```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle


data = pd.read_csv('function2.csv')
x = data['x'].to_numpy()
y = data['y'].to_numpy()

DEGREE = 2


def get_polynomial_data(x_data, degree=1):
    poly_data = []
    for a in x_data:
        curr = []
        curr_val = 1
        for i in range(degree + 1):
            curr.append(curr_val)
            curr_val *= a
        poly_data.append(curr)
    return np.array(poly_data)


def normalize_data(x):
    x[:, 1:] = (x[:, 1:] - np.mean(x[:, 1:], axis=0)) / np.std(x[:, 1:], axis=0)
    return x


def predict(x, weights):
    return np.matmul(x, weights)


def compute_error(x, y, weights):
    preds = np.matmul(x, weights)
    err = np.subtract(preds, y)
    return err


def fit(x_data, y_data, degree=1, lambda_=0.1):
    phi = get_polynomial_data(x_data, degree=degree)
    x1 = np.matmul(phi.T, phi) + lambda_ * np.identity(degree + 1)
    x2 = np.matmul(phi.T, y_data)
    weights = np.matmul(np.linalg.inv(x1), x2)
    return weights


def train(x_data, y_data, degree=1, num_iters=100, learning_rate=0.1):
    x_data = get_polynomial_data(x_data, degree=degree)
    x_data = normalize_data(x_data)
    weights = np.random.randn(degree + 1)
    for i in range(num_iters):
        err = compute_error(x_data, y_data, weights)
        grads = np.matmul(x_data.T, err)
        weights -= learning_rate * grads
    return weights


def test(weights, degree=2, num_points=50):
    x_test = np.linspace(-1.0, 1.0, num=num_points)
    x_test_poly = get_polynomial_data(x_test, degree)
    x_test_poly = normalize_data(x_test_poly)
    y_test = np.matmul(x_test_poly, weights)
    return x_test, y_test
```

```
def cross_validate(weights, x_val, y_val, degree):
    x_val = get_polynomial_data(x_val, degree=degree)
    y_pred = predict(x_val, weights)
    err = 0
    for i in range(x_val.shape[0]):
        err += (y_pred[i] - y_val[i]) ** 2
    return (2 * err / x_val.shape[0]) ** 0.5


# Quad reg: size 10 - degree 6, 0.1 size 200 - degree 9, 0.001
# No reg: size 10 - degree 3, size 200 - degree 9

DATASET_SIZE = 10
CROSSVAL_SIZE = 10
TEST_SIZE = 5
degrees = [2, 3, 6, 9]
# lambdas = [0.001, 0.01, 0.1, 1.0]
lambdas = [0]
fig, ax = plt.subplots()
min_error = 9999
best_weights = None
best_degree = 2
best_lambda = 0.0
table = []
# for degree in degrees:
#     for lambda_ in lambdas:
#         res = fit(x[0:DATASET_SIZE], y[0:DATASET_SIZE], degree=degree, lambda_=lambda_)
#         train_err = cross_validate(res, x[0:DATASET_SIZE], y[0:DATASET_SIZE], degree)
#         cross_val_err = cross_validate(res, x[DATASET_SIZE:DATASET_SIZE+CROSSVAL_SIZE],
y[DATASET_SIZE:DATASET_SIZE+CROSSVAL_SIZE], degree)
#         row = str(degree) + ' ' + str(lambda_) + ' ' + str(train_err) + ' ' + str(cross_val_err)
#         table.append(row)
#         if cross_val_err < min_error:
#             min_error = cross_val_err
#             best_weights = res
#             best_degree = degree
#             best_lambda = lambda_
        # x_test, y_test = test(res, degree=degree, num_points=50)
        # ax.plot(x_test, y_test, label='Lambda=' + str(lambda_))
# plt.legend(loc='best')
# plt.xlabel('x')
# plt.ylabel('y')
# plt.show()
# print(best_weights)
# print(min_error)
# print(best_degree)
# print(best_lambda)
# with open('dataset1_no_reg_10_erms.txt', 'w') as f:
#     f.write('\n'.join(table))

res = fit(x[0:DATASET_SIZE], y[0:DATASET_SIZE], degree=3, lambda_=0)
test_err = cross_validate(res,
x[DATASET_SIZE+CROSSVAL_SIZE:DATASET_SIZE+CROSSVAL_SIZE+TEST_SIZE],
y[DATASET_SIZE+CROSSVAL_SIZE:DATASET_SIZE+CROSSVAL_SIZE+TEST_SIZE], 3)
print(test_err)
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D


data = pd.read_csv('function2_2d.csv')
x1 = data['x1'].to_numpy()
```

```
x2 = data['x2'].to_numpy()
y = data['y'].to_numpy()
data = data.drop(['y'], axis=1)
data = data.drop(['Unnamed: 0'], axis=1)
x = data.to_numpy()
print(x.shape)


def get_phi(x1, x2, degree=2):
    phi = []
    for i in range(x1.shape[0]):
        curr = []
        curr_x1 = x1[i]
        curr_x2 = x2[i]
        curr.append(1)
        for j in range(1, degree + 1):
            num_terms = j + 1
            for k in range(num_terms):
                pow_x1 = math.pow(curr_x1, k)
                pow_x2 = math.pow(curr_x2, j - k)
                curr.append(pow_x1 * pow_x2)
        phi.append(curr)
    return np.array(phi)


def predict(x, weights):
    return np.matmul(x, weights)


def fit(x1, x2, y, degree=2, lambda_=0.0):
    phi = get_phi(x1, x2, degree=degree)
    print(phi.shape)
    num_terms = 0
    for i in range(degree + 1):
        num_terms += i + 1
    w1 = np.matmul(phi.T, phi) + lambda_ * np.identity(num_terms)
    w2 = np.matmul(phi.T, y)
    weights = np.matmul(np.linalg.inv(w1), w2)
    return weights


def quadratic_regularization(phi, y, lambda_=0.0):
    w1 = np.matmul(phi.T, phi) + lambda_ * np.identity(phi.shape[1])
    w2 = np.matmul(phi.T, y)
    weights = np.matmul(np.linalg.inv(w1), w2)
    return weights


def tikhonov_regularization(phi, y, mus, sigma=1, lambda_=0.0):
    phi_prime = np.zeros((phi.shape[1], phi.shape[1]))
    den = sigma ** 2
    for i in range(mus.shape[0]):
        for j in range(mus.shape[0]):
            if i == j:
                phi_prime[i, j] = 1
            else:
                phi_prime[i, j] = np.exp(-np.linalg.norm(mus[i] - mus[j]) ** 2 / den)
    w1 = np.matmul(phi.T, phi) + lambda_ * phi_prime
    w2 = np.matmul(phi.T, y)
    weights = np.matmul(np.linalg.inv(w1), w2)
    return weights


def test(weights, x1_, x2_, degree=2, num_points=50):
    phi = get_phi(x1_, x2_, degree=degree)
    y_test = np.matmul(phi, weights)
    return x1_, x2_, y_test
```

```
def compute_error(phi, weights, y):
    error = 0.0
    for i in range(phi.shape[0]):
        point = phi[i]
        pred = np.matmul(point, weights)
        error += (y[i] - pred) ** 2
    return (2 * error / phi.shape[0]) ** 0.5


def get_kmeans(k, x, num_iters=2):
    init_means = np.random.randint(0, x.shape[0], size=k)
    kmeans = np.zeros((k, x.shape[1]))
    for i in range(k):
        kmeans[i] = x[init_means[i]]
    point_means = np.zeros(x.shape)
    tol = 0.001
    for i in range(num_iters):
        for j in range(x.shape[0]):
            point = x[j]
            min_dist = 999
            curr_mean = kmeans[0]
            for k in range(kmeans.shape[0]):
                curr_dist = np.linalg.norm(point - kmeans[k])
                if curr_dist < min_dist:
                    curr_mean = kmeans[k]
                    min_dist = curr_dist
            point_means[j] = curr_mean
        err = 0
        for j in range(kmeans.shape[0]):
            curr_sum = np.zeros(x.shape[1])
            curr_num = 0
            for k in range(x.shape[0]):
                if (point_means[k] == kmeans[j]).all():
                    curr_sum += x[k]
                    curr_num += 1
            new_mean = curr_sum / curr_num
            err += np.linalg.norm(kmeans[j] - new_mean)
            kmeans[j] = new_mean
        if err < tol:
            break
    sigma = 0
    for j in range(kmeans.shape[0]):
        curr_sum = 0
        curr_count = 0
        for k in range(x.shape[0]):
            if (point_means[k] == kmeans[j]).all():
                curr_sum += np.linalg.norm(x[k] - kmeans[j]) ** 2
                curr_count += 1
        sigma += curr_sum
    return kmeans, sigma / x.shape[0]


def get_gaussian_phi(kmeans, x, sigma):
    phi = []
    den = sigma ** 2
    for point in x:
        curr = []
        for mu in kmeans:
            curr.append(np.exp(-(np.linalg.norm(point - mu) ** 2) / den))
        phi.append(curr)
    return np.array(phi)


def surface_test(weights, degree=2):
    x1_test = np.linspace(-10, 10, 50)
```

```
    x2_test = np.linspace(-10, 10, 50)
    X, Y = np.meshgrid(x1_test, x2_test)
    _, __, Z = test(weights, np.ravel(X), np.ravel(Y), degree=degree)
    Z = Z.reshape(X.shape)
    return X, Y, Z


# START = 300
# DATASET_SIZE = 500
# CROSS_VAL_SIZE = 30
# degrees = [2, 3, 6, 9]
# lamdas = [0, 0.01, 0.1, 1.0]
# DEGREE = 2
# min_error = 9999
# best_weights = None
# best_degree = 2
# best_lambda = 0.0
# table = []
# res = fit(x1[START:START+DATASET_SIZE], x2[START:START+DATASET_SIZE],
y[START:START+DATASET_SIZE], degree=2, lambda_=0)
# test_phi = get_phi(x1[START+DATASET_SIZE:START+DATASET_SIZE+CROSS_VAL_SIZE],
x2[START+DATASET_SIZE:START+DATASET_SIZE+CROSS_VAL_SIZE], degree=2)
# val_err = compute_error(test_phi, res, y[START+DATASET_SIZE:START+DATASET_SIZE+CROSS_VAL_SIZE])
# print(val_err)
# preds = predict(phi, res)
# fig, ax = plt.subplots()
# ax.scatter(y[START+DATASET_SIZE:START+DATASET_SIZE+CROSS_VAL_SIZE], preds)
# plt.xlabel('Target Output')
# plt.ylabel('Model Output')
# plt.show()
# for degree in degrees:
#     for lambda_ in lamdas:
#         res = fit(x1[0:DATASET_SIZE], x2[0:DATASET_SIZE], y[0:DATASET_SIZE], degree=degree, lambda_=lambda_)
#         x1_, x2_,  y_ = test(res, x1[100:100 + DATASET_SIZE], x2[100:100 + DATASET_SIZE], degree=degree)
#         phi = get_phi(x1[0:DATASET_SIZE], x2[0:DATASET_SIZE], degree=degree)
#         cross_phi = get_phi(x1[DATASET_SIZE:DATASET_SIZE+CROSS_VAL_SIZE],
x2[DATASET_SIZE:DATASET_SIZE+CROSS_VAL_SIZE], degree=degree)
#         train_err = compute_error(phi, res, y[0:DATASET_SIZE])
#         val_err = compute_error(cross_phi, res, y[DATASET_SIZE:DATASET_SIZE+CROSS_VAL_SIZE])
#         row = str(degree) + ' ' + str(lambda_) + ' ' + str(train_err) + ' ' + str(val_err)
#         table.append(row)
#         if val_err < min_error:
#             min_error = val_err
#             best_weights = res
#             best_lambda = lambda_
#             best_degree = degree
#         # X, Y, Z = surface_test(res, degree=degree)
#         # fig = plt.figure()
#         # ax = fig.add_subplot(111, projection='3d')
#         # ax.scatter(x1, x2, y)
#         # ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
#         # ax.plot_trisurf(x1_, x2_, y_, linewidth=0.1)
#         # ax.set_xlabel('x1')
#         # ax.set_ylabel('x2')
#         # ax.set_zlabel('y')
#         # filename = 'degree_' + str(degree) + '_lamdbda_' + str(lambda_) + '.png'
#         # fig.savefig('plots/Task 2/Dataset Size 500/' + filename, bbox_inches ="tight")
# print('Best degree:', best_degree)
# print('Best lambda:', best_lambda)
# print('Min error:', min_error)
# with open('dataset2_50_train_val_erms.txt', 'w') as f:
#     f.write('\n'.join(table))
DATASET_SIZE = x.shape[0]
TRAIN_SIZE = int(0.7 * DATASET_SIZE)
VAL_SIZE = int(0.2 * DATASET_SIZE)
TEST_SIZE = int(0.1 * DATASET_SIZE)
mean_sizes = [10, 25, 30]
```

```
# mean_sizes = [20, 60, 100]
lambdas = [0.0001, 0.1, 1.0]
sigmas = [8, 20, 30]
# sigmas = [2, 5, 8, 9]
rmse = []
table = []
for k in mean_sizes:
    for lambda_ in lambdas:
        for sigma in sigmas:
            try:
                print(k, lambda_, sigma)
                kmeans, _ = get_kmeans(k, x[0:TRAIN_SIZE], num_iters=100)
                phi = get_gaussian_phi(kmeans, x, 20)
                # weights = tikhonov_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], kmeans, sigma, lambda_=lambda_)
                weights = quadratic_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], lambda_=lambda_)
                err = compute_error(phi[0:TRAIN_SIZE], weights, y[0:TRAIN_SIZE])
                val_err = compute_error(phi[TRAIN_SIZE:TRAIN_SIZE+VAL_SIZE], weights,
y[TRAIN_SIZE:TRAIN_SIZE+VAL_SIZE])
                rmse.append(err)
                row = str(k) + ' ' + str(lambda_) + ' ' + str(sigma) + ' ' + str(err) + ' ' + str(val_err)
                table.append(row)
            except Exception as e:
                print(e)
            # print('K:', k, 'lambda:', 0.0001, 'sigma:', 20)
            # print('Pred:', np.matmul(phi[100], weights))
            # print('Actual:', y[100])
            # print('Error:', err)

with open('task3_dataset2_quad_erms.txt', 'w') as f:
    f.write('\n'.join(table))
#
# fig, ax = plt.subplots()
# ax.plot(mean_sizes, rmse)
# plt.xlabel('Cluster Size')
# plt.ylabel('Erms')
# plt.show()

# kmeans, _ = get_kmeans(100, x, num_iters=100)
# phi = get_gaussian_phi(kmeans, x, 8)
# weights = quadratic_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], lambda_=0.0001)
# preds = np.matmul(phi[0:TRAIN_SIZE], weights)
#
# fig, ax = plt.subplots()
# ax.scatter(y[0:TRAIN_SIZE], preds)
# plt.xlabel('Target Output')
# plt.ylabel('Model Output')
# plt.show()
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy

def predict(x, weights):
    return np.matmul(x, weights)

def quadratic_regularization(phi, y, lambda_=0.0):
    w1 = np.matmul(phi.T, phi) + lambda_ * np.identity(phi.shape[1])
    w2 = np.matmul(phi.T, y)
    weights = np.matmul(np.linalg.inv(w1), w2)
    return weights


def tikhonov_regularization(phi, y, mus, sigma, lambda_):
    phi_prime = np.zeros((phi.shape[1], phi.shape[1]))
    den = sigma ** 2
    for i in range(mus.shape[0]):
```

```
        for j in range(mus.shape[0]):
            if i == j:
                phi_prime[i, j] = 1
            else:
                phi_prime[i, j] = np.exp(-np.linalg.norm(mus[i] - mus[j]) ** 2 / den)
    w1 = np.matmul(phi.T, phi) + lambda_ * phi_prime
    w2 = np.matmul(phi.T, y)
    weights = np.matmul(np.linalg.inv(w1), w2)
    return weights


def compute_error(phi, weights, y):
    error = 0.0
    for i in range(phi.shape[0]):
        point = phi[i]
        pred = np.matmul(point, weights)
        error += (y[i] - pred) ** 2
    return (error / phi.shape[0]) ** 0.5


def get_kmeans(k, x, num_iters=2):
    print('Computing K Means...')
    init_means = np.random.randint(0, x.shape[0], size=k)
    kmeans = np.zeros((k, x.shape[1]))
    for i in range(k):
        kmeans[i] = x[init_means[i]]
    point_means = np.zeros(x.shape)
    tol = 0.1
    for i in range(num_iters):
        print(i)
        for j in range(x.shape[0]):
            point = x[j]
            min_dist = 999
            curr_mean = kmeans[0]
            for k in range(kmeans.shape[0]):
                curr_dist = np.linalg.norm(point - kmeans[k])
                if curr_dist < min_dist:
                    curr_mean = kmeans[k]
                    min_dist = curr_dist
            point_means[j] = curr_mean
        err = 0
        for j in range(kmeans.shape[0]):
            curr_sum = np.zeros(x.shape[1])
            curr_num = 0
            for k in range(x.shape[0]):
                if (point_means[k] == kmeans[j]).all():
                    curr_sum += x[k]
                    curr_num += 1
            new_mean = curr_sum / curr_num
            err += np.linalg.norm(kmeans[j] - new_mean)
            kmeans[j] = new_mean
        if err < tol:
            break
    sigma = 0
    for j in range(kmeans.shape[0]):
        curr_sum = 0
        curr_count = 0
        for k in range(x.shape[0]):
            if (point_means[k] == kmeans[j]).all():
                curr_sum += np.linalg.norm(x[k] - kmeans[j]) ** 2
                curr_count += 1
        sigma += curr_sum
    return kmeans, sigma / x.shape[0]


def get_gaussian_phi(kmeans, x, sigma):
    phi = []
```

```
    den = sigma ** 2
    for point in x:
        curr = []
        for mu in kmeans:
            curr.append(np.exp(-(np.linalg.norm(point - mu) ** 2) / den))
        phi.append(curr)
    return np.array(phi)




data_sc = pd.read_csv('0_superconductor.csv')
cols = ['wtd_std_Valence', 'critical_temp']
# print(data_sc.columns)
y = data_sc[cols].to_numpy()
# y1 = data_sc['wtd_std_Valence'].to_numpy()
# y2 = data_sc['critical_temp'].to_numpy()
# print(cols)
# print(data_sc.columns)
x = data_sc.drop(cols, axis=1)
x = x.to_numpy()


DATASET_SIZE = x.shape[0]
TRAIN_SIZE = int(0.7 * DATASET_SIZE)
VAL_SIZE = int(0.2 * DATASET_SIZE)
TEST_SIZE = int(0.1 * DATASET_SIZE)
mean_sizes = [50, 70, 100]
# mean_sizes = [50]
lambdas = [0.0001, 0.001, 0.01, 0.1, 1.0]
# lambdas = [0.001]
sigmas = [20, 50, 100, 150, 200]
# sigmas = [200]
# rmse1 = []
# rmse2 = []
min_error = [999999, 999999]
models = []
for k in mean_sizes:
    kmeans, _ = get_kmeans(k, x[0:TRAIN_SIZE], num_iters=100)
    for sigma in sigmas:
        phi = get_gaussian_phi(kmeans, x, sigma)
        for ld in lambdas:
            try:
                # weights = tikhonov_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], kmeans, sigma, ld)
                w1 = np.matmul(phi.T, phi)
                w2 = np.matmul(phi.T, y)
                weights = np.matmul(np.linalg.inv(w1), w2)
                # weights = quadratic_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], ld)
                err_train = compute_error(phi[0:TRAIN_SIZE], weights, y[0:TRAIN_SIZE])
                err_val = compute_error(phi[TRAIN_SIZE:TRAIN_SIZE+VAL_SIZE], weights,
y[TRAIN_SIZE:TRAIN_SIZE+VAL_SIZE])
                err_test = compute_error(phi[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE], weights,
y[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE])
                models.append([k+1, sigma, ld, err_train[0], err_train[1], err_val[0], err_val[1], err_test[0], err_test[1]])
                if sum(err_val) < sum(min_error):
                    min_error = copy.copy(err_val)
                    best_weights = copy.copy(weights)
                    best_sigma = sigma
                    best_lambda = ld
                    best_k = k
                    best_phi = copy.deepcopy(phi)


                print('K:', k, 'lambda:', ld, 'sigma:', sigma)
            except Exception as e:
                print(e)
```

```
print(best_weights)
print(best_sigma)
print(best_lambda)
print(best_k)


print('Pred:', np.matmul(best_phi[100], best_weights))
print('Actual:', y[100])
print('Error:', min_error)
print(models)
# pd.DataFrame(models).to_csv("models.csv")
# , columns=['Dimension', 'width', 'lamda', 'Erms_train1', 'Erms_train2', 'Erms_val1', 'Erms_val2', 'Erms_test1',
'Erms_test2']
models = np.asarray(models)
np.savetxt("models.csv", models, delimiter=",")
np.savetxt("phi.csv", best_phi, delimiter=",")
np.savetxt("weights.csv", weights, delimiter=",")

# fig, ax = plt.subplots()
# ax.plot(mean_sizes, rmse1)
# # ax.plot(mean_sizes, rmse2)
# plt.xlabel('Cluster Size')
# plt.ylabel('Erms')
# plt.show()

# kmeans, _ = get_kmeans(100, x, num_iters=100)
# phi = get_gaussian_phi(kmeans, x, 8)
# weights = quadratic_regularization(phi[0:TRAIN_SIZE], y[0:TRAIN_SIZE], lambda_=0.0001)
# preds = np.matmul(phi[0:TRAIN_SIZE], weights)
#
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy

data_sc = pd.read_csv('0_superconductor.csv')
DATASET_SIZE = data_sc.shape[0]
TRAIN_SIZE = int(0.7 * DATASET_SIZE)
VAL_SIZE = int(0.2 * DATASET_SIZE)
TEST_SIZE = int(0.1 * DATASET_SIZE)

y1 = data_sc['wtd_std_Valence']
y2 = data_sc['critical_temp']

phi = pd.read_csv('phi (2).csv')
phi = phi.to_numpy()

weights = pd.read_csv('weights (2).csv')
weights1 = weights['w1'].to_numpy()
weights2 = weights['w2'].to_numpy()
# print(weights1)
print(y1.shape)

pred_y1 = []
pred_y2 = []
y = []
for i in range(len(y1)):
    pred_y1.append(np.matmul(phi[i], weights1))
    pred_y2.append(np.matmul(phi[i], weights2))
```

```
# def modify_y(x,y):
#     y_modified = []
#     for i in x:
#         summ = 0
#         l = 0
#         for j in range(len(x)):
#             if x[j]==i:
#                 summ = summ + y[j]
#                 l = l+1
#         avg = summ / l
#         y_modified.append(avg)
#     return y_modified

# print(y1[10000], pred_y1[10000])
# for i in range(len(y1)):
#     print(y1[i], pred_y1[i])

# pred_y1 = np.asarray(pred_y1)
# print(pred_y1.shape)
# print(weights1.shape)
#
fig, ax = plt.subplots()
ax.scatter(y1[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE],
pred_y1[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE])
plt.xlabel('Target Output y1')
plt.ylabel('Model Output y1')
plt.show()


fig, ax = plt.subplots()
ax.scatter(y2[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE],
pred_y2[TRAIN_SIZE+VAL_SIZE:TRAIN_SIZE+VAL_SIZE+TEST_SIZE])
plt.xlabel('Target Output y2')
plt.ylabel('Model Output y2')
plt.show()
```