In [1]:

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


from sklearn.preprocessing import scale, StandardScaler
from sklearn import model_selection
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error

import itertools
from itertools import combinations

import mlxtend
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
```

Problem a

In [2]:

```
# normal distribution mean and standard deviation
mu = 0
sigma = 1

# X: 100 random samples from normal distribution N(mu = 0, std = 1)
x = np.random.normal(mu, sigma, 100)

# noise: 100 random samples from normal distribution N(mu = 0, std = 1)
noise = np.random.normal(mu, sigma, 100)
```

Problem b

In [3]:

```
# Response: y = 1 + 2*x + 3*x^2 + 4*x^3 + noise
y = 1 + 2*x + 3*x**2 + 4*x**3 + noise
```

Problem c

In [4]:

```
# Make dataset of desired predictors
predictors = {
            'x': x,
            'x^2': x**2,
            'x^3': x**3,
            'x^4': x**4,
            'x^5': x**5,
            'x^6': x**6,
            'x^7': x**7,
            'x^8': x**8,
            'x^9': x**9,
            'x^10': x**10,
           }
data = pd.DataFrame(predictors)
```

Cp Scoring

In [5]:

```
# Calculate sigma_hat_squared
linreg = LinearRegression()
linreg.fit(data, y)
y_pred = linreg.predict(data)
sigma_hat_squared = np.sum((y - y_pred)**2)/(y.shape[0] - data.shape[1] - 1)


# --------------- BEST SUBSET SELECTION (USING Cp SCORING) -------------
n_features = data.shape[1]
max_size = 10
num_observations = len(y)

subsets = (combinations(range(n_features), k + 1) for k in range(min(n_features, max_size)))

best_size_subset = []
best_score_list = []
best_coeff_list = []
best_intercept_list = []
for subsets_k in subsets:  # for each list of subsets of the same size
    best_score = np.inf
    best_subset = None
```

```
    best_coeffs = None
    best_intercept = None
    for subset in subsets_k: # for each subset of size k
        LinRegr_classifier = LinearRegression()
        LinRegr_classifier.fit(data.iloc[:, list(subset)], y)

        curr_interc = LinRegr_classifier.intercept_
        curr_coeffs = LinRegr_classifier.coef_

        # Calculate Cp Score
        curr_num_predictors = len(list(subset))
        curr_y_pred = LinRegr_classifier.predict(data.iloc[:, list(subset)])
        current_RSS = np.sum((y - curr_y_pred)**2)

        Cp_score = (current_RSS + 2*curr_num_predictors*sigma_hat_squared)/num_observations
        if Cp_score < best_score:
            best_score, best_subset, best_coeffs, best_intercept = 
                                Cp_score, subset, curr_coeffs, curr_interc

    # to compare subsets of different sizes we must use CV
    # first store the best subset of each size
    best_size_subset.append(best_subset)
    best_score_list.append(best_score)
    best_coeff_list.append(best_coeffs)
    best_intercept_list.append(best_intercept)

# compare best subsets of each size and choose model with lowest Cp score
best_score = np.inf
best_subset = None
best_coeffs = None
best_intercept = None
for subset, score, coeffs, intercept in zip(best_size_subset,
                                            best_score_list,
                                            best_coeff_list,
                                            best_intercept_list):
    if score < best_score:
        best_score, best_subset, best_coeffs, best_intercept = 
                                score, subset, coeffs, intercept


# best_subset, best_score, best_size_subset, best_score_list
print('The best subset is determined to be ' + repr(best_subset))
print('The best Cp score is determined to be ' + repr(best_score))
print('The Intercept is ' + repr(best_intercept))
print('The coefficients are determined to be ' + repr(best_coeffs))
```

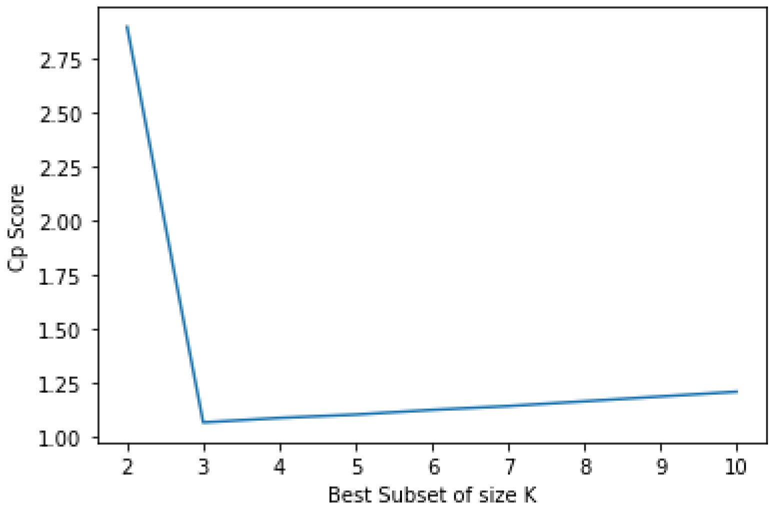
```
The best subset is determined to be (0, 1, 2)
The best Cp score is determined to be 1.0698153750635868
The Intercept is 1.0507720229291904
The coefficients are determined to be array([2.27720155, 2.97504281, 3.96782749])
```

In [6]:

```
plt.plot(np.linspace(2,10,9), best_score_list[1:10])
plt.xlabel('Best Subset of size K')
plt.ylabel('Cp Score')
```

Out[6]:

```
Text(0, 0.5, 'Cp Score')
```



BIC Scoring

In [7]:

```
# Calculate sigma_hat_squared
linreg = LinearRegression()
linreg.fit(data, y)
y_pred = linreg.predict(data)
sigma_hat_squared = np.sum((y - y_pred)**2)/(y.shape[0] - data.shape[1] - 1)



# --------------- BEST SUBSET SELECTION (USING BIC SCORING) -------------
n_features = data.shape[1]
max_size = 10
num_observations = len(y)
```

```
subsets = (combinations(range(n_features), k + 1) for k in range(min(n_features, max_size)))

best_size_subset = []
best_score_list = []
best_coeff_list = []
best_intercept_list = []
for subsets_k in subsets:  # for each list of subsets of the same size
    best_score = np.inf
    best_subset = None
    best_coeffs = None
    best_intercept = None
    for subset in subsets_k: # for each subset of size k
        LinRegr_classifier = LinearRegression()
        LinRegr_classifier.fit(data.iloc[:, list(subset)], y)

        curr_interc = LinRegr_classifier.intercept_
        curr_coeffs = LinRegr_classifier.coef_

        # Calculate BIC Score
        curr_num_predictors = len(list(subset))
        curr_y_pred = LinRegr_classifier.predict(data.iloc[:, list(subset)])
        current_RSS = np.sum((y - curr_y_pred)**2)
        BIC_score = (current_RSS + np.log(num_observations)*curr_num_predictors*sigma_hat_squared)/
                                            (num_observations*sigma_hat_squared)

        if BIC_score < best_score:
            best_score, best_subset, best_coeffs, best_intercept =
                                        BIC_score, subset, curr_coeffs, curr_interc

    # to compare subsets of different sizes we must use CV
    # first store the best subset of each size
    best_size_subset.append(best_subset)
    best_score_list.append(best_score)
    best_coeff_list.append(best_coeffs)
    best_intercept_list.append(best_intercept)

# compare best subsets of each size and choose model with lowest Cp score
best_score = np.inf
best_subset = None
best_coeffs = None
best_intercept = None
for subset, score, coeffs, intercept in zip(best_size_subset,
                                            best_score_list,
                                            best_coeff_list,
```

```
                                                         best_intercept_list):
        if score < best_score:
            best_score, best_subset, best_coeffs, best_intercept =
                                                         score, subset, coeffs, intercept


# best_subset, best_score, best_size_subset, best_score_list
print('The best subset is determined to be ' + repr(best_subset))
print('The best BIC score is determined to be ' + repr(best_score))
print('The Intercept is ' + repr(best_intercept))
print('The coefficients are determined to be ' + repr(best_coeffs))
```

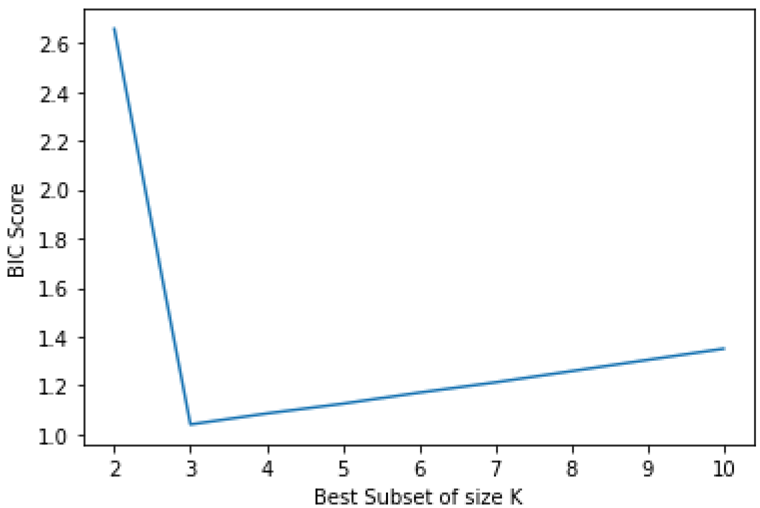
```
The best subset is determined to be (0, 1, 2)
The best BIC score is determined to be 1.040514984077876
The Intercept is 1.0507720229291904
The coefficients are determined to be array([2.27720155, 2.97504281, 3.96782749])
```

In [8]:

```
plt.plot(np.linspace(2,10,9), best_score_list[1:10])
plt.xlabel('Best Subset of size K')
plt.ylabel('BIC Score')
```

Out[8]:

```
Text(0, 0.5, 'BIC Score')
```



Adjusted R^2 Scoring

In [ ]:
```
y_mean = np.mean(y)
TSS = np.sum((y - y_mean)**2)


# --------------- BEST SUBSET SELECTION (USING Adjusted R^2 SCORING) -------------
n_features = data.shape[1]
max_size = 10
num_obs = len(y)

subsets = (combinations(range(n_features), k + 1) for k in range(min(n_features, max_size)))

best_size_subset = []
best_score_list = []
best_coeff_list = []
best_intercept_list = []
for subsets_k in subsets:  # for each list of subsets of the same size
    best_score = -np.inf
    best_subset = None
    best_coeffs = None
    best_intercept = None
    for subset in subsets_k: # for each subset of size k
        LinRegr_classifier = LinearRegression()
        LinRegr_classifier.fit(data.iloc[:, list(subset)], y)

        curr_interc = LinRegr_classifier.intercept_
        curr_coeffs = LinRegr_classifier.coef_

        # Calculate Adjusted R^2 Score
        curr_num_predictors = len(list(subset))
        curr_y_pred = LinRegr_classifier.predict(data.iloc[:, list(subset)])
        current_RSS = np.sum((y - curr_y_pred)**2)
        Adj_R_Sq_score = 1 - (current_RSS/(num_obs-curr_num_predictors-1))/(TSS/(num_obs-1))

        if Adj_R_Sq_score > best_score:
            best_score, best_subset, best_coeffs, best_intercept = 
                                    Adj_R_Sq_score, subset, curr_coeffs, curr_interc

    # to compare subsets of different sizes we must use CV
    # first store the best subset of each size
    best_size_subset.append(best_subset)
    best_score_list.append(best_score)
    best_coeff_list.append(best_coeffs)
    best_intercept_list.append(best_intercept)

# compare best subsets of each size and choose model with lowest Cp score
```

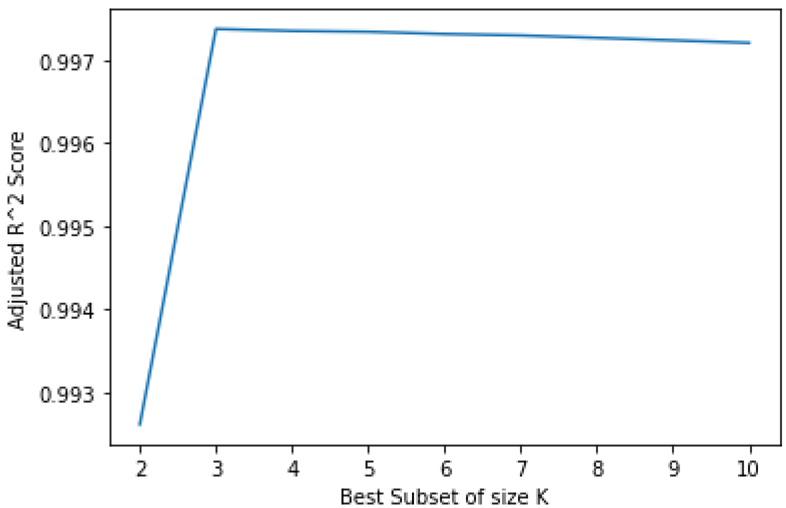
```
best_score = -np.inf
best_subset = None
best_coeffs = None
best_intercept = None
for subset, score, coeffs, intercept in zip(best_size_subset,
                                            best_score_list,
                                            best_coeff_list,
                                            best_intercept_list):
    if score > best_score:
        best_score, best_subset, best_coeffs, best_intercept =
                                            score, subset, coeffs, intercept


# best_subset, best_score, best_size_subset, best_score_list
print('The best subset is determined to be ' + repr(best_subset))
print('The best Adjusted R^2 score is determined to be ' + repr(best_score))
print('The Intercept is ' + repr(best_intercept))
print('The coefficients are determined to be ' + repr(best_coeffs))
```

In [10]:

```
plt.plot(np.linspace(2,10,9), best_score_list[1:10])
plt.xlabel('Best Subset of size K')
plt.ylabel('Adjusted R^2 Score')
```

Out[10]:

```
Text(0, 0.5, 'Adjusted R^2 Score')
```



Problem d

Forward Sequential Selection

In [11]:

```
LinRegr_classifier = LinearRegression()
sfs = SFS(LinRegr_classifier,
          k_features=10,
          forward=True,
          floating=False,
          scoring='neg_mean_squared_error',
          cv=10)

sfs = sfs.fit(data, y)
sfs.subsets_
```

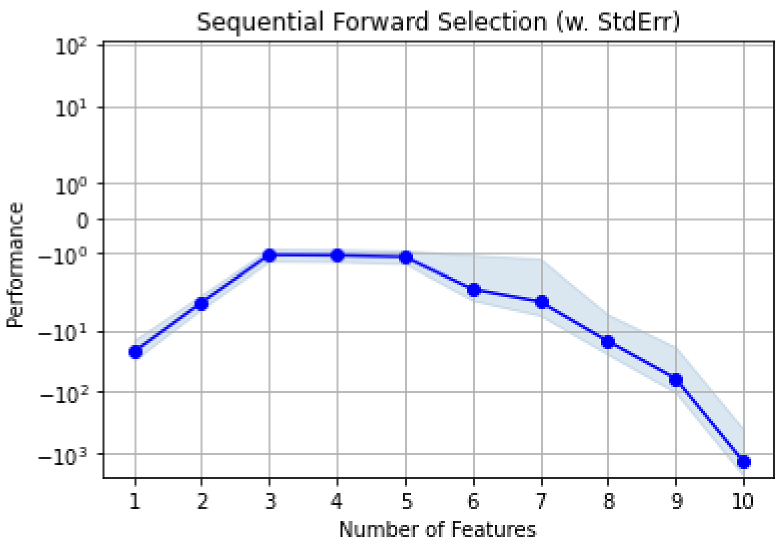
Out[11]:

```
{1: {'feature_idx': (2,),
  'cv_scores': array([-10.38683694,  -7.07054494, -14.96058696, -91.86858365,
         -27.92270419, -30.03314831, -12.06552408,  -4.79907348,
         -15.06003096, -11.16832237]),
  'avg_score': -22.533535588375493,
  'feature_names': ('x^3',)},
 2: {'feature_idx': (1, 2),
  'cv_scores': array([ -2.51428293,  -2.75693949,  -2.85836064,  -3.63104781,
          -3.28933565, -11.52895664,  -0.84929777,  -3.88200934,
          -1.724788  ,  -2.81319539]),
  'avg_score': -3.584821365461052,
  'feature_names': ('x^2', 'x^3')},
 3: {'feature_idx': (0, 1, 2),
  'cv_scores': array([-1.03626395, -0.76614438, -0.86824782, -1.84689057, -0.93916351,
         -1.06509699, -0.30513156, -2.1670272 , -0.4671812 , -1.35455305]),
  'avg_score': -1.0815700227998917,
  'feature_names': ('x', 'x^2', 'x^3')},
 4: {'feature_idx': (0, 1, 2, 4),
  'cv_scores': array([-1.0349656 , -0.76547127, -0.86903353, -1.9115353 , -0.93933611,
         -1.09347467, -0.32170475, -2.17376917, -0.47136474, -1.35531045]),
  'avg_score': -1.0935965598790287,
  'feature_names': ('x', 'x^2', 'x^3', 'x^5')},
 5: {'feature_idx': (0, 1, 2, 4, 6),
  'cv_scores': array([-1.03860068, -0.76774345, -0.89019082, -1.97386464, -0.9463927 ,
         -1.30146509, -0.34854737, -2.19620565, -0.47256175, -1.45611702]),
  'avg_score': -1.1391689178431006,
  'feature_names': ('x', 'x^2', 'x^3', 'x^5', 'x^7')},
```

```
 6: {'feature_idx': (0, 1, 2, 3, 4, 6),
  'cv_scores': array([ -1.00974422,  -0.74208992,  -0.85077681,  -1.7933438 ,
          -0.9524472 , -11.96715318,  -0.32480635,  -2.22518701,
          -0.54193228,  -1.42794963]),
  'avg_score': -2.183543041207798,
  'feature_names': ('x', 'x^2', 'x^3', 'x^4', 'x^5', 'x^7')},
 7: {'feature_idx': (0, 1, 2, 3, 4, 5, 6),
  'cv_scores': array([ -1.02922195,  -0.74714956,  -0.8569936 ,  -2.49264849,
          -1.05923674, -23.9059594 ,  -0.32493197,  -2.24313899,
          -0.54195837,  -1.48309907]),
  'avg_score': -3.4684338144061386,
  'feature_names': ('x', 'x^2', 'x^3', 'x^4', 'x^5', 'x^6', 'x^7')},
 8: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7),
  'cv_scores': array([ -1.04881372,  -0.8258835 ,  -0.86801568, -82.72377229,
          -1.05149397, -59.20074582,  -0.33897933,  -2.34178469,
          -0.5188395 ,  -1.45552043]),
  'avg_score': -15.037384894665502,
  'feature_names': ('x', 'x^2', 'x^3', 'x^4', 'x^5', 'x^6', 'x^7', 'x^8')},
 9: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8),
  'cv_scores': array([  -1.12371303,   -0.85306668,   -0.8501749 , -209.83047972,
           -1.06438893, -382.26540742,   -0.38476699,   -2.38356079,
           -0.52717677,   -1.55620798]),
  'avg_score': -60.083894319001594,
  'feature_names': ('x',
   'x^2',
   'x^3',
   'x^4',
   'x^5',
   'x^6',
   'x^7',
   'x^8',
   'x^9')},
 10: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8, 9),
  'cv_scores': array([-1.20758910e+00, -8.89022753e-01, -8.57041958e-01, -4.65588060e+03,
         -1.07076765e+00, -8.48545773e+03, -3.83948558e-01, -2.46527221e+00,
         -5.33335349e-01, -1.70571541e+00]),
  'avg_score': -1315.045101752545,
  'feature_names': ('x',
   'x^2',
   'x^3',
   'x^4',
   'x^5',
   'x^6',
   'x^7',
   'x^8',
```

```
 'x^9',
 'x^10')}}
```

In [12]:
```
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')

plt.title('Sequential Forward Selection (w. StdErr)')
plt.yscale('symlog')
plt.grid()
plt.show()
```



Backward Sequential Selection

In [15]:
```
LinRegr_classifier = LinearRegression()
sfs = SFS(LinRegr_classifier,
          k_features=1,
          forward=False,
          floating=False,
          scoring='neg_mean_squared_error',
          cv=10)

sfs = sfs.fit(data, y)
sfs.subsets_
```

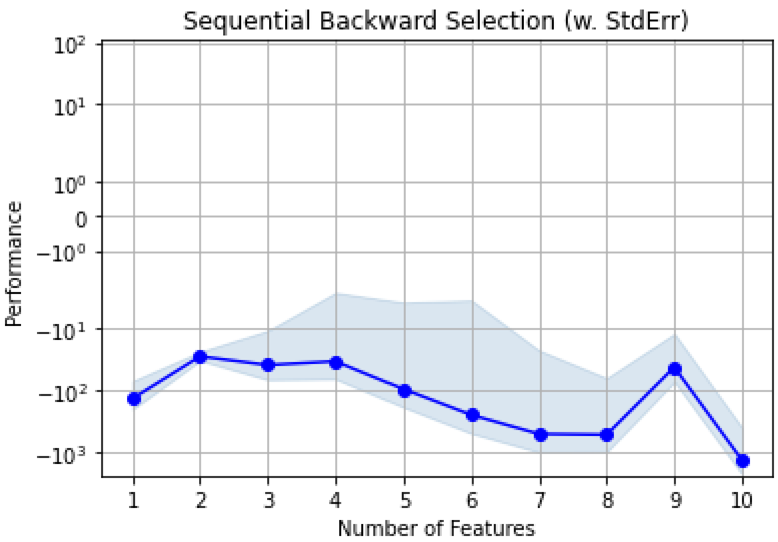
```
{10: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8, 9),
```

```
Out[15]:   'cv_scores': array([-1.20758910e+00, -8.89022753e-01, -8.57041958e-01, -4.65588060e+03,
                 -1.07076765e+00, -8.48545773e+03, -3.83948558e-01, -2.46527221e+00,
                 -5.33335349e-01, -1.70571541e+00]),
            'avg_score': -1315.045101752545,
            'feature_names': ('x',
             'x^2',
             'x^3',
             'x^4',
             'x^5',
             'x^6',
             'x^7',
             'x^8',
             'x^9',
             'x^10')},
           9: {'feature_idx': (0, 1, 3, 4, 5, 6, 7, 8, 9),
            'cv_scores': array([  -1.22826999,   -1.07600249,   -0.6119963 , -285.45656861,
                   -0.86861302, -120.75445255,   -0.48229377,   -2.027363  ,
                   -0.54007906,   -1.90653826]),
            'avg_score': -41.495217704390186,
            'feature_names': ('x',
             'x^2',
             'x^4',
             'x^5',
             'x^6',
             'x^7',
             'x^8',
             'x^9',
             'x^10')},
           8: {'feature_idx': (0, 1, 4, 5, 6, 7, 8, 9),
            'cv_scores': array([-1.29174696e+00, -1.03074805e+00, -7.97307363e-01, -5.82246820e+02,
                 -8.43682779e-01, -4.53157379e+03, -5.34197674e-01, -1.98154746e+00,
                 -5.69993333e-01, -1.85416389e+00]),
            'avg_score': -512.2723992456882,
            'feature_names': ('x', 'x^2', 'x^5', 'x^6', 'x^7', 'x^8', 'x^9', 'x^10')},
           7: {'feature_idx': (0, 1, 4, 5, 6, 7, 8),
            'cv_scores': array([-1.50563807e+00, -1.16482603e+00, -1.55023487e+00, -1.92757252e+02,
                 -8.15173027e-01, -4.77723321e+03, -6.70195467e-01, -2.17464975e+00,
                 -4.12701782e-01, -1.55609860e+00]),
            'avg_score': -497.98399844233415,
            'feature_names': ('x', 'x^2', 'x^5', 'x^6', 'x^7', 'x^8', 'x^9')},
           6: {'feature_idx': (0, 1, 4, 5, 6, 8),
            'cv_scores': array([-1.19810454e+00, -1.11134497e+00, -1.04988312e+00, -2.17641681e+01,
                 -1.17998830e+00, -2.41622573e+03, -6.63949613e-01, -2.30952260e+00,
                 -5.50605887e-01, -1.11837556e+00]),
            'avg_score': -244.71716688146245,
```

```
 'feature_names': ('x', 'x^2', 'x^5', 'x^6', 'x^7', 'x^9')},
5: {'feature_idx': (0, 1, 4, 5, 6),
 'cv_scores': array([  -2.26337076,   -1.56815666,   -4.31108752,  -17.5935654 ,
          -1.14122721, -904.33129804,   -1.58543297,   -3.03744977,
          -0.91734191,   -1.16428973]),
 'avg_score': -93.79132199833931,
 'feature_names': ('x', 'x^2', 'x^5', 'x^6', 'x^7')},
4: {'feature_idx': (0, 1, 4, 6),
 'cv_scores': array([  -1.74210545,   -1.69877003,   -2.62726339,   -7.95162899,
          -1.61343557, -310.23799687,   -1.68204831,   -3.64361511,
          -1.35745995,   -0.99560083]),
 'avg_score': -33.35499244941956,
 'feature_names': ('x', 'x^2', 'x^5', 'x^7')},
3: {'feature_idx': (0, 1, 4),
 'cv_scores': array([  -6.78484896,   -7.60641904,  -10.02372535,  -28.6710969 ,
         -11.9357874 , -286.50417498,   -9.83002202,   -8.1910553 ,
          -9.03612605,   -4.28724581]),
 'avg_score': -38.28705018119772,
 'feature_names': ('x', 'x^2', 'x^5')},
2: {'feature_idx': (0, 4),
 'cv_scores': array([-21.96847743, -14.6266542 , -41.70206449, -48.78096159,
        -36.80028691, -44.21299072, -32.1009667 , -11.57178202,
        -14.4554826 , -11.49726289]),
 'avg_score': -27.77169295735037,
 'feature_names': ('x', 'x^5')},
1: {'feature_idx': (4,),
 'cv_scores': array([ -94.74707938,  -18.3081568 , -114.50733765,  -61.64772281,
        -129.07240819, -705.29408739, -108.16240927,  -26.80256095,
         -35.76947362,  -44.52045795]),
 'avg_score': -133.8831694031052,
 'feature_names': ('x^5',)}}
```

In [16]:

```
fig = plot_sfs(sfs.get_metric_dict(), kind='std_err')
plt.title('Sequential Backward Selection (w. StdErr)')
plt.yscale('symlog')
plt.grid()
plt.show()
```



Problem e

In [ ]:

```
# Make dataset of desired predictors as numpy array
data_array = np.asarray(data)

# determine best value for lambda tuning parameter
possible_lambdas = np.linspace(0.01,10,1000)
cv_error_list = []
best_lambda = 0
best_score = np.inf
K = 5

for value in (possible_lambdas):  # Obtain test error rate for each value of Lambda
    kfold = KFold(n_splits=K, shuffle=True)

    sum_test_errors = 0
    for train_index, test_index in kfold.split(data_array):
    #      print("TRAIN:", train_index, "TEST:", test_index)
        X_train_curr, X_test_curr = data_array[train_index], data_array[test_index]
        y_train_curr, y_test_curr = y[train_index], y[test_index]

        lasso_clf = Lasso(alpha=value, tol=0.01, max_iter=10000)
        lasso_clf.fit(X_train_curr, y_train_curr)

        y_pred = lasso_clf.predict(X_test_curr)
```

```
            test_error = np.sum((y_test_curr - y_pred)**2)/len(y_test_curr)

            sum_test_errors = sum_test_errors + test_error

        current_test_error = sum_test_errors/K

        cv_error_list.append(current_test_error)

        if current_test_error < best_score:
            best_score = current_test_error
            best_lambda = value
```

In [18]:

```
print("The best lambda value is " + repr(best_lambda))

# Find test erro using best lambda value
Lasso_classifier = Lasso(alpha=best_lambda)
Lasso_classifier.fit(data, y)
y_pred = Lasso_classifier.predict(data)
test_error = np.sum((y - y_pred)**2)/len(y)

print("The testing error for the Lasso Regression is " + repr(test_error))
print("The Intercept is " + repr(Lasso_classifier.intercept_))
print("The Coefficients are " + repr(Lasso_classifier.coef_))
```

```
The best lambda value is 0.02
The testing error for the Lasso Regression is 0.9978872639287711
The Intercept is 1.0450097644243055
The Coefficients are array([ 2.24855517e+00,  2.95931602e+00,  3.97474038e+00,  4.43239410e-02,
       -0.00000000e+00, -1.16806938e-02, -1.01532763e-03, -6.25466780e-05,
        1.33872341e-04,  8.93535892e-05])
```

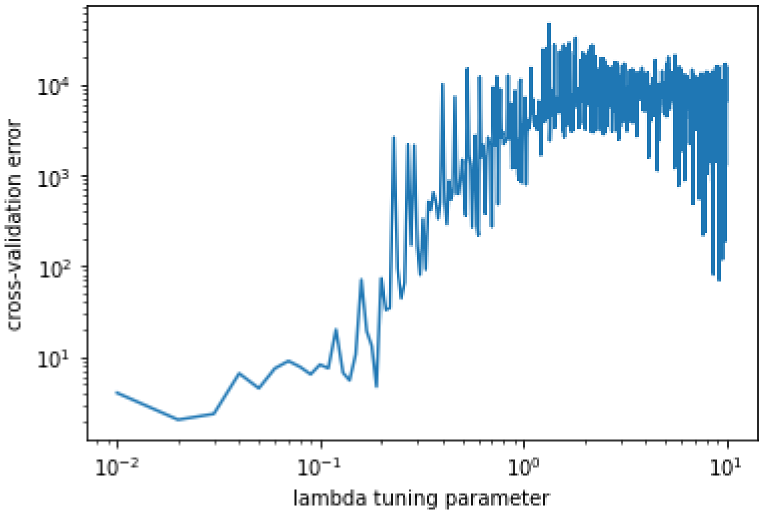
```
C:\Users\Adam Yang\anaconda3\envs\ml_homework\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: Converge
nceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the featu
res or consider increasing regularisation. Duality gap: 6.439e+01, tolerance: 3.942e+00
  model = cd_fast.enet_coordinate_descent(
```

In [19]:

```
plt.plot(possible_lambdas, cv_error_list)
plt.yscale('log')
plt.xscale('log')
plt.xlabel('lambda tuning parameter')
plt.ylabel('cross-validation error')
```

Out[19]:

```
Text(0, 0.5, 'cross-validation error')
```



Problem f

In [20]:

```
# Response: y = 1 + 7*x^7 + noise
y_new = 1 + 7*x**7 + noise
```

Best Subset (BIC)

In [21]:

```
# Calculate sigma_hat_squared
linreg = LinearRegression()
linreg.fit(data, y_new)
y_pred = linreg.predict(data)
sigma_hat_squared = np.sum((y_new - y_pred)**2)/(y_new.shape[0] - data.shape[1] - 1)



# --------------- BEST SUBSET SELECTION (USING BIC SCORING) -------------
n_features = data.shape[1]
max_size = 10
num_observations = len(y_new)

subsets = (combinations(range(n_features), k + 1) for k in range(min(n_features, max_size)))
```

```
best_size_subset = []
best_score_list = []
best_coeff_list = []
best_intercept_list = []
for subsets_k in subsets:  # for each list of subsets of the same size
    best_score = np.inf
    best_subset = None
    best_coeffs = None
    best_intercept = None
    for subset in subsets_k: # for each subset of size k
        LinRegr_classifier = LinearRegression()
        LinRegr_classifier.fit(data.iloc[:, list(subset)], y_new)

        curr_interc = LinRegr_classifier.intercept_
        curr_coeffs = LinRegr_classifier.coef_

        # Calculate BIC Score
        curr_num_predictors = len(list(subset))
        curr_y_pred = LinRegr_classifier.predict(data.iloc[:, list(subset)])
        current_RSS = np.sum((y_new - curr_y_pred)**2)
        BIC_score = (current_RSS + np.log(num_observations)*curr_num_predictors*sigma_hat_squared)/
                                                                (num_observations*sigma_hat_squared)

        if BIC_score < best_score:
            best_score, best_subset, best_coeffs, best_intercept =
                                            BIC_score, subset, curr_coeffs, curr_interc

    # to compare subsets of different sizes we must use CV
    # first store the best subset of each size
    best_size_subset.append(best_subset)
    best_score_list.append(best_score)
    best_coeff_list.append(best_coeffs)
    best_intercept_list.append(best_intercept)

# compare best subsets of each size and choose model with lowest Cp score
best_score = np.inf
best_subset = None
best_coeffs = None
best_intercept = None
for subset, score, coeffs, intercept in zip(best_size_subset,
                                            best_score_list,
                                            best_coeff_list,
                                            best_intercept_list):
    if score < best_score:
        best_score, best_subset, best_coeffs, best_intercept =
```

```
                                              score, subset, coeffs, intercept


# best_subset, best_score, best_size_subset, best_score_list
print('The best subset is determined to be ' + repr(best_subset))
print('The best BIC score is determined to be ' + repr(best_score))
print('The Intercept is ' + repr(best_intercept))
print('The coefficients are determined to be ' + repr(best_coeffs))
```

```
The best subset is determined to be (6,)
The best BIC score is determined to be 0.9840769684555608
The Intercept is 1.0098198202748847
The coefficients are determined to be array([7.00006324])
```

Lasso Regression

In [22]:

```
# Make dataset of desired predictors as numpy array
data_array = np.asarray(data)

# determine best value for lambda tuning parameter
possible_lambdas = np.linspace(0.01,10,1000)
cv_error_list = []
best_lambda = 0
best_score = np.inf
K = 5

for value in (possible_lambdas):  # Obtain test error rate for each value of Lambda
    kfold = KFold(n_splits=K, shuffle=True)

    sum_test_errors = 0
    for train_index, test_index in kfold.split(data_array):
    #      print("TRAIN:", train_index, "TEST:", test_index)
        X_train_curr, X_test_curr = data_array[train_index], data_array[test_index]
        y_train_curr, y_test_curr = y_new[train_index], y_new[test_index]

        lasso_clf = Lasso(alpha=value, tol=0.01, max_iter=10000)
        lasso_clf.fit(X_train_curr, y_train_curr)

        y_pred = lasso_clf.predict(X_test_curr)
        test_error = np.sum((y_test_curr - y_pred)**2)/len(y_test_curr)

        sum_test_errors = sum_test_errors + test_error
```

```
    current_test_error = sum_test_errors/K

    cv_error_list.append(current_test_error)

    if current_test_error < best_score:
        best_score = current_test_error
        best_lambda = value
```

In [23]:

```
print("The best lambda value is " + repr(best_lambda))

# Find test erro using best lambda value
Lasso_classifier = Lasso(alpha=best_lambda)
Lasso_classifier.fit(data, y_new)
y_pred = Lasso_classifier.predict(data)
test_error = np.sum((y_new - y_pred)**2)/len(y_new)

print("The testing error for the Lasso Regression is " + repr(test_error))
print("The Intercept is " + repr(Lasso_classifier.intercept_))
print("The Coefficients are " + repr(Lasso_classifier.coef_))
```

```
The best lambda value is 4.22
The testing error for the Lasso Regression is 37.88136360301897
The Intercept is 1.404923235979794
The Coefficients are array([-0.        , -0.        ,  0.        , -4.16825788,  5.8989894 ,
        2.27136505,  4.16372382, -0.12439829,  0.2978138 , -0.02629599])
```

In [ ]: