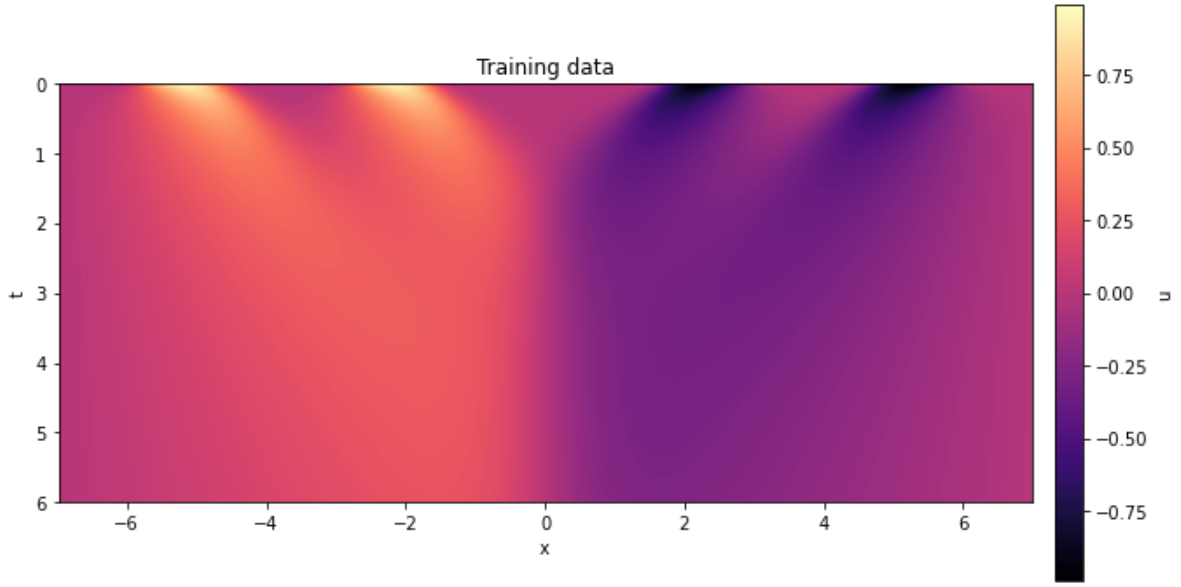
在这个练习中，我们观察到了状态变量$u(x,t)$在某个时间向量t上的1D系统。因此，控制这个系统的偏微分方程（PDE）具有某种形式：

$$u_t = \mathcal{F}(u) + \epsilon$$

其中$u_t = \frac{\partial u}{\partial t}$是$u$的时间导数，$\mathcal{F}()$表示$u$的一些非线性函数及其空间导数。（例如，$\frac{\partial u}{\partial x}$，$\frac{\partial^2 u}{\partial x^2}$），并且$\epsilon$是加性噪声。**我们的任务是对u的观察中学习$\mathcal{F}$的形式。**一旦我们有了$\mathcal{F}$的估计，我们将在验证数据集上测试我们的方程。

让我们首先加载训练数据集。该数据集包含我们的系统状态u和相应的时间向量t的观测值。

```
# 加载u
u = np.loadtxt('u_train.txt')
# 加载t
t = np.loadtxt('t_train.txt')
# 加载空间坐标x
x = np.loadtxt('coordinates.txt')

# 可视化数据
fig, ax = plt.subplots(figsize=(10, 6))
im = ax.imshow(u, extent=[x[0], x[-1], t[-1], t[0]], cmap='magma')
plt.colorbar(im, ax=ax, pad=0.02, shrink=0.8, label='u')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_title('Training data')
plt.tight_layout()
```



我们可以看到u是一个二维数组，时间沿纵轴，空间沿横轴。在时间$t = 0$时，u字段具有其最高值，在域的左侧有两个正峰值（$x < 0$），在域右侧有两个负峰值（$x > 0$）。随着时间的推移，这些峰值向中心汇聚，u变得更加扩散（分散），值越低。这些定性观察应该已经给了你一个关于PDE应该是什么形式的线索。

另一种可视化数据的方法是在不同的时间步长绘制u的属性。

构造从数据中学习偏微分方程的线性系统

SINDy方法（非线性动力学的备用识别）将PDE表示为线性系统$\mathbf{Az} = \mathbf{b}$，其中：

- $\mathbf{b}$ = u_t: u的时间导数向量
- $\mathbf{A}$ = [u, u_x, u_{xx}, ...]: u矩阵及其空间导数的非线性组合
- $\mathbf{z}$: 乘以$\mathbf{A}$列的系数向量

例如，假设PDE具有以下形式：$u_t = 1.25 * u_{xx} + 0.5 * u$。我将使用以下内容创建$\mathbf{A}$和$\mathbf{b}$：

```
# 计算时间导数
u_t = np.gradient(u, t, axis=0, edge_order=2)
# 计算空间导数
u_x = np.gradient(u, x, axis=1, edge_order=2)
u_xx = np.gradient(u_x, x, axis=1, edge_order=2)

# 构造A
A = np.column_stack((u_xx.ravel(), u.ravel()))

# 构造b
b = u_t.ravel()
```

在这种情况下，向量$\mathbf{z}$的值为$\mathbf{z} = [1.25,\ 0.5]$。

在一般情况下，**我们不知道在矩阵A**中放入什么函数。SINDy方法通过首先用u的*多个*不同函数及其空间导数（尽可能多地适合数据）构造$\mathbf{A}$来解决这个问题。然后，执行线性回归，并添加额外的惩罚，以鼓励非零系数的*稀疏个数*（非常少）。换句话说，我们求解以下成本函数：

$$C(\mathbf{z}) = \|\mathbf{b} - \mathbf{Az}\|^2 + \alpha\|\mathbf{z}\|_1$$

其中α是控制稀疏性惩罚强度的标量参数：较高的α导致$\mathbf{z}$中的非零系数较少（但与数据的拟合可能更差），而较低的α将导致更多的非零系数。在本笔记本中，我们将使用`sklearn.lineral_model.Lasso`模型来求解上述成本函数。

TODO:

修改下面的`construct_A_b`函数来计算u的时间和空间导数，以形成矩阵**A**和向量**b**。对于矩阵**A**，包括您认为拟合数据所需的尽可能多的项（例如，~10或更多）。

```
def construct_A_b(u, t, N=2000, normalize=True):
    """
    Construct matrix A and vector b for the linear system A * z = b.

    Parameters
    ----------

    Returns
    -------
    A: ndarray
        Matrix A.
    b: ndarray
        Vector b.
    column_scales: ndarray, optional
        Scale factors for normalizing the columns of A (if normalize=True)
    """
    # 存储形状
    Nt, Nx = u.shape
```

```
    # 计算时间导数u_t
    u_t = np.gradient(u, t, axis=0, edge_order=2)

    #---------------------------------------------------------
    # TODO: 计算各种空间导数u_x、u_x等。
    u_x = np.gradient(u, x, axis=1, edge_order=2)
    u_xx = np.gradient(u_x, x, axis=1, edge_order=2)
    #---------------------------------------------------------

    # 构造A和b的随机样本u及其导数
    rng = np.random.default_rng(1337)
    inds = rng.choice(Nt*Nx, size=N, replace=False)
    u, u_t, u_x, u_xx = [arr.ravel()[inds] for arr in (u, u_t, u_x, u_xx)]

    #---------------------------------------------------------
    # 构造A和b的随机样本u及其导数
    A = np.column_stack((u_xx.ravel(), u.ravel()))
    #---------------------------------------------------------

    # 可选：计算每列的最大绝对值
    # 我们可以使用这些来归一化每一列，这样每个特征
    # 对预测A*z的贡献大致相等
    if normalize:
        column_scales = np.max(np.abs(A), axis=0)
        A = A / column_scales
        return A, u_t, column_scales
    else:
        return A, u_t
```

训练

现在，让我们使用我们编写的函数来创建用于训练的**A**和**b**，以估计**z**和PDE的形式。本练习的一个关键约束是，**我们将仅使用u的1秒观测跨度**进行训练。

```
# 选择数据的一秒钟时间跨度进行训练
# 让我们先来看看最后一秒的模拟数据
mask = t > 5
u1 = u[mask, :]
t1 = t[mask]

# 为此子集创建A和b
# 注意：我们会跟踪归一化项，以便稍后使用
A_train, b_train, column_scales = construct_A_b(u1, t1, normalize=True)
```

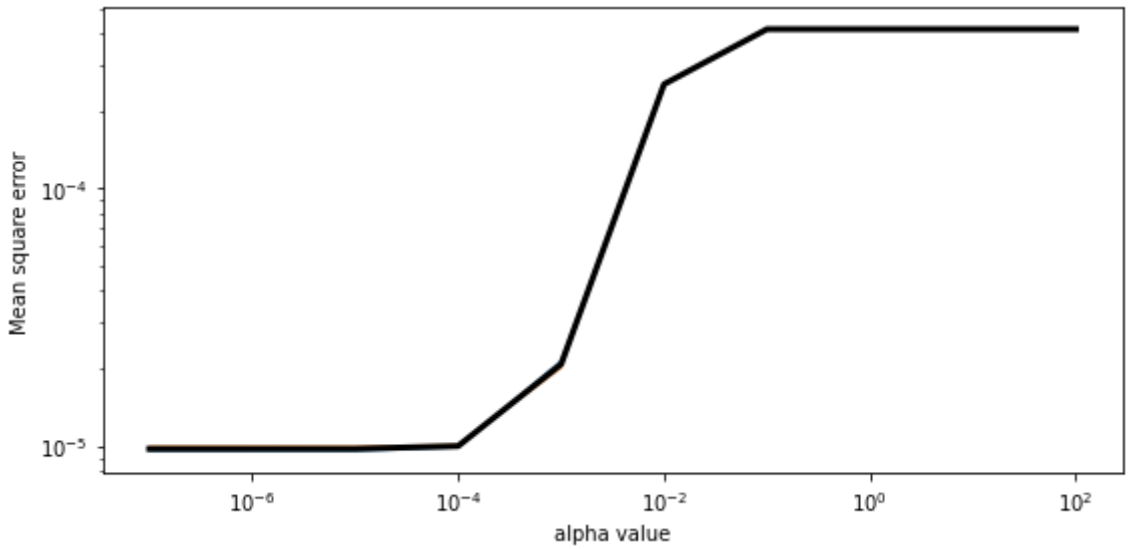
交叉验证

为了使用 `sklearn.lineral_model.Lasso` 模型和求解器，我们需要为惩罚参数α指定一个值。但是，什么值是输出数据的最佳值？为了解决这个问题，我们将使用 `sklearn.lineral_model.LasoCV` 类来执行K折叠交叉验证。

TODO:

修改以下代码块以调整 `LassoCV` 测试的α的值。理想情况下，α的最佳值是当均方误差（MSE）最低时（下图中的黑色曲线）。如果没有明显的最小值，请选择仍然产生相对较低MSE的α的最高值。

```
#--------------------------------------------------------------
# TODO: 修改LassoCV的实例化以选择适当的alpha值范围
# 来测试和适当数量的交叉验证折叠
cv = LassoCV(alphas=[10**n for n in range (-7,3)],
             fit_intercept=False,
             cv=2,
             max_iter=10000,
             random_state=50)
cv.fit(A_train, b_train);
#--------------------------------------------------------------

# 现在绘制交叉验证性能
fig, ax = plt.subplots(figsize=(8, 4))
ax.loglog(cv.alphas_, cv.mse_path_, lw=0.75, alpha=0.75)
ax.loglog(cv.alphas_, cv.mse_path_.mean(axis=-1), 'k', lw=3)
ax.set_xlabel('alpha value')
ax.set_ylabel('Mean square error')
plt.tight_layout()
```



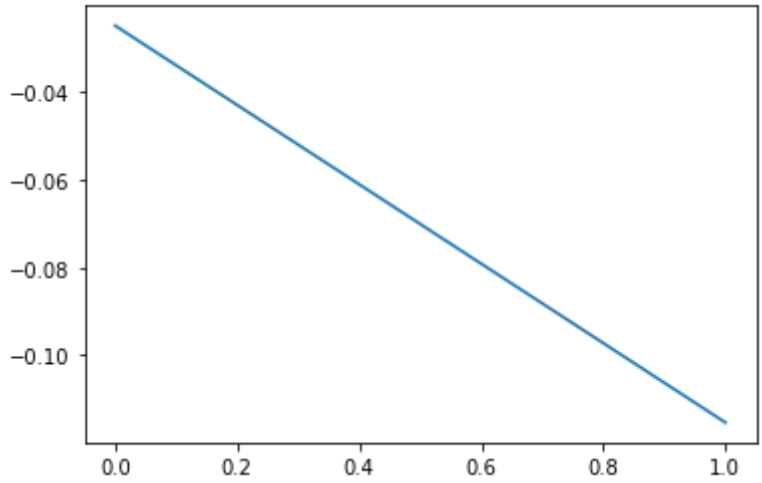
使用Lasso fit找出最适合的α

在选择α的最佳值后，我们现在可以调用 `Lasso` 模型并打印出z中的系数。理想情况下，z的低值表明A中的那些函数不善于预测数据中的信号。

```
#--------------------------------------------------------------
# TODO: 用你的最佳alpha值进行Lasso fit
# 得到交叉验证曲线
lasso = Lasso(alpha=1e-4,
              max_iter=10000)
lasso.fit(A_train, b_train)
#--------------------------------------------------------------

# 绘制系数图
# 注意：我们需要按顺序除以column_scales
# 来说明A的归一化
plt.plot(lasso.coef_ / column_scales)

# 打印出系统的系数时，再次确保考虑
# A的归一化
print('The estimated coefficients are:')
```

```
for i in range(lasso.coef_.size):
    print(i, lasso.coef_[i] / column_scales[i])
```

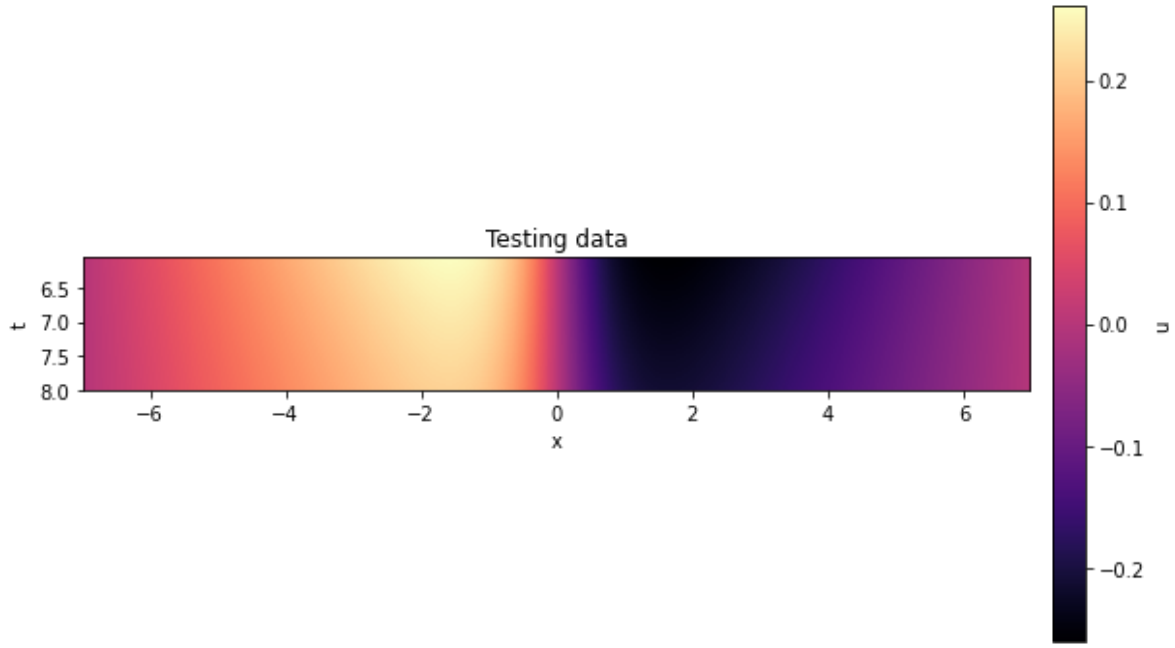
```
The estimated coefficients are:
0 -0.024802203518588858
1 -0.1154012679351321
```



验证

现在我们有了一个估计的PDE（使用**z**中的非零系数），让我们将PDE应用于验证数据集。该数据集是用于训练的模拟的最后两秒。让我们先来看看数据：

```
# 加载u
u_test = np.loadtxt('u_test.txt')
# 加载t
t_test = np.loadtxt('t_test.txt')

# 可视化测试数据
fig, ax = plt.subplots(figsize=(9, 6))
im = ax.imshow(u_test, extent=[x[0], x[-1], t_test[-1], t_test[0]],
cmap='magma')
plt.colorbar(im, ax=ax, pad=0.02, shrink=0.8, label='u')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_title('Testing data')
plt.tight_layout()
```



为了验证模型，我们将计算测试数据的时间导数，然后使用您认为最适合训练数据的PDE将其与预测的时间导数进行比较。

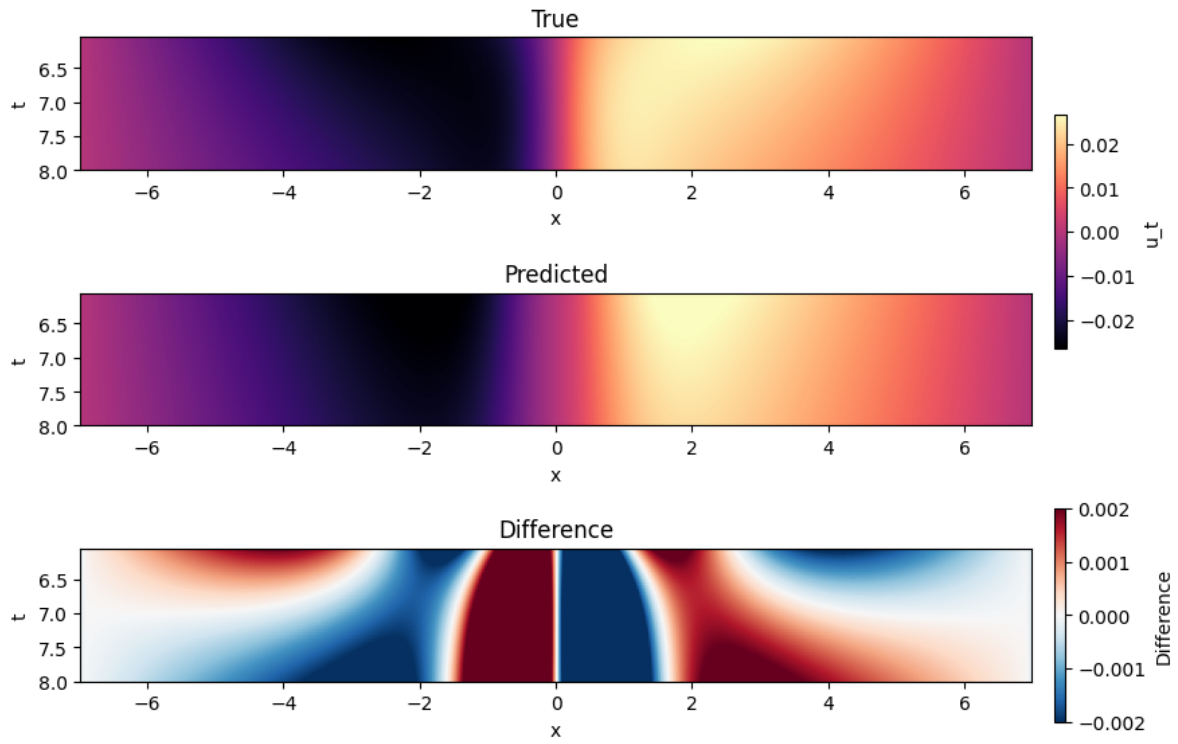
TODO:

修改以下代码块，将估计模型应用于测试数据，以获得预测的u_t。然后绘制真实和预测u_t的图像以及预测误差。模型在测试数据上的表现如何?

```
# 测试数据的时间导数是目标值
u_t_test = np.gradient(u_test, t_test, axis=0, edge_order=2)

#------------------------------------------------------------
# TODO: 计算估计的物理方程的必要组成部分
# 注意：必须在u_test上计算空间梯度
u_x = np.gradient(u_test, x, axis=1, edge_order=2)
u_xx = np.gradient(u_x, x, axis=1, edge_order=2)
u_t_pred = lasso.coef_[0] / column_scales[0] * u_xx + lasso.coef_[1] /
column_scales[1] * u_test
# 手动输入估计模型，例如 u_t_pred = 0.1 * u_xx + 2 * u
#------------------------------------------------------------

# 绘图
fig, (ax1, ax2, ax3) = plt.subplots(nrows=3, figsize=(11, 7))
extent = [x[0], x[-1], t_test[-1], t_test[0]]

im1 = ax1.imshow(u_t_test, extent=extent, cmap='magma')
im2 = ax2.imshow(u_t_pred, extent=extent, clim=(im1.get_clim()),
cmap='magma')

im3 = ax3.imshow(u_t_pred - u_t_test, extent=extent, cmap='RdBu_r', clim=
(-0.002, 0.002))

cbar1 = plt.colorbar(im1, ax=[ax1, ax2], pad=0.02, shrink=0.5, label='u_t')
cbar2 = plt.colorbar(im3, ax=ax3, pad=0.02, label='Difference')

for ax in (ax1, ax2, ax3):
    ax.set_xlabel('x')
    ax.set_ylabel('t')
ax1.set_title('True')
```

```
ax2.set_title('Predicted')
ax3.set_title('Difference')
```

```
Text(0.5, 1.0, 'Difference')
```



TODO:

- 用**不同的1秒时间跨度**训练数据重复上述分析。具体来说，使用与最初用于训练的时间段具有不同动态特性的时间段。例如，如果最初使用$t < 1$，请尝试使用$t > 5$。
- 你估计的模型有变化吗？您的交叉验证曲线和最佳α值是否发生变化？解释任何变化并提供一些物理解释。