Import

In [1]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.utils.validation import check_is_fitted
import numpy as np
from numpy.linalg import inv
import pandas as pd
from itertools import cycle, islice
import matplotlib.pyplot as plt
import random
import math
```

In [2]:
```
plt.rcParams["font.family"] = "Times New Roman"
```

Data generation

In [3]:
```
num_interval = 90
length_per_interval = 100

c1 = np.linspace(-50, 50, length_per_interval*num_interval)
c2 = np.linspace(0, 5, length_per_interval*num_interval)
c3 = [5] * length_per_interval * num_interval
```

In [4]:
```
x1_variation = ["low", "mid", "high"]
x1_variation_all_intervals = list(islice(cycle(x1_variation), num_interval))
x2_variation = ["low", "mid", "high"]
x2_variation_all_intervals = list(islice(cycle(np.array(x2_variation).repeat(len
```

In [5]:
```
rng = np.random.default_rng(12345)

x1 = []
for x1_variation_per_interval in x1_variation_all_intervals:
    if x1_variation_per_interval == "low":
        x1.extend(rng.uniform(4.9, 5.1, length_per_interval))
    elif x1_variation_per_interval == "mid":
        x1.extend(rng.uniform(3, 7, length_per_interval))
    elif x1_variation_per_interval == "high":
        x1.extend(rng.uniform(0, 10, length_per_interval))
```

In [6]:
```
rng = np.random.default_rng(222)

x2 = []
for x2_variation_per_interval in x2_variation_all_intervals:
    if x2_variation_per_interval == "low":
        x2.extend(rng.uniform(49, 51, length_per_interval))
    elif x2_variation_per_interval == "mid":
        x2.extend(rng.uniform(30, 70, length_per_interval))
    elif x2_variation_per_interval == "high":
        x2.extend(rng.uniform(0, 100, length_per_interval))
```

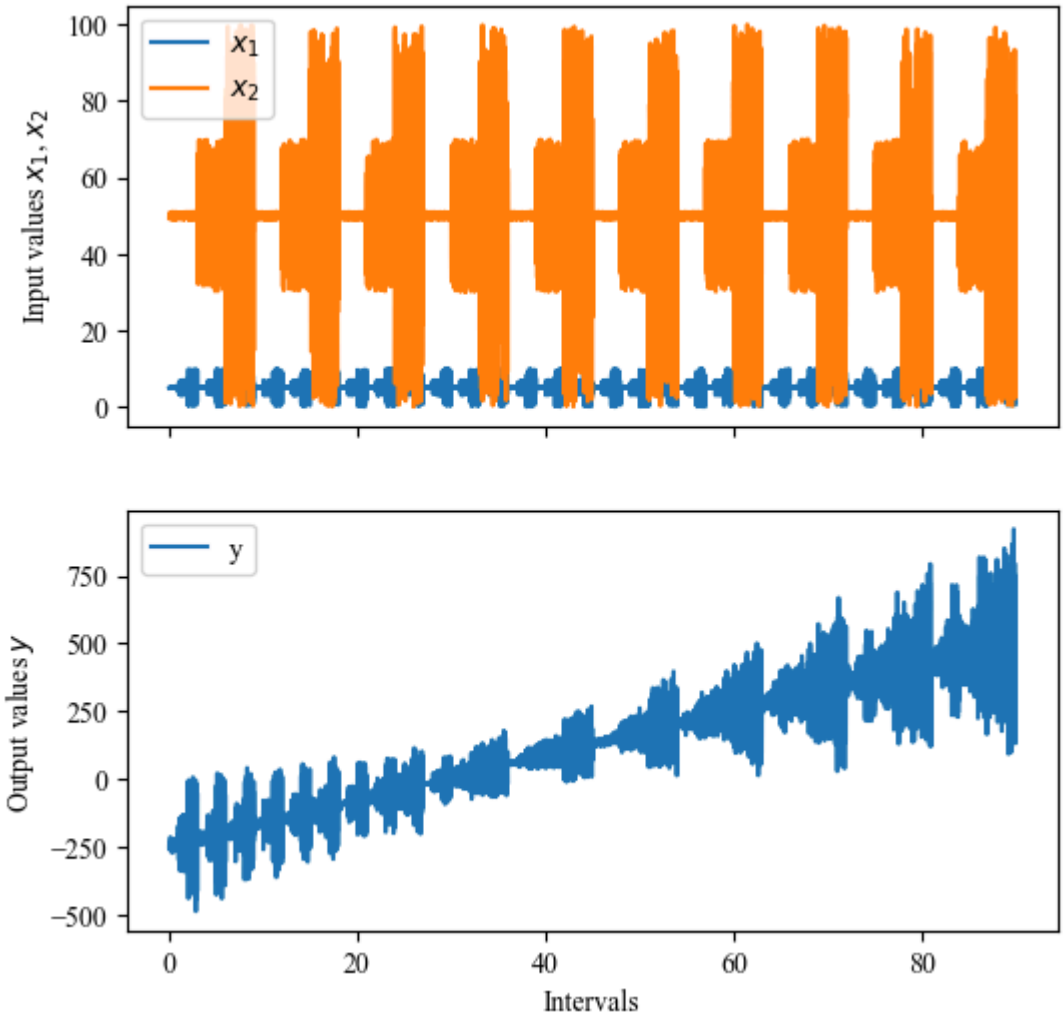
In [7]:
```
y = c1 * x1 + c2 * x2 + c3
noise = y * rng.normal(size=length_per_interval * num_interval) * 0.05
y = y + noise
```

In [8]:
```
data = pd.DataFrame({"y": y, "x1": x1, "x2": x2, "c1": c1, "c2": c2, "c3": c3})
data.head()
```

Out[8]:

	y	x1	x2	c1	c2	c3
0	-249.675179	4.945467	49.754418	-50.000000	0.000000	5
1	-233.751821	4.963352	50.407285	-49.988888	0.000556	5
2	-232.712062	5.059473	50.444397	-49.977775	0.001111	5
3	-247.779734	5.035251	50.732941	-49.966663	0.001667	5
4	-247.876188	4.978222	50.791086	-49.955551	0.002222	5

In [9]:
```
fig, axs = plt.subplots(2, 1, figsize=(6,6), sharex=True)
axs[0].plot(data.index/length_per_interval, data.x1, label=r"$x_1$")
axs[0].plot(data.index/length_per_interval, data.x2, label=r"$x_2$")
axs[0].legend()
axs[0].set_ylabel(r"Input values $x_1$, $x_2$")

axs[1].plot(data.index/length_per_interval, data.y, label="y")
axs[1].legend()
axs[1].set_ylabel(r"Output values $y$")
axs[1].set_xlabel("Intervals")
plt.show()
```



In [10]:
```
x1_ref = 8
x2_ref = 80
```

Transfer linear regression (TLR) algorithm

In [11]:
```
class TransferLinearRegression:
    def __init__(self, lambda_, fit_intercept=True, penalty_matrix=None):
        """
        :param lambda_: setting lambda_ = 0 leads to fitting the target dataset
        setting lambda_ to infinite constrains gamma = 0.
        :param fit_intercept:
        :param penalty_matrix: array-like of shape (n_features,), or if fit_inte
        """
        self.lambda_ = lambda_
        self.fit_intercept = fit_intercept
        self.penalty_matrix = penalty_matrix

    def fit(self, X, y, base_coefficients):
        """
        Fit the model: the base coefficients will be adapted with the given X an
        :param X: array-like of shape (n_samples, n_features) Training data.
        :param y: array-like of shape (n_samples,) Target values.
        :param base_coefficients: array-like of shape (n_features,) Starting coe
        :return: object. A adapted linear regression model.
        """
        if self.fit_intercept:
```

```
            Xs = np.concatenate([X, np.ones((len(X), 1))], axis=1)
        else:
            Xs = X

        if not self.penalty_matrix:
            Xt = np.eye(Xs.shape[1])  # a unit matrix
        else:
            Xt = np.diag(self.penalty_matrix)

        beta = base_coefficients
        Ys = y
        Y = Ys - (Xs @ beta)

        self.gamma = inv(Xs.T @ Xs + self.lambda_ * Xt.T @ Xt) @ Xs.T @ Y
        self.coef_ = beta + self.gamma
        return self

    def predict(self, X):
        check_is_fitted(self, ["coef_"])
        if self.fit_intercept:
            X = np.concatenate([X, np.ones((len(X), 1))], axis=1)
        return X @ self.coef_
```

Get initial model parameters β_0

By model fitting with the first 10% of the data.

In [12]:
```
features = ["x1", "x2"]
```

In [13]:
```
data_init = data.head(round(len(data)/10))
X = data_init[features]
y = data_init[["y"]]
```

In [14]:
```
reg_init = LinearRegression().fit(X, y)
```

In [15]:
```
model_coef = pd.DataFrame(zip(features, reg_init.coef_[0]))
model_coef = model_coef.rename(columns={0: "features", 1: "coefficients"})
model_coef.set_index("features", inplace=True)
model_coef.at["intercept", "coefficients"] = reg_init.intercept_

print("initial coefficients:")
display(model_coef)
```

```
initial coefficients:
```

features	coefficients
x1	-43.969875
x2	0.389758
intercept	-7.090351

Get the 1^{st} term of X_T (variable "penalty_matrix")

Do plain linear regresison (PLR) fitting for all intervals, to get the expected variation of β_p over time.

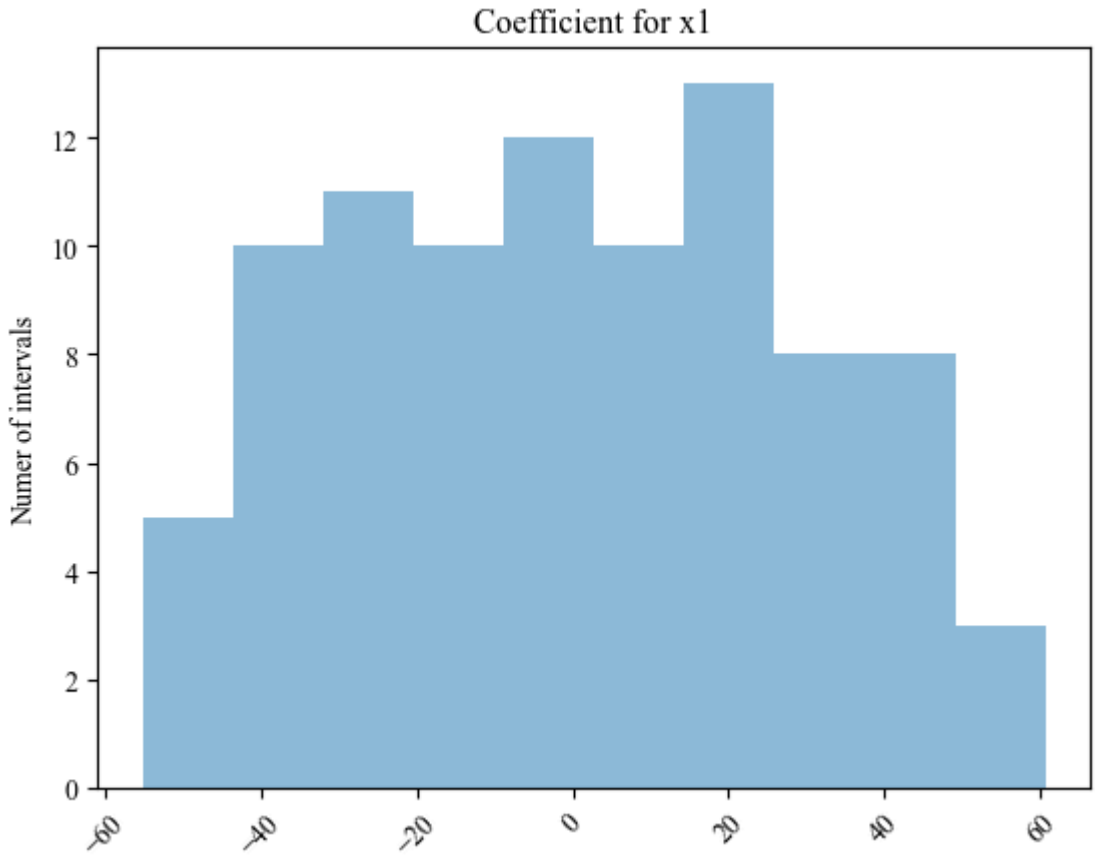
In [16]:
```
fitting_results = pd.DataFrame()

for interval in range(0, num_interval):

    start_index = interval * length_per_interval
    end_index = (interval + 1) * length_per_interval
    data_this_interval = data.query(f"{start_index}<=index<={end_index}")


    X = data_this_interval[features]
    y = data_this_interval[["y"]]

    # Plain linear regression
    reg = LinearRegression().fit(X, y)
    y_pred = reg.predict(X)
    fitting_results.loc[interval, "R2"] = r2_score(y, y_pred)
    fitting_results.loc[interval, "MSE"] = mean_squared_error(y, y_pred)

    for i in range(len(features)):
        fitting_results.loc[interval, features[i]] = reg.coef_[0][i]
        fitting_results.loc[interval, f"c{i+1}"] = data_this_interval[f"c{i+1}"]

    fitting_results.loc[interval, "intercept"] = reg.intercept_[0]
    fitting_results.loc[interval, f"c3"] = data_this_interval[f"c3"].mean()

fitting_results.eval(f"y_ref_estimated = x1 * {x1_ref} + x2 * {x2_ref} + interce
fitting_results.eval(f"y_ref_true = c1 * {x1_ref} + c2 * {x2_ref} + c3", inplace
```

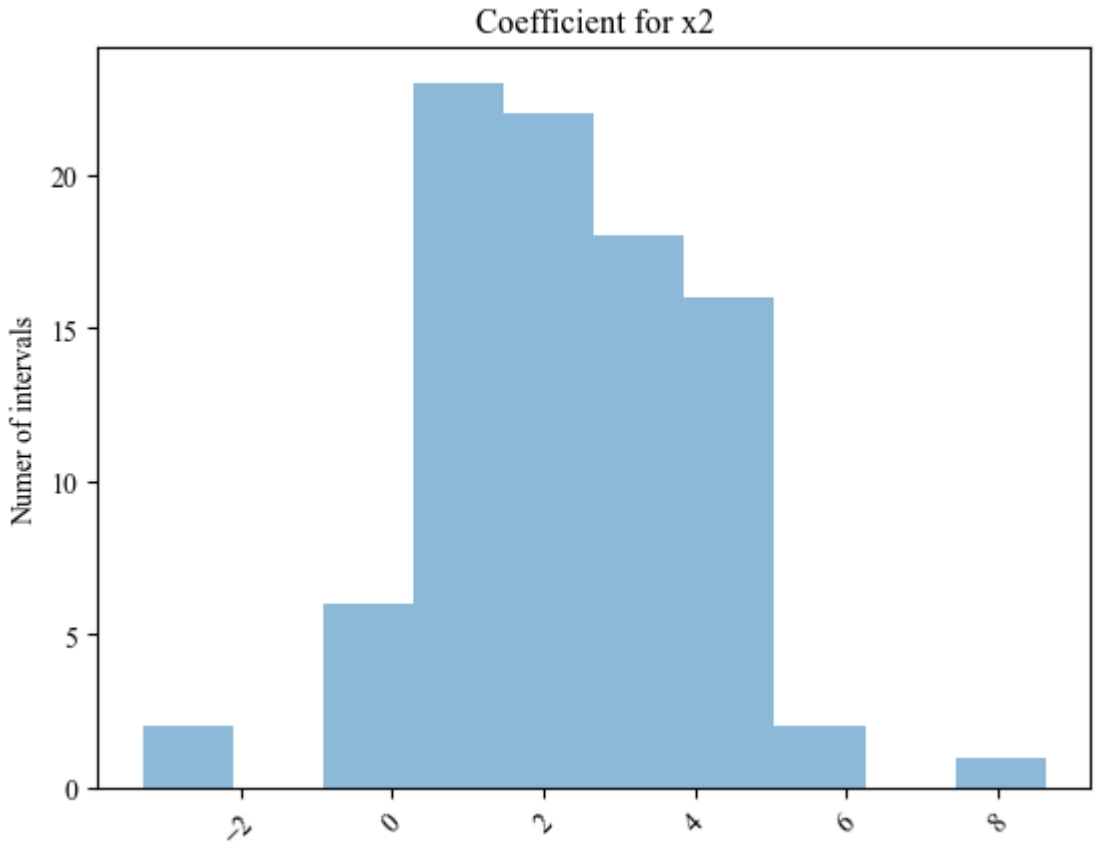
In [17]:
```
fitting_results.head()
```

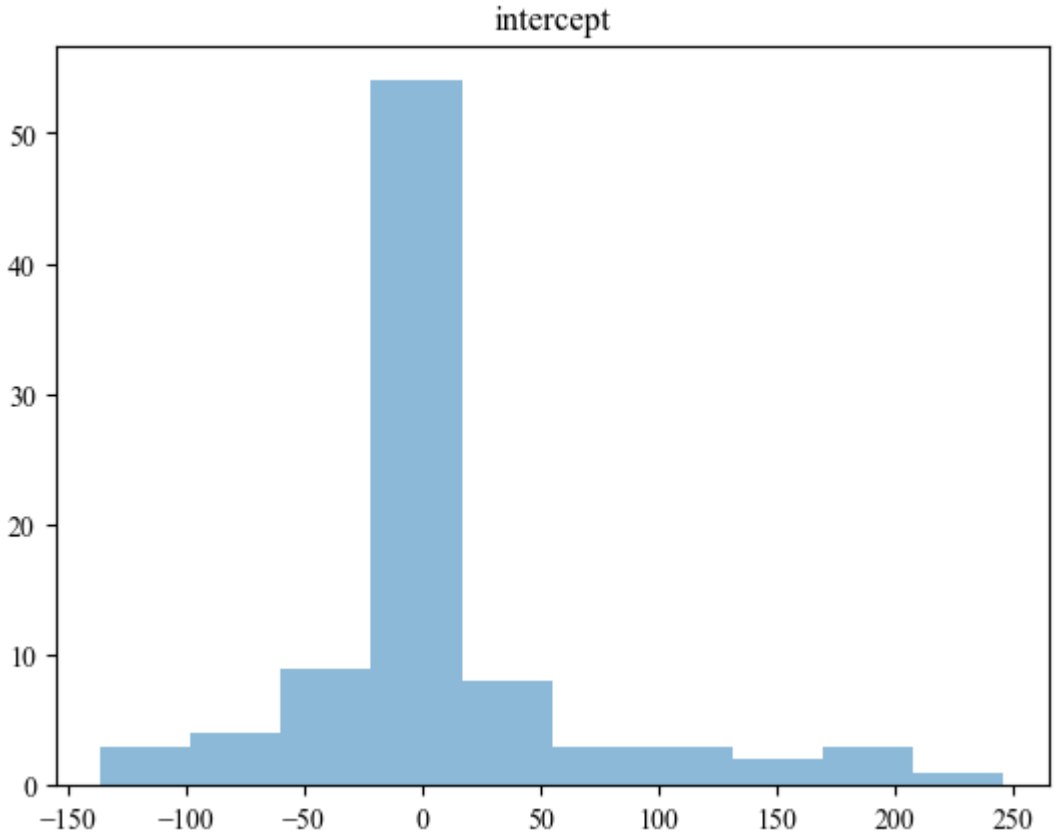
Out[17]:

	R2	MSE	x1	c1	x2	c2	intercept	c3	y
0	0.309657	142.723479	-55.192206	-49.444383	-3.017265	0.027781	184.691631	5.0	
1	0.952517	159.214352	-46.978651	-48.333148	-0.009936	0.083343	5.105926	5.0	
2	0.992186	147.547744	-46.334740	-47.221914	-3.290412	0.138904	174.566751	5.0	
3	0.111926	102.425993	-34.515435	-46.110679	0.240793	0.194466	-56.800798	5.0	
4	0.971916	75.159414	-46.062276	-44.999444	0.366649	0.250028	3.604023	5.0	

In [18]:
```
for feature in features:
    plt.hist(fitting_results.loc[:, feature], alpha = 0.5)
    plt.title(f"Coefficient for {feature}")
    plt.ylabel("Numer of intervals")
    plt.xticks(rotation=45)
    plt.show()
plt.hist(fitting_results.loc[:, "intercept"], alpha = 0.5)
plt.title("intercept")
plt.show()
```


Coefficient for x1
Numer of intervals
0
2
4
6
8
10
12
−60
−40
−20
0
20
40
60


Coefficient for x2
Numer of intervals
0
5
10
15
20
−2
0
2
4
6
8



In [19]:
```
penalty_matrix = []
for feature in features:
    low = np.quantile(fitting_results[feature], 0.25)
    high = np.quantile(fitting_results[feature], 0.75)
    print(f"{feature} range: {low} - {high}")
    print(f"--- Diff: {abs(low-high)}")
    penalty_matrix.append(1/abs(low-high))


low = np.quantile(fitting_results["intercept"], 0.25)
high = np.quantile(fitting_results["intercept"], 0.75)
print(f"intercept range: {low} - {high}")
print(f"--- Diff: {abs(low-high)}")
penalty_matrix.append(1/abs(low-high))
```

```
x1 range: -23.48628032344597 - 22.881966804779864
--- Diff: 46.36824712822583
x2 range: 1.0799642144401722 - 3.650035166095603
--- Diff: 2.5700709516554308
intercept range: -0.3283325213405135 - 14.718771681561734
--- Diff: 15.047104202902247
```

In [20]:
```
penalty_matrix
```

Out[20]: [0.021566482710347445, 0.38909431638682246, 0.06645797001971467]

In [21]:
```
fitting_results_PLR = fitting_results.copy()
```

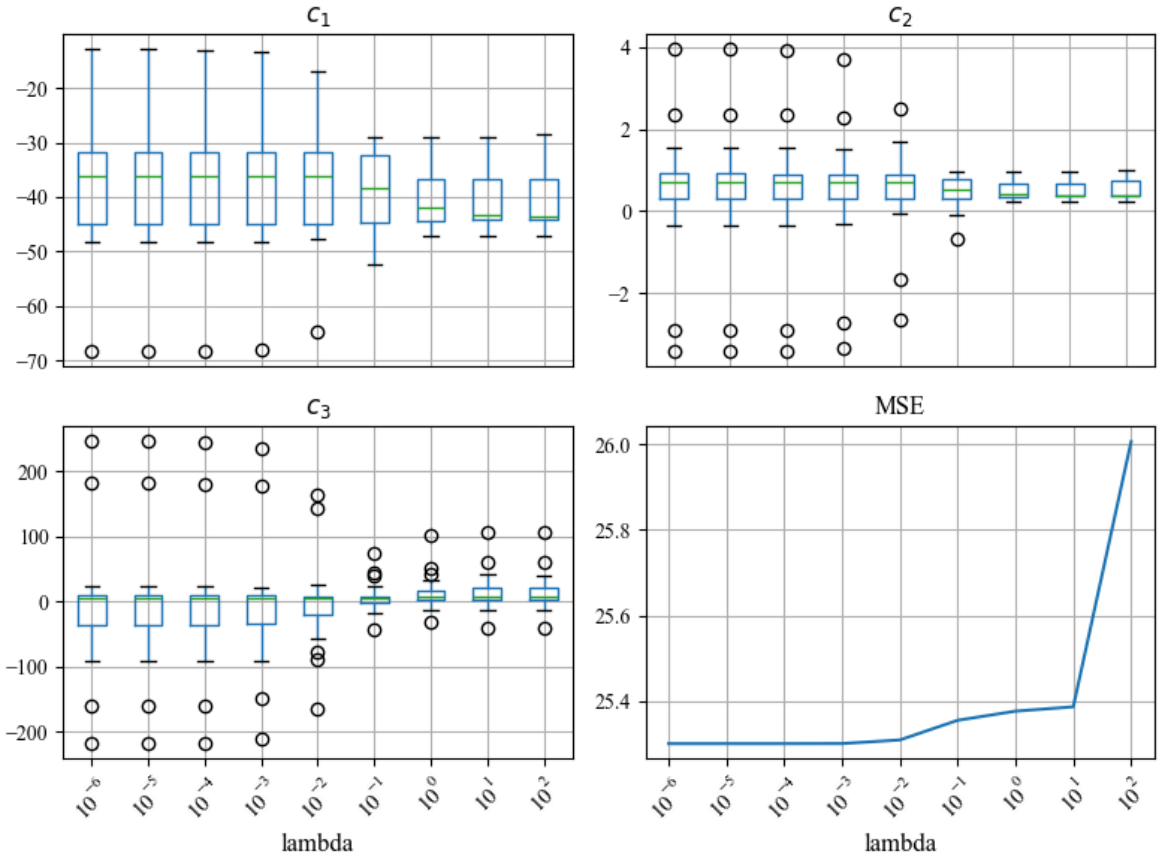
Select λ

In [22]:
```
op_range = {"x1": [0, 10], "x2": [0, 100]}
```

In [23]:
```
lambda_list = [10**(-6), 10**(-5),10**(-4), 10**(-3), 0.01, 0.1, 1, 10, 100]

coef_dict ={}
for feature in model_coef.index:
    coef_dict[feature] = pd.DataFrame()

for interval in range(20):
    start_index = interval * length_per_interval
    end_index = (interval + 1) * length_per_interval
    data_this_interval = data.query(f"{start_index}<=index<{end_index}")

    mse_list = []

    for lambda_ in lambda_list:

        X = data_this_interval[features]

        y = data_this_interval[["y"]].squeeze()

        penalty_matrix_this_interval = penalty_matrix.copy()
        penalty_for_intercept = 1
        for i in range(len(features)):
            feature = features[i]
            q3, q1 = np.percentile(data_this_interval[feature], [75 ,25])
            op_range_feature = op_range[feature]
            if len(op_range_feature) == 2:
                # little variation -> high penalty -> less flexibility
                penalty_for_variation = (op_range_feature[1]-op_range_feature[0]
                penalty_matrix_this_interval[i] = penalty_matrix[i] * (penalty_f

                penalty_for_intercept = penalty_for_intercept * (abs(q1))

        reg = TransferLinearRegression(lambda_, penalty_matrix=penalty_matrix_th

        y_pred = reg.predict(X)

        mse_list.append(mean_squared_error(y, y_pred))
        for feature, coef in zip(model_coef.index, reg.coef_):
            coef_dict[feature].loc[interval, lambda_] = coef
```

In [24]:
```
fig, axes = plt.subplots(2, 2, figsize=(8,6))

ax = axes[0, 0]
coef_dict["x1"].boxplot(ax=ax)
ax.set_title(f"$c_1$")
ax.tick_params(bottom=False, labelbottom=False)

ax = axes[0, 1]
coef_dict["x2"].boxplot(ax=ax)
ax.set_title(f"$c_2$")
ax.tick_params(bottom=False, labelbottom=False)

ax = axes[1, 0]
coef_dict["intercept"].boxplot(ax=ax)
ax.set_title(f"$c_3$")
ax.set_xlabel("lambda", fontsize=12)

ax = axes[1, 1]
ax.plot(lambda_list, mse_list)
```

```
ax.set_title(f"MSE")
ax.set_xscale("symlog", linthresh=1e-6)
ax.set_xlabel("lambda", fontsize=12)
ax.tick_params(axis='x', which='major', labelsize=11, rotation=45)
ax.grid()

axes[1, 0].set_xticklabels(axes[1, 1].get_xticklabels()[1:], fontsize=11, rotati
plt.tight_layout()
plt.show()
```



TLR model fitting

In [25]:
```
lambda_ = 10

reg = TransferLinearRegression(lambda_)
base_coefficients = model_coef.coefficients
fitting_results_TLR = pd.DataFrame()

for interval in range(0, num_interval):

    start_index = interval * length_per_interval
    end_index = (interval + 1) * length_per_interval
    data_this_interval = data.query(f"{start_index}<=index<{end_index}")

    X = data_this_interval[features]
    y = data_this_interval[["y"]].squeeze()

    # Determine penalty matrix based on the variation (Interquartile Range) of t
    # If variation is small -> the coefficient of the feature can't be well dete
    # -> need to set a high penalty matrix to avoid adaptation on that feature

    penalty_matrix_this_interval = penalty_matrix.copy()
```

```
    penalty_for_intercept = 1
    for i in range(len(features)):
        feature = features[i]
        q3, q1 = np.percentile(data_this_interval[feature], [75 ,25])
        op_range_feature = op_range[feature]
        if len(op_range_feature) == 2:
            # little variation -> high penalty -> less flexibility
            penalty_for_variation = (op_range_feature[1]-op_range_feature[0]) /
            penalty_matrix_this_interval[i] = penalty_matrix[i] * (penalty_for_v

            penalty_for_intercept = penalty_for_intercept * (abs(q1))

    reg = TransferLinearRegression(lambda_, penalty_matrix=penalty_matrix_this_i

    y_pred = reg.predict(X)

    fitting_results_TLR.loc[interval, "base_coef"] = str(dict(round(base_coeffic
    fitting_results_TLR.loc[interval, "R2"] = r2_score(y, y_pred)
    fitting_results_TLR.loc[interval, "MSE"] = mean_squared_error(y, y_pred)
    fitting_results_TLR.loc[interval, "penalty_matrix"] = str([round(num, 1) for


    base_coefficients = reg.coef_


    for i in range(len(model_coef.index)):
        fitting_results_TLR.loc[interval, model_coef.index[i]] = reg.coef_.iloc[
        fitting_results_TLR.loc[interval, f"c{i+1}"] = data_this_interval[f"c{i+

fitting_results_TLR.eval(f"y_ref_estimated = x1 * {x1_ref} + x2 * {x2_ref} + int
fitting_results_TLR.eval(f"y_ref_true = c1 * {x1_ref} + c2 * {x2_ref} + c3", inp
```

In [26]:
```
fitting_results_TLR.head()
```

Out[26]:

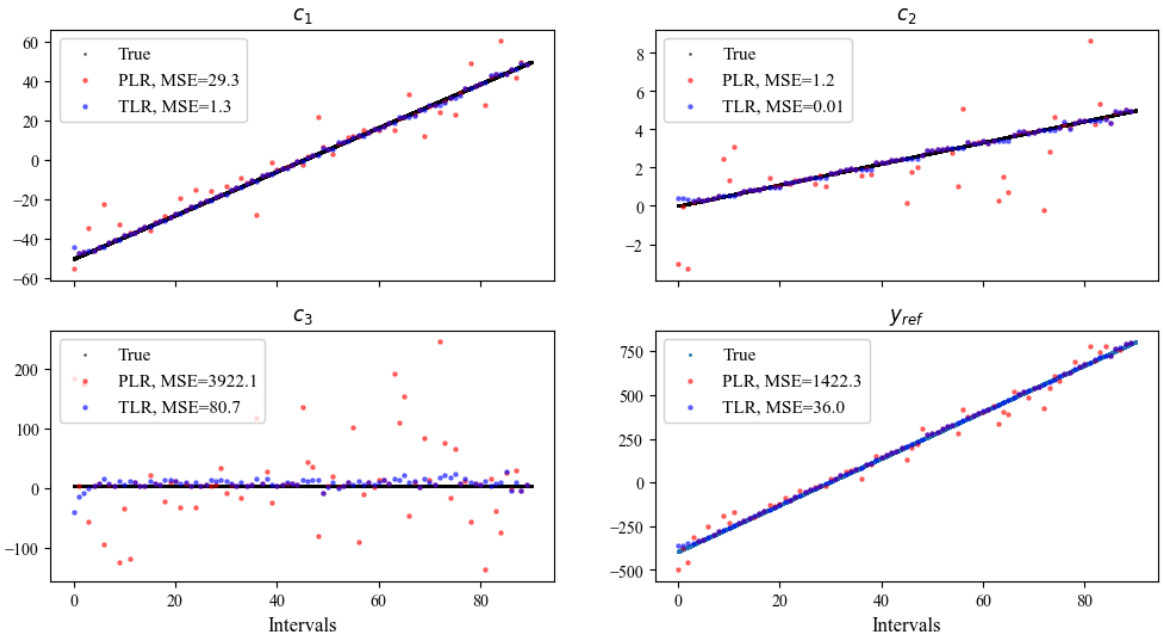
	base_coef	R2	MSE	penalty_matrix	x1	c1	x2
0	{'x1': -43.969875, 'x2': 0.389758, 'intercept'...	0.073831	149.137514	[2.5, 44.8, 0.1]	-44.208180	-49.449939	0.380575
1	{'x1': -44.20818, 'x2': 0.380575, 'intercept':...	0.953520	156.833579	[0.1, 42.5, 0.1]	-47.074203	-48.338704	0.382616
2	{'x1': -47.074203, 'x2': 0.382616, 'intercept'...	0.991861	155.208051	[0.0, 33.2, 0.1]	-46.293341	-47.227470	0.368829
3	{'x1': -46.293341, 'x2': 0.368829, 'intercept'...	0.139230	100.096331	[2.2, 2.0, 0.1]	-46.264180	-46.116235	0.254538
4	{'x1': -46.26418, 'x2': 0.254538, 'intercept':...	0.971160	75.716140	[0.1, 2.3, 0.1]	-46.130168	-45.005001	0.364032

Results

In [27]:

```
fig, axes = plt.subplots(2, 2, figsize=(12,6), sharex=True)

feature = "x1"
axes[0, 0].set_title(f"$c_1$")
axes[0, 0].scatter(data.index/length_per_interval, c1, label="True", alpha= 0.5,
axes[0, 0].scatter(fitting_results.index, fitting_results[feature], label=f"PLR,
axes[0, 0].scatter(fitting_results_TLR.index, fitting_results_TLR[feature], labe
axes[0, 0].legend(loc="upper left", fontsize=11)

feature = "x2"
axes[0, 1].set_title(f"$c_2$")
axes[0, 1].scatter(data.index/length_per_interval, c2, label="True", alpha= 0.5,
axes[0, 1].scatter(fitting_results.index, fitting_results[feature], label=f"PLR,
axes[0, 1].scatter(fitting_results_TLR.index, fitting_results_TLR[feature], labe
axes[0, 1].legend(loc="upper left", fontsize=11)

feature = "intercept"
axes[1, 0].set_title(f"$c_3$")
axes[1, 0].scatter(data.index/length_per_interval, c3, label="True", alpha= 0.5,
axes[1, 0].scatter(fitting_results.index, fitting_results[feature], label=f"PLR,
axes[1, 0].scatter(fitting_results_TLR.index, fitting_results_TLR[feature], labe
axes[1, 0].legend(loc="upper left", fontsize=11)
```

```
y_ref_true = c1 * x1_ref + c2 * x2_ref + c3
axes[1, 1].set_title(r"$y_{ref}$")
axes[1, 1].scatter(data.index/length_per_interval, y_ref_true, label="True", s=1
axes[1, 1].scatter(fitting_results.index, fitting_results.y_ref_estimated, label
axes[1, 1].scatter(fitting_results_TLR.index, fitting_results_TLR.y_ref_estimate
axes[1, 1].legend(loc="upper left", fontsize=11)

axes[1, 0].set_xlabel('Intervals', fontsize=12)
axes[1, 1].set_xlabel('Intervals', fontsize=12)
plt.show()
```



In [ ]: