In [1]:
```
%load_ext autoreload
%autoreload 2

from itertools import product
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')

from basis_expansions import (Binner,
                              GaussianKernel,
                              Polynomial,
                              LinearSpline,
                              CubicSpline,
                              NaturalCubicSpline)
from dftransformers import ColumnSelector, FeatureUnion, Intercept, Ma

from simulation import (run_simulation_expreiment,
                        plot_simulation_expreiment,
                        make_random_train_test,
                        run_residual_simulation)
```

Examples Applying to Series, Creating Data Frames

In [2]:
```
s1 = pd.Series([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], name='s',
               index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
s2 = pd.Series([2, 3, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24], name='s',
               index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',

df = pd.DataFrame({'s1': s1, 's2': s2})
```

In [3]:
```
df.shape
```

Out[3]:
```
(12, 2)
```

In [4]:
```
t = FeatureUnion([
    ('intercept', Intercept()),
    ('s1_pipe', Pipeline([
        ('s1_selector', ColumnSelector(name='s1')),
        ('s1_features', FeatureUnion([
            ('s1_spline', NaturalCubicSpline(knots=[4, 8, 10])),
            ('s1_indicator', MapFeature(lambda t: t % 2 == 0, 'is_ever
        ]))
    ])),
    ('s2_pipe', Pipeline([
        ('s2_selector', ColumnSelector(name='s2')),
        ('s2_spline', NaturalCubicSpline(knots=[4, 8, 10]))
    ]))
])
```

In [5]:
```
t.fit(df)
t.transform(df)
```

Out[5]:

	intercept	s1_spline_linear	s1_spline_0	is_even	s2_spline_linear	s2_spline_0
a	1.0	0.0	0.000000	1.0	2.0	0.000000
b	1.0	1.0	0.000000	0.0	3.0	0.000000
c	1.0	2.0	0.000000	1.0	6.0	1.333333
d	1.0	3.0	0.000000	0.0	8.0	10.666667
e	1.0	4.0	0.000000	1.0	10.0	32.000000
f	1.0	5.0	0.166667	0.0	12.0	56.000000
g	1.0	6.0	1.333333	1.0	14.0	80.000000
h	1.0	7.0	4.500000	0.0	16.0	104.000000
i	1.0	8.0	10.666667	1.0	18.0	128.000000
j	1.0	9.0	20.333333	0.0	20.0	152.000000
k	1.0	10.0	32.000000	1.0	22.0	176.000000
l	1.0	11.0	44.000000	0.0	24.0	200.000000

In [6]:
```
t = FeatureUnion([
    ('intercept', Intercept()),
    ('s1_pipe', Pipeline([
        ('s1_selector', ColumnSelector(name='s1')),
        ('s1_features', FeatureUnion([
            ('s1_spline', Polynomial(degree=3)),
            ('s1_indicator', MapFeature(lambda t: t % 2 == 0, 'is_ever
        ]))
    ])),
    ('s2_pipe', Pipeline([
        ('s2_selector', ColumnSelector(name='s2')),
        ('s2_spline', Polynomial(degree=2))
    ]))
])
```

In [7]:
```
t.fit(df)
t.transform(df)
```

Out[7]:

	intercept	s1_degree_1	s1_degree_2	s1_degree_3	is_even	s2_degree_1	s2_degree_2
a	1.0	0.0	0.0	0.0	1.0	2.0	4.0
b	1.0	1.0	1.0	1.0	0.0	3.0	9.0
c	1.0	2.0	4.0	8.0	1.0	6.0	36.0
d	1.0	3.0	9.0	27.0	0.0	8.0	64.0
e	1.0	4.0	16.0	64.0	1.0	10.0	100.0
f	1.0	5.0	25.0	125.0	0.0	12.0	144.0
g	1.0	6.0	36.0	216.0	1.0	14.0	196.0
h	1.0	7.0	49.0	343.0	0.0	16.0	256.0
i	1.0	8.0	64.0	512.0	1.0	18.0	324.0
j	1.0	9.0	81.0	729.0	0.0	20.0	400.0
k	1.0	10.0	100.0	1000.0	1.0	22.0	484.0
l	1.0	11.0	121.0	1331.0	0.0	24.0	576.0

In [8]:
```
t = FeatureUnion([
    ('intercept', Intercept()),
    ('s1_pipe', Pipeline([
        ('s1_selector', ColumnSelector(name='s1')),
        ('s1_features', FeatureUnion([
            ('s1_spline', LinearSpline(knots=[4, 8, 10])),
            ('s1_indicator', MapFeature(lambda t: t % 2 == 0, 'is_ever
        ]))
    ])),
    ('s2_pipe', Pipeline([
        ('s2_selector', ColumnSelector(name='s2')),
        ('s2_spline', Polynomial(degree=2))
    ]))
])
```

In [9]:
```
t.fit(df)
t.transform(df)
```

Out[9]:

	intercept	s1_spline_linear	s1_spline_0	s1_spline_1	s1_spline_2	is_even	s2_degree_1	s2_d
a	1.0	0.0	0.0	0.0	0.0	1.0	2.0	
b	1.0	1.0	0.0	0.0	0.0	0.0	3.0	
c	1.0	2.0	0.0	0.0	0.0	1.0	6.0	
d	1.0	3.0	0.0	0.0	0.0	0.0	8.0	
e	1.0	4.0	0.0	0.0	0.0	1.0	10.0	
f	1.0	5.0	1.0	0.0	0.0	0.0	12.0	
g	1.0	6.0	2.0	0.0	0.0	1.0	14.0	
h	1.0	7.0	3.0	0.0	0.0	0.0	16.0	
i	1.0	8.0	4.0	0.0	0.0	1.0	18.0	
j	1.0	9.0	5.0	1.0	0.0	0.0	20.0	
k	1.0	10.0	6.0	2.0	0.0	1.0	22.0	
l	1.0	11.0	7.0	3.0	1.0	0.0	24.0	

In [10]:
```
t = FeatureUnion([
    ('intercept', Intercept()),
    ('s1_pipe', Pipeline([
        ('s1_selector', ColumnSelector(name='s1')),
        ('s1_features', FeatureUnion([
            ('s1_spline', CubicSpline(knots=[4, 8, 10])),
            ('s1_indicator', MapFeature(lambda t: t % 2 == 0, 'is_ever
        ]))
    ])),
    ('s2_pipe', Pipeline([
        ('s2_selector', ColumnSelector(name='s2')),
        ('s2_spline', Polynomial(degree=2))
    ]))
])
```

In [11]:
```
t.fit(df)
t.transform(df)
```

Out[11]:

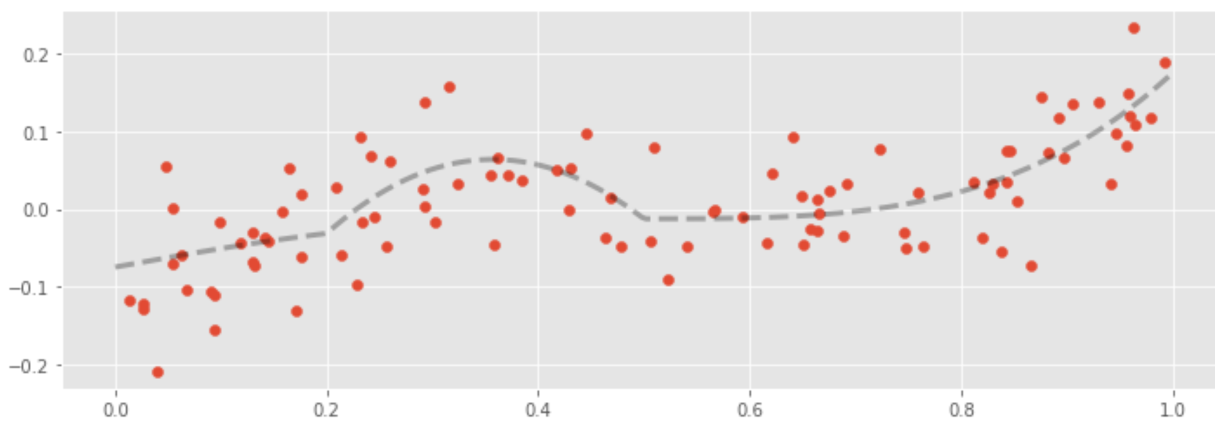
	intercept	s1_spline_linear	s1_spline_quadratic	s1_spline_cubic	s1_spline_0	s1_spline_1	s1
a	1.0	0.0	0.0	0.0	0.0	0.0	
b	1.0	1.0	1.0	1.0	0.0	0.0	
c	1.0	2.0	4.0	8.0	0.0	0.0	
d	1.0	3.0	9.0	27.0	0.0	0.0	
e	1.0	4.0	16.0	64.0	0.0	0.0	
f	1.0	5.0	25.0	125.0	1.0	0.0	
g	1.0	6.0	36.0	216.0	8.0	0.0	
h	1.0	7.0	49.0	343.0	27.0	0.0	
i	1.0	8.0	64.0	512.0	64.0	0.0	
j	1.0	9.0	81.0	729.0	125.0	1.0	
k	1.0	10.0	100.0	1000.0	216.0	8.0	
l	1.0	11.0	121.0	1331.0	343.0	27.0	

Examples of Fitting

In [12]:
```
def signal(x):
    return (x*x*x*(x-1)
              + 2*(1/(1 + np.exp(-0.5*(x - 0.5))))
              - 3.5*(x > 0.2)*(x < 0.5)*(x - 0.2)*(x - 0.5)
              - 0.95)

x = np.random.uniform(size=100)
y = signal(x) + np.random.normal(scale=0.05, size=100)
```

In [13]:
```
fig, ax = plt.subplots(figsize=(12, 4))
t = np.linspace(0, 1, num=250)

ax.scatter(x, y)
ax.plot(t, signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.3)
```

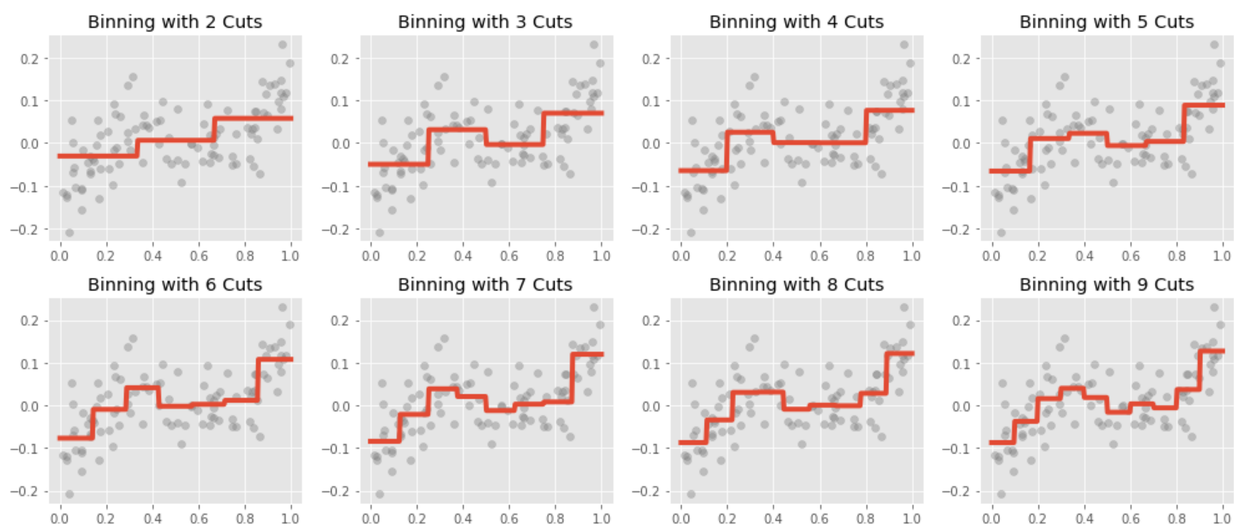
Out[13]: [<matplotlib.lines.Line2D at 0x1303ecfa0>]



Binned regression with dummy varaibles.

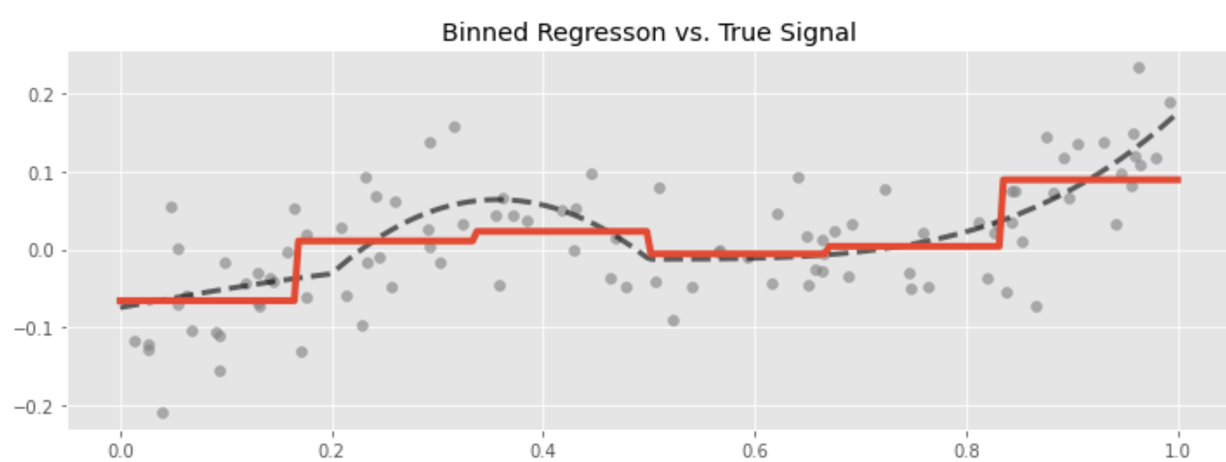
In [14]:
```
def make_binned_regression(n_cuts):
    return Pipeline([
        ('binner', Binner(0, 1, n_cuts=n_cuts)),
        ('regression', LinearRegression(fit_intercept=False))
    ])

regressions = {}
for n_cuts in range(1, 24):
    regressions[n_cuts] = make_binned_regression(n_cuts)
    regressions[n_cuts].fit(x, y)
```

In [15]:
```
fig, ax = plt.subplots(2, 4, figsize=(14, 6))

t = np.linspace(0, 1, 250)
for n_cuts, ax in enumerate(ax.flatten(), start=2):
    ax.plot(t, regressions[n_cuts].predict(t.reshape(-1, 1)), linewidt
    ax.scatter(x, y, alpha=0.4, color="grey")
    ax.set_title("Binning with {} Cuts".format(n_cuts))

fig.tight_layout()
```



In [16]:
```
fig, ax = plt.subplots(figsize=(12, 4))
t = np.linspace(0, 1, num=250)

ax.scatter(x, y, alpha=0.6, color="grey")
ax.plot(t, signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.6)
ax.plot(t, regressions[5].predict(t.reshape(-1, 1)), linewidth=4)
ax.set_title("Binned Regresson vs. True Signal")
```

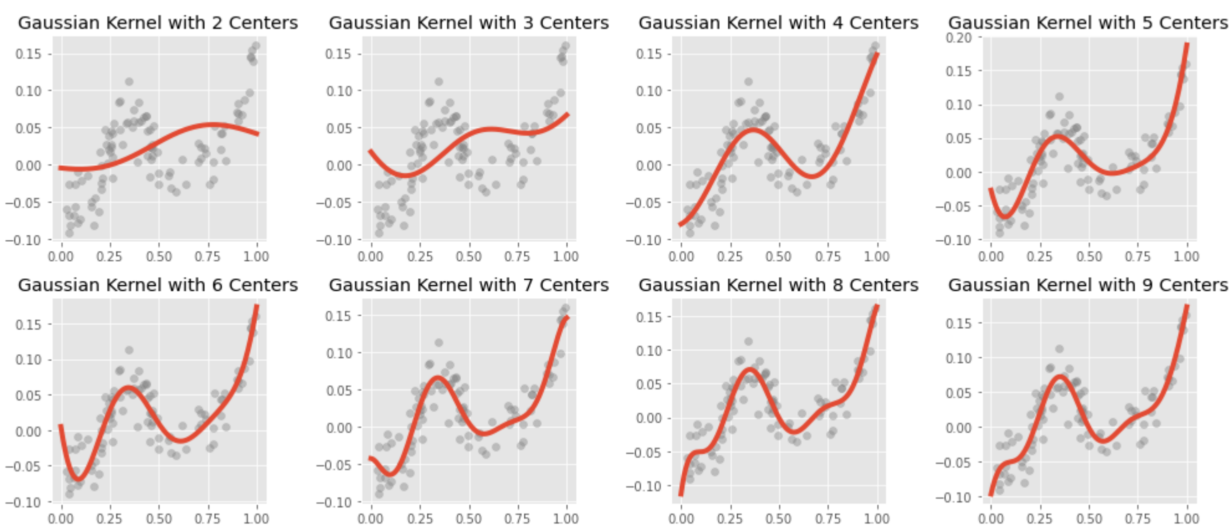
Out[16]: Text(0.5, 1.0, 'Binned Regresson vs. True Signal')



Regression with Gaussian Kernel Basis.

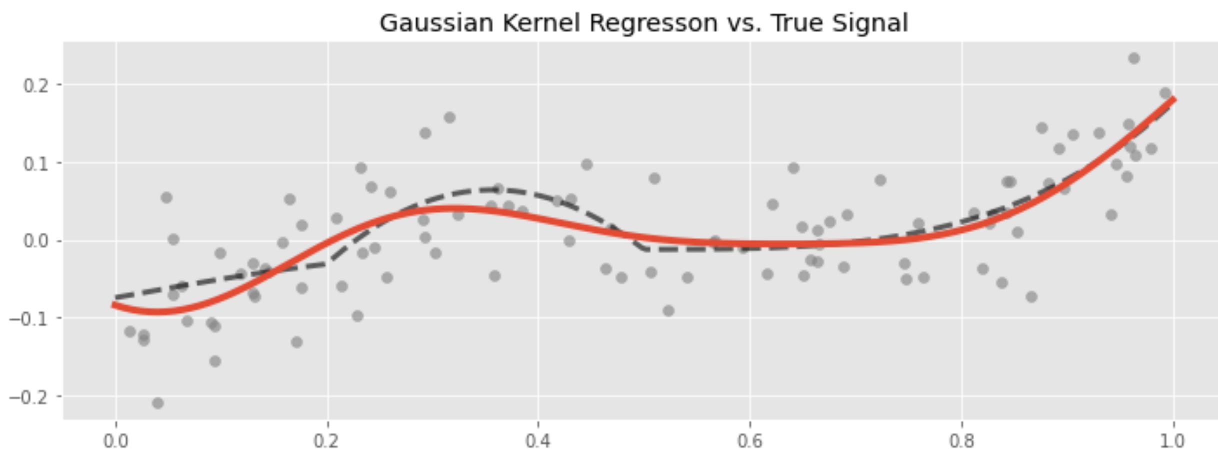
In [76]:
```
def make_gaussian_regression(n_centers):
    return Pipeline([
        ('binner', GaussianKernel(0, 1, n_centers=n_centers, bandwidth
        ('regression', LinearRegression(fit_intercept=True))
    ])

regressions = {}
for n_centers in range(2, 10):
    regressions[n_centers] = make_gaussian_regression(n_centers)
    regressions[n_centers].fit(x, y)
```

In [77]:
```
fig, ax = plt.subplots(2, 4, figsize=(14, 6))

t = np.linspace(0, 1, 250)
for n_centers, ax in enumerate(ax.flatten(), start=2):
    ax.plot(t, regressions[n_centers].predict(t.reshape(-1, 1)), linew
    ax.scatter(x, y, alpha=0.4, color="grey")
    ax.set_title("Gaussian Kernel with {} Centers".format(n_centers))

fig.tight_layout()
```



In [19]:
```
fig, ax = plt.subplots(figsize=(12, 4))
t = np.linspace(0, 1, num=250)

ax.scatter(x, y, alpha=0.6, color="grey")
ax.plot(t, signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.6)
ax.plot(t, regressions[6].predict(t.reshape(-1, 1)), linewidth=4)
ax.set_title("Gaussian Kernel Regresson vs. True Signal")
```

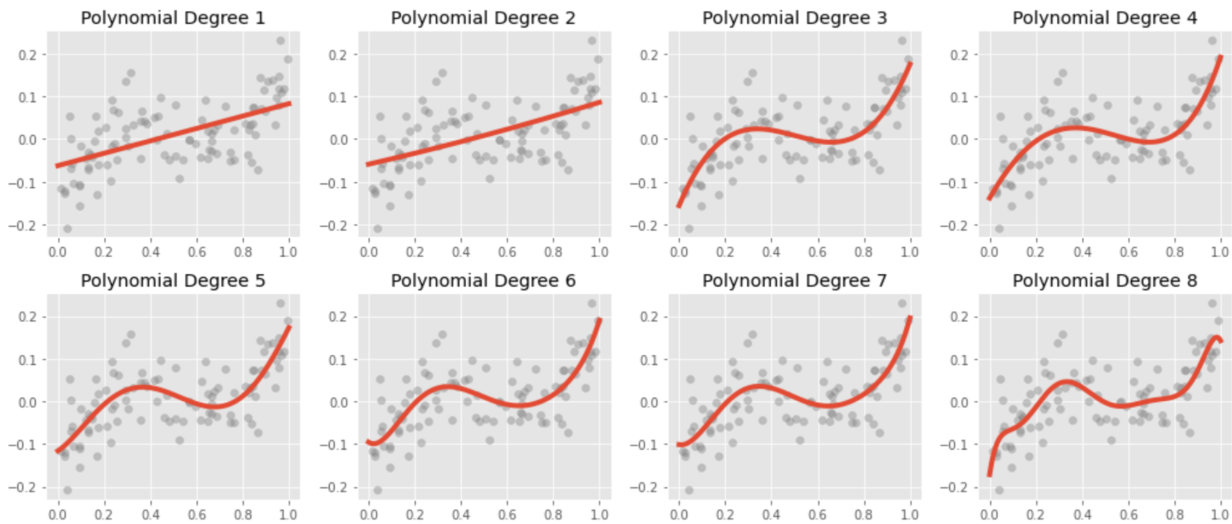
Out[19]:
```
Text(0.5, 1.0, 'Gaussian Kernel Regresson vs. True Signal')
```



Regression with polynomial expansion

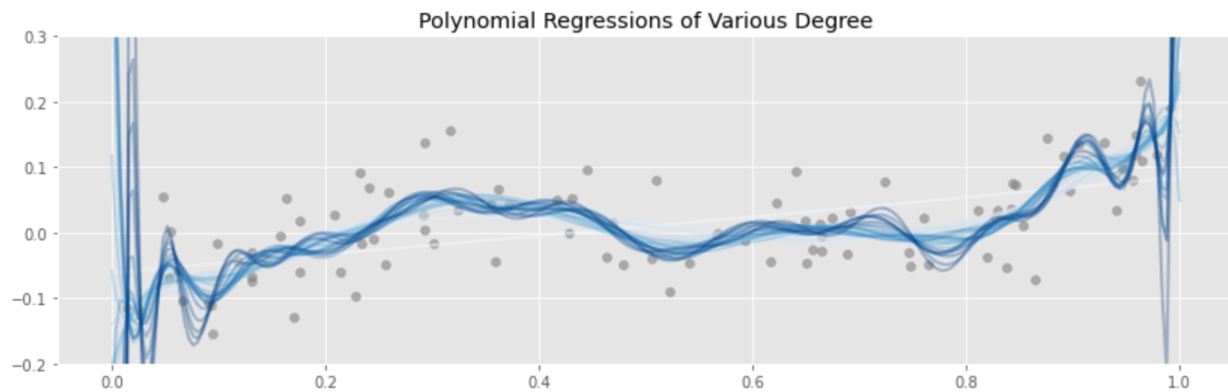
In [20]:
```
def make_polynomial_regression(degree):
    return Pipeline([
        ('std', StandardScaler()),
        ('poly', Polynomial(degree=degree)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

regressions = {}
for degree in range(1, 30):
    regressions[degree] = make_polynomial_regression(degree)
    regressions[degree].fit(x.reshape(-1, 1), y)
```

In [21]:
```
fig, ax = plt.subplots(2, 4, figsize=(14, 6))

t = np.linspace(0, 1, 250)
for i, ax in enumerate(ax.flatten()):
    degree = i + 1
    ax.plot(t, regressions[degree].predict(t.reshape(-1, 1)), linewidt
    ax.scatter(x, y, alpha=0.4, color="grey")
    ax.set_title("Polynomial Degree {}".format(degree))

fig.tight_layout()
```



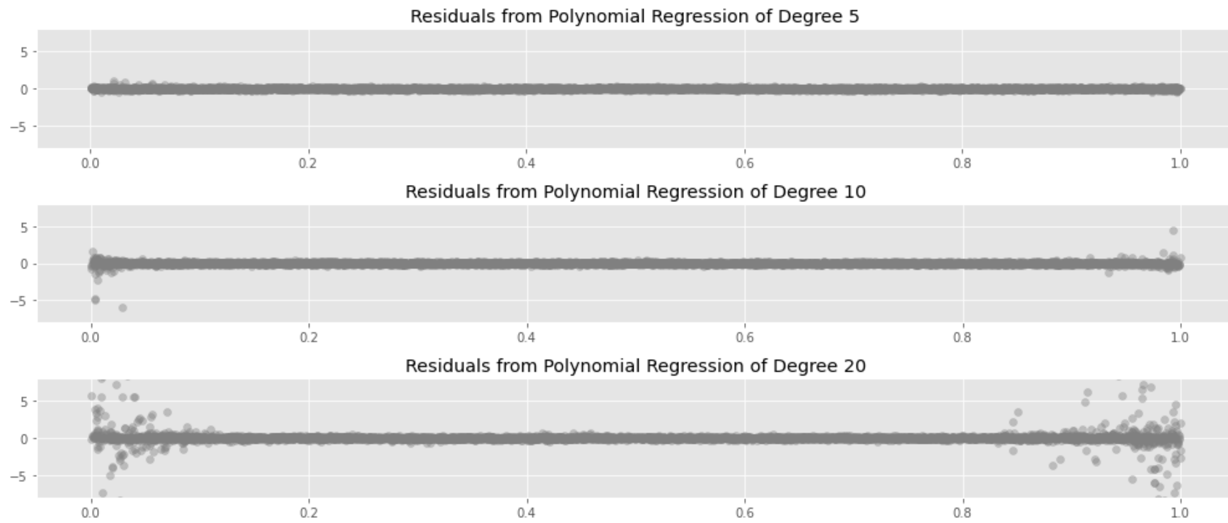
In [22]:
```
fig, ax = plt.subplots(figsize=(14, 4))

from matplotlib import cm

ax.scatter(x, y, alpha=0.6, color="grey")
colors = [cm.Blues(t) for t in np.linspace(0.0, 1.0, 30)]

for i, degree in enumerate(range(1, 30)):
    ax.plot(t, regressions[degree].predict(t.reshape(-1, 1)),
            linewidth=2, alpha=0.33, color=colors[i])
ax.set_ylim(-0.2, 0.3)
ax.set_title("Polynomial Regressions of Various Degree")
```

Out[22]:
```
Text(0.5, 1.0, 'Polynomial Regressions of Various Degree')
```



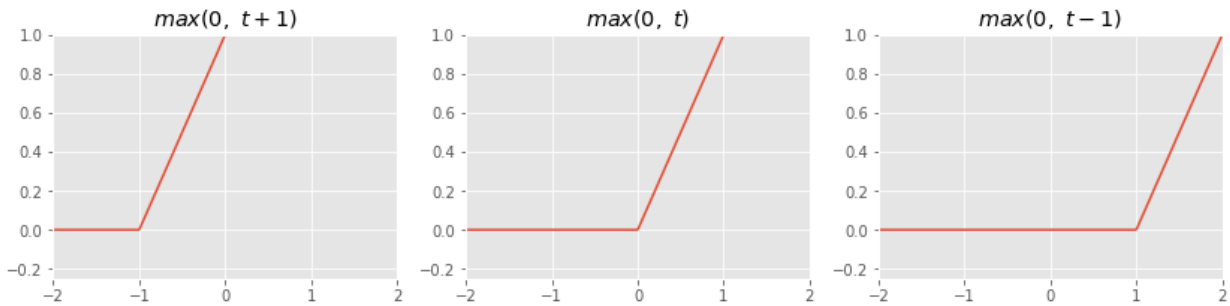
In [23]:
```
degrees = [5, 10, 20]
regressors = [make_polynomial_regression(degree) for degree in degrees
test_xs, test_errors = run_residual_simulation(signal, regressors, 50,
```

In [24]:
```
fig, axs = plt.subplots(len(degrees), figsize=(14, 6))

for (i, degree), sim in product(enumerate(degrees), range(100)):
    axs[i].scatter(test_xs[i, sim, :], test_errors[i, sim, :], color="
    axs[i].set_title("Residuals from Polynomial Regression of Degree {
    axs[i].set_ylim(-8, 8)
fig.tight_layout()
```



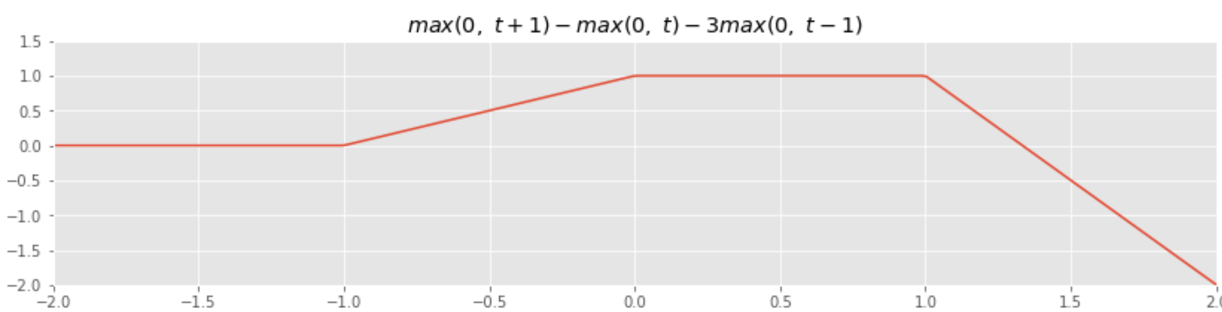
Regression with piecewise linear expansion

In [25]:
```
t = np.linspace(-2, 2, num=250)
knots = [-1, 0, 1]
titles = ["$max(0, \ t + 1)$", "$max(0, \ t)$", "$max(0, \ t - 1)$"]

fig, axs = plt.subplots(1, 3, figsize=(14, 3))

for i, (knot, title) in enumerate(zip(knots, titles)):
    axs[i].plot(t, np.maximum(0, t - knot))
    axs[i].set_ylim(-0.25, 1)
    axs[i].set_xlim(-2, 2)
    axs[i].set_title(title)
```



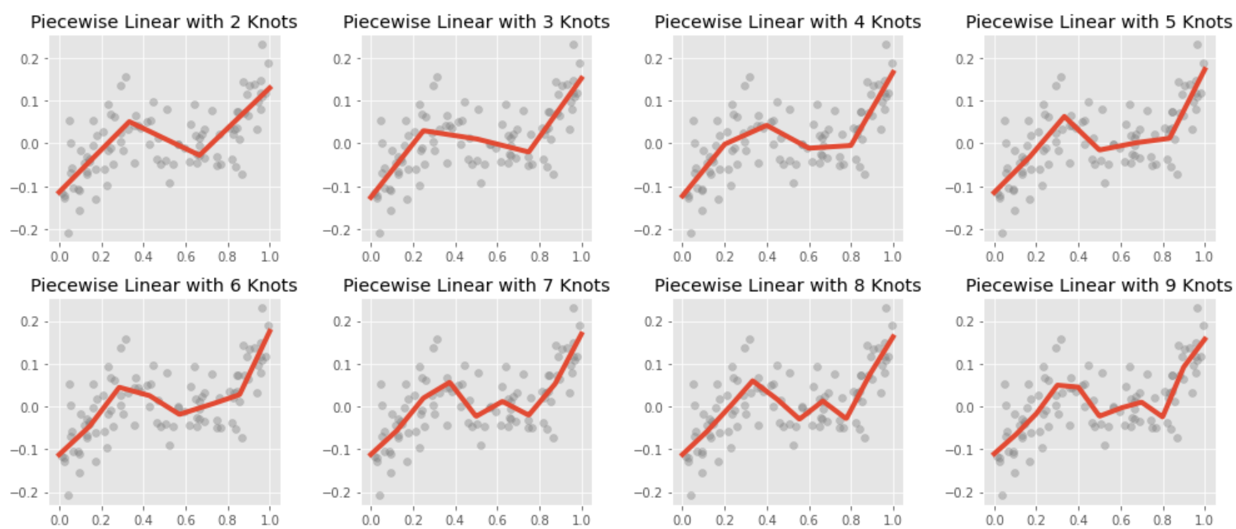
In [26]:
```
fig, ax = plt.subplots(figsize=(14, 3))

ax.plot(t, np.maximum(0, t + 1) - np.maximum(0, t) - 3*np.maximum(0, t
ax.set_ylim(-2, 1.5)
ax.set_xlim(-2, 2)
ax.set_title("$max(0, \ t + 1) - max(0, \ t) - 3 max(0, \ t - 1)$")
```

Out[26]:
```
Text(0.5, 1.0, '$max(0, \\ t + 1) - max(0, \\ t) - 3 max(0, \\ t - 1)
$')
```



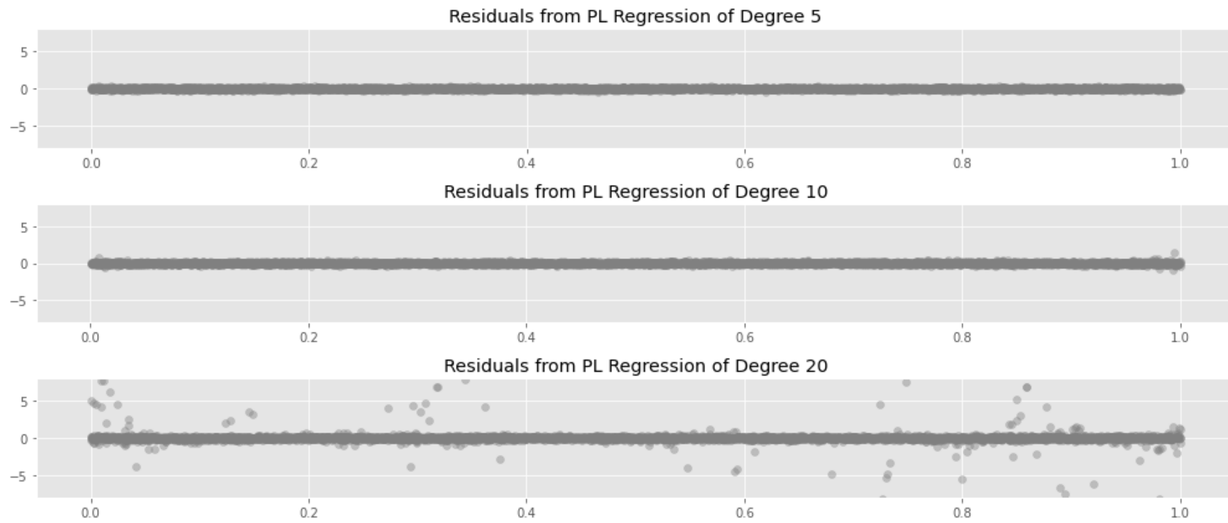
In [27]:
```
def make_pl_regression(n_knots):
    return Pipeline([
        ('pl', LinearSpline(0, 1, n_knots=n_knots)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

regressions = {}
for n_knots in range(2, 25):
    regressions[n_knots] = make_pl_regression(n_knots)
    regressions[n_knots].fit(x.reshape(-1, 1), y)
```

In [28]:
```
fig, ax = plt.subplots(2, 4, figsize=(14, 6))

t = np.linspace(0, 1, 250)
for i, ax in enumerate(ax.flatten()):
    n_knots = i + 2
    ax.plot(t, regressions[n_knots].predict(t.reshape(-1, 1)), linewic
    ax.scatter(x, y, alpha=0.4, color="grey")
    ax.set_title("Piecewise Linear with {} Knots".format(n_knots))

fig.tight_layout()
```



In [29]:
```
n_knots = [5, 10, 20]
regressors = [make_pl_regression(n_knot) for n_knot in n_knots]
test_xs, test_errors = run_residual_simulation(signal, regressors, 50,
```

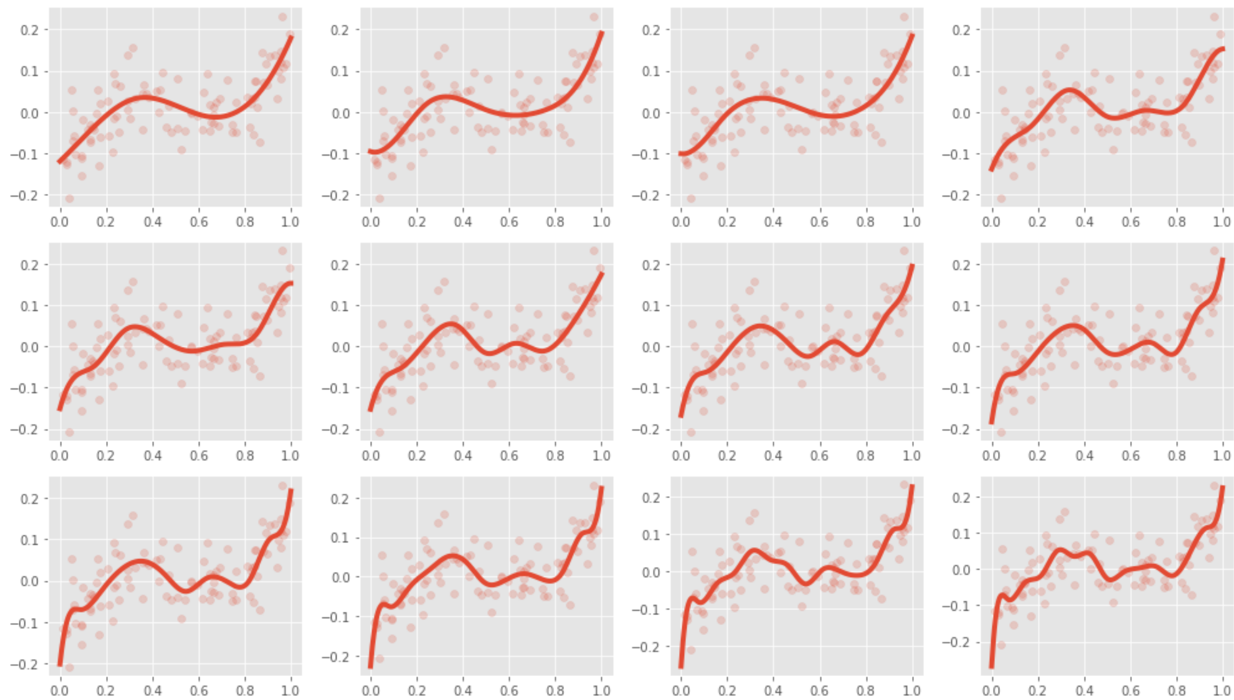
In [30]:
```
fig, axs = plt.subplots(len(degrees), figsize=(14, 6))

for (i, n_knot), sim in product(enumerate(n_knots), range(100)):
    axs[i].scatter(test_xs[i, sim, :], test_errors[i, sim, :], color="
    axs[i].set_title("Residuals from PL Regression of Degree {}".forma
    axs[i].set_ylim(-8, 8)
fig.tight_layout()
```



Regression with piecewise cubic expansion

In [31]:
```
def make_pw_cubic_regression(n_knots):
    return Pipeline([
        ('pw_cubic', CubicSpline(0, 1, n_knots=n_knots)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

regressions = {}
for n_knots in range(2, 25):
    regressions[n_knots] = make_pw_cubic_regression(n_knots)
    regressions[n_knots].fit(x.reshape(-1, 1), y)
```

In [32]:
```
fig, ax = plt.subplots(3, 4, figsize=(14, 8))

t = np.linspace(0, 1, 250)
for i, ax in enumerate(ax.flatten()):
    n_knots = i + 2
    ax.plot(t, regressions[n_knots].predict(t.reshape(-1, 1)), linewic
    ax.scatter(x, y, alpha=0.2)



fig.tight_layout()
```



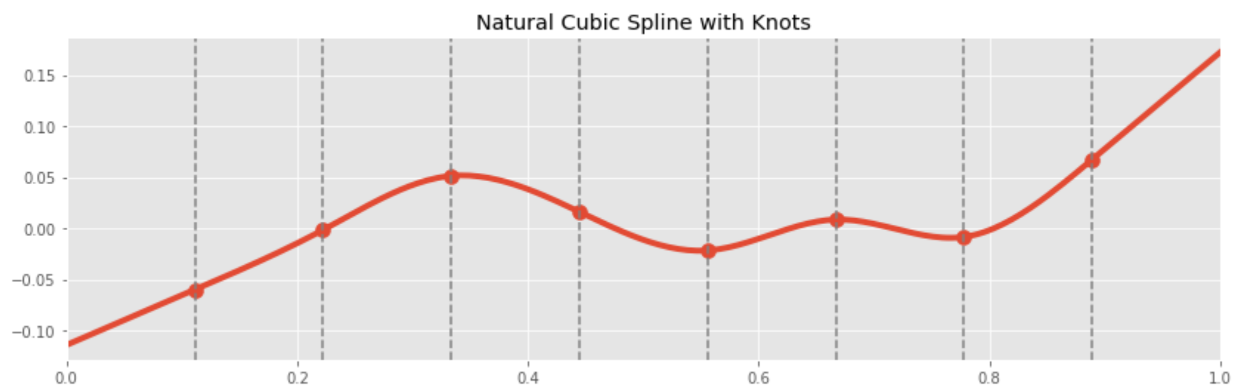
Regression with piecewise natural cubic expansion

In [33]:
```
def make_natural_cubic_regression(n_knots):
    return Pipeline([
        ('nat_cubic', NaturalCubicSpline(0, 1, n_knots=n_knots)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

regressions = {}
for n_knots in range(2, 25):
    regressions[n_knots] = make_natural_cubic_regression(n_knots)
    regressions[n_knots].fit(x.reshape(-1, 1), y)
```

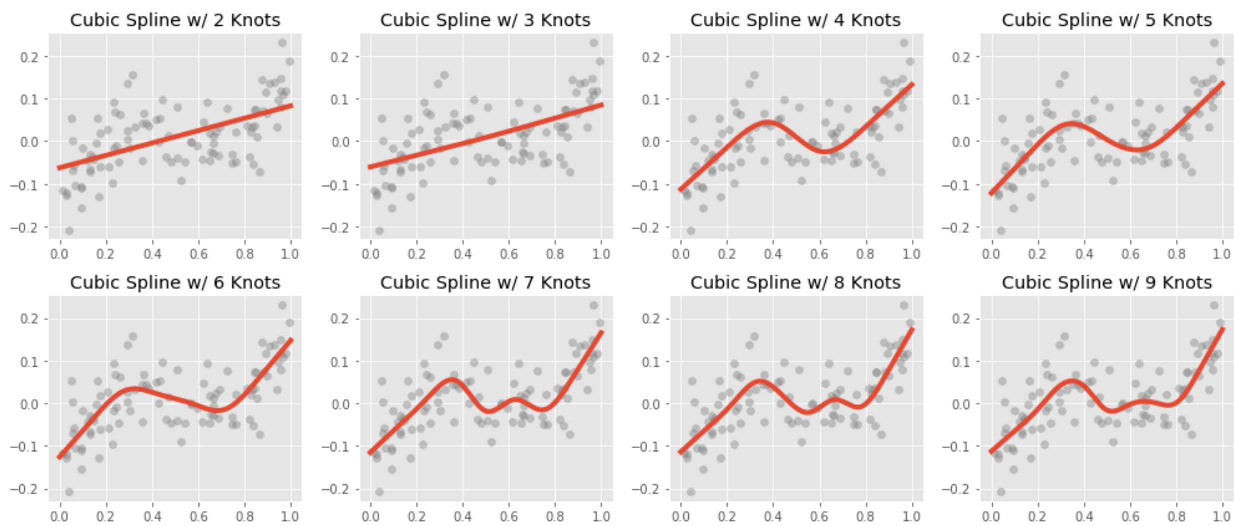
In [34]:
```
fig, ax = plt.subplots(figsize=(14, 4))
ax.plot(t, regressions[8].predict(t.reshape(-1, 1)), linewidth=4)

knots = regressions[8].get_params()['nat_cubic__knots']
ax.scatter([knots], regressions[8].predict(np.array(knots).reshape(-1,

for knot in knots:
    ax.axvline(knot, linestyle='--', color='grey')
ax.set_xlim(0, 1)
ax.set_title("Natural Cubic Spline with Knots")
```

```
/Users/mdrury/.pyenv/versions/3.8.0/envs/basis-expansions/lib/python3
.8/site-packages/sklearn/base.py:193: FutureWarning: From version 0.2
4, get_params will raise an AttributeError if a parameter cannot be r
etrieved as an instance attribute. Previously it would return None.
  warnings.warn('From version 0.24, get_params will raise an '
```

Out[34]:
```
Text(0.5, 1.0, 'Natural Cubic Spline with Knots')
```

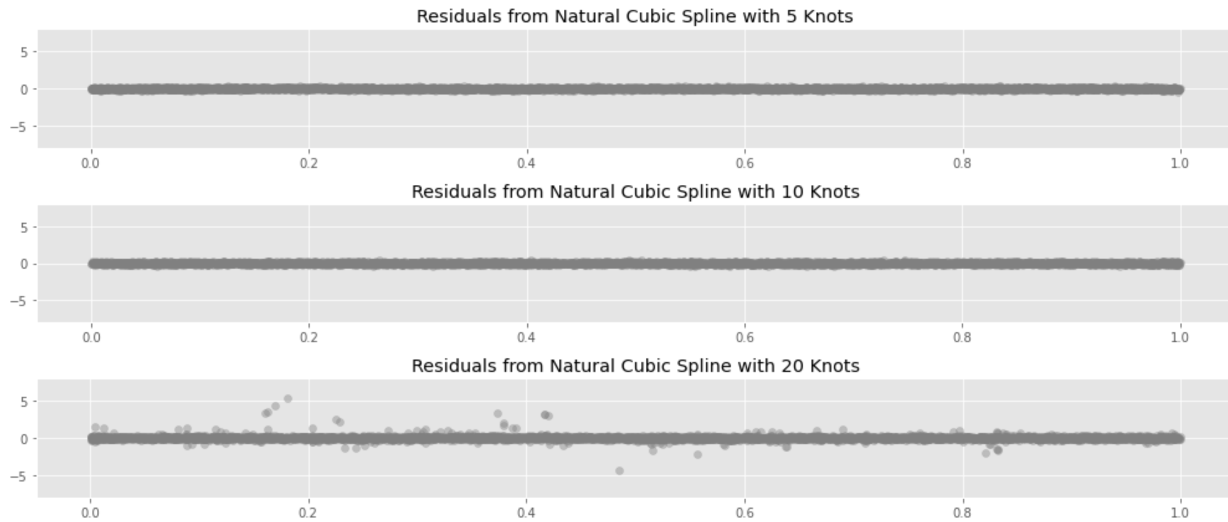


In [35]:
```
fig, ax = plt.subplots(2, 4, figsize=(14, 6))

t = np.linspace(0, 1, 250)
for i, ax in enumerate(ax.flatten()):
    n_knots = i + 2
    ax.plot(t, regressions[n_knots].predict(t.reshape(-1, 1)), linewic
    ax.scatter(x, y, alpha=0.4, color="grey")
    ax.set_title("Cubic Spline w/ {} Knots".format(n_knots))

fig.tight_layout()
```



In [36]:
```
n_knots = [5, 10, 20]
regressors = [make_natural_cubic_regression(n_knot) for n_knot in n_kr
test_xs, test_errors = run_residual_simulation(signal, regressors, 50,
```

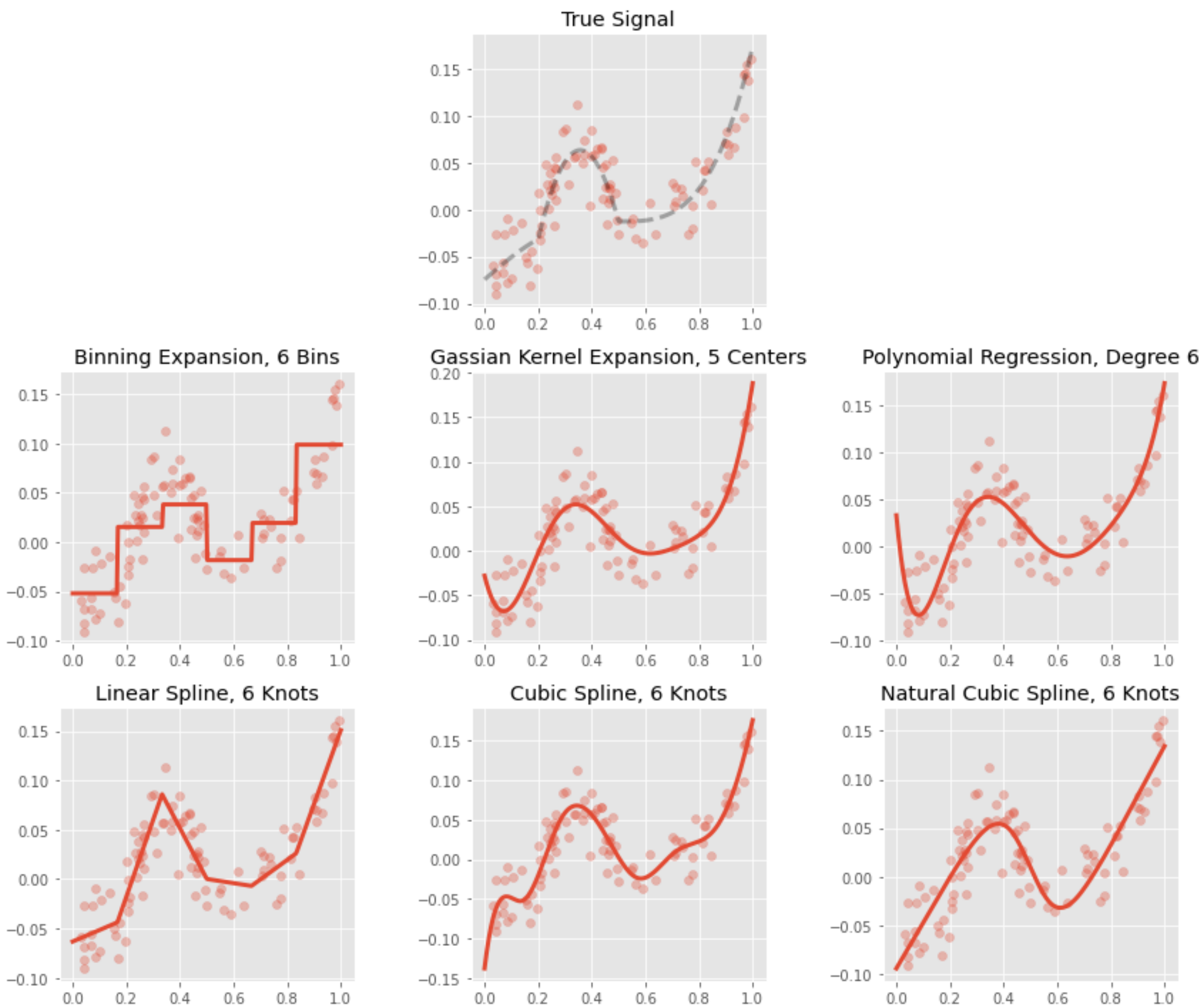
In [37]:
```
fig, axs = plt.subplots(len(degrees), figsize=(14, 6))

for (i, n_knot), sim in product(enumerate(n_knots), range(100)):
    axs[i].scatter(test_xs[i, sim, :], test_errors[i, sim, :], color="
    axs[i].set_title("Residuals from Natural Cubic Spline with {} Knot
    axs[i].set_ylim(-8, 8)
fig.tight_layout()
```



Examples of all

In [78]:
```
regressions = [
    {'model': make_binned_regression(5),
         'title': "Binning Expansion, 6 Bins"},

    {'model': make_gaussian_regression(5),
         'title': "Gassian Kernel Expansion, 5 Centers"},

    {'model': make_polynomial_regression(6) ,
         'title': "Polynomial Regression, Degree 6"},

    {'model': make_pl_regression(6),
          'title': "Linear Spline, 6 Knots"},

    {'model': make_pw_cubic_regression(6) ,
         'title': "Cubic Spline, 6 Knots"},

    {'model': make_natural_cubic_regression(6),
         'title': "Natural Cubic Spline, 6 Knots"}
]

for reg in regressions:
    reg['model'].fit(x.reshape(-1, 1), y)
```

In [79]:
```
fig, axs = plt.subplots(3, 3, figsize=(12, 10))

axs[0, 1].scatter(x, y, alpha=0.33)
axs[0, 1].plot(t, signal(t), linewidth=3, linestyle="--",
               color="black", alpha=0.3)
axs[0, 1].set_title("True Signal")

axs[0, 0].axis('off')
axs[0, 2].axis('off')

for i, ax in enumerate(axs[1:, :].flatten()):
    ax.scatter(x, y, alpha=0.33)
    ax.plot(t, regressions[i]['model'].predict(t.reshape(-1, 1)), line
    ax.set_title(regressions[i]['title'])

fig.tight_layout()
```



Investigating Performance with Different Smoothers

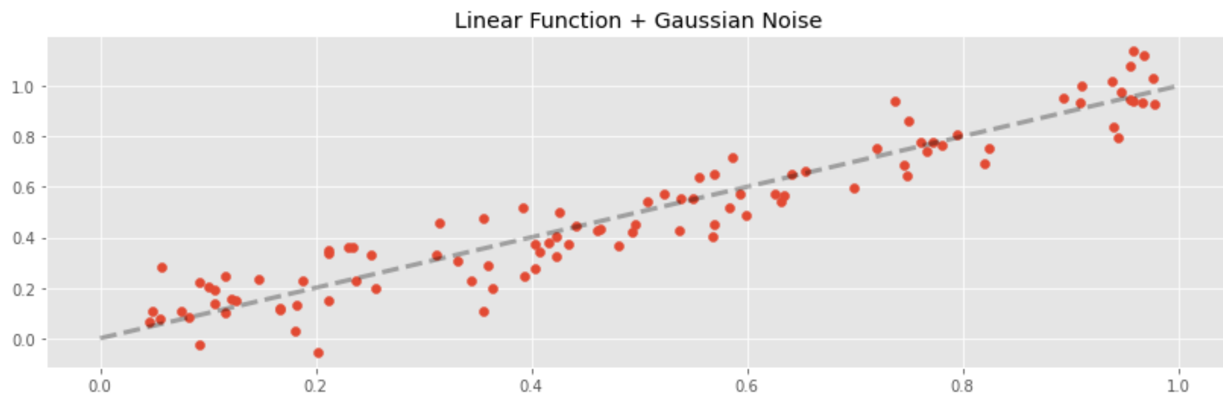
In [42]:
```
def linear_signal(x):
    return x

def sin_signal(x):
    return np.sin(2*np.pi*x)

cutpoints = sorted(np.random.uniform(size=6))
def broken_sin_signal(x):
    return  (np.sin(2*np.pi*x)
             - (cutpoints[0] <= x)*(x <= cutpoints[2])
             - (cutpoints[1] <= x)*(x <= cutpoints[2])
             - 2*(cutpoints[3] <= x)*(x <= cutpoints[4]))

def weird_signal(x):
    return (x*x*x*(x-1)
            + 2*(1/(1 + np.exp(-0.5*(x - 0.5))))
            - 3.5*(x > 0.2)*(x < 0.5)*(x - 0.2)*(x - 0.5)
            - 0.95)
```

In [43]:
```
degrees_of_freedom = list(range(2, 30))

def make_binned_regression(n_params):
    return Pipeline([
        ('binner', Binner(0, 1, n_params=n_params)),
        ('regression', LinearRegression(fit_intercept=False))
    ])

def make_polynomial_regression(n_params):
    return Pipeline([
        ('std', StandardScaler()),
        ('poly', Polynomial(n_params=n_params)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

def make_pl_regression(n_params):
    return Pipeline([
        ('pl', LinearSpline(0, 1, n_params=n_params)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

def make_natural_cubic_regression(n_params):
    return Pipeline([
        ('nat_cubic', NaturalCubicSpline(0, 1, n_params=n_params)),
        ('regression', LinearRegression(fit_intercept=True))
    ])

def make_non_linear_regressions(regression_maker, degrees_of_freedom):
    return [regression_maker(dof) for dof in degrees_of_freedom]

binned_regressors = make_non_linear_regressions(make_binned_regressior
                                                degrees_of_freedom)
polynomial_regressors = make_non_linear_regressions(make_polynomial_re
                                                    degrees_of_freedom
pl_regressors = make_non_linear_regressions(make_pl_regression,
                                            degrees_of_freedom)
ncs_regressors = make_non_linear_regressions(make_natural_cubic_regres
                                             degrees_of_freedom)

regressors = {
    "binned": binned_regressors,
    "polynomial": polynomial_regressors,
    "pl": pl_regressors,
    "ncs": ncs_regressors
}
```

Fitting to a Linear Signal

In [44]:
```
fig, ax = plt.subplots(figsize=(14, 4))
t = np.linspace(0, 1, num=250)

x = np.random.uniform(size=100)
y = linear_signal(x) + np.random.normal(scale=0.1, size=100)
ax.scatter(x, y)
ax.plot(t, linear_signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.3)

ax.set_title("Linear Function + Gaussian Noise")
```

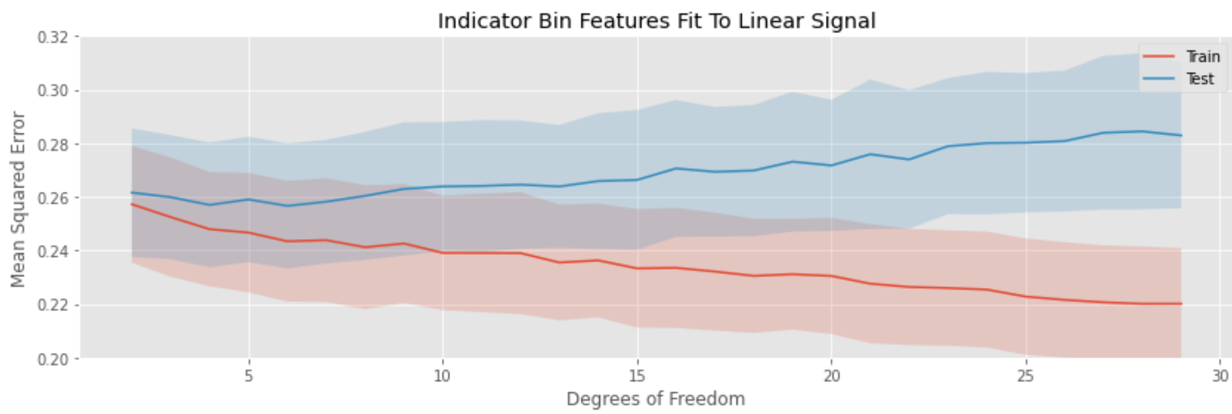
Out[44]:
```
Text(0.5, 1.0, 'Linear Function + Gaussian Noise')
```



In [45]:
```
binned_mean_errors, binned_std_errors = run_simulation_expreiment(
    linear_signal, binned_regressors, sd=0.5)
polynomial_mean_errors, polynomial_std_errors = run_simulation_expreim
    linear_signal, polynomial_regressors, sd=0.5)
pl_mean_errors, pl_std_errors = run_simulation_expreiment(
    linear_signal, pl_regressors, sd=0.5)
ncs_mean_errors, ncs_std_errors = run_simulation_expreiment(
    linear_signal, ncs_regressors, sd=0.5)
```

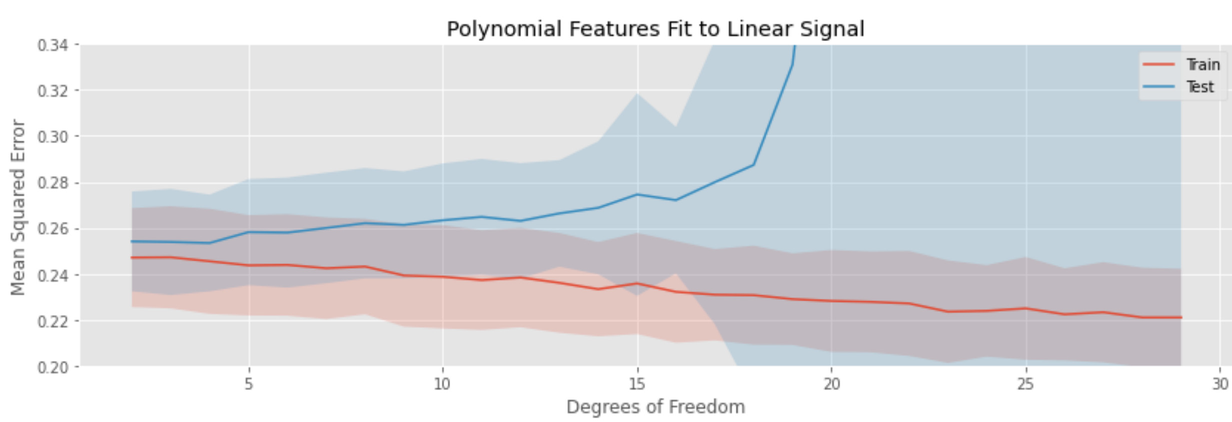
In [46]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, binned_mean_errors, binned_std_errors)
ax.set_ylim(0.2, 0.32)
ax.set_title("Indicator Bin Features Fit To Linear Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[46]:
```
Text(0, 0.5, 'Mean Squared Error')
```



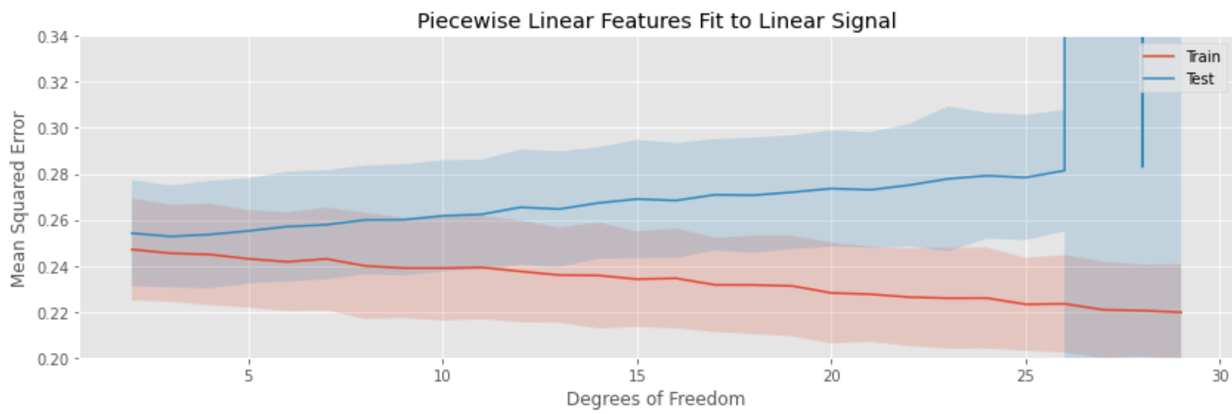
In [47]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, polynomial_mean_errors, polynomial_std_errors)
ax.set_ylim(0.2, 0.34)
ax.set_title("Polynomial Features Fit to Linear Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[47]:
```
Text(0, 0.5, 'Mean Squared Error')
```



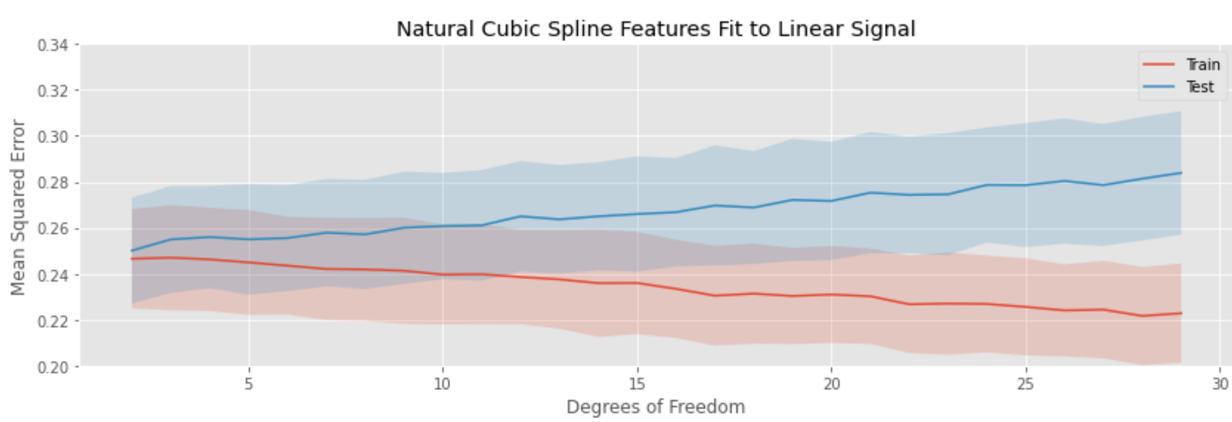
In [48]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, pl_mean_errors, pl_std_errors)
ax.set_ylim(0.2, 0.34)
ax.set_title("Piecewise Linear Features Fit to Linear Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[48]: Text(0, 0.5, 'Mean Squared Error')



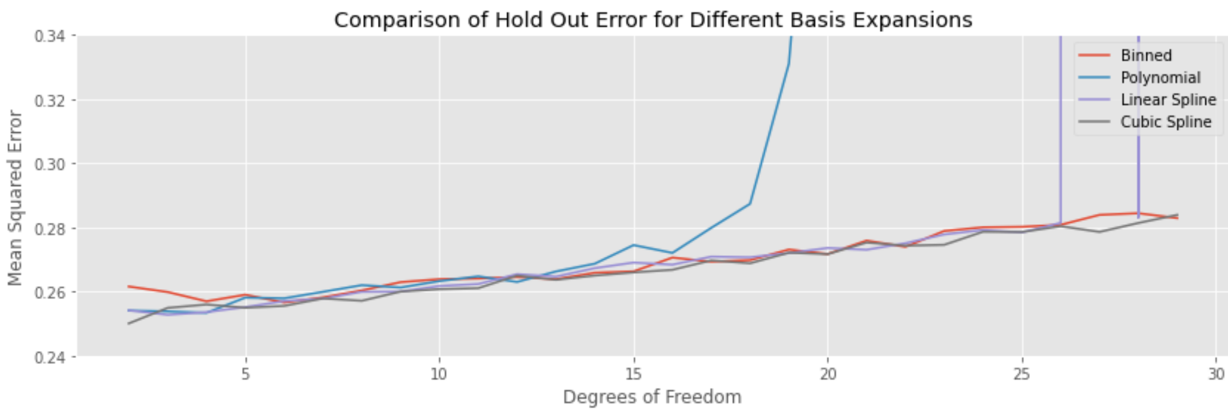
In [49]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, ncs_mean_errors, ncs_std_errors)
ax.set_ylim(0.2, 0.34)
ax.set_title("Natural Cubic Spline Features Fit to Linear Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[49]: Text(0, 0.5, 'Mean Squared Error')



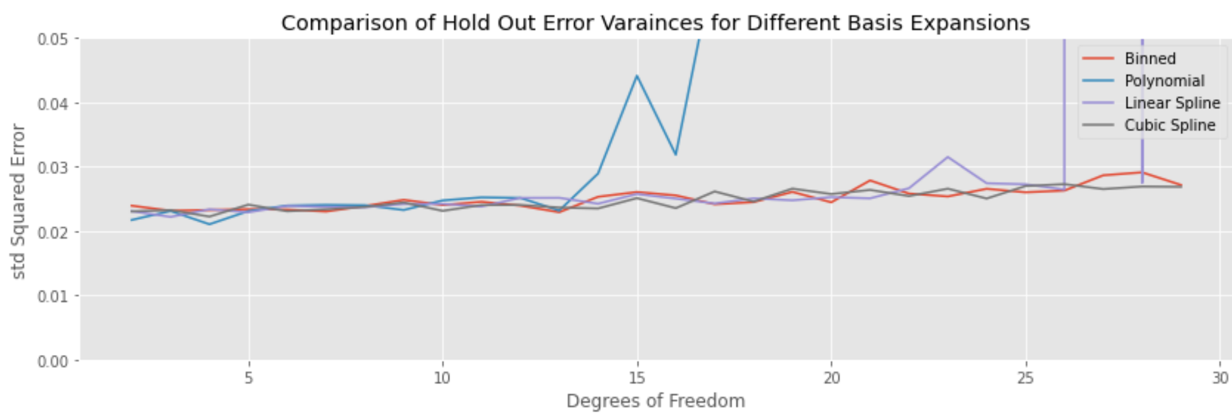
In [50]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_mean_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_mean_errors[1], label="Polynomi
ax.plot(degrees_of_freedom, pl_mean_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_mean_errors[1], label="Cubic Spline")
ax.set_ylim(0.24, 0.34)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
ax.set_title("Comparison of Hold Out Error for Different Basis Expansi
```

Out[50]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error for Different Basis Expa
nsions')
```



In [51]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_std_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_std_errors[1], label="Polynomia
ax.plot(degrees_of_freedom, pl_std_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_std_errors[1], label="Cubic Spline")
ax.set_ylim(0.0, 0.05)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("std Squared Error")
ax.set_title("Comparison of Hold Out Error Varainces for Different Bas
```

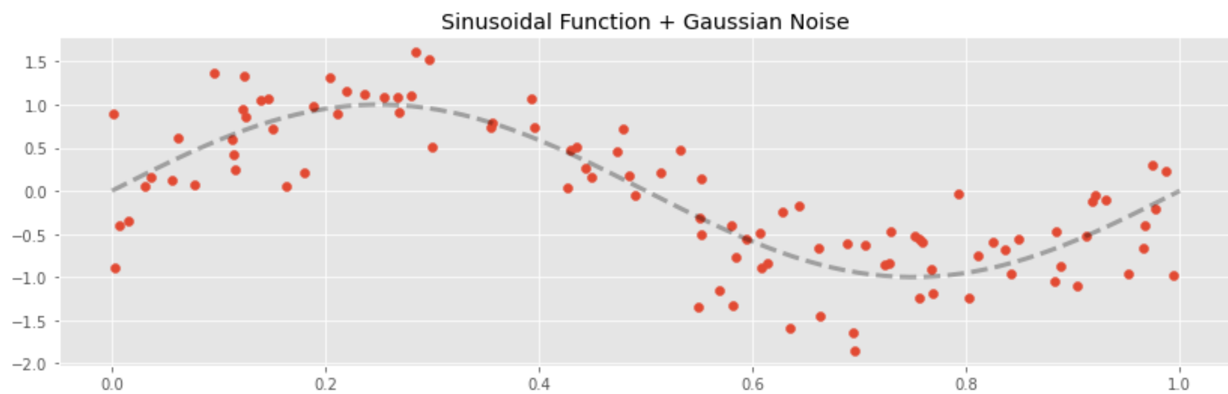
Out[51]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error Varainces for Different
Basis Expansions')
```



Fitting to a Sinusoidal Signal

In [52]:
```
fig, ax = plt.subplots(figsize=(14, 4))
t = np.linspace(0, 1, num=250)

x = np.random.uniform(size=100)
y = sin_signal(x) + np.random.normal(scale=0.5, size=100)
ax.scatter(x, y)
ax.plot(t, sin_signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.3)
ax.set_title("Sinusoidal Function + Gaussian Noise")
```

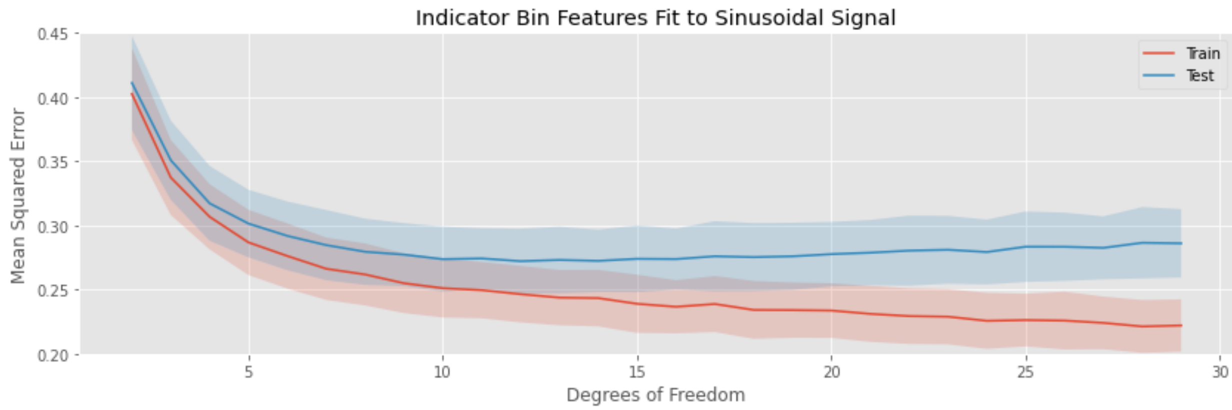
Out[52]:
```
Text(0.5, 1.0, 'Sinusoidal Function + Gaussian Noise')
```



In [53]:
```
binned_mean_errors, binned_std_errors = run_simulation_expreiment(
    sin_signal, binned_regressors, sd=0.5)
polynomial_mean_errors, polynomial_std_errors = run_simulation_expreim
    sin_signal, polynomial_regressors, sd=0.5)
pl_mean_errors, pl_std_errors = run_simulation_expreiment(
    sin_signal, pl_regressors, sd=0.5)
ncs_mean_errors, ncs_std_errors = run_simulation_expreiment(
    sin_signal, ncs_regressors, sd=0.5)
```

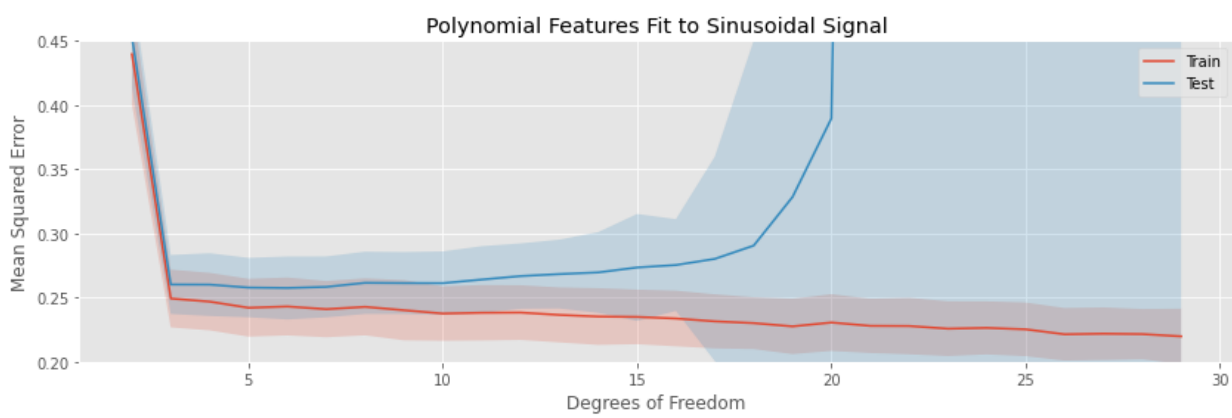
In [54]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, binned_mean_errors, binned_std_errors)
ax.set_ylim(0.2, 0.45)
ax.set_title("Indicator Bin Features Fit to Sinusoidal Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[54]: Text(0, 0.5, 'Mean Squared Error')



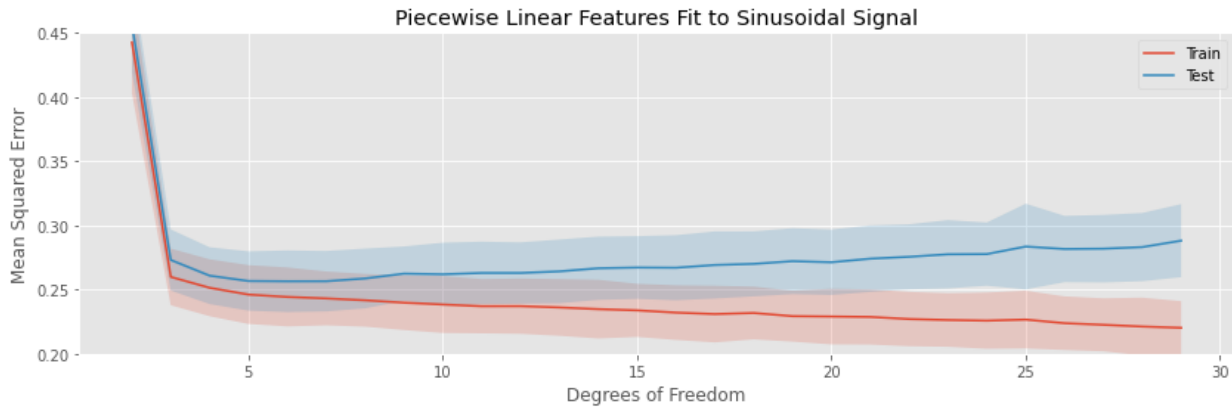
In [55]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, polynomial_mean_errors, polynomial_std_errors)
ax.set_ylim(0.2, 0.45)
ax.set_title("Polynomial Features Fit to Sinusoidal Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[55]: Text(0, 0.5, 'Mean Squared Error')



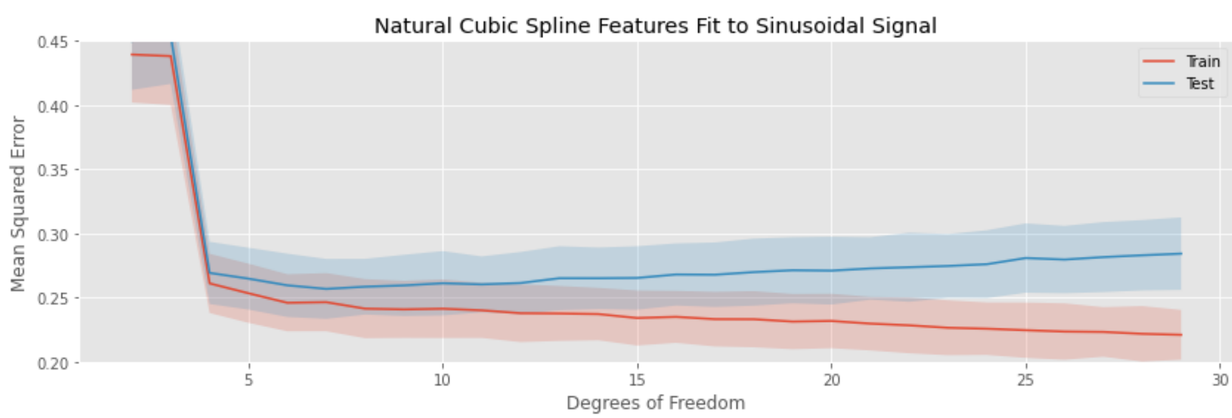
In [56]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, pl_mean_errors, pl_std_errors)
ax.set_ylim(0.2, 0.45)
ax.set_title("Piecewise Linear Features Fit to Sinusoidal Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[56]:
```
Text(0, 0.5, 'Mean Squared Error')
```



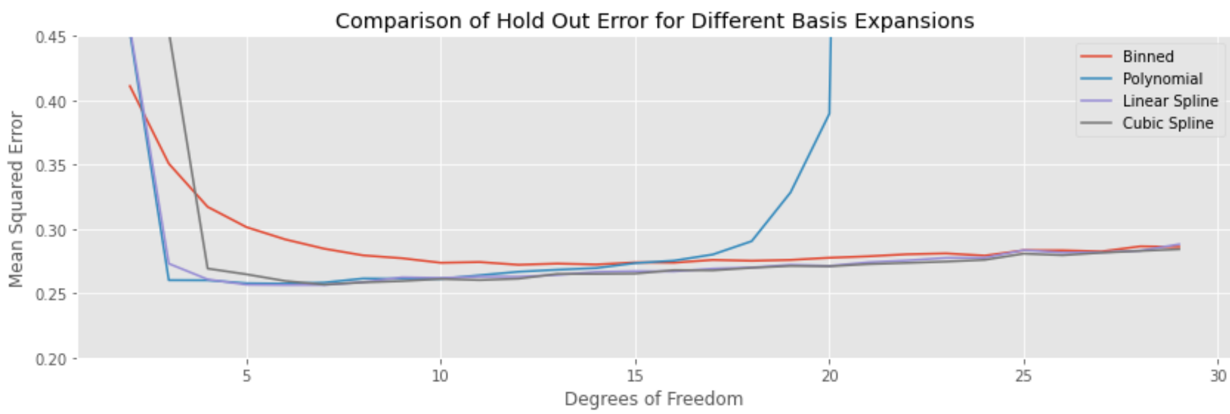
In [57]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, ncs_mean_errors, ncs_std_errors)
ax.set_ylim(0.2, 0.45)
ax.set_title("Natural Cubic Spline Features Fit to Sinusoidal Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[57]:
```
Text(0, 0.5, 'Mean Squared Error')
```



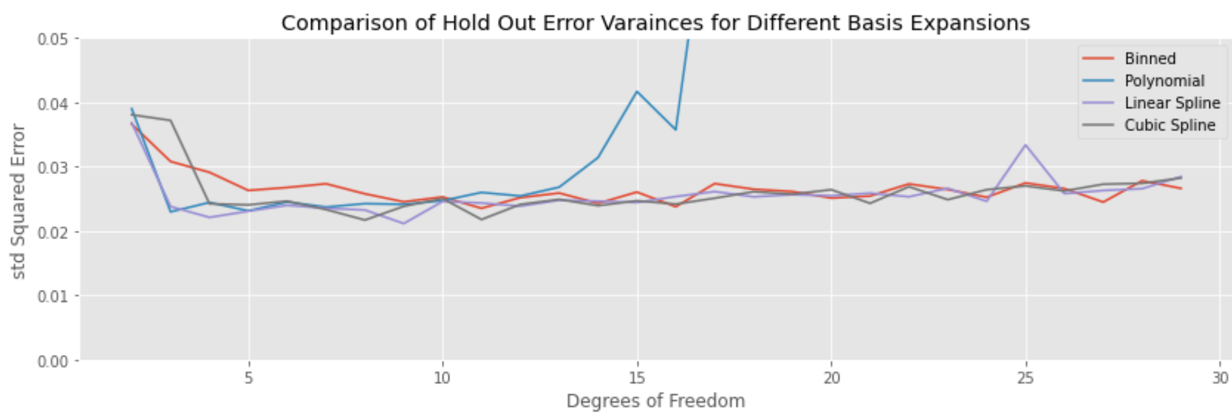
In [58]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_mean_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_mean_errors[1], label="Polynomi
ax.plot(degrees_of_freedom, pl_mean_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_mean_errors[1], label="Cubic Spline")
ax.set_ylim(0.2, 0.45)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
ax.set_title("Comparison of Hold Out Error for Different Basis Expansi
```

Out[58]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error for Different Basis Expa
nsions')
```



In [59]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_std_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_std_errors[1], label="Polynomia
ax.plot(degrees_of_freedom, pl_std_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_std_errors[1], label="Cubic Spline")
ax.set_ylim(0.0, 0.05)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("std Squared Error")
ax.set_title("Comparison of Hold Out Error Varainces for Different Bas
```

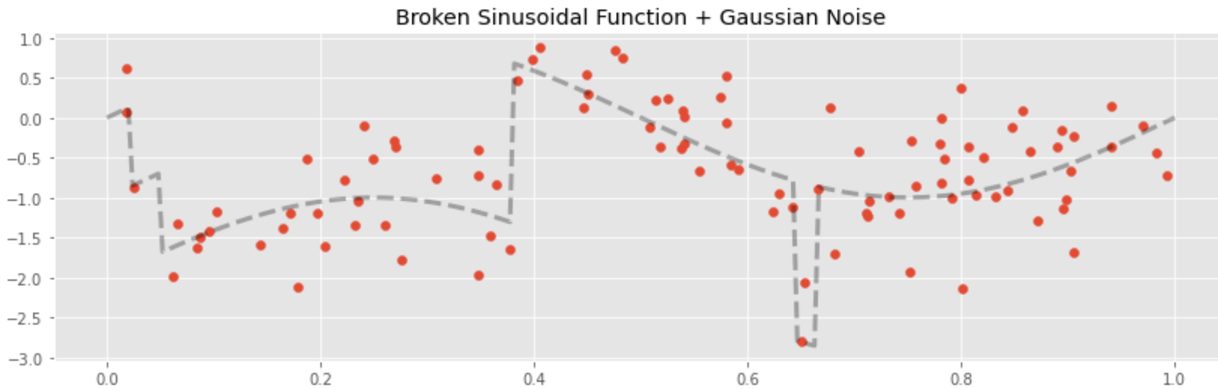
Out[59]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error Varainces for Different
Basis Expansions')
```



Fitting to a Broken Sin Signal

In [60]:
```
fig, ax = plt.subplots(figsize=(14, 4))
t = np.linspace(0, 1, num=250)

x = np.random.uniform(size=100)
y = broken_sin_signal(x) + np.random.normal(scale=0.5, size=100)
ax.scatter(x, y)
ax.plot(t, broken_sin_signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.3)
ax.set_title("Broken Sinusoidal Function + Gaussian Noise")
```

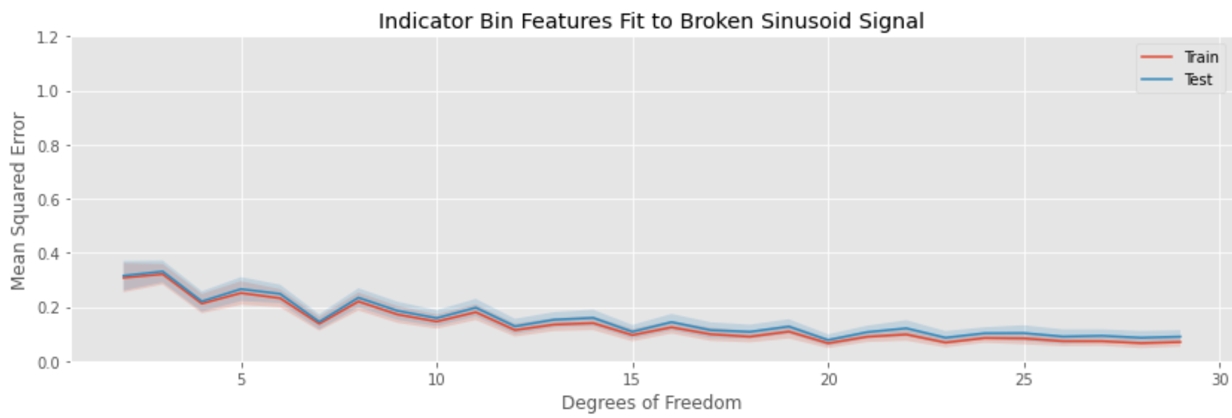
Out[60]:
```
Text(0.5, 1.0, 'Broken Sinusoidal Function + Gaussian Noise')
```



In [61]:
```
binned_mean_errors, binned_std_errors = run_simulation_expreiment(
    broken_sin_signal, binned_regressors, sd=0.05)
polynomial_mean_errors, polynomial_std_errors = run_simulation_expreim
    broken_sin_signal, polynomial_regressors, sd=0.05)
pl_mean_errors, pl_std_errors = run_simulation_expreiment(
    broken_sin_signal, pl_regressors, sd=0.05)
ncs_mean_errors, ncs_std_errors = run_simulation_expreiment(
    broken_sin_signal, ncs_regressors, sd=0.05)
```

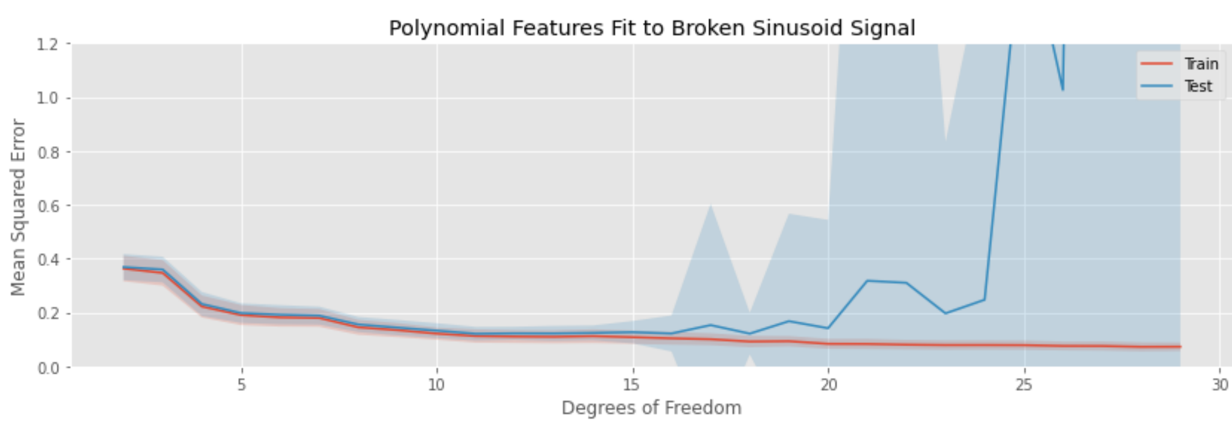
In [62]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, binned_mean_errors, binned_std_errors)
ax.set_ylim(0.0, 1.2)
ax.set_title("Indicator Bin Features Fit to Broken Sinusoid Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[62]: Text(0, 0.5, 'Mean Squared Error')



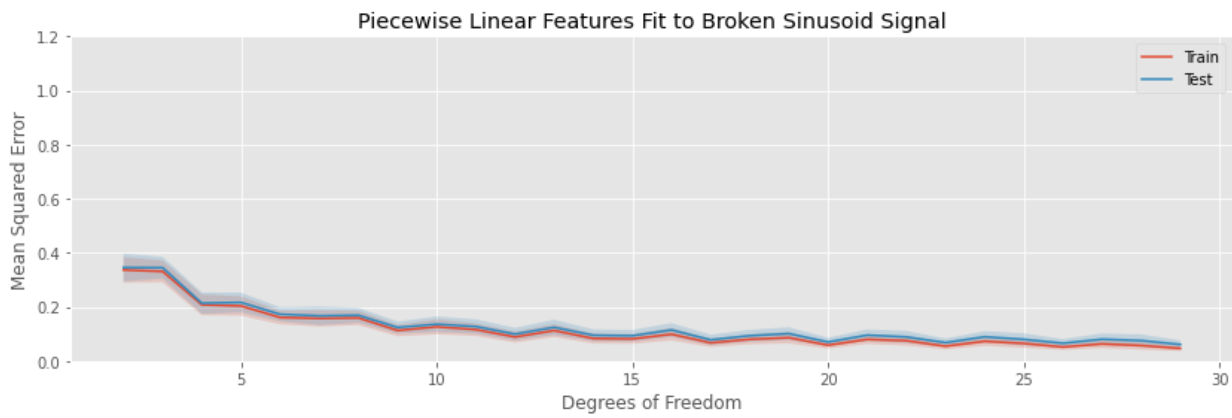
In [63]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, polynomial_mean_errors, polynomial_std_errors)
ax.set_ylim(0.0, 1.2)
ax.set_title("Polynomial Features Fit to Broken Sinusoid Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[63]: Text(0, 0.5, 'Mean Squared Error')



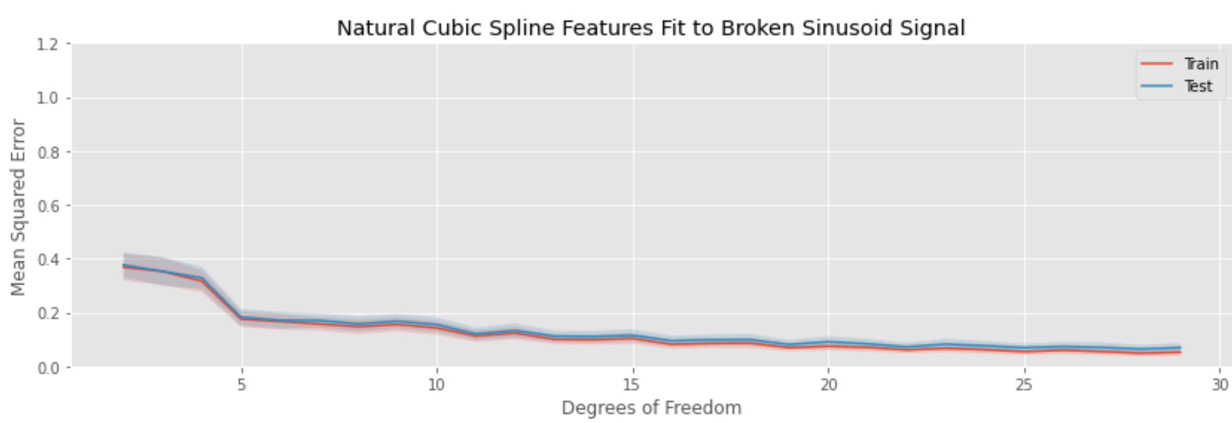
In [64]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, pl_mean_errors, pl_std_errors)
ax.set_ylim(0.0, 1.2)
ax.set_title("Piecewise Linear Features Fit to Broken Sinusoid Signal"
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[64]: Text(0, 0.5, 'Mean Squared Error')



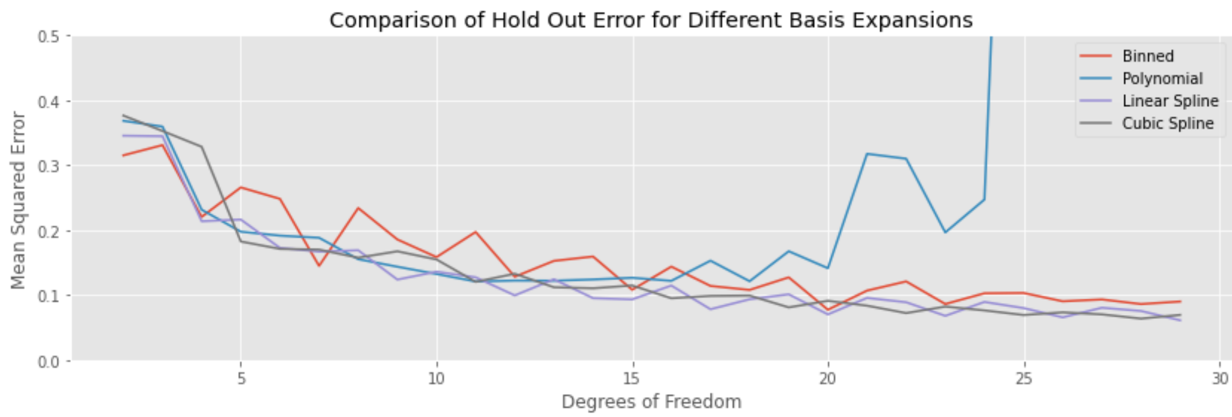
In [65]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, ncs_mean_errors, ncs_std_errors)
ax.set_ylim(0.0, 1.2)
ax.set_title("Natural Cubic Spline Features Fit to Broken Sinusoid Sig
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[65]: Text(0, 0.5, 'Mean Squared Error')



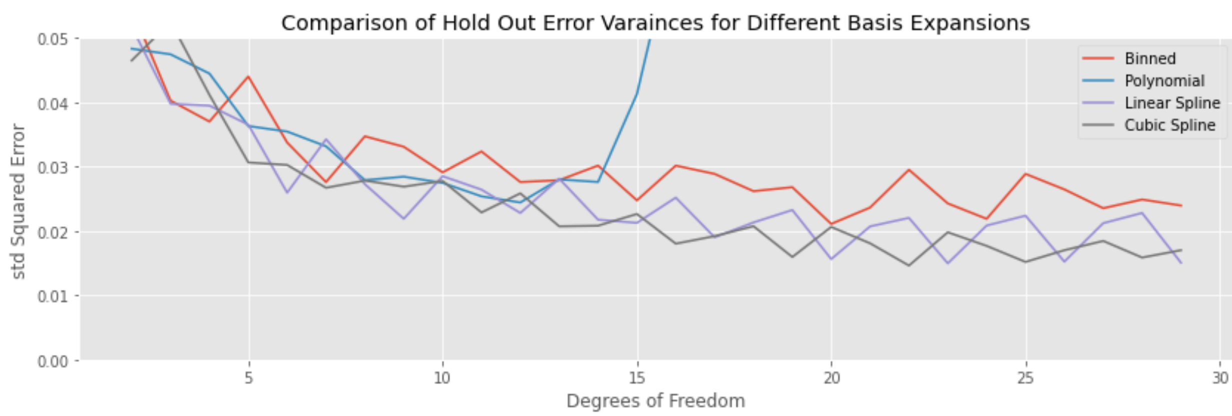
In [66]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_mean_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_mean_errors[1], label="Polynomi
ax.plot(degrees_of_freedom, pl_mean_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_mean_errors[1], label="Cubic Spline")
ax.set_ylim(0.0, 0.5)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
ax.set_title("Comparison of Hold Out Error for Different Basis Expansi
```

Out[66]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error for Different Basis Expa
nsions')
```



In [67]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_std_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_std_errors[1], label="Polynomia
ax.plot(degrees_of_freedom, pl_std_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_std_errors[1], label="Cubic Spline")
ax.set_ylim(0.0, 0.05)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("std Squared Error")
ax.set_title("Comparison of Hold Out Error Varainces for Different Bas
```

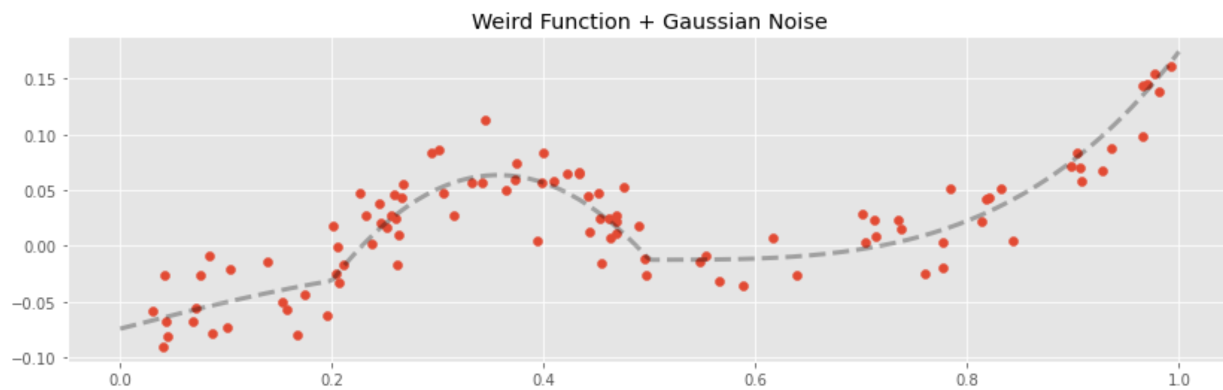
Out[67]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error Varainces for Different
Basis Expansions')
```



Fitting to a weird signal

In [68]:
```
fig, ax = plt.subplots(figsize=(14, 4))
t = np.linspace(0, 1, num=250)

x = np.random.uniform(size=100)
y = weird_signal(x) + np.random.normal(scale=0.025, size=100)
ax.scatter(x, y)
ax.plot(t, weird_signal(t), linewidth=3, linestyle="--",
        color="black", alpha=0.3)
ax.set_title("Weird Function + Gaussian Noise")
```

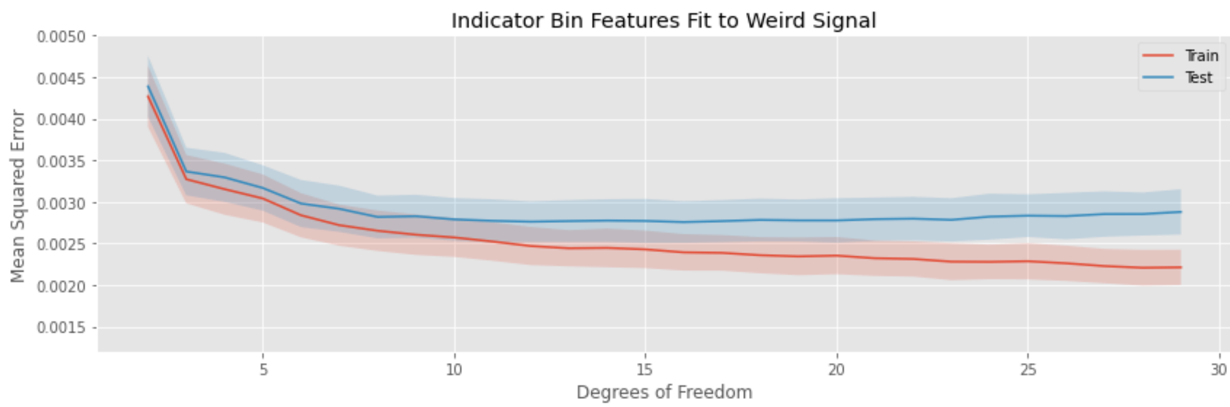
Out[68]:
```
Text(0.5, 1.0, 'Weird Function + Gaussian Noise')
```



In [69]:
```
binned_mean_errors, binned_std_errors = run_simulation_expreiment(
    weird_signal, binned_regressors, sd=0.05)
polynomial_mean_errors, polynomial_std_errors = run_simulation_expreim
    weird_signal, polynomial_regressors, sd=0.05)
pl_mean_errors, pl_std_errors = run_simulation_expreiment(
    weird_signal, pl_regressors, sd=0.05)
ncs_mean_errors, ncs_std_errors = run_simulation_expreiment(
    weird_signal, ncs_regressors, sd=0.05)
```

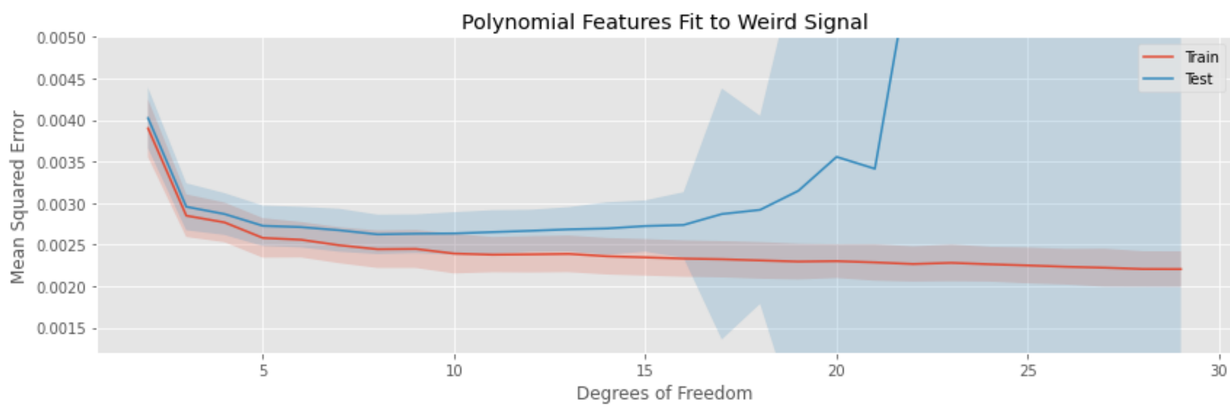
In [70]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, binned_mean_errors, binned_std_errors)
ax.set_ylim(0.0012, 0.005)
ax.set_title("Indicator Bin Features Fit to Weird Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[70]:
```
Text(0, 0.5, 'Mean Squared Error')
```



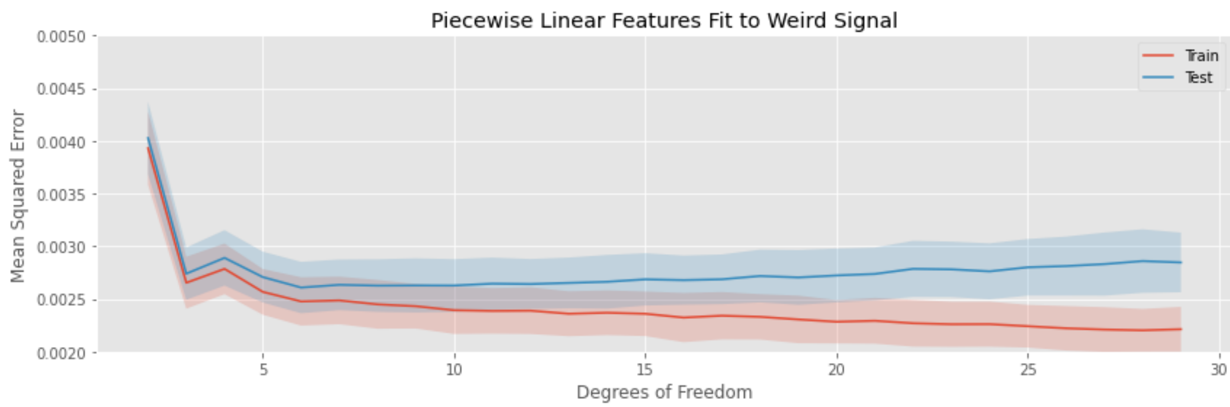
In [71]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, polynomial_mean_errors, polynomial_std_errors)
ax.set_ylim(0.0012, 0.005)
ax.set_title("Polynomial Features Fit to Weird Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[71]:
```
Text(0, 0.5, 'Mean Squared Error')
```



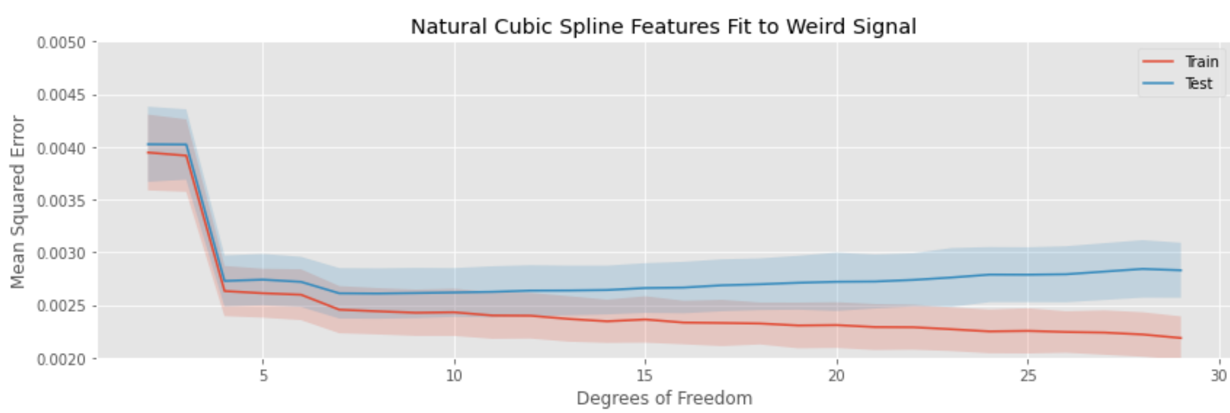
In [72]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, pl_mean_errors, pl_std_errors)
ax.set_ylim(0.002, 0.005)
ax.set_title("Piecewise Linear Features Fit to Weird Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[72]:
```
Text(0, 0.5, 'Mean Squared Error')
```



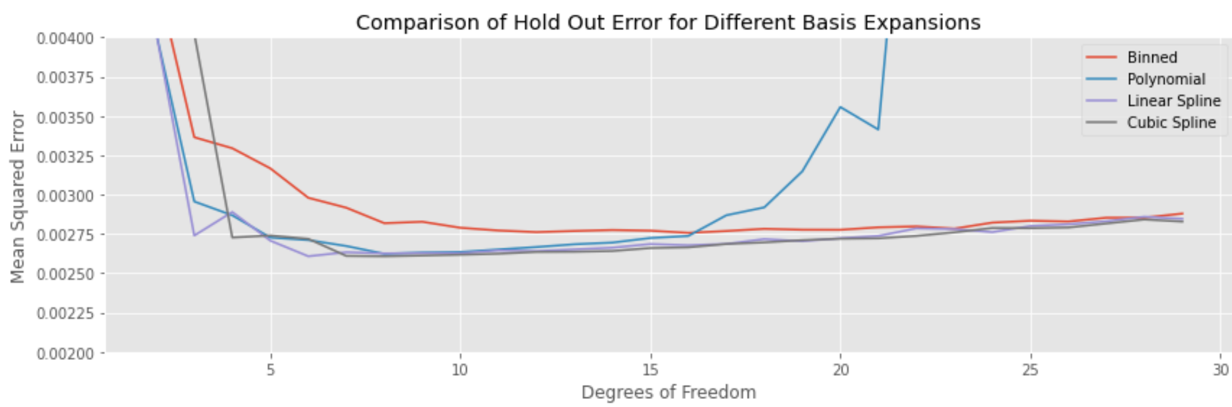
In [73]:
```
fig, ax = plt.subplots(figsize=(14, 4))
plot_simulation_expreiment(ax,
    degrees_of_freedom, ncs_mean_errors, ncs_std_errors)
ax.set_ylim(0.002, 0.005)
ax.set_title("Natural Cubic Spline Features Fit to Weird Signal")
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
```

Out[73]:
```
Text(0, 0.5, 'Mean Squared Error')
```



In [74]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_mean_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_mean_errors[1], label="Polynomi
ax.plot(degrees_of_freedom, pl_mean_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_mean_errors[1], label="Cubic Spline")
ax.set_ylim(0.002, 0.004)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("Mean Squared Error")
ax.set_title("Comparison of Hold Out Error for Different Basis Expansi
```

Out[74]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error for Different Basis Expa
nsions')
```



In [75]:
```
fig, ax = plt.subplots(figsize=(14, 4))

ax.plot(degrees_of_freedom, binned_std_errors[1], label="Binned")
ax.plot(degrees_of_freedom, polynomial_std_errors[1], label="Polynomia
ax.plot(degrees_of_freedom, pl_std_errors[1], label="Linear Spline")
ax.plot(degrees_of_freedom, ncs_std_errors[1], label="Cubic Spline")
ax.set_ylim(0.0, 0.001)

ax.legend()
ax.set_xlabel("Degrees of Freedom")
ax.set_ylabel("std Squared Error")
ax.set_title("Comparison of Hold Out Error Varainces for Different Bas
```

Out[75]:
```
Text(0.5, 1.0, 'Comparison of Hold Out Error Varainces for Different
Basis Expansions')
```