In [1]:

```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
from statsmodels.api import OLS
import sklearn.model_selection as skm
import sklearn.linear_model as skl
from sklearn.preprocessing import StandardScaler
from ISLP import load_data
from ISLP.models import ModelSpec as MS
from functools import partial
```

In [2]:

```
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from ISLP.models import \
    (Stepwise,
     sklearn_selected,
     sklearn_selection_path)
from l0bnb import fit_path
```

In [3]:

```
from sklearn.exceptions import ConvergenceWarning
import warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)
warnings.filterwarnings("ignore", category=UserWarning)
```

In [4]:

```
Hitters = load_data('Hitters')
np.isnan(Hitters['Salary']).sum()
```

Out[4]:

```
59
```

In [5]:

```
Hitters = Hitters.dropna()
Hitters.shape
```

Out[5]:

```
(263, 20)
```

In [6]:

```
def nCp(sigma2, estimator, X, Y):
    "Negative Cp statistic"
    n, p = X.shape
    Yhat = estimator.predict(X)
    RSS = np.sum((Y - Yhat)**2)
    return -(RSS + 2 * p * sigma2) / n
```

In [7]:

```
design = MS(Hitters.columns.drop('Salary')).fit(Hitters)
Y = np.array(Hitters['Salary'])
X = design.transform(Hitters)
sigma2 = OLS(Y, X).fit().scale
```

In [8]:

```
neg_Cp = partial(nCp, sigma2)
```

In [9]:

```
strategy = Stepwise.first_peak(design,
                               direction='forward',
                               max_terms=len(design.terms))
```

In [10]:

```
hitters_MSE = sklearn_selected(OLS,
                               strategy)
hitters_MSE.fit(Hitters, Y)
hitters_MSE.selected_state_
```

Out[10]:

```
('Assists',
 'AtBat',
 'CAtBat',
 'CHits',
 'CHmRun',
 'CRBI',
 'CRuns',
 'CWalks',
 'Division',
 'Errors',
 'Hits',
 'HmRun',
 'League',
 'NewLeague',
 'PutOuts',
 'RBI',
 'Runs',
 'Walks',
 'Years')
```

In [11]:

```
hitters_Cp = sklearn_selected(OLS,
                              strategy,
                              scoring=neg_Cp)
hitters_Cp.fit(Hitters, Y)
hitters_Cp.selected_state_
```

Out[11]:

```
('Assists',
 'AtBat',
 'CAtBat',
 'CRBI',
 'CRuns',
 'CWalks',
 'Division',
 'Hits',
 'PutOuts',
 'Walks')
```

In [12]:

```
strategy = Stepwise.fixed_steps(design,
                                len(design.terms),
                                direction='forward')
full_path = sklearn_selection_path(OLS, strategy)
```

In [13]:

```
full_path.fit(Hitters, Y)
Yhat_in = full_path.predict(Hitters)
```

```
Yhat_in.shape
```

Out[13]:

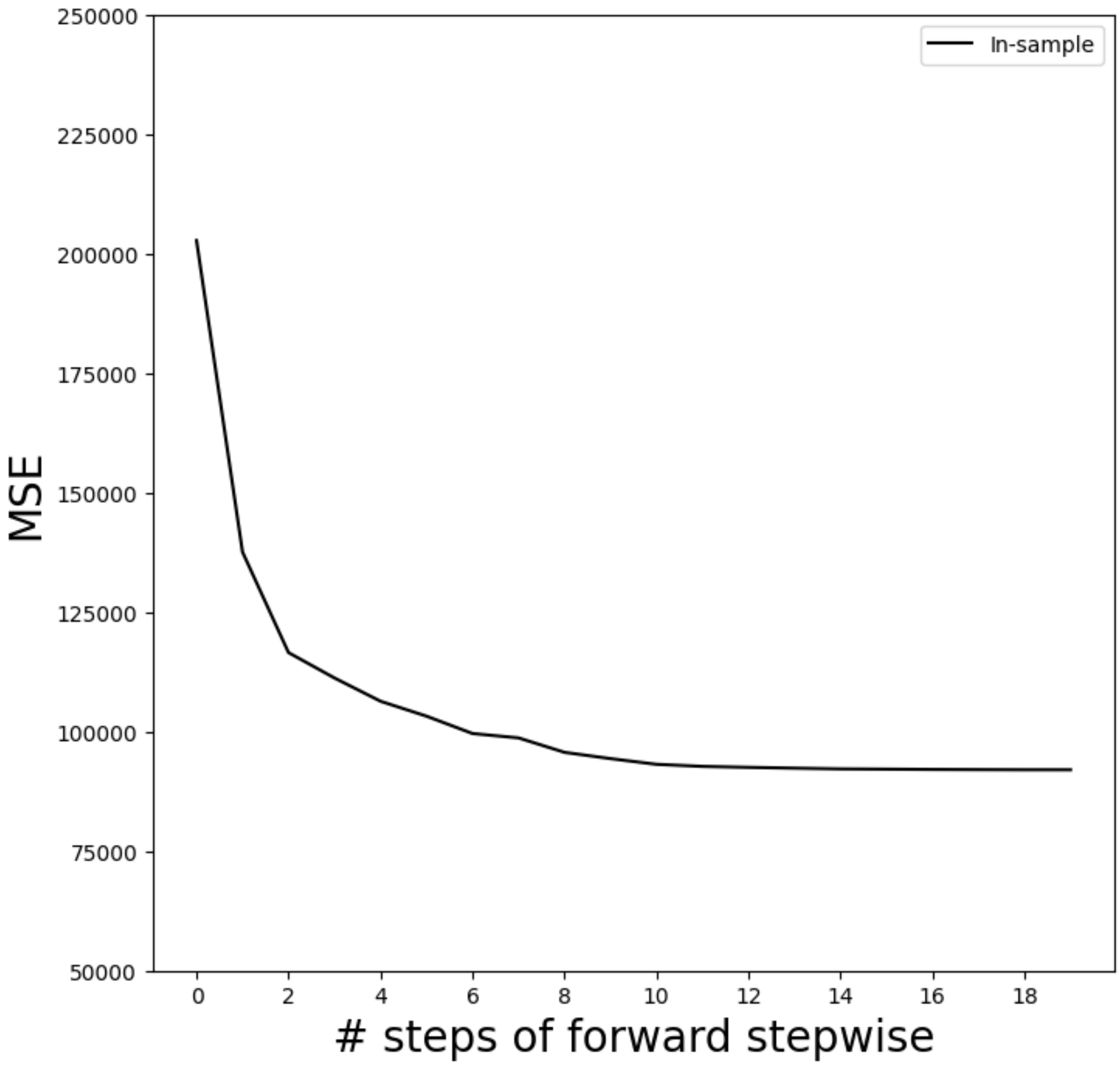
```
(263, 20)
```

In [14]:

```
mse_fig, ax = subplots(figsize=(8, 8))
insample_mse = ((Yhat_in - Y[:, None]) ** 2).mean(0)
n_steps = insample_mse.shape[0]
ax.plot(np.arange(n_steps),
        insample_mse,
        'k',  # color black
        label='In-sample')
ax.set_ylabel('MSE',
              fontsize=20)
ax.set_xlabel('# steps of forward stepwise',
              fontsize=20)
ax.set_xticks(np.arange(n_steps)[::2])
ax.legend()
ax.set_ylim([50000, 250000])
```

Out[14]:

```
(50000.0, 250000.0)
```



In [15]:

```
K = 5
kfold = skm.KFold(K,
                  random_state=0,
                  shuffle=True)
Yhat_cv = skm.cross_val_predict(full_path,
                                Hitters,
                                Y,
                                cv=kfold)
Yhat_cv.shape
```

Out[15]:

```
(263, 20)
```

In [16]:

```
cv_mse = []
for train_idx, test_idx in kfold.split(Y):
    errors = (Yhat_cv[test_idx] - Y[test_idx, None]) ** 2
    cv_mse.append(errors.mean(0))  # column means
cv_mse = np.array(cv_mse).T
cv_mse.shape
```

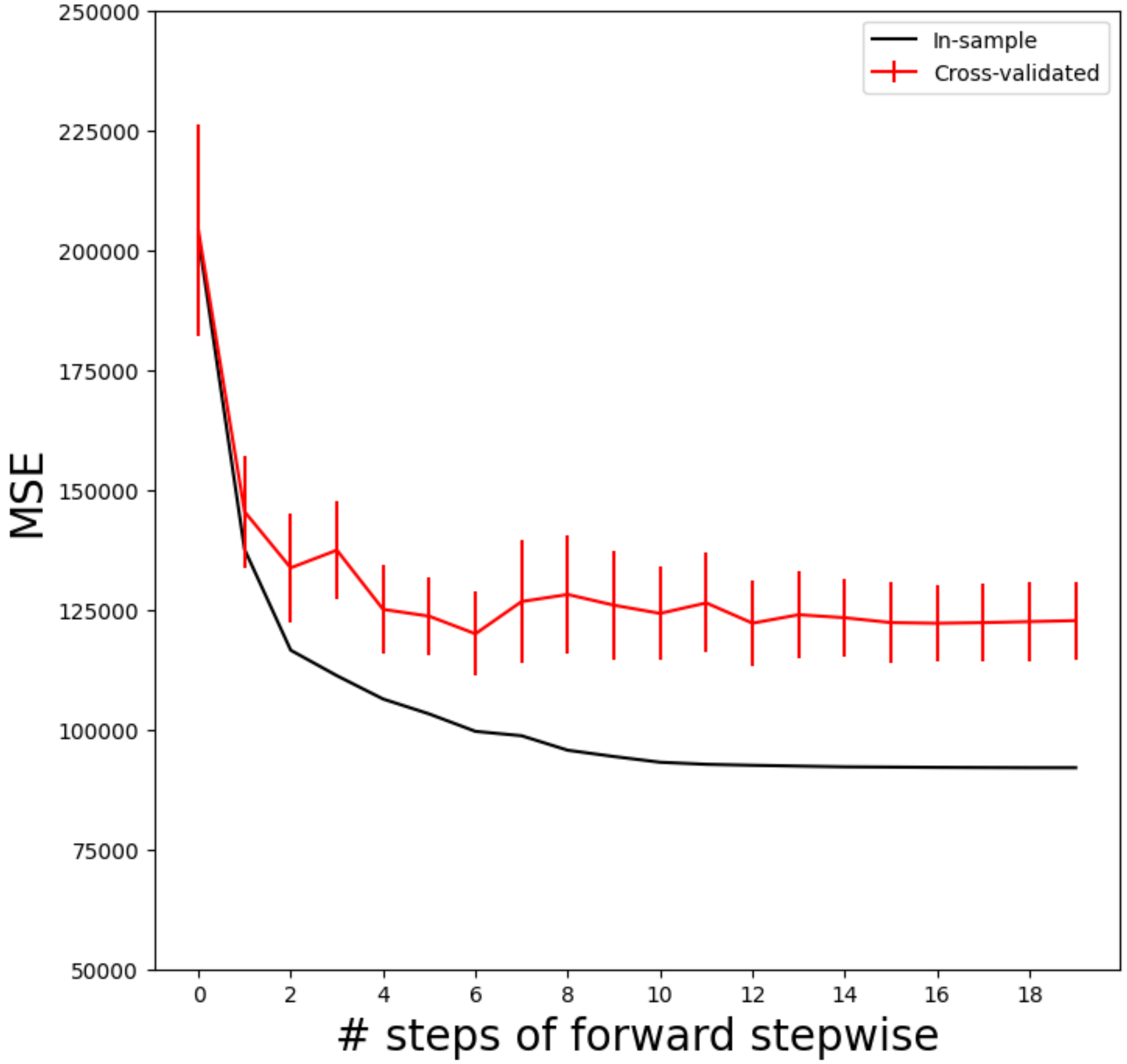
Out[16]:

```
(20, 5)
```

In [17]:

```
ax.errorbar(np.arange(n_steps),
            cv_mse.mean(1),
            cv_mse.std(1) / np.sqrt(K),
            label='Cross-validated',
            c='r')  # color red
ax.set_ylim([50000, 250000])
ax.legend()
mse_fig
```

Out[17]:



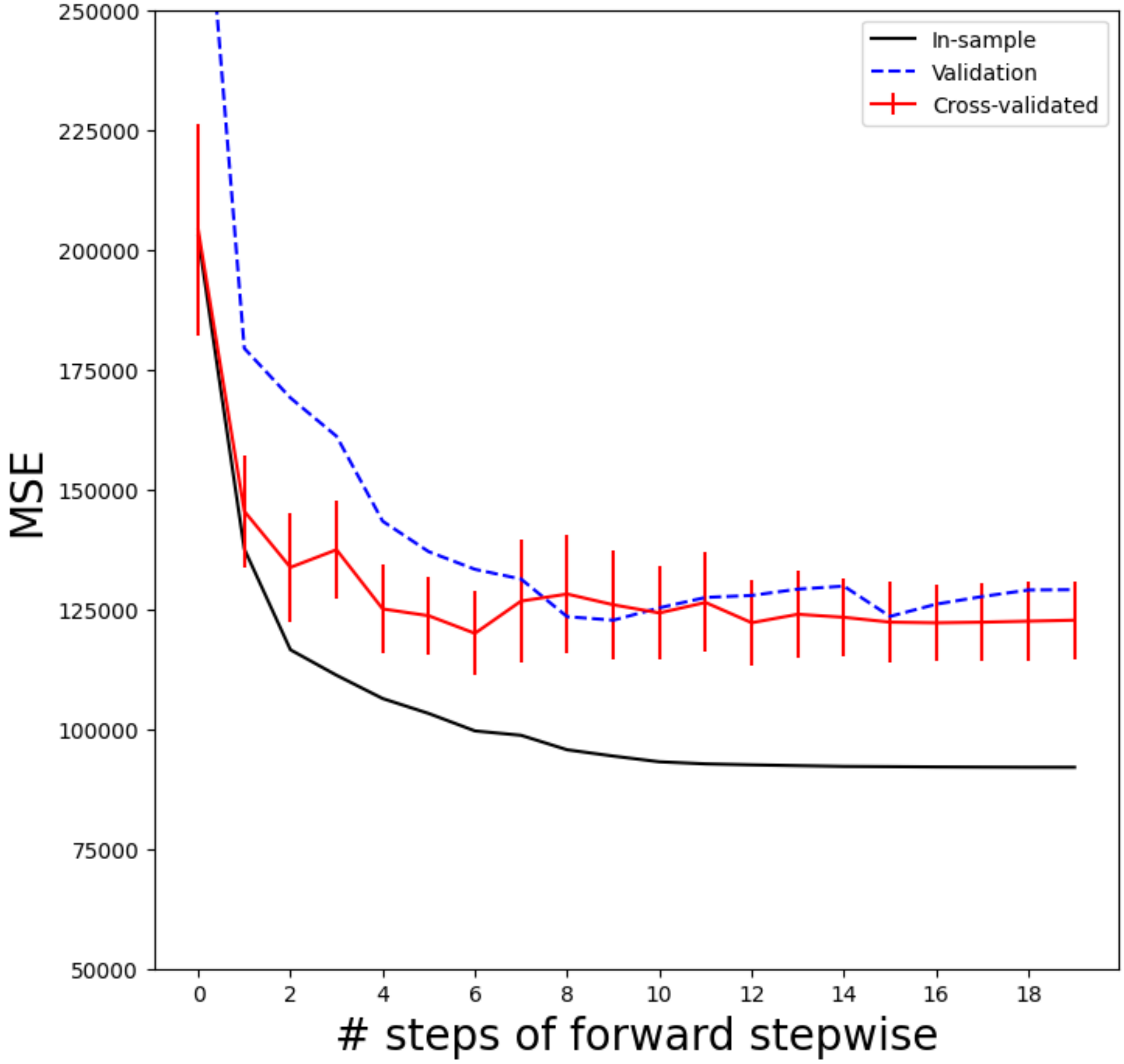
In [18]:

```
validation = skm.ShuffleSplit(n_splits=1,
                              test_size=0.2,
                              random_state=0)
for train_idx, test_idx in validation.split(Y):
    full_path.fit(Hitters.iloc[train_idx],
                  Y[train_idx])
    Yhat_val = full_path.predict(Hitters.iloc[test_idx])
    errors = (Yhat_val - Y[test_idx, None]) ** 2
    validation_mse = errors.mean(0)
```

In [19]:

```
ax.plot(np.arange(n_steps),
        validation_mse,
        'b--',  # color blue , broken line
        label='Validation')
ax.set_xticks(np.arange(n_steps)[::2])
ax.set_ylim([50000, 250000])
ax.legend()
mse_fig
```

Out[19]:



In [20]:

```
D = design.fit_transform(Hitters)
D = D.drop('intercept', axis=1)
X = np.asarray(D)
```

In [21]:

```
path = fit_path(X,
                Y,
                max_nonzeros=X.shape[1])
```

```
Preprocessing Data.
BnB Started.
Iteration: 1. Number of non-zeros:  1
Iteration: 2. Number of non-zeros:  2
Iteration: 3. Number of non-zeros:  2
Iteration: 4. Number of non-zeros:  2
Iteration: 5. Number of non-zeros:  3
Iteration: 6. Number of non-zeros:  3
Iteration: 7. Number of non-zeros:  4
Iteration: 8. Number of non-zeros:  9
Iteration: 9. Number of non-zeros:  9
Iteration: 10. Number of non-zeros:  9
Iteration: 11. Number of non-zeros:  9
Iteration: 12. Number of non-zeros:  9
Iteration: 13. Number of non-zeros:  9
Iteration: 14. Number of non-zeros:  9
Iteration: 15. Number of non-zeros:  9
Iteration: 16. Number of non-zeros:  9
Iteration: 17. Number of non-zeros:  9
Iteration: 18. Number of non-zeros:  17
Iteration: 19. Number of non-zeros:  19
```

In [22]:

```python
path[3]
```

Out[22]:

```
{'B': array([0.        , 3.25484367, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.67775265, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        ]),
 'B0': -38.98216739555551,
 'lambda_0': 0.011416248027450178,
 'M': 0.5829861733382012,
 'Time_exceeded': False}
```

In [23]:

```python
Xs = X - X.mean(0)[None, :]
X_scale = X.std(0)
Xs = Xs / X_scale[None, :]
lambdas = 10 ** np.linspace(8, -2, 100) / Y.std()
soln_array = skl.ElasticNet.path(Xs,
                                 Y,
                                 l1_ratio=0.,
                                 alphas=lambdas)[1]
soln_array.shape
```

Out[23]:

```
(19, 100)
```

In [24]:

```python
soln_path = pd.DataFrame(soln_array.T,
                         columns=D.columns,
                         index=-np.log(lambdas))
soln_path.index.name = 'negative log(lambda)'
soln_path
```

Out[24]:

	AtBat	Hits	HmRun	Runs	RBI	Walk
negative log(lambda)						
-12.310855	0.000800	0.000889	0.000695	0.000851	0.000911	0.00090
-12.078271	0.001010	0.001122	0.000878	0.001074	0.001150	0.00113
-11.845686	0.001274	0.001416	0.001107	0.001355	0.001451	0.00143
-11.613102	0.001608	0.001787	0.001397	0.001710	0.001831	0.00180
-11.380518	0.002029	0.002255	0.001763	0.002158	0.002310	0.00228
...	...	...	...	...	...	.
9.784658	-290.823989	336.929968	37.322686	-59.748520	-26.507086	134.85591
10.017243	-290.879272	337.113713	37.431373	-59.916820	-26.606957	134.90054
10.249827	-290.923382	337.260446	37.518064	-60.051166	-26.686604	134.93613
10.482412	-290.958537	337.377455	37.587122	-60.158256	-26.750044	134.96447
10.714996	-290.986528	337.470648	37.642077	-60.243522	-26.800522	134.98702

100 rows × 19 columns

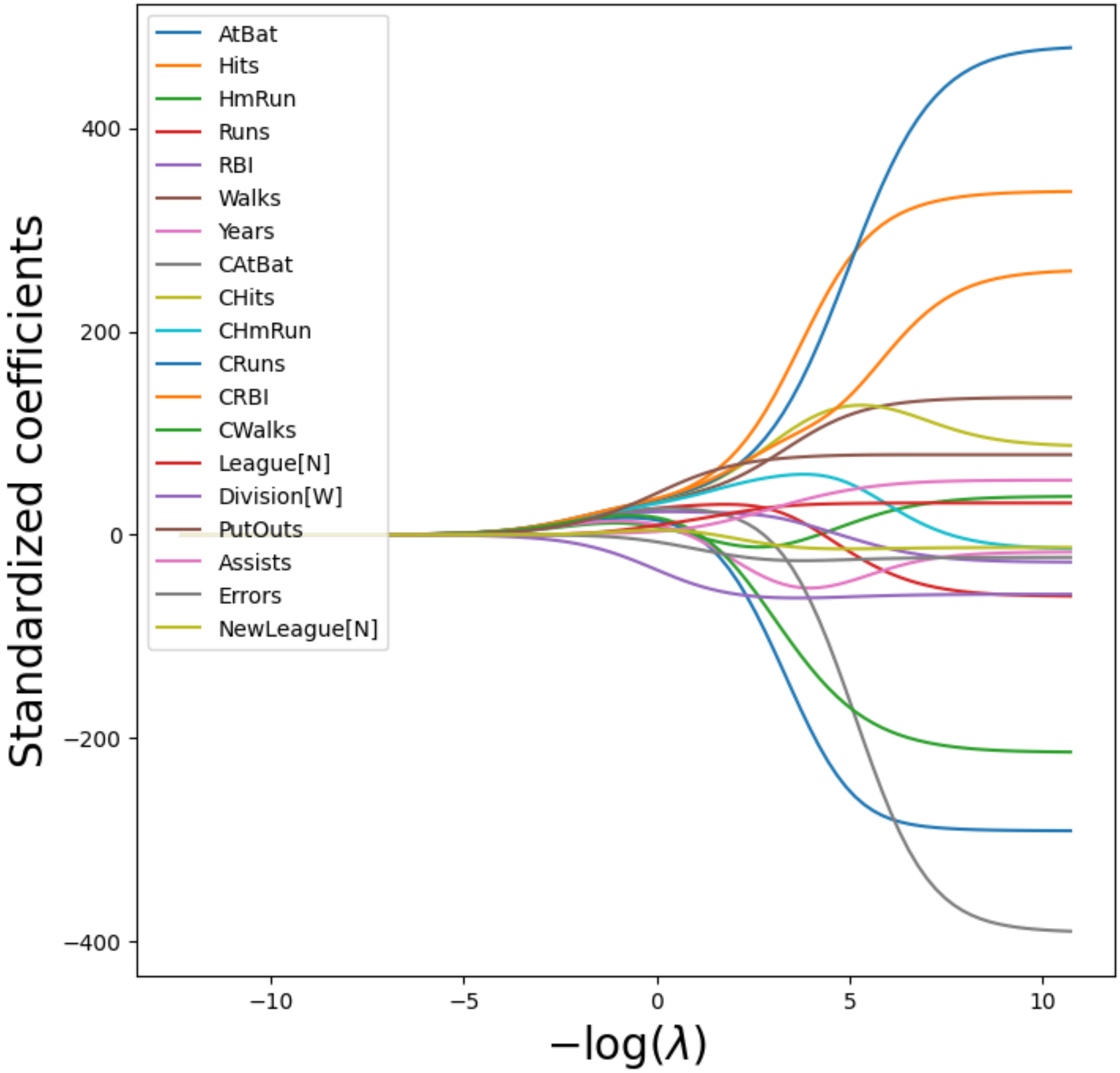
In [25]:

```
path_fig, ax = subplots(figsize=(8, 8))
soln_path.plot(ax=ax, legend=False)
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Standardized coefficients', fontsize=20)
ax.legend(loc='upper left')
```

```
<>:3: SyntaxWarning: invalid escape sequence '\l'
<>:3: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/2938439178.
py:3: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[25]:

```
<matplotlib.legend.Legend at 0x304386690>
```



In [26]:
```
beta_hat = soln_path.loc[soln_path.index[39]]
lambdas[39], beta_hat
```

```
Out[26]: (25.53538897200662,
          AtBat           5.433750
          Hits            6.223582
          HmRun           4.585498
          Runs            5.880855
          RBI             6.195921
          Walks           6.277975
          Years           5.299767
          CAtBat          7.147501
          CHits           7.539495
          CHmRun          7.182344
          CRuns           7.728649
          CRBI            7.790702
          CWalks          6.592901
          League[N]       0.042445
          Division[W]    -3.107159
          PutOuts         4.605263
          Assists         0.378371
          Errors         -0.135196
          NewLeague[N]    0.150323
          Name: -3.240065292879872, dtype: float64)
```

```
In [27]: np.linalg.norm(beta_hat)
```

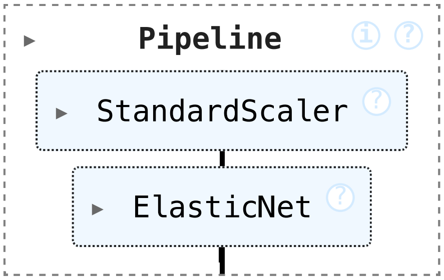
```
Out[27]: 24.17061720144378
```

```
In [28]: beta_hat = soln_path.loc[soln_path.index[59]]
         lambdas[59], np.linalg.norm(beta_hat)
```

```
Out[28]: (0.24374766133488554, 160.42371017725839)
```

```
In [29]: ridge = skl.ElasticNet(alpha=lambdas[59], l1_ratio=0)
         scaler = StandardScaler(with_mean=True, with_std=True)
         pipe = Pipeline(steps=[('scaler', scaler), ('ridge', ridge)])
         pipe.fit(X, Y)
```

Out[29]:



```
In [30]: np.linalg.norm(ridge.coef_)
```

```
Out[30]: 160.4237101772591
```

```
In [31]: validation = skm.ShuffleSplit(n_splits=1,
                                       test_size=0.5,
                                       random_state=0)
```

```
ridge.alpha = 0.01
results = skm.cross_validate(ridge,
                             X,
                             Y,
                             scoring='neg_mean_squared_error',
                             cv=validation)
-results['test_score']
```

Out[31]:

```
array([134214.00419204])
```

In [32]:

```
ridge.alpha = 1e10
results = skm.cross_validate(ridge,
                             X,
                             Y,
                             scoring='neg_mean_squared_error',
                             cv=validation)
-results['test_score']
```

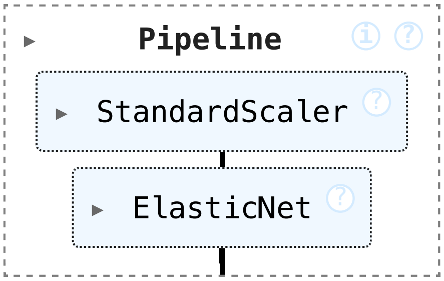
Out[32]:

```
array([231788.32155285])
```

In [33]:

```
param_grid = {'ridge__alpha': lambdas}
grid = skm.GridSearchCV(pipe,
                        param_grid,
                        cv=validation,
                        scoring='neg_mean_squared_error')
grid.fit(X, Y)
grid.best_params_['ridge__alpha']
grid.best_estimator_
```

Out[33]:



▸ **Pipeline**

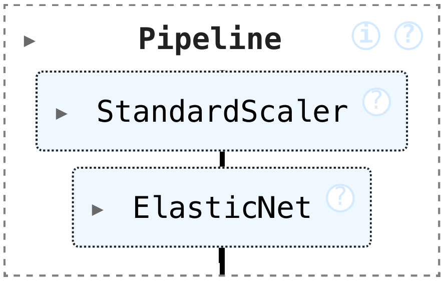
▸ StandardScaler

▸ ElasticNet

In [34]:

```
grid = skm.GridSearchCV(pipe,
                        param_grid,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
grid.fit(X, Y)
grid.best_params_['ridge__alpha']
grid.best_estimator_
```

Out[34]:



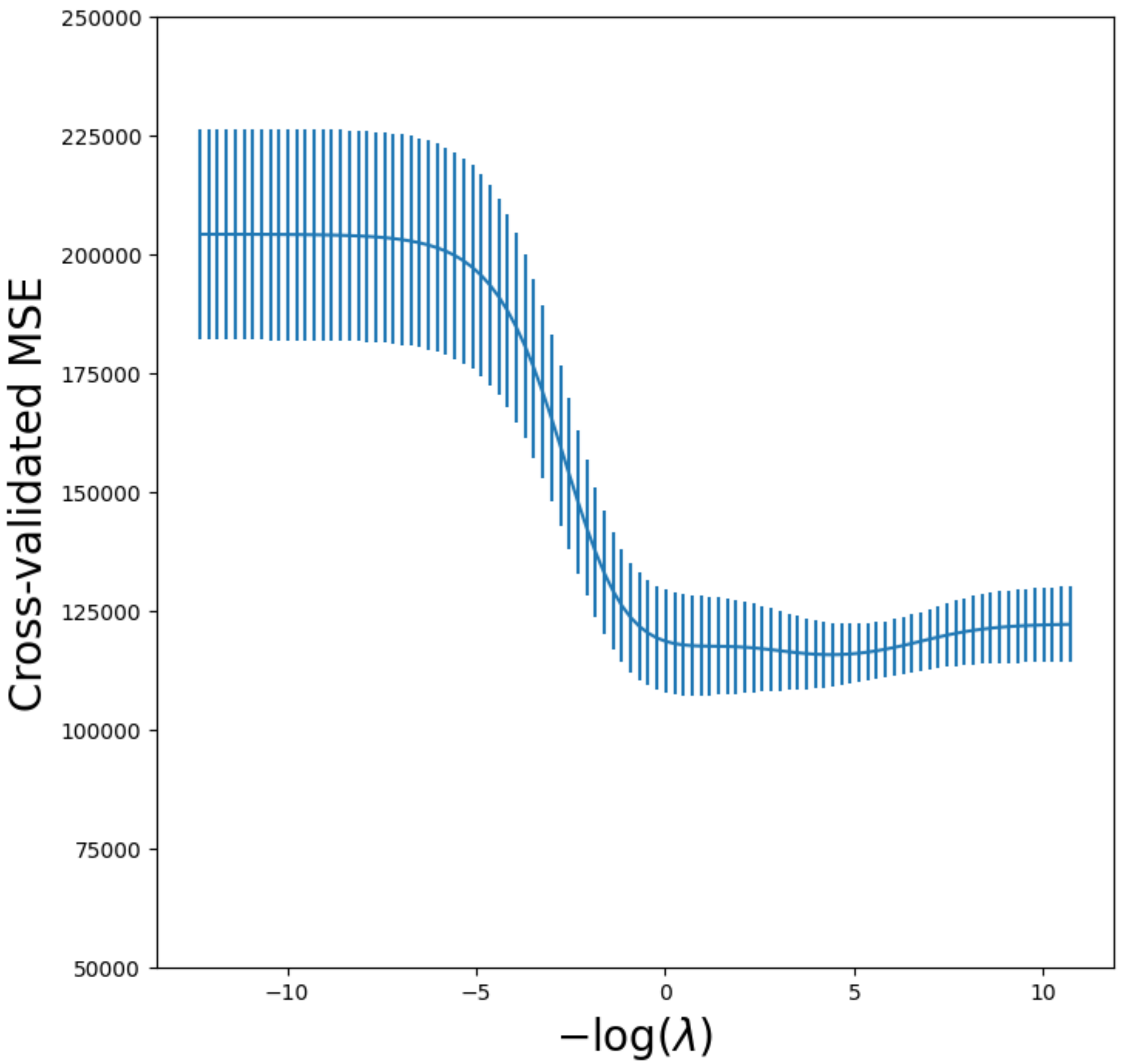
In [35]:

```
ridge_fig, ax = subplots(figsize=(8, 8))
ax.errorbar(-np.log(lambdas),
            -grid.cv_results_['mean_test_score'],
            yerr=grid.cv_results_['std_test_score'] / np.sqrt(K))
ax.set_ylim([50000, 250000])
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20)
```

```
<>:6: SyntaxWarning: invalid escape sequence '\l'
<>:6: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/252384649.p
y:6: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[35]:

```
Text(0, 0.5, 'Cross-validated MSE')
```



In [36]:

```python
grid_r2 = skm.GridSearchCV(pipe,
                           param_grid,
                           cv=kfold)
grid_r2.fit(X, Y)
```

Out[36]:

```
▸        GridSearchCV
▸ best_estimator_: Pipeline
    ▸ StandardScaler
      ▸ ElasticNet
```

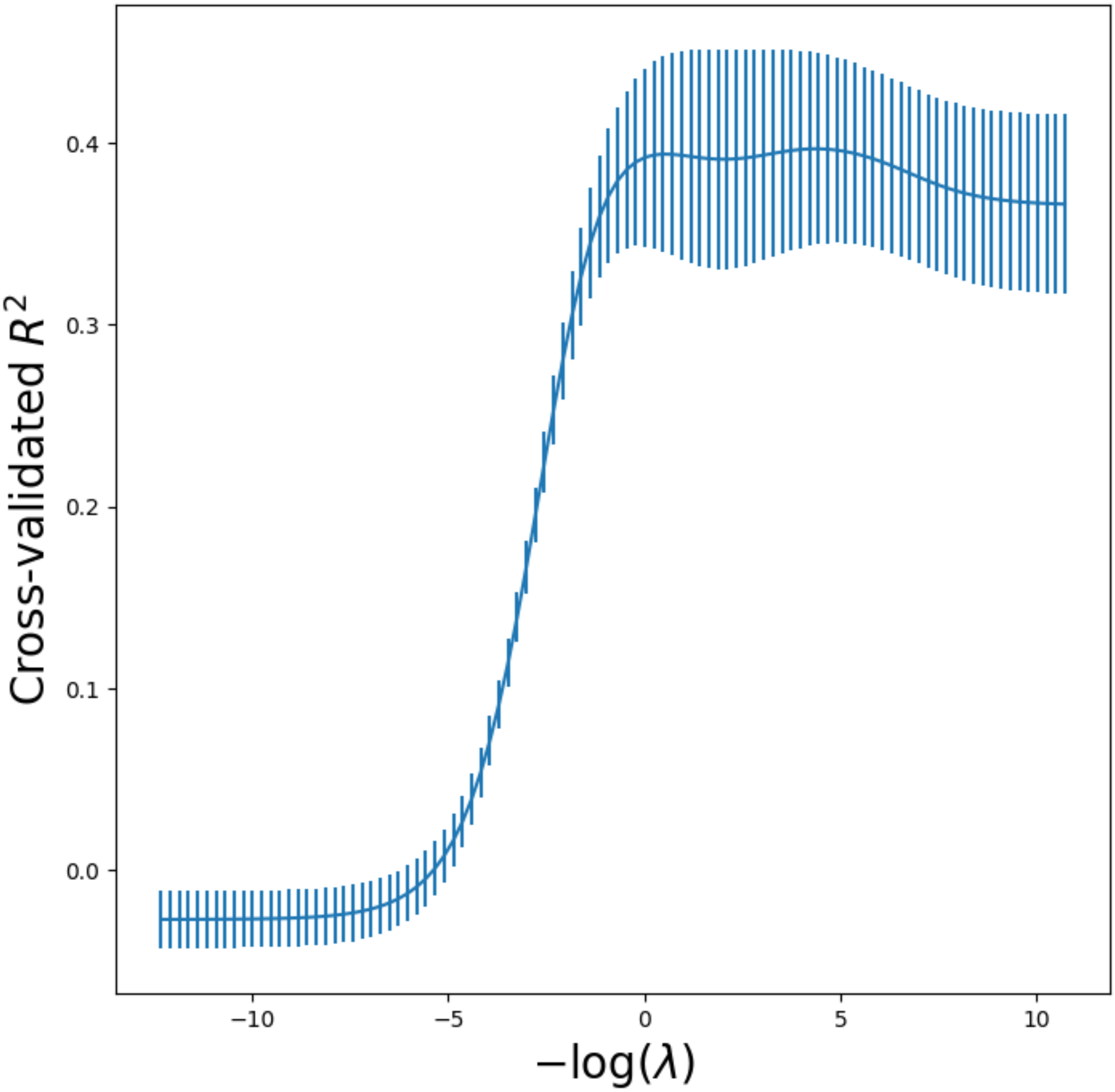
In [37]:

```python
r2_fig, ax = subplots(figsize=(8, 8))
ax.errorbar(-np.log(lambdas),
             grid_r2.cv_results_['mean_test_score'],
             yerr=grid_r2.cv_results_['std_test_score'] / np.sqrt(K)
             )
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated $R^2$', fontsize=20)
```

```
<>:6: SyntaxWarning: invalid escape sequence '\l'
<>:6: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/4088780906.
py:6: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[37]:

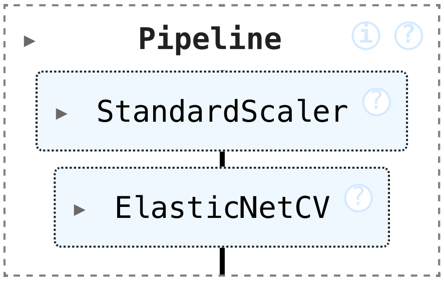
```
Text(0, 0.5, 'Cross-validated $R^2$')
```



In [38]:

```
ridgeCV = skl.ElasticNetCV(alphas=lambdas,
                           l1_ratio=0,
                           cv=kfold)
pipeCV = Pipeline(steps=[('scaler', scaler),
                         ('ridge', ridgeCV)])
pipeCV.fit(X, Y)
```

Out[38]:



▸ Pipeline

▸ StandardScaler

▸ ElasticNetCV

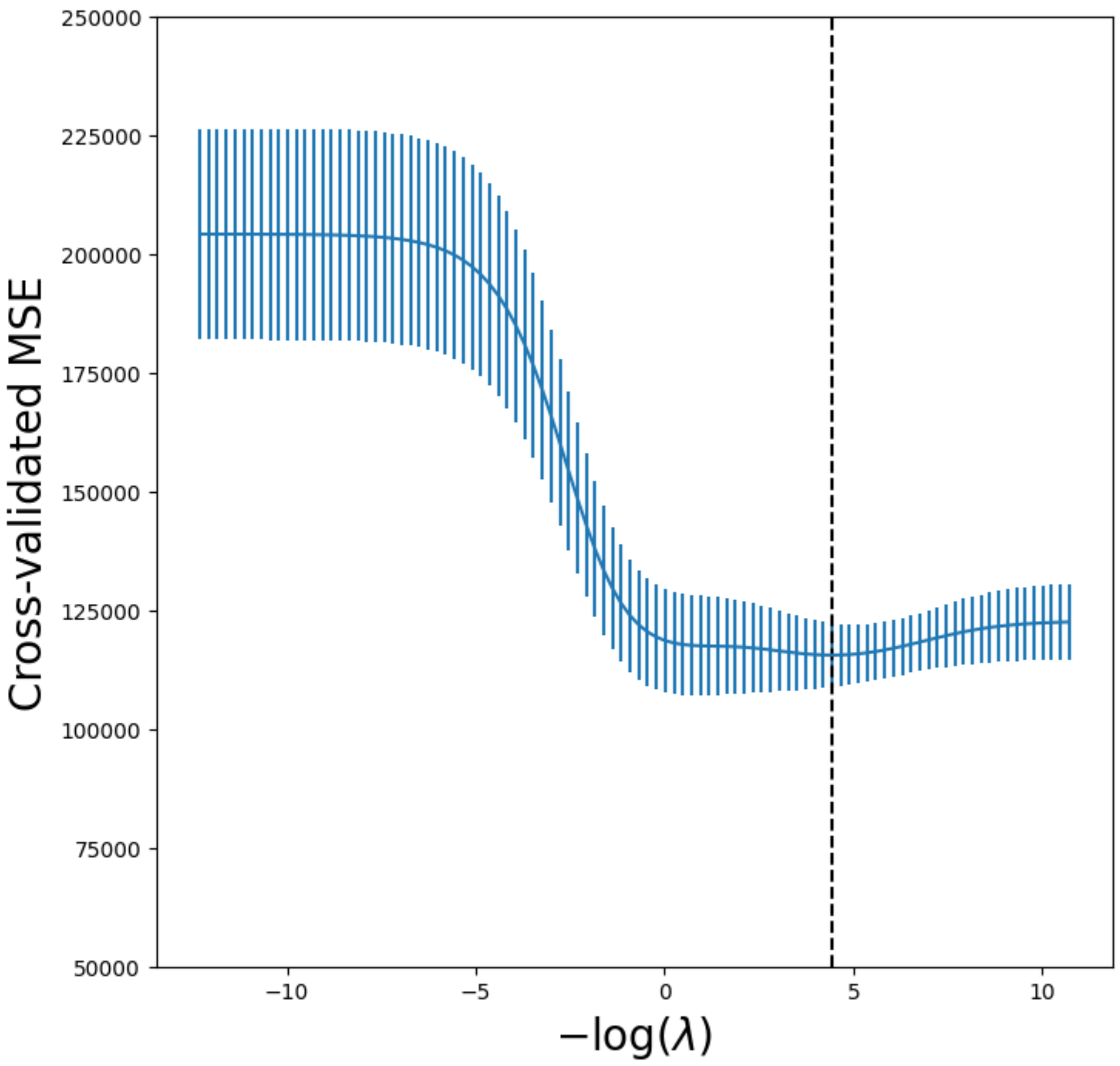
In [39]:

```
tuned_ridge = pipeCV.named_steps['ridge']
ridgeCV_fig, ax = subplots(figsize=(8, 8))
ax.errorbar(-np.log(lambdas),
             tuned_ridge.mse_path_.mean(1),
             yerr=tuned_ridge.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_ridge.alpha_), c='k', ls='--')
ax.set_ylim([50000, 250000])
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20)
```

```
<>:8: SyntaxWarning: invalid escape sequence '\l'
<>:8: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/35476348.p
y:8: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[39]:

```
Text(0, 0.5, 'Cross-validated MSE')
```



```
In [40]: np.min(tuned_ridge.mse_path_.mean(1))
```

```
Out[40]: 115526.70630987917
```

```
In [41]: tuned_ridge.coef_
```

```
Out[41]: array([-222.80877051,  238.77246614,    3.21103754,   -2.93050845,
                   3.64888723,  108.90953869,  -50.81896152, -105.15731984,
                 122.00714801,   57.1859509 ,  210.35170348,  118.05683748,
                -150.21959435,   30.36634231,  -61.62459095,   77.73832472,
                  40.07350744,  -25.02151514,  -13.68429544])
```

```
In [42]: outer_valid = skm.ShuffleSplit(n_splits=1,
                                        test_size=0.25,
                                        random_state=1)
         inner_cv = skm.KFold(n_splits=5,
                              shuffle=True,
                              random_state=2)
```

```
ridgeCV = skl.ElasticNetCV(alphas=lambdas,
                           l1_ratio=0,
                           cv=inner_cv)
pipeCV = Pipeline(steps=[('scaler', scaler),
                         ('ridge', ridgeCV)])
```

In [43]:
```
results = skm.cross_validate(pipeCV,
                             X,
                             Y,
                             cv=outer_valid,
                             scoring='neg_mean_squared_error')
-results['test_score']
```

Out[43]:
```
array([132393.84003227])
```

In [44]:
```
lassoCV = skl.ElasticNetCV(n_alphas=100,
                           l1_ratio=1,
                           cv=kfold)
pipeCV = Pipeline(steps=[('scaler', scaler),
                         ('lasso', lassoCV)])
pipeCV.fit(X, Y)
tuned_lasso = pipeCV.named_steps['lasso']
tuned_lasso.alpha_
```

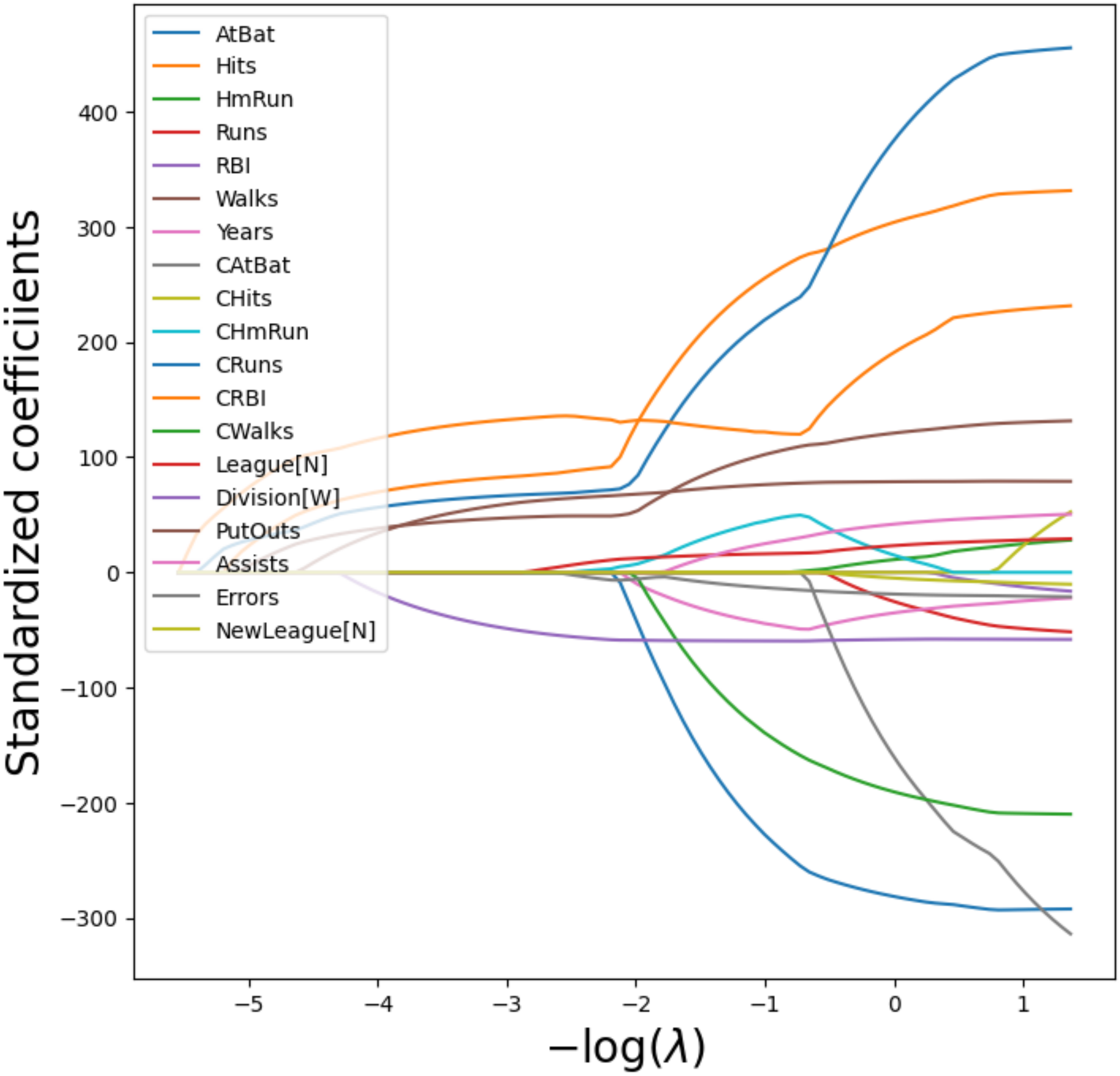
Out[44]:
```
3.1472370031649866
```

In [45]:
```
lambdas, soln_array = skl.Lasso.path(Xs,
                                     Y,
                                     l1_ratio=1,
                                     n_alphas=100)[:2]
soln_path = pd.DataFrame(soln_array.T,
                         columns=D.columns,
                         index=-np.log(lambdas))
```

In [46]:
```
path_fig, ax = subplots(figsize=(8, 8))
soln_path.plot(ax=ax, legend=False)
ax.legend(loc='upper left')
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Standardized coefficiients', fontsize=20)
```

```
<>:4: SyntaxWarning: invalid escape sequence '\l'
<>:4: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/1325931816.
py:4: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[46]:
```
Text(0, 0.5, 'Standardized coefficiients')
```



In [47]:
```
np.min(tuned_lasso.mse_path_.mean(1))
```

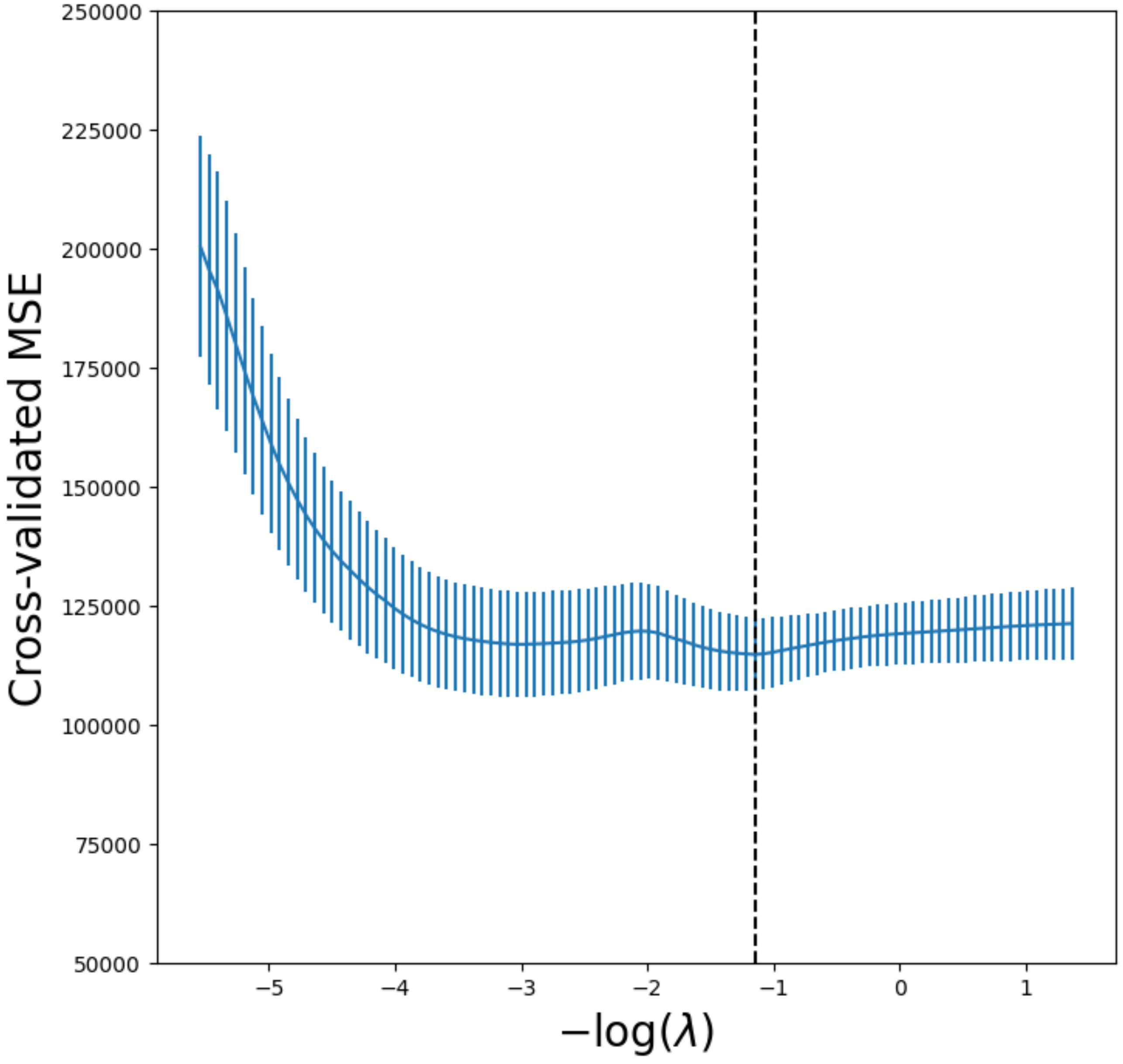
Out[47]:
```
114690.73118253727
```

In [48]:
```
lassoCV_fig, ax = subplots(figsize=(8, 8))
ax.errorbar(-np.log(tuned_lasso.alphas_),
            tuned_lasso.mse_path_.mean(1),
            yerr=tuned_lasso.mse_path_.std(1) / np.sqrt(K))
ax.axvline(-np.log(tuned_lasso.alpha_), c='k', ls='--')
ax.set_ylim([50000, 250000])
ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
ax.set_ylabel('Cross-validated MSE', fontsize=20)
```

```
<>:7: SyntaxWarning: invalid escape sequence '\l'
<>:7: SyntaxWarning: invalid escape sequence '\l'
/var/folders/97/23ltc4v96g31pp78_gyv6dvm0000gn/T/ipykernel_10519/3583132959.
py:7: SyntaxWarning: invalid escape sequence '\l'
  ax.set_xlabel('$-\log(\lambda)$', fontsize=20)
```

Out[48]:

```
Text(0, 0.5, 'Cross-validated MSE')
```



In [49]:

```
tuned_lasso.coef_
```

Out[49]:

```
array([-210.01008773,  243.4550306 ,    0.        ,    0.        ,
          0.        ,   97.69397357,  -41.52283116,   -0.        ,
          0.        ,   39.62298193,  205.75273856,  124.55456561,
       -126.29986768,   15.70262427,  -59.50157967,   75.24590036,
         21.62698014,  -12.04423675,   -0.        ])
```

In [50]:

```
pca = PCA(n_components=2)
linreg = skl.LinearRegression()
pipe = Pipeline([('pca', pca),
                     ('linreg', linreg)])
pipe.fit(X, Y)
pipe.named_steps['linreg'].coef_
```

Out[50]:

```
array([0.09846131, 0.4758765 ])
```

In [51]:

```
pipe = Pipeline([('scaler', scaler),
                 ('pca', pca),
                 ('linreg', linreg)])
pipe.fit(X, Y)
pipe.named_steps['linreg'].coef_
```

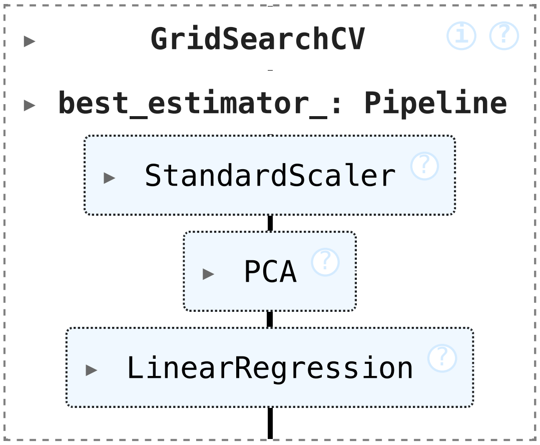
Out[51]:

```
array([106.36859204,  21.60350456])
```

In [52]:

```
param_grid = {'pca__n_components': range(1, 20)}
grid = skm.GridSearchCV(pipe,
                        param_grid,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
grid.fit(X, Y)
```

Out[52]:



▸ GridSearchCV

▸ best_estimator_: Pipeline

▸ StandardScaler

▸ PCA

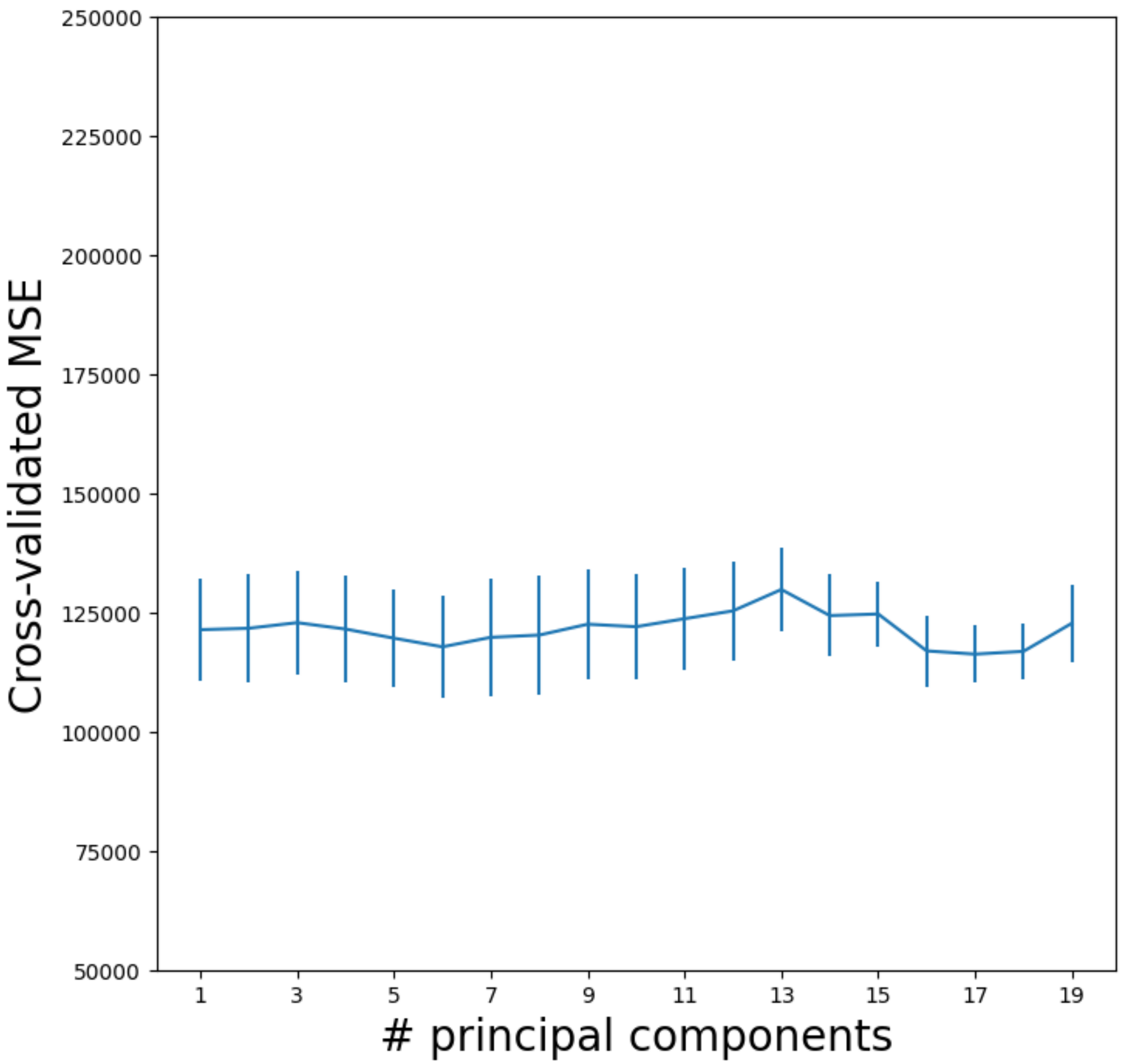
▸ LinearRegression

In [53]:

```
pcr_fig, ax = subplots(figsize=(8, 8))
n_comp = param_grid['pca__n_components']
ax.errorbar(n_comp,
            -grid.cv_results_['mean_test_score'],
            grid.cv_results_['std_test_score'] / np.sqrt(K))
ax.set_ylabel('Cross-validated MSE', fontsize=20)
ax.set_xlabel('# principal components', fontsize=20)
ax.set_xticks(n_comp[::2])
ax.set_ylim([50000, 250000])
```

Out[53]:

```
(50000.0, 250000.0)
```



```
In [54]: Xn = np.zeros((X.shape[0], 1))
         cv_null = skm.cross_validate(linreg,
                                      Xn,
                                      Y,
                                      cv=kfold,
                                      scoring='neg_mean_squared_error')
         -cv_null['test_score'].mean()
```

```
Out[54]: 204139.30692994667
```

```
In [55]: pipe.named_steps['pca'].explained_variance_ratio_
```

```
Out[55]: array([0.3831424 , 0.21841076])
```

```
In [56]: pls = PLSRegression(n_components=2,
                             scale=True)
         pls.fit(X, Y)
```

Out[56]:

```
▾ PLSRegression

PLSRegression()
```

In [57]:

```python
param_grid = {'n_components': range(1, 20)}
grid = skm.GridSearchCV(pls,
                        param_grid,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
grid.fit(X, Y)
```

Out[57]:

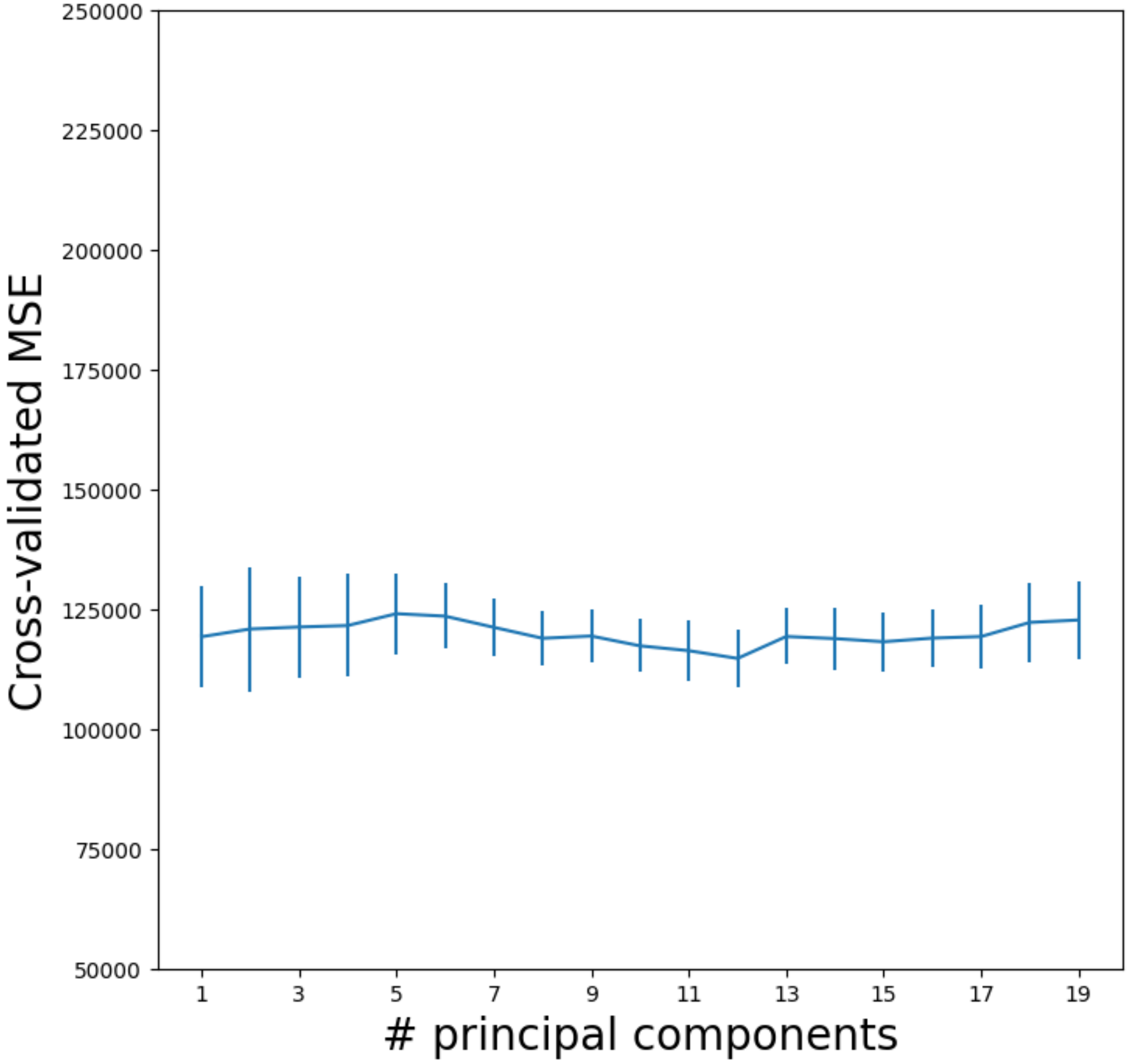
```
▸         GridSearchCV

▸ best_estimator_: PLSRegression

      ▸ PLSRegression
```

In [58]:

```python
pls_fig, ax = subplots(figsize=(8, 8))
n_comp = param_grid['n_components']
ax.errorbar(n_comp,
            -grid.cv_results_['mean_test_score'],
            grid.cv_results_['std_test_score'] / np.sqrt(K))
ax.set_ylabel('Cross-validated MSE', fontsize=20)
ax.set_xlabel('# principal components', fontsize=20)
ax.set_xticks(n_comp[::2])
ax.set_ylim([50000, 250000])
```

Out[58]:

```
(50000.0, 250000.0)
```


250000
225000
200000
175000
150000
125000
100000
75000
50000
Cross-validated MSE
1
3
5
7
9
11
13
15
17
19
# principal components