compare_all_Rmodels_QNJL

Compare performance on nonlinear data (with all regression algorithms)¶

Quistian Navarro Juan Luis \ A341807@alumnos.uaslp.mx \ Ing. Sistemas Inteligentes, Gen 2021 \ Machine Learning, Group 281601

02/19/24¶

Abstract¶

In this notebook, we delve into a comprehensive analysis of regression models. Our exploration focuses on comparing the performance of various regression models using a dataset that includes real data (DS-5-1-1-GAP-0-1-N-0_v2) along with noise data from DS-5-1-1-GAP-1-1-1-N-1_v2 and DS-5-1-GAP-5-1-1-N-3_v2.

The activity focuses on employing 100 realizations per noise level. Our evaluation criteria are multifaceted, incorporating metrics such as Mean Squared Error (MSE) for both the training and testing phases in the initial activity. In addition, in the second activity, we augment our evaluation to encompass bias and variance considerations, thus providing a more complete understanding of model performance under varying conditions. Through this comparative analysis, we aim to provide information on the effectiveness and robustness of different regression models, thus helping to make informed decisions in real-world applications.

In [1]:

```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import  linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, SplineTransformer
from sklearn.linear_model import Ridge
```

```
from sklearn.linear_model import LinearRegression
from scipy.fft import fft
import math as m
```

```
C:\Users\juanq\AppData\Local\Temp\ipykernel_20112\300021418.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (par
(to allow more performant data types, such as the Arrow string type, and better interoperabi
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

In [2]:

```
# dataset phat

DATA_PHAT = '../dataset/'
DATA_PATH_TRUE = 'DS-5-1-GAP-0-1-N-0_v2.csv'
DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
DATA_PATH_NOISE2 = 'DS-5-1-GAP-5-1-N-3_v2.csv'
```

In [3]:

```
df_true = pd.read_csv(DATA_PHAT + DATA_PATH_TRUE, header=None)
df_noise1 = pd.read_csv(DATA_PHAT + DATA_PATH_NOISE1,header=None)
df_noise2 = pd.read_csv(DATA_PHAT + DATA_PATH_NOISE2,header=None)
```

In [4]:

```
df_true.head(5)
```

Out[4]:

	0	1	2
0	0.00	17.49	17.04
1	2.12	17.65	17.17
2	3.06	17.70	17.24
3	4.16	17.73	17.33
4	4.93	17.75	17.39

In [5]:

```
df_noise1.head(5)
```

Out[5]:

	0	1	2	3	4	5	6	7	8	9	...	191	192	193
0	0.00	17.49	17.50	17.49	17.49	17.50	17.49	17.50	17.49	17.49	...	17.04	17.04	17.05
1	2.12	17.65	17.65	17.64	17.64	17.65	17.65	17.65	17.64	17.65	...	17.16	17.17	17.18
2	3.06	17.69	17.70	17.70	17.69	17.69	17.70	17.69	17.69	17.70	...	17.25	17.24	17.24
3	4.16	17.74	17.73	17.74	17.73	17.74	17.74	17.74	17.74	17.73	...	17.33	17.32	17.33
4	4.93	17.75	17.74	17.73	17.74	17.74	17.75	17.75	17.75	17.74	...	17.38	17.39	17.39

5 rows × 201 columns

In [6]:

```
df_noise2.head(5)
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	191	192	193
0	0.00	17.61	17.55	17.48	17.46	17.43	17.53	17.35	17.66	17.60	...	17.16	17.03	17.12
1	2.12	17.71	17.55	17.70	17.52	17.67	17.62	17.76	17.73	17.63	...	17.24	17.17	17.24
2	3.06	17.68	17.77	17.61	17.72	17.73	17.78	17.80	17.81	17.68	...	17.42	17.27	17.22
3	4.16	17.62	17.72	17.66	17.69	17.75	17.65	17.82	17.78	17.86	...	17.32	17.28	17.25
4	4.93	17.80	17.54	17.66	17.71	17.82	17.69	17.70	17.81	17.82	...	17.33	17.46	17.39

5 rows × 201 columns

Activity: Use 100 realizations per level of noise¶

Data: DS-5-1-GAP-1-1-N-1¶

Polynomial Regression¶ In [7]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
# Load data
X_test = df_true[0].to_numpy()[:,np.newaxis]
Y_test = df_true[1].to_numpy()[:,np.newaxis]

X_train = df_noise1[0].to_numpy()[:,np.newaxis]

print(X_test.shape)
print(Y_test.shape)
print(X_train.shape)

Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()
```

```
degrees = list(range(2,20))

mean_bias = np.zeros(len(degrees))
mean_variance = np.zeros(len(degrees))

for j, degree in enumerate(degrees):
    bias = []
    y_pred_all = []

    for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        # create model
        model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
        # training
        model.fit(X_train, y_i)

        y_pred_all.append(model.predict(X_test))

        bias.append(abs(Y_test - y_pred_all[i]))

    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)
```

```
(50, 1)
(50, 1)
(45, 1)
Mean Bias:
[0.10361983 0.08780442 0.03847091 0.01668811 0.0102933  0.00898785
 0.00737389 0.00669004 0.01615585 0.01651086 0.01407579 0.01633646
 0.0263673  0.03827158 0.05005268 0.08763278 0.09121493 0.09245241]

Mean Variance:
```

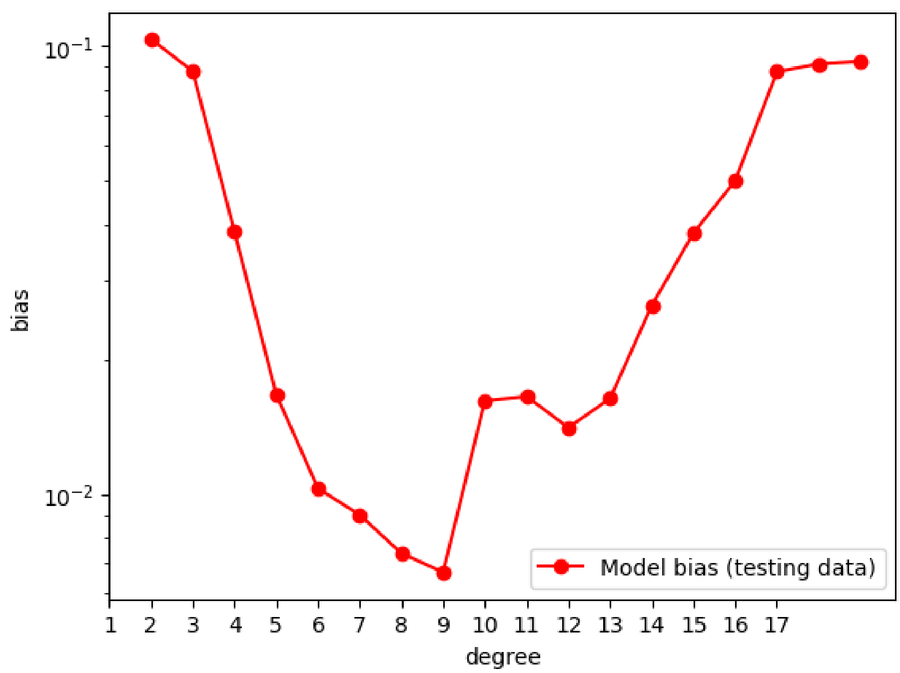
```
[0.00147208 0.00171471 0.00189878 0.00203802 0.00223529 0.00238363
 0.00254565 0.00394676 0.0026084  0.00242827 0.00238885 0.00235898
 0.00234475 0.00243909 0.00241474 0.00212611 0.00210574 0.0020848 ]
```

In [8]:

```
plt.xlabel('degree')
plt.ylabel('bias')
plt.plot(degrees, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(1,len(degrees)))
plt.show
```

Out[8]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



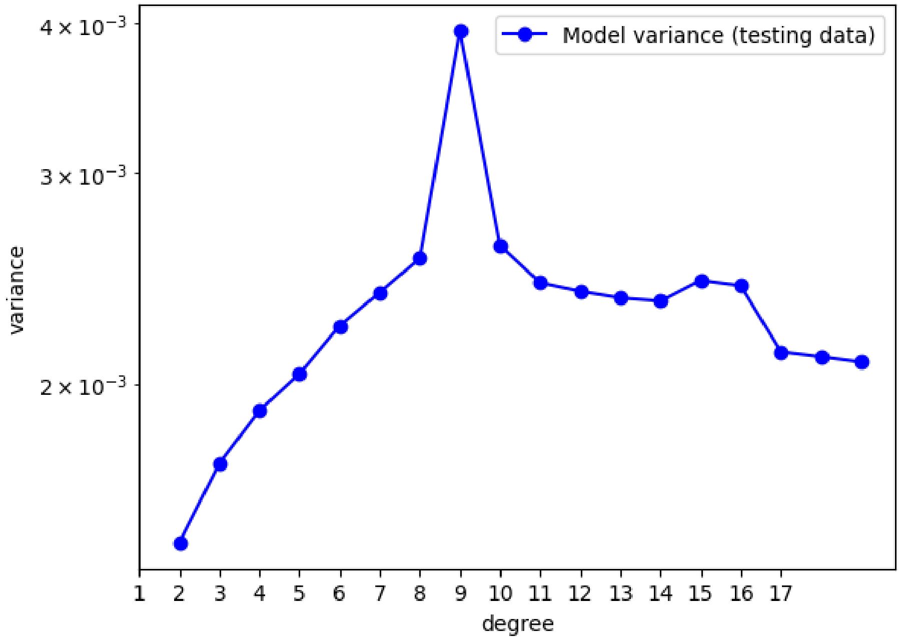
In [9]:

```
plt.xlabel('degree')
plt.ylabel('variance')
plt.plot(degrees, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(1,len(degrees)))
```

```
plt.show
```

Out[9]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [10]:

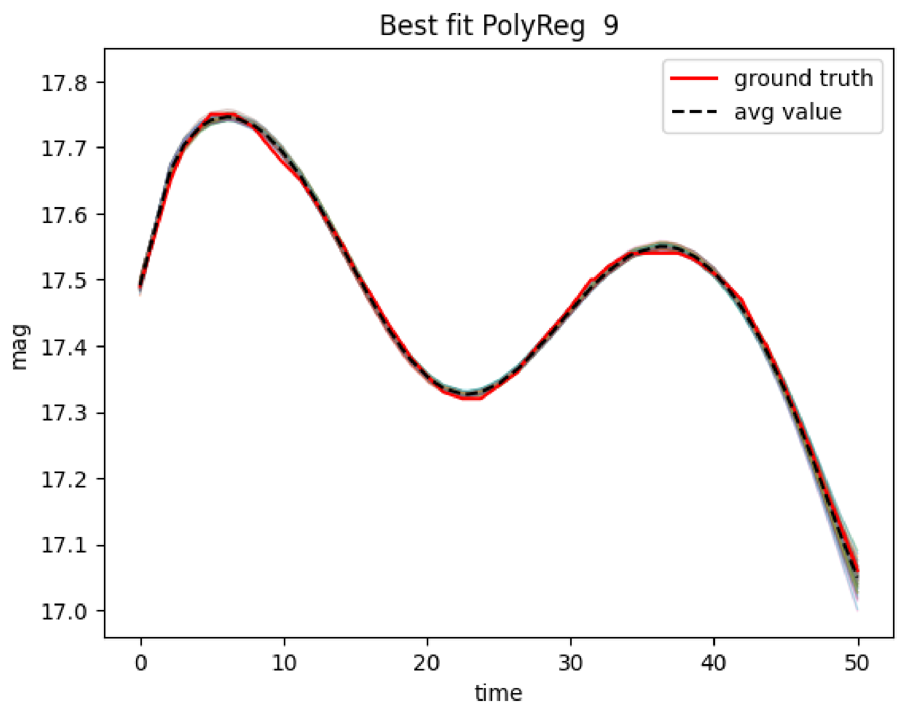
```
best_degree = degrees[np.argmin(mean_bias)]
print(best_degree)
index_min =np.argmin(mean_bias)
print(index_min)
```

```
9
7
```

In [11]:

```
plt.title( f'Best fit PolyReg  {best_degree}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)


MSE_list_test = []
MSE_list_train =[]
y_pred_all = []

for i in range(0, 100):
        y_i = y[:, i]
```

```
        y_i = y_i[:, np.newaxis]

        # create model
        model = make_pipeline(PolynomialFeatures(best_degree), LinearRegression())
        # training
        model.fit(X_train, y_i)

        #Testing
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict(X_test)

        MSE_train = mean_squared_error(y_i,Y_pred_train)
        MSE_test = mean_squared_error(Y_test,Y_pred_test)

        MSE_list_test.append(MSE_test)
        MSE_list_train.append(MSE_train)

        y_pred_all.append(Y_pred_test)
        plt.plot(X_test,Y_pred_test,linewidth = 1, alpha = 0.3)

pred_mean = np.mean(y_pred_all, axis=0)
pred_variance = np.std(y_pred_all, axis=0)



plt.plot(X_test, Y_test, 'r', label = 'ground truth')

plt.plot(X_test, pred_mean, '--k', label = 'avg value')
plt.xlabel('time')
plt.ylabel('mag')
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.006690036291529672
variance 0.003946759748134829
```

In [12]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  8.512111257842205e-05
MSE_test:  7.283567745835302e-05
```

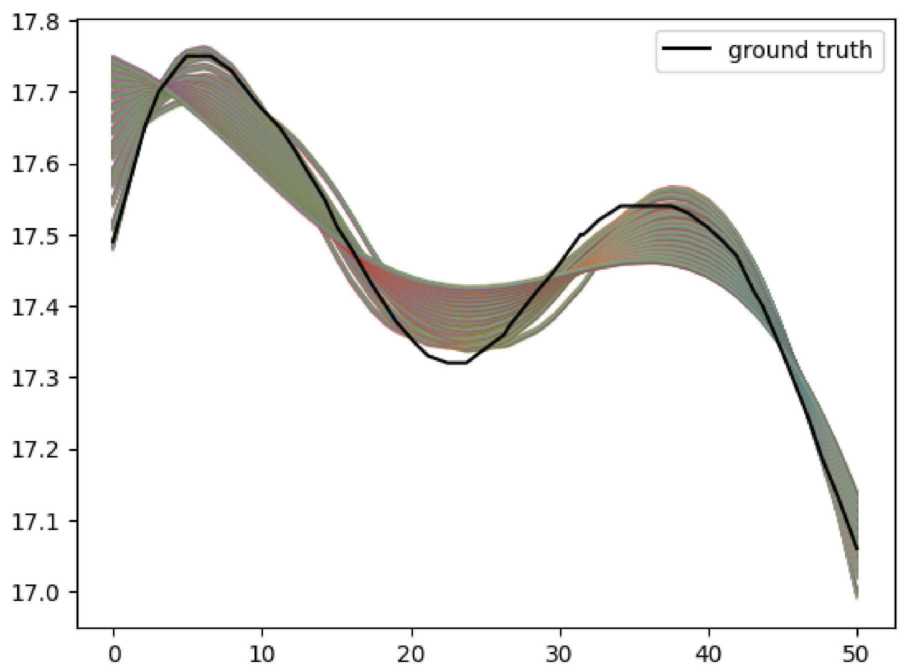
Splines (B-spline)¶ In [13]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
# Load data
X_test = df_true[0].to_numpy()[:,np.newaxis]
Y_test = df_true[1].to_numpy()[:,np.newaxis]

X_train = df_noise1[0].to_numpy()[:,np.newaxis]
```

```
print(X_test.shape)
print(Y_test.shape)
print(X_train.shape)



Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()

degrees = list(range(2,20))

mean_bias = np.zeros(len(degrees))
mean_variance = np.zeros(len(degrees))

for j, degree in enumerate(degrees):
    bias = []
    y_pred_all = []

    for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        #create model
        model = make_pipeline(SplineTransformer(n_knots=4, degree=degree), Ridge(alpha=1e-3)
        #training
        model.fit(X_train, y_i)

        #predictions
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict (X_test)

        y_pred_all.append(Y_pred_test)

        bias.append(abs(Y_test - y_pred_all[i]))

        plt.plot(X_test, Y_pred_test,linewidth = 1, alpha = 0.3)

    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)
```

```
plt.plot(X_test, Y_test, label='ground truth', color = 'k')
plt.legend()
plt.show()

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)
```

```
(50, 1)
(50, 1)
(45, 1)
```



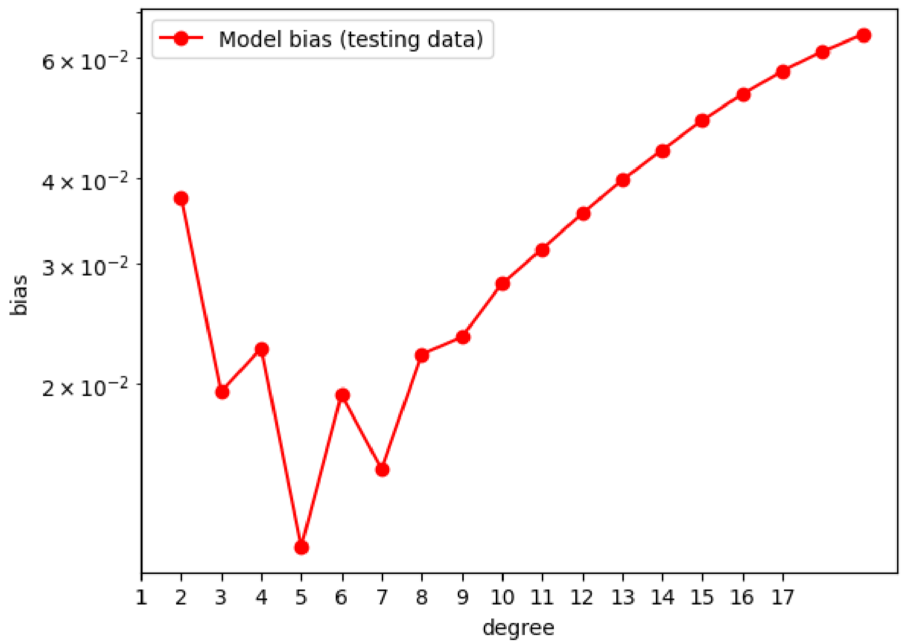

```
Mean Bias:
[0.03753138 0.01947464 0.02256899 0.01156363 0.01928863 0.0150147
 0.02209184 0.02341411 0.02802274 0.03148421 0.03548916 0.03972975
 0.04398461 0.04858668 0.05308472 0.0573883  0.06131321 0.0648641 ]

Mean Variance:
[0.00192881 0.00200077 0.00195926 0.00199313 0.0019467  0.00194404
 0.00191057 0.00188872 0.00186714 0.00184289 0.00182251 0.00180042
 0.00178091 0.00176207 0.0017452  0.00172973 0.00171575 0.00170293]
```

In [14]:

```
plt.xlabel('degree')
plt.ylabel('bias')
plt.plot(degrees, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(1,len(degrees)))
plt.show
```

Out[14]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

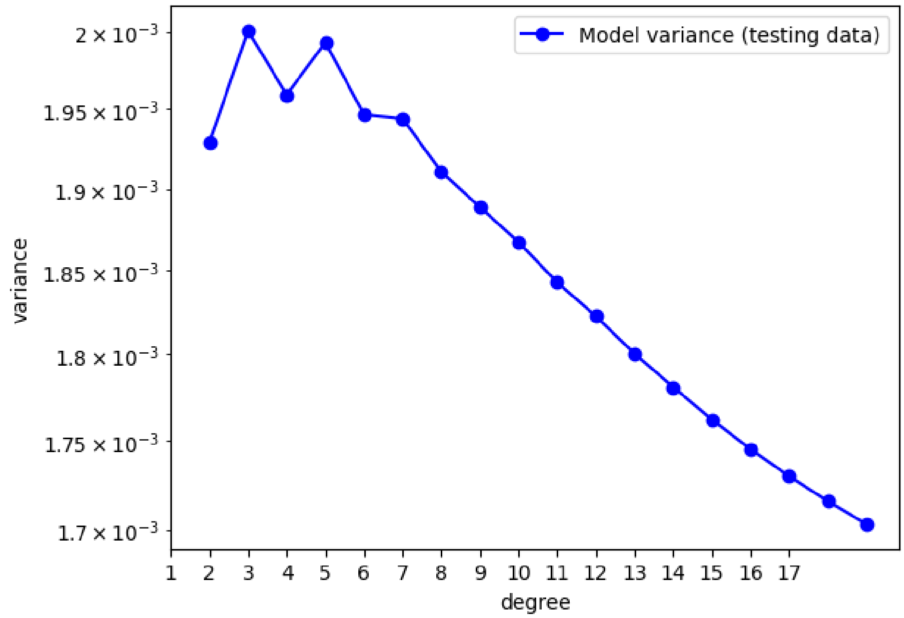


In [15]:

```
plt.xlabel('degree')
plt.ylabel('variance')
plt.plot(degrees, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(1,len(degrees)))
plt.show
```

Out[15]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [16]:

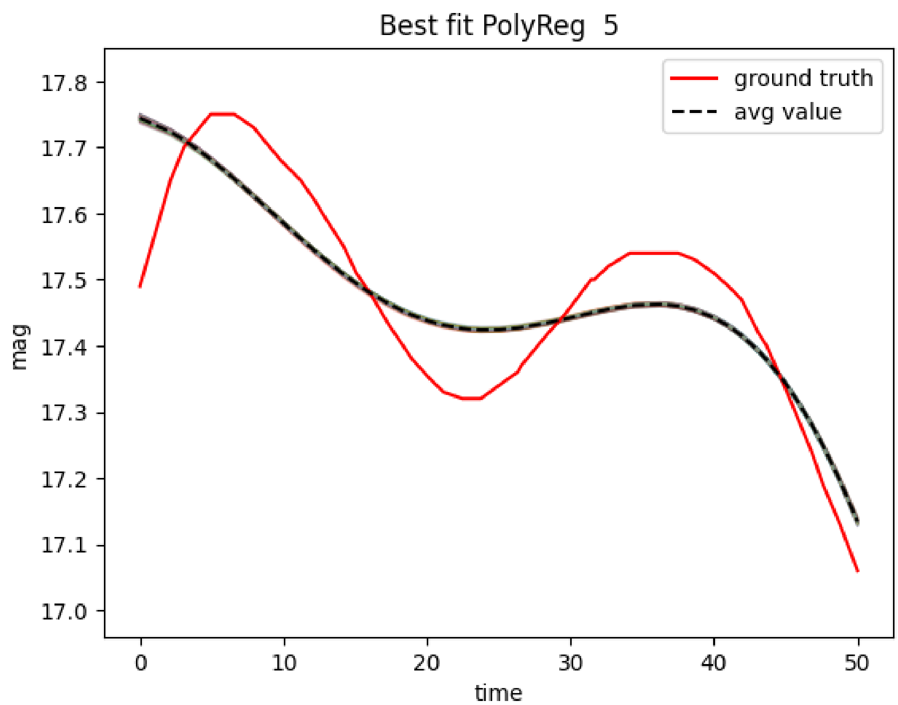
```
best_degree = degrees[np.argmin(mean_bias)]
print(best_degree)

index_min = np.argmin(mean_bias)
print(index_min)
```

```
5
3
```

In [17]:

```
plt.title( f'Best fit PolyReg  {best_degree}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)

y_pred_all = []

MSE_list_train = []
MSE_list_test = []

for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        #create model
        model = make_pipeline(SplineTransformer(n_knots=4, degree=degree), Ridge(alpha=1e-3)
```

```
        #training
        model.fit(X_train, y_i)

        #predictions
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict (X_test)

        #MSE
        MSE_train = mean_squared_error(y_i, Y_pred_train)
        MSE_test = mean_squared_error(Y_test, Y_pred_test)

        MSE_list_train.append(MSE_train)
        MSE_list_test.append(MSE_test)

        y_pred_all.append(Y_pred_test)
        plt.plot(X_test,Y_pred_test,linewidth = 1, alpha = 0.3)

pred_mean = np.mean(y_pred_all, axis=0)
pred_variance = np.std(y_pred_all, axis=0)



plt.plot(X_test, Y_test, 'r', label = 'ground truth')

plt.plot(X_test, pred_mean, '--k', label = 'avg value')
plt.xlabel('time')
plt.ylabel('mag')
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.011563634204581886
variance 0.001993125121183679
```

In [18]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.00606479808328759
MSE_test:  0.005814183614339369
```

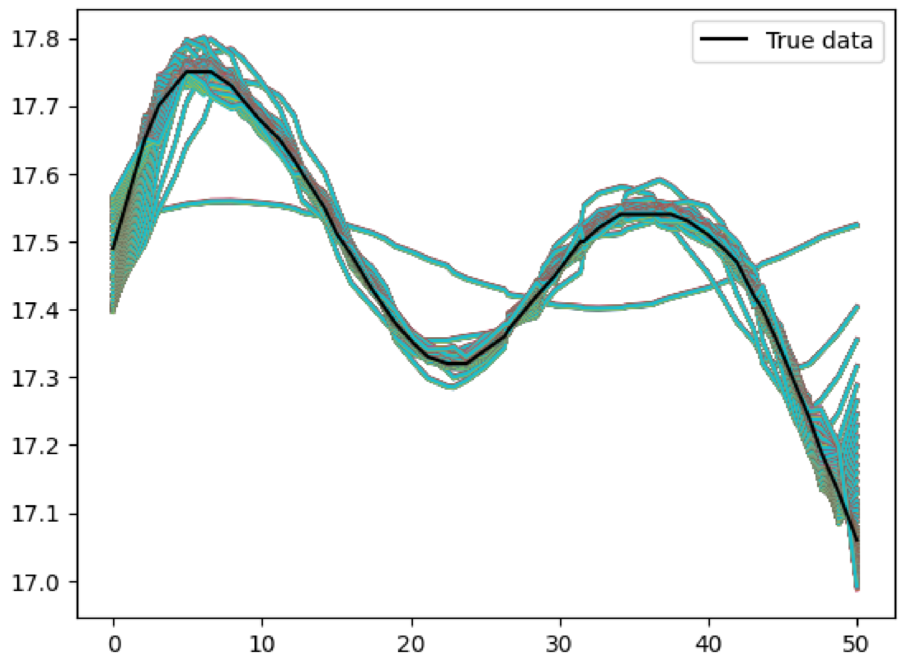
Fourier¶ In [19]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
# Load data
X_test = df_true[0].to_numpy()
Y_test = df_true[1].to_numpy()

X_train = df_noise1[0].to_numpy()
```

```
Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()
y_i = y[:, 1]

components = list(range(2,40))
mean_bias = np.zeros(len(components))
mean_variance = np.zeros(len(components))

for j, component in enumerate(components):
    bias = []
    y_pred_all = []
    for i in range(0,100):
        y_i = y[:, i]

        # ---------------train
        # Calculate the Fourier transform
        Y_train_fft = fft(y_i)

        # Select Fourier components
        n_components = component
        Y_train_fft_filtered = np.zeros_like(Y_train_fft)
        Y_train_fft_filtered[:n_components] = Y_train_fft[:n_components]

        n_samples = y_i.size

        # Build the design matrix
        X_train_fourier = np.abs(np.fft.ifft(Y_train_fft_filtered).real)

        # Fit the linear regression model
        model = LinearRegression()
        model.fit(X_train_fourier[:, None], y_i)

        # ---------------test
        # Calculate the Fourier transform
        Y_test_fft = fft(Y_test)

        # # Select Fourier components
        Y_test_fft_filtered = np.zeros_like(Y_test_fft)
        Y_test_fft_filtered[:n_components] = Y_test_fft[:n_components]


        # Build the design matrix
        X_test_fourier = np.abs(np.fft.ifft(Y_test_fft_filtered).real)

        # Predictions
```

```
        Y_pred_test = model.predict(X_test_fourier[:, None])
        Y_pred_train = model.predict(X_train_fourier[:,None])
        y_pred_all.append(Y_pred_test)

        #bias
        bias.append(abs(Y_test - y_pred_all[i]))

        #plot
        plt.plot(X_test, Y_pred_test)

    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)

plt.plot(X_test, Y_test, label='True data', color = 'k')
plt.legend()
plt.show()
```

```
Mean Bias:
[0.12328409 0.05799202 0.04568209 0.03358446 0.02751162 0.02318978
 0.02085649 0.01848195 0.01694538 0.01594828 0.01494815 0.0133464
 0.01309501 0.01172961 0.0110446  0.0109774  0.00888897 0.0087832
 0.00863953 0.00763873 0.00716563 0.00612747 0.00536327 0.00366718
 0.0037745  0.00538773 0.00618906 0.00716355 0.00772859 0.0086652
 0.00931838 0.00882928 0.01104786 0.01170972 0.01296795 0.01454332
 0.01578212 0.01656416]

Mean Variance:
[0.00083865 0.00083865 0.00083865 0.00083865 0.00083865 0.00083865
 0.00083865 0.00083865 0.00083865 0.00083865 0.00083865 0.00083865
 0.00083865 0.00083865 0.00083865 0.00083865 0.00083865 0.00083865
 0.00083865 0.00083865 0.00083865 0.00083865 0.00084317 0.00085053
 0.00085262 0.00085149 0.00085086 0.00085419 0.00086405 0.00087533
 0.0008864  0.00088785 0.00088734 0.00087677 0.00088009 0.00089215
 0.00090651 0.00088769]
```

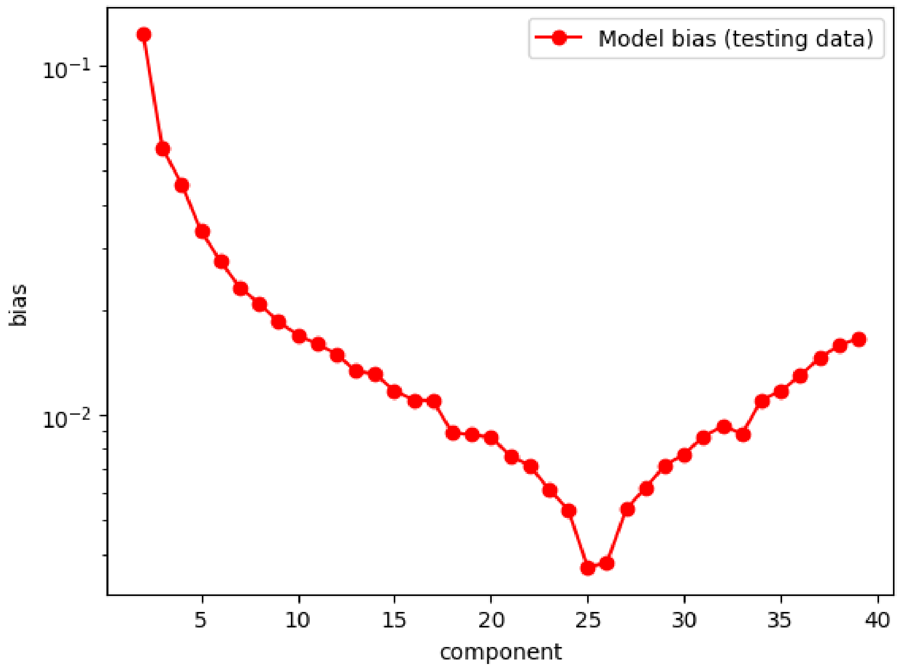


In [20]:

```
plt.xlabel('component')
plt.ylabel('bias')
plt.plot(components, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[20]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

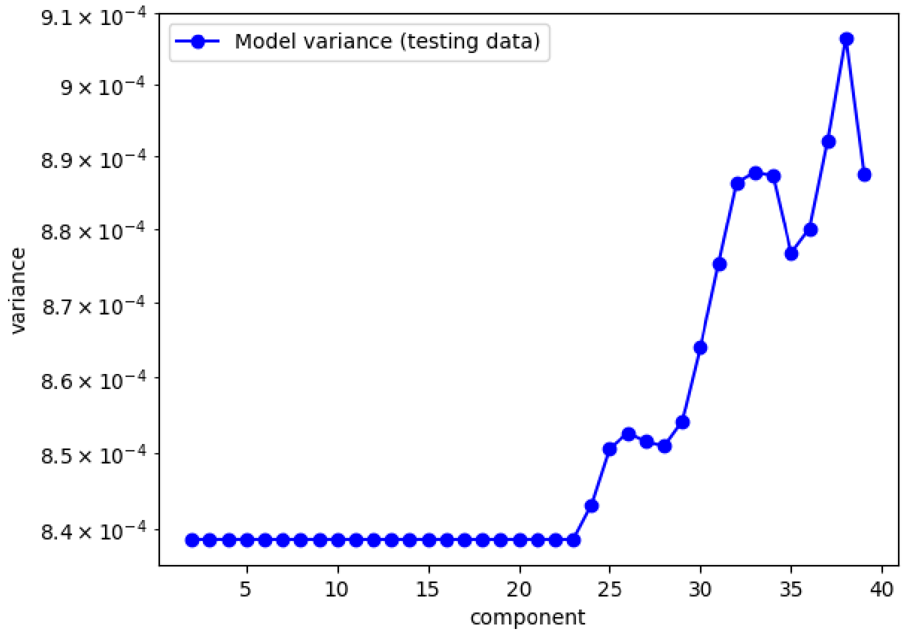


In [21]:

```
plt.xlabel('component')
plt.ylabel('variance')
plt.plot(components, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[21]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [22]:

```
best_ncomponents = components[np.argmin(mean_bias)]
print(best_ncomponents)

index_min = np.argmin(mean_bias)
print(index_min)
```

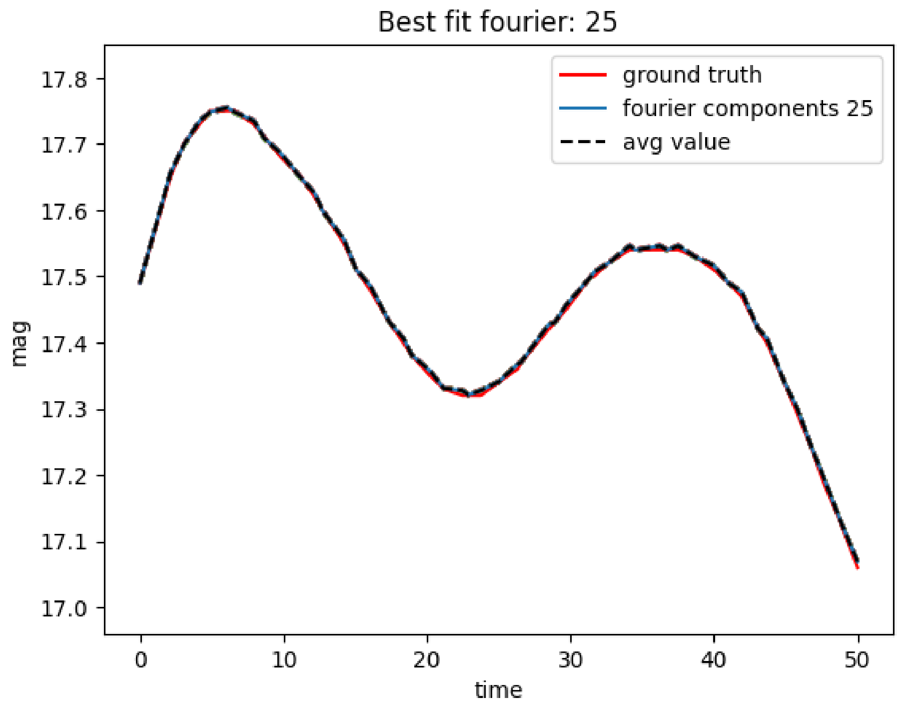
```
25
23
```

In [23]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()

y_pred_all = []

MSE_list_train = []
MSE_list_test = []

plt.title( f'Best fit fourier: {best_ncomponents}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)

for i in range(0,100):
```

```
    y_i = y[:, i]
    # ---------------train
    # Calculate the Fourier transform
    Y_train_fft = fft(y_i)
    # Select Fourier components
    n_components = best_ncomponents
    Y_train_fft_filtered = np.zeros_like(Y_train_fft)
    Y_train_fft_filtered[:n_components] = Y_train_fft[:n_components]
    n_samples = y_i.size
    # Build the design matrix
    X_train_fourier = np.abs(np.fft.ifft(Y_train_fft_filtered).real)
    # Fit the linear regression model
    model = LinearRegression()
    model.fit(X_train_fourier[:, None], y_i)
    # ---------------test
    # Calculate the Fourier transform
    Y_test_fft = fft(Y_test.squeeze())
    # # Select Fourier components
    Y_test_fft_filtered = np.zeros_like(Y_test_fft)
    Y_test_fft_filtered[:n_components] = Y_test_fft[:n_components]
    # Build the design matrix
    X_test_fourier = np.abs(np.fft.ifft(Y_test_fft_filtered).real)
    # Predictions
    Y_pred_test = model.predict(X_test_fourier[:, None])
    Y_pred_train = model.predict(X_train_fourier[:,None])
    y_pred_all.append(Y_pred_test)

    # Calculate MSE on training set
    mse_train = mean_squared_error(y_i, Y_pred_train)
    MSE_list_train.append(mse_train)

    # Calculate MSE on test set
    mse_test = mean_squared_error(Y_test, Y_pred_test)
    MSE_list_test.append(mse_test)

    y_pred_all.append(Y_pred_test)

    plt.plot(X_test, Y_pred_test, linestyle='--')

pred_mean = np.mean(y_pred_all, axis=0)

plt.plot(X_test, Y_test, 'r', label = 'ground truth')
plt.plot(X_test, Y_pred_test,label = f'fourier components {best_ncomponents}')
plt.plot(X_test, pred_mean,'--k', label = 'avg value' )
plt.xlabel('time')
plt.ylabel('mag')
```

```
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.003667180746496816
variance 0.0008505280393994924
```

In [24]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)


MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  8.60851666319718e-05
MSE_test:  2.1620287806004973e-05
```

Kernel methods¶ In [25]:

```
X = df_noise1[0]  #time
```

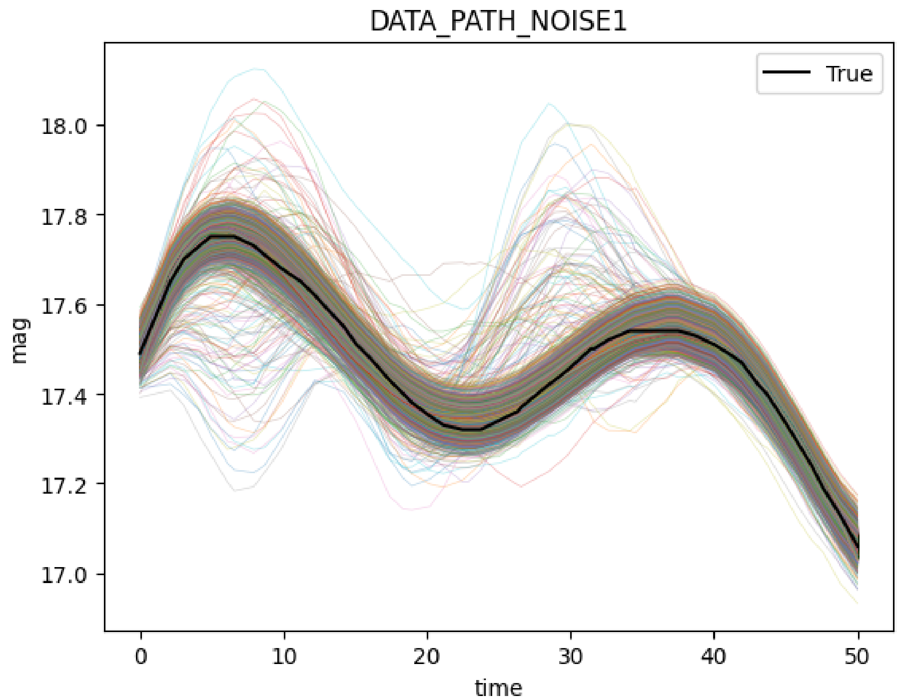
```
x = X.values.reshape(-1,1) #X[:, np.newaxis]
Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()

X_test = df_true[0]
Y_test = df_true[1]
x_test = X_test.values.reshape(-1,1) #x_test = X_test[:, np.newaxis]
y_test = Y_test.values.reshape(-1,1) #y_test = Y_test[:, np.newaxis]

in_Nsigma = 5

sigmas = list(range(in_Nsigma,200))

mean_bias = np.zeros(len(sigmas))
mean_variance = np.zeros(len(sigmas))

for j,sigma in enumerate(sigmas):
  bias = []
  y_pred_all = []
  for i in range(0,100):
    y_i = y[:, i]
    ones = []
    for k in range(0,Y.size):
      ones.append(1)

    #Do an array of ones
    ones1 = np.array(ones)

    #make de function K1 where
    #ex = data time
    #n = number of points
    #c = Centers of gaussians
    #k = number of kernels
    #d = kernels width
    def K1(ex,n,c,km,d):
      matrix = [[0 for _ in range(n)] for _ in range(km)]
      for i in range(0,km):
        for j in range (0,n):
          matrix[i][j] = m.exp(-(abs(ex[j]-c[i])**2/(d[j]**2))) #the kernel function
      return matrix

    #First Step: make the Gran_Matrix
    Gram_matrix = K1(X, X.size, X, X.size, ones1*sigma)
    #This function returns a Matrix instead an array
    Gram_matrix_M = np.asarray(Gram_matrix)
```

```
    #The pseudo inverse of the matrix wiht the function np.linalg.pinv
    #gettin the H matrix
    pinvGram_matrix_M = np.linalg.pinv(np.transpose(Gram_matrix_M))
    #In this point we can calculate the alpha becase we have the pseudo-inverse of the matri
    alpha = pinvGram_matrix_M.dot(y_i)

    #getting H from Alpha
    alphaT = np.transpose(alpha)
    #Remove axes of length one from alphaTD.
    alphaTD = np.squeeze(alphaT)

    #make the kernel method
    h = alphaTD.dot(Gram_matrix_M)

    #Remove axes of length one from h.
    hArray = np.squeeze(np.asarray(np.transpose(h)))

    #Metrics
    MSE_train = mean_squared_error(y_i,np.transpose(h))

    y_pred_all.append(hArray)
    #bias
    Y_test2 = Y_test[:45]
    bias.append(abs(Y_test2 - y_pred_all[i]))
    plt.plot(x, hArray,linewidth = 0.5, alpha = 0.3)
  # bias
  pred_mean = np.mean(bias, axis=0)
  mean_bias[j] = np.mean(pred_mean)

  # variance
  pred_variance = np.std(y_pred_all, axis=0)
  mean_variance[j] = np.mean(pred_variance)

#Plotting
plt.plot(x_test,y_test, color='k', label="True")

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)

plt.xlabel("time")
plt.ylabel("mag")
plt.legend(loc="best")
plt.title("DATA_PATH_NOISE1")
```

```
plt.show()
print(Y_test.shape)
print(hArray.shape)
```

```
Mean Bias:
[0.13908042 0.07060755 0.06383995 0.06141816 0.0866933  0.0613356
 0.05564368 0.05901141 0.06451631 0.05922657 0.05540365 0.05591826
 0.05828562 0.05759666 0.06171231 0.0558539  0.05912731 0.0557128
 0.05721058 0.06170412 0.0559286  0.05591755 0.05698291 0.05584264
 0.05590898 0.05582431 0.05758183 0.05856733 0.05572657 0.0560088
 0.05793442 0.05587957 0.0598416  0.06208465 0.05606221 0.05597787
 0.05592396 0.05595288 0.05621826 0.0572921  0.05702172 0.05932586
 0.05944751 0.05696796 0.05700905 0.05705444 0.0571439  0.0569047
 0.05680114 0.05707955 0.0574363  0.05749291 0.05713807 0.06146673
 0.06575109 0.06058765 0.0628852  0.05663303 0.05661888 0.05659251
 0.05666905 0.05677919 0.05670288 0.05668683 0.05673678 0.05675433
 0.05660941 0.05725247 0.05697677 0.05682166 0.05731358 0.05726997
 0.0593252  0.05907722 0.05890002 0.06283977 0.06054976 0.0600279
 0.05757039 0.05756255 0.0575742  0.05758282 0.05761623 0.05756505
 0.05755468 0.05762378 0.0576418  0.05759473 0.05753756 0.05759497
 0.05776361 0.05773326 0.05768504 0.05766525 0.05757749 0.05782167
 0.05766012 0.05786068 0.05760913 0.05760407 0.05792942 0.05843603
 0.05814362 0.05765345 0.05803226 0.05812004 0.06123024 0.05830193
 0.05846662 0.06106534 0.05907771 0.06064515 0.06513108 0.0623963
 0.06447293 0.05551225 0.0555013  0.05550398 0.05550881 0.05550259
 0.05547188 0.0555108  0.05551342 0.05544905 0.05547619 0.05550799
 0.05550694 0.05547233 0.05547926 0.0554577  0.05552164 0.05546071
 0.05552128 0.05551724 0.05548116 0.05547812 0.05548507 0.0555588
 0.05543501 0.05542031 0.05544168 0.05539203 0.05547322 0.0554579
 0.05538306 0.05541431 0.05567085 0.05537061 0.05546797 0.05583106
 0.05538343 0.05543258 0.05552822 0.05593504 0.05601875 0.05542533
 0.05593696 0.05552298 0.05645222 0.05548114 0.05685485 0.05656385
 0.05701484 0.0559432  0.05566737 0.05620338 0.05674115 0.05751677
 0.05605174 0.05599405 0.05656562 0.05672848 0.05808313 0.05709974
 0.05681599 0.05745899 0.05671767 0.06533764 0.06477695 0.05882768
 0.05947949 0.06975477 0.05930934 0.05907613 0.05981361 0.06860251
 0.05908379 0.05907963 0.05908025 0.05908914 0.05907774 0.05908782
 0.05908816 0.05908677 0.0590946 ]

Mean Variance:
[0.10397006 0.02316916 0.01672528 0.01464568 0.03466662 0.02550255
 0.0077391  0.01032862 0.01358225 0.00844551 0.00523556 0.00399422
 0.00865286 0.00542654 0.01822119 0.00471568 0.01338196 0.00378626
 0.00615856 0.01700655 0.00351795 0.00543491 0.011045   0.00318435
 0.00349449 0.00488029 0.00770251 0.01360716 0.00306311 0.003204
 0.00384877 0.00478833 0.00732814 0.01332291 0.00288862 0.00296667
```

```
0.00313923 0.00341657 0.00423161 0.0058842  0.00835717 0.01242355
0.01930707 0.00271997 0.00274733 0.00276557 0.00283229 0.00286885
0.00327761 0.00351303 0.00469935 0.00577873 0.00597908 0.00879519
0.0127052  0.01493871 0.0181589  0.00253297 0.00254523 0.00256887
0.00256508 0.00263942 0.00267483 0.00280132 0.0030263  0.00289201
0.0031567  0.00378083 0.00386531 0.00452782 0.00599056 0.00656134
0.00740559 0.00890533 0.00966894 0.01081479 0.01671784 0.01837733
0.00237473 0.00237185 0.00239631 0.00237992 0.00238535 0.00238079
0.00242829 0.00242264 0.00243146 0.00243717 0.00246766 0.00248885
0.00253568 0.00258277 0.00259037 0.00276037 0.00278546 0.00293159
0.0030592  0.00376096 0.00348205 0.00456809 0.00413457 0.0048467
0.00544792 0.00525095 0.00598154 0.00859469 0.00783672 0.0087684
0.01102487 0.0111031  0.01347972 0.01856042 0.01625697 0.01823473
0.02080956 0.00222881 0.00223004 0.00222505 0.00223505 0.00222967
0.00222179 0.00224155 0.00224221 0.00224052 0.00222359 0.00224471
0.00224711 0.00225138 0.00225067 0.00223733 0.002255   0.00225291
0.00230177 0.00225817 0.00231551 0.00229599 0.00233151 0.00228999
0.00235269 0.00238387 0.00238393 0.00245483 0.00245073 0.00241136
0.00249233 0.00278821 0.00263794 0.00267687 0.00271208 0.00271475
0.00297994 0.00293184 0.00307141 0.0034226  0.00349043 0.00363948
0.00396885 0.00428476 0.00387792 0.00391677 0.00436786 0.00544053
0.00567668 0.00573003 0.00581777 0.00616882 0.00641519 0.00747047
0.00729314 0.00762106 0.0088249  0.00991576 0.01025649 0.01020175
0.01066517 0.01185668 0.01088661 0.0139711  0.01422509 0.01605268
0.01743909 0.01871994 0.01974059 0.02029238 0.02185387 0.02531813
0.00204687 0.00204891 0.002049   0.00204947 0.00204899 0.00205093
0.00204536 0.00205022 0.00204523]
```



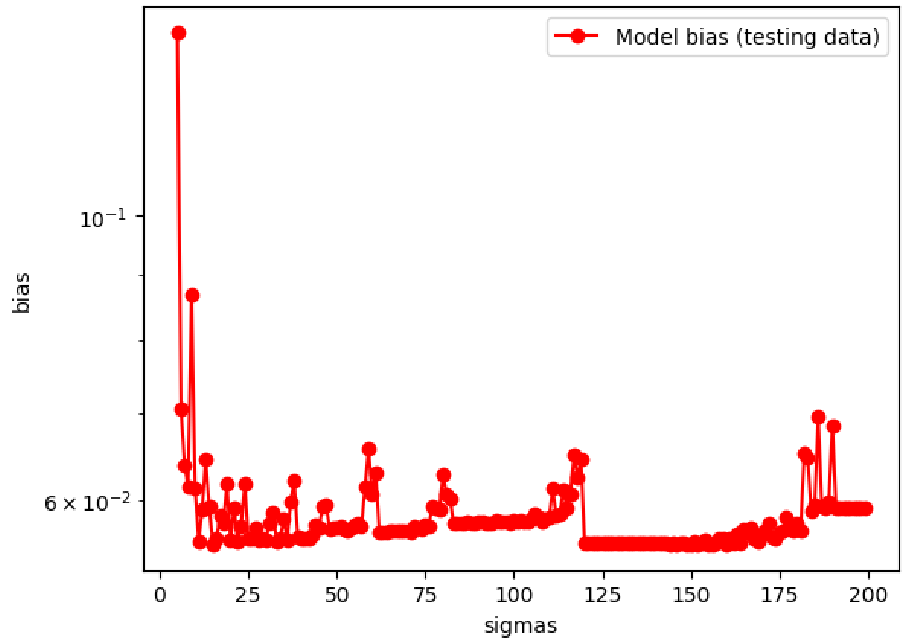

```
(50,)
(45,)
```

In [26]:

```
plt.xlabel('sigmas')
plt.ylabel('bias')
plt.plot(sigmas, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
# plt.xticks(range(in_Nsigma,len(sigmas)))
plt.show
```

Out[26]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

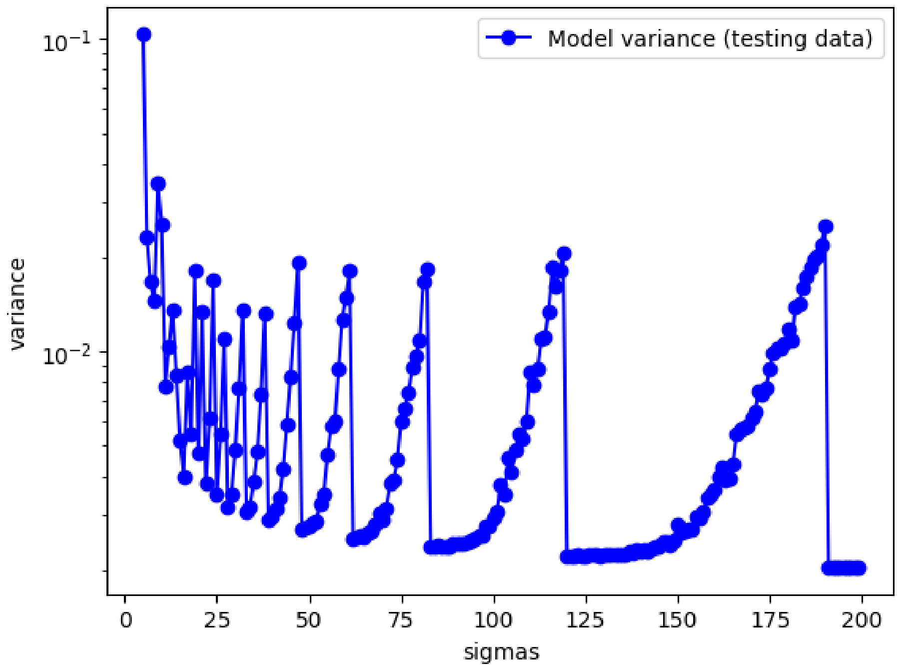


In [27]:

```
plt.xlabel('sigmas')
plt.ylabel('variance')
plt.plot(sigmas, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
# plt.xticks(range(in_Nsigma,len(sigmas)))
plt.show
```

Out[27]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [28]:

```
best_sigma = sigmas[np.argmin(mean_bias)]
print(best_sigma)
index_min = np.argmin(mean_bias)
print(index_min)
```

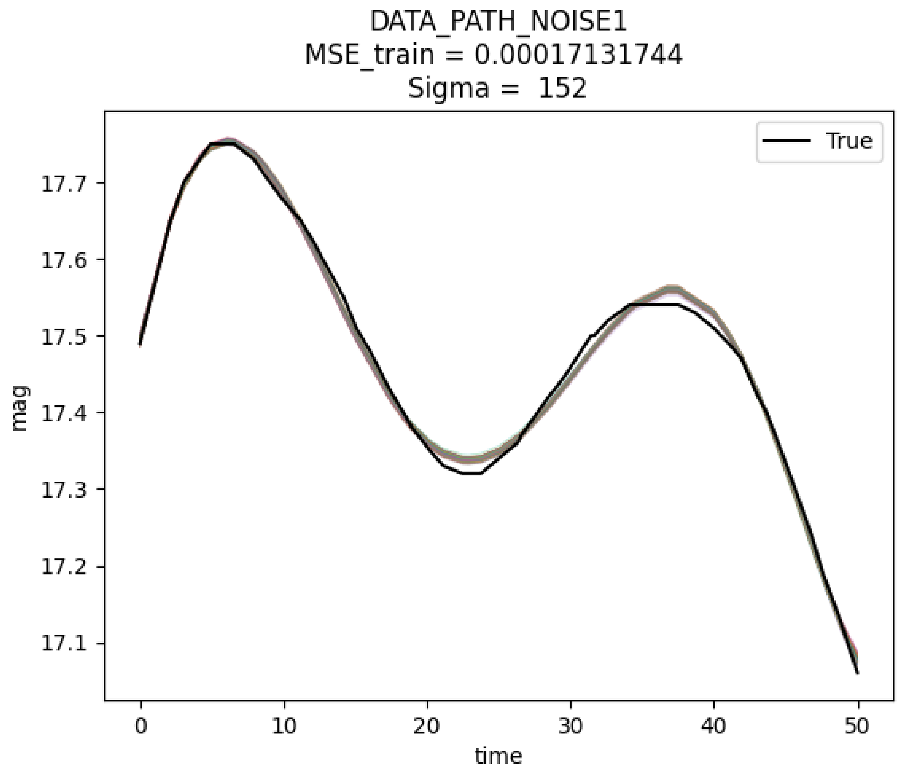
```
152
147
```

In [29]:

```
X = df_noise1[0]  #time
x = X.values.reshape(-1,1) #X[:, np.newaxis]
Y = df_noise1.iloc[:,1:101]
y = Y.to_numpy()

X_test = df_true[0]
Y_test = df_true[1]
x_test = X_test.values.reshape(-1,1) #x_test = X_test[:, np.newaxis]
y_test = Y_test.values.reshape(-1,1) #y_test = Y_test[:, np.newaxis]

sigma=best_sigma

MSE_train_list = []
```

```
for i in range(0,100):
    y_i = y[:, i]
    ones = []
    for k in range(0,Y.size):
      ones.append(1)

    #Do an array of ones
    ones1 = np.array(ones)

    #make de function K1 where
    #ex = data time
    #n = number of points
    #c = Centers of gaussians
    #k = number of kernels
    #d = kernels width
    def K1(ex,n,c,km,d):
      matrix = [[0 for _ in range(n)] for _ in range(km)]
      for i in range(0,km):
        for j in range (0,n):
          matrix[i][j] = m.exp(-(abs(ex[j]-c[i])**2/(d[j]**2))) #the kernel function
      return matrix

    #First Step: make the Gran_Matrix
    Gram_matrix = K1(X, X.size, X, X.size, ones1*sigma)
    #This function returns a Matrix instead an array
    Gram_matrix_M = np.asarray(Gram_matrix)

    #The pseudo inverse of the matrix wiht the function np.linalg.pinv
    #gettin the H matrix
    pinvGram_matrix_M = np.linalg.pinv(np.transpose(Gram_matrix_M))
    #In this point we can calculate the alpha becase we have the pseudo-inverse of the matri
    alpha = pinvGram_matrix_M.dot(y_i)

    #getting H from Alpha
    alphaT = np.transpose(alpha)
    #Remove axes of length one from alphaTD.
    alphaTD = np.squeeze(alphaT)

    #make the kernel method
    h = alphaTD.dot(Gram_matrix_M)

    #Remove axes of length one from h.
    hArray = np.squeeze(np.asarray(np.transpose(h)))

    #Metrics
```

```
    MSE_train = mean_squared_error(y_i,np.transpose(h))

    #----------------------------
    Gram_matrix_test = K1(X_test, X_test.size, X, X.size, ones1 * sigma)

    # Calculate predictions for test data
    h_test = alphaTD.dot(Gram_matrix_test)

    # Calculate MSE for test data
    MSE_test = mean_squared_error(y_test, np.transpose(h_test))

    MSE_list_train.append(MSE_train)
    MSE_list_test.append(MSE_test)
    plt.plot(x, hArray,linewidth = 0.5, alpha = 0.3)

#Plotting
plt.plot(x_test,y_test, color='k', label="True")
plt.xlabel("time")
plt.ylabel("mag")
plt.legend(loc="best")
plt.title("DATA_PATH_NOISE1\nMSE_train = {:.8} \nSigma =  {:}".format(MSE_train,sigma))
plt.show()
print(Y_test.shape)
print(hArray.shape)


print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
(50,)
(45,)
bias:  0.05537060621473526
variance 0.0026768734322475634
```

In [30]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.00011098043278959848
MSE_test:  8.327705547474614e-05
```

Table: Dataset: DS-5-1-GAP-1-1-N-1¶

Regression	MSE training	MSE Testing (ground truth)	Bias
Polynomial (degree = 9)	8.512111257842205e-05	7.283567745835302e-05	0.00669003629152
Splines (degree = 5)	0.00606479808328759	0.005814183614339369	0.01156363420458

Regression	MSE training	MSE Testing (ground truth)	Bias
Fourier(n_components = 25)	8.60851666319718e-05	2.1620287806004973e-05	0.003667180746496
Kenerl method(sigma = 59)	0.00016743339978391236	....	0.05616772

Data: DS-5-1-GAP-5-1-N-3¶

Plynomial regression¶ In [31]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
X_test = df_true[0].to_numpy()[:,np.newaxis]
Y_test = df_true[1].to_numpy()[:,np.newaxis]

X_train = df_noise2[0].to_numpy()[:,np.newaxis]

print(X_test.shape)
print(Y_test.shape)
print(X_train.shape)

Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()


degrees = list(range(2,20))

mean_bias = np.zeros(len(degrees))
mean_variance = np.zeros(len(degrees))

for j, degree in enumerate(degrees):
    bias = []
    y_pred_all = []

    for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        # create model
        model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
        # training
        model.fit(X_train, y_i)

        y_pred_all.append(model.predict(X_test))

        bias.append(abs(Y_test - y_pred_all[i]))
```

```
    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)
```

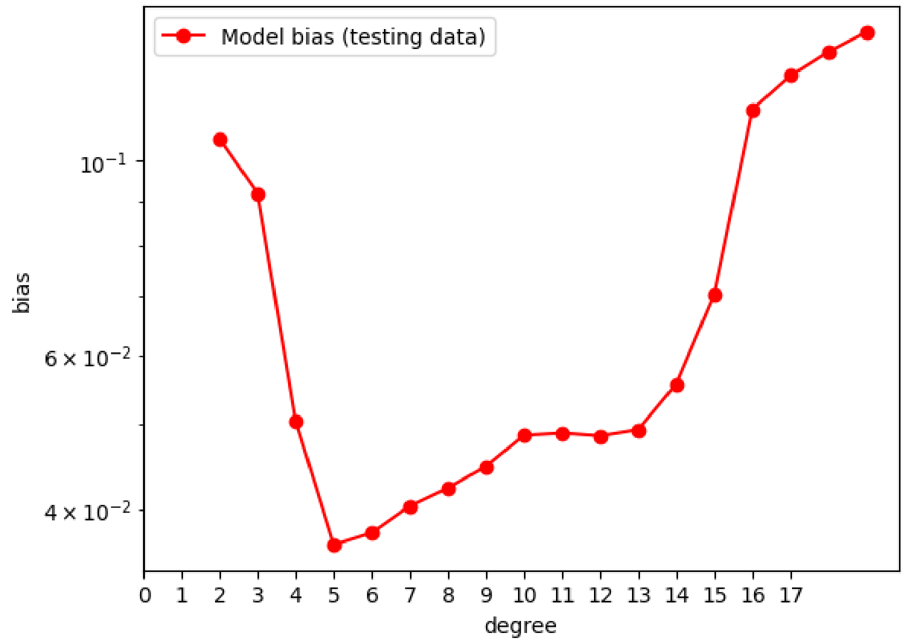
```
(50, 1)
(50, 1)
(25, 1)
Mean Bias:
[0.10607181 0.09187968 0.05050545 0.03643462 0.03758568 0.04032173
 0.04228031 0.04477442 0.0485861  0.04888251 0.04851896 0.04930965
 0.05563294 0.0704391  0.11455676 0.12518624 0.13329452 0.14040393]

Mean Variance:
[0.02560118 0.03167672 0.03527781 0.0402986  0.04559276 0.04941269
 0.0526682  0.05583096 0.05445432 0.04969405 0.04968259 0.05091983
 0.05372675 0.05759329 0.04780424 0.0507615  0.05431813 0.05835109]
```

In [32]:

```
plt.xlabel('degree')
plt.ylabel('bias')
plt.plot(degrees, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(0,len(degrees)))
plt.show
```

Out[32]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

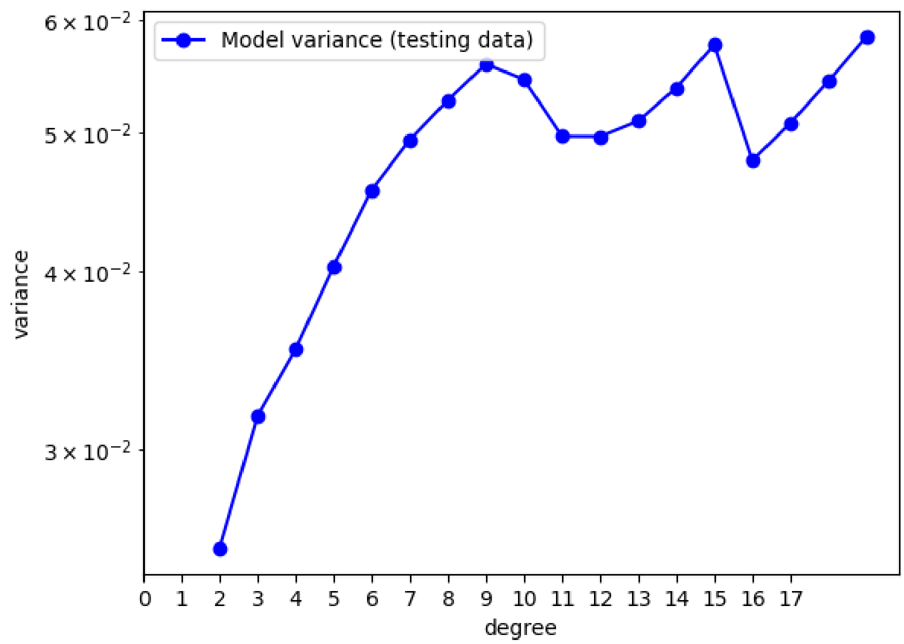


In [33]:

```
plt.xlabel('degree')
plt.ylabel('variance')
plt.plot(degrees, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.xticks(range(0,len(degrees)))
plt.show
```

Out[33]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [34]:

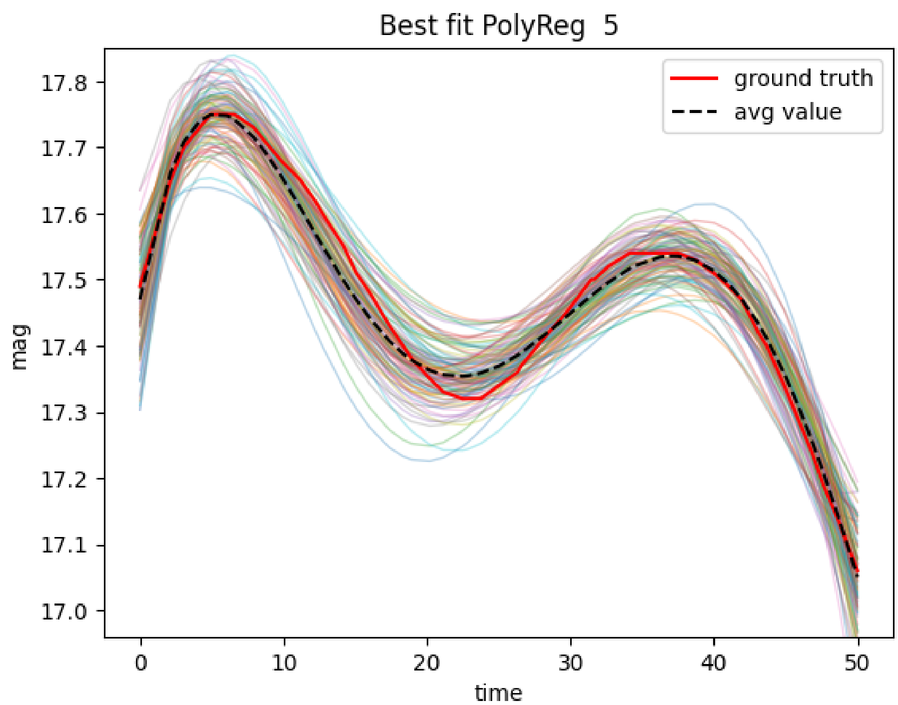
```
best_degree = degrees[np.argmin(mean_bias)]
print(best_degree)

index_min = np.argmin(mean_bias)
print(index_min)
```

```
5
3
```

In [35]:

```
plt.title( f'Best fit PolyReg  {best_degree}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)


MSE_list_test = []
MSE_list_train =[]
y_pred_all = []

for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        # create model
```

```
        model = make_pipeline(PolynomialFeatures(best_degree), LinearRegression())
        # training
        model.fit(X_train, y_i)

        #Testing
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict(X_test)

        MSE_train = mean_squared_error(y_i,Y_pred_train)
        MSE_test = mean_squared_error(Y_test,Y_pred_test)

        MSE_list_test.append(MSE_test)
        MSE_list_train.append(MSE_train)

        y_pred_all.append(Y_pred_test)
        plt.plot(X_test,Y_pred_test,linewidth = 1, alpha = 0.3)

pred_mean = np.mean(y_pred_all, axis=0)
pred_variance = np.std(y_pred_all, axis=0)



plt.plot(X_test, Y_test, 'r', label = 'ground truth')

plt.plot(X_test, pred_mean, '--k', label = 'avg value')
plt.xlabel('time')
plt.ylabel('mag')
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.036434619162285936
variance 0.04029860246793506
```

In [36]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.005457059069296167
MSE_test:  0.0022353617074447 09
```

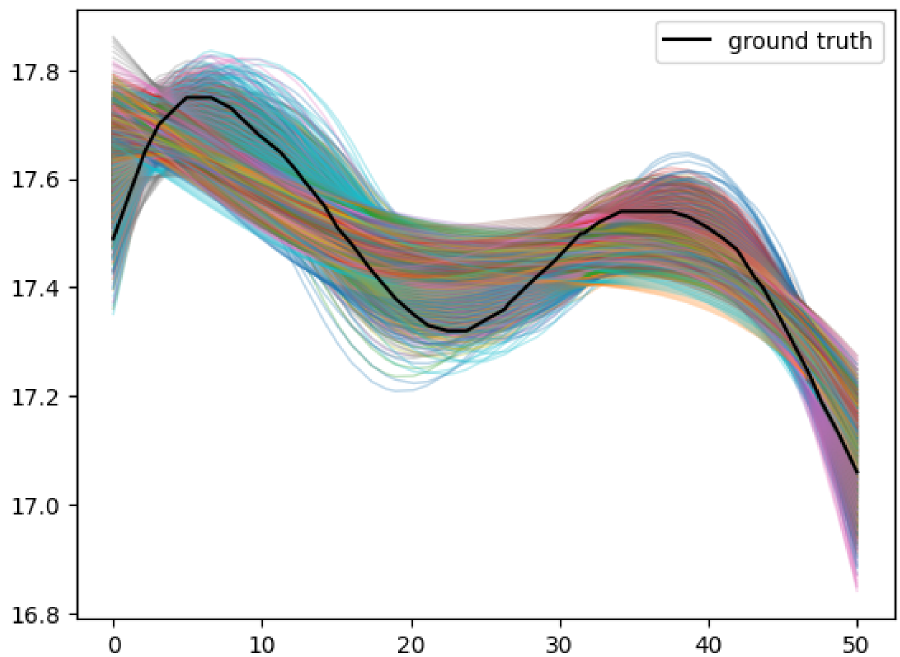
Splines (B-spline)¶ In [37]:

```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'

X_test = df_true[0].to_numpy()[:,np.newaxis]
Y_test = df_true[1].to_numpy()[:,np.newaxis]

X_train = df_noise2[0].to_numpy()[:,np.newaxis]
```

```
print(X_test.shape)
print(Y_test.shape)
print(X_train.shape)



Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()

degrees = list(range(2,20))

mean_bias = np.zeros(len(degrees))
mean_variance = np.zeros(len(degrees))

for j, degree in enumerate(degrees):
    bias = []
    y_pred_all = []

    for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        #create model
        model = make_pipeline(SplineTransformer(n_knots=4, degree=degree), Ridge(alpha=1e-3)
        #training
        model.fit(X_train, y_i)

        #predictions
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict (X_test)

        y_pred_all.append(Y_pred_test)

        bias.append(abs(Y_test - y_pred_all[i]))

        plt.plot(X_test, Y_pred_test,linewidth = 1, alpha = 0.3)

    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)
```

```
plt.plot(X_test, Y_test, label='ground truth', color = 'k')
plt.legend()
plt.show()

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)
```

```
(50, 1)
(50, 1)
(25, 1)
```



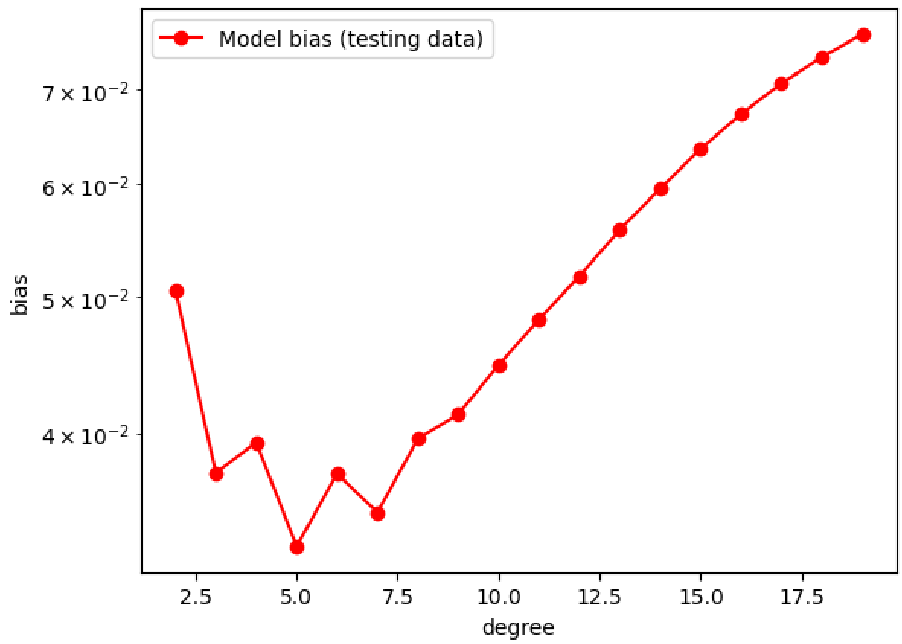

```
Mean Bias:
[0.05050902 0.0375114  0.03942022 0.03327819 0.03746337 0.0351709
 0.03965597 0.04126685 0.04471864 0.04815637 0.0516782  0.05572757
 0.05959692 0.0635737  0.06726199 0.07070857 0.07380965 0.07660179]

Mean Variance:
[0.03605991 0.03911008 0.03660843 0.03837006 0.03631333 0.03642909
 0.03537963 0.03481744 0.03424572 0.03364857 0.03314303 0.03262976
 0.0321779  0.03175405 0.03137057 0.03101514 0.03068416 0.03037109]
```

In [38]:

```
plt.xlabel('degree')
plt.ylabel('bias')
plt.plot(degrees, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[38]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

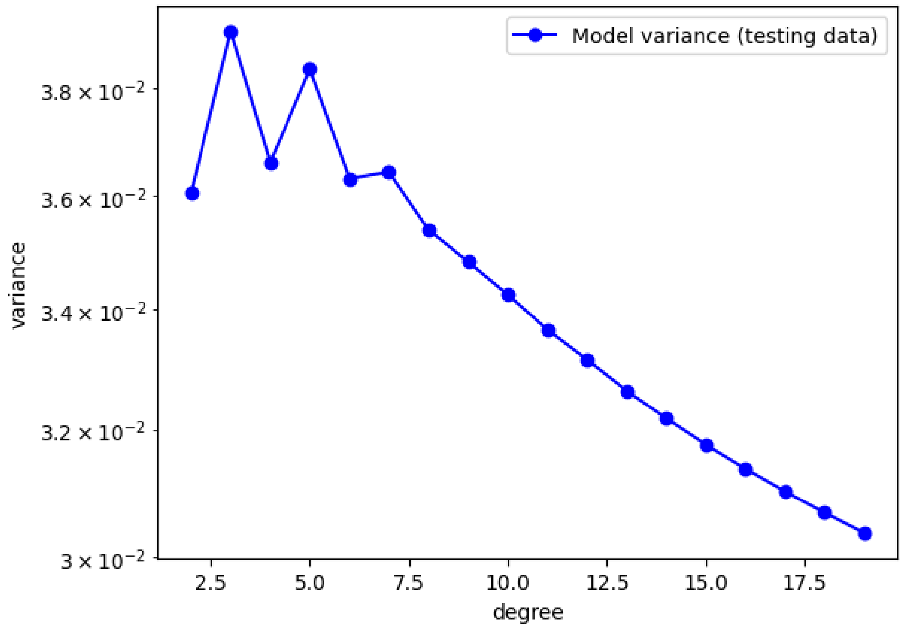


In [39]:

```
plt.xlabel('degree')
plt.ylabel('variance')
plt.plot(degrees, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[39]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [40]:

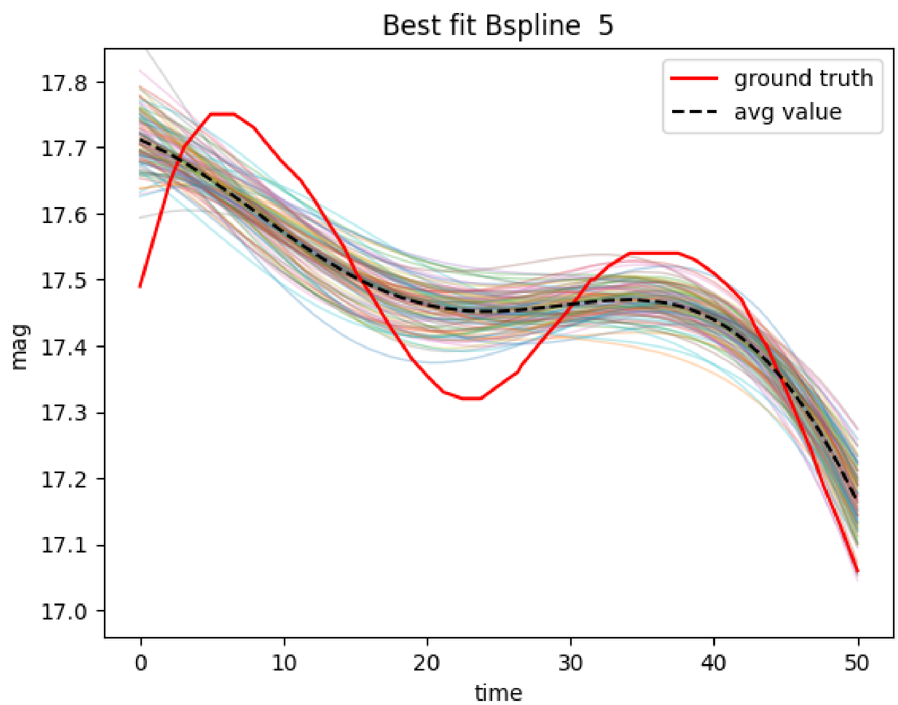
```
best_degree = degrees[np.argmin(mean_bias)]
print(best_degree)

index_min = np.argmin(mean_bias)
print(index_min)
```

```
5
3
```

In [41]:

```
plt.title( f'Best fit Bspline  {best_degree}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)

y_pred_all = []

MSE_list_train = []
MSE_list_test = []

for i in range(0, 100):
        y_i = y[:, i]
        y_i = y_i[:, np.newaxis]

        #create model
        model = make_pipeline(SplineTransformer(n_knots=4, degree=degree), Ridge(alpha=1e-3)
```

```
        #training
        model.fit(X_train, y_i)

        #predictions
        Y_pred_train = model.predict(X_train)
        Y_pred_test = model.predict (X_test)

        #MSE
        MSE_train = mean_squared_error(y_i, Y_pred_train)
        MSE_test = mean_squared_error(Y_test, Y_pred_test)

        MSE_list_train.append(MSE_train)
        MSE_list_test.append(MSE_test)

        y_pred_all.append(Y_pred_test)
        plt.plot(X_test,Y_pred_test,linewidth = 1, alpha = 0.3)

pred_mean = np.mean(y_pred_all, axis=0)
pred_variance = np.std(y_pred_all, axis=0)



plt.plot(X_test, Y_test, 'r', label = 'ground truth')

plt.plot(X_test, pred_mean, '--k', label = 'avg value')
plt.xlabel('time')
plt.ylabel('mag')
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.03327818833873526
variance 0.03837005736971363
```

In [42]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.01311374761653855
MSE_test:  0.00822981194506003
```

Fourier¶ In [43]:

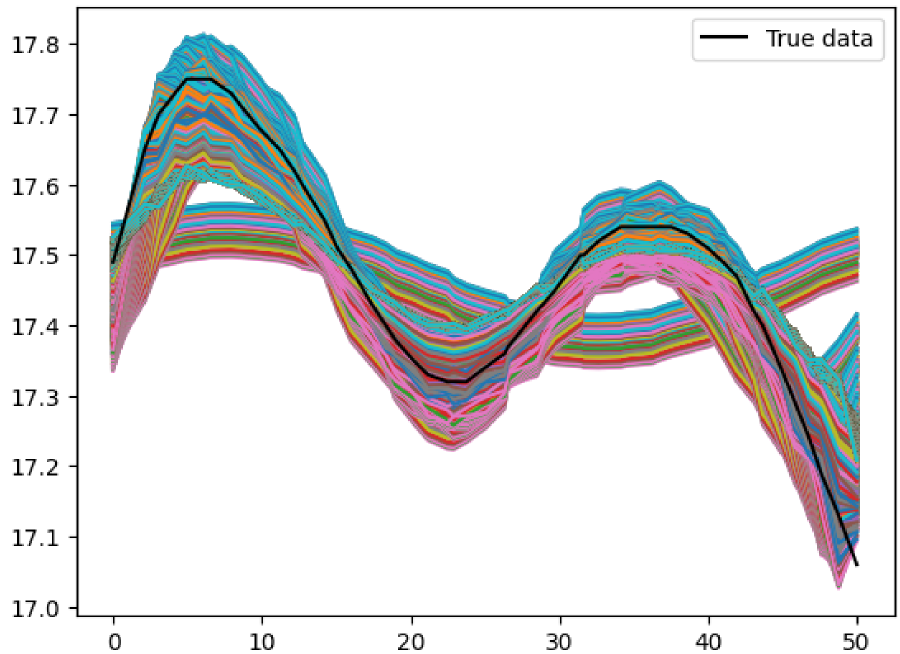
```
#DATA_PATH_NOISE1 = 'DS-5-1-GAP-1-1-N-1_v2.csv'
# Load data
X_test = df_true[0].to_numpy()
Y_test = df_true[1].to_numpy()

X_train = df_noise2[0].to_numpy()
```

```
Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()
y_i = y[:, 1]

components = list(range(2,40))
mean_bias = np.zeros(len(components))
mean_variance = np.zeros(len(components))

for j, component in enumerate(components):
    bias = []
    y_pred_all = []
    for i in range(0,100):
        y_i = y[:, i]

        # ---------------train
        # Calculate the Fourier transform
        Y_train_fft = fft(y_i)

        # Select Fourier components
        n_components = component
        Y_train_fft_filtered = np.zeros_like(Y_train_fft)
        Y_train_fft_filtered[:n_components] = Y_train_fft[:n_components]

        n_samples = y_i.size

        # Build the design matrix
        X_train_fourier = np.abs(np.fft.ifft(Y_train_fft_filtered).real)

        # Fit the linear regression model
        model = LinearRegression()
        model.fit(X_train_fourier[:, None], y_i)

        # ---------------test
        # Calculate the Fourier transform
        Y_test_fft = fft(Y_test)

        # # Select Fourier components
        Y_test_fft_filtered = np.zeros_like(Y_test_fft)
        Y_test_fft_filtered[:n_components] = Y_test_fft[:n_components]


        # Build the design matrix
        X_test_fourier = np.abs(np.fft.ifft(Y_test_fft_filtered).real)

        # Predictions
```

```
        Y_pred_test = model.predict(X_test_fourier[:, None])
        Y_pred_train = model.predict(X_train_fourier[:,None])
        y_pred_all.append(Y_pred_test)

        #bias
        bias.append(abs(Y_test - y_pred_all[i]))

        #plot
        plt.plot(X_test, Y_pred_test)

    # bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)

plt.plot(X_test, Y_test, label='True data', color = 'k')
plt.legend()
plt.show()
```

```
Mean Bias:
[0.12577774 0.061138   0.04949484 0.03898689 0.03380951 0.03049073
 0.02840981 0.02662245 0.02579574 0.02477627 0.02373892 0.02310106
 0.02223931 0.021583   0.0215506  0.02323303 0.02555745 0.02804311
 0.03144035 0.03777056 0.04103929 0.04645779 0.06225689 0.064384
 0.064248   0.06420337 0.06420024 0.06438641 0.06450803 0.06442076
 0.06436768 0.06437901 0.06444212 0.06464926 0.06442197 0.06437676
 0.06430318 0.06396704]

Mean Variance:
[1.59824999e-02 1.59824999e-02 1.59824999e-02 1.59824999e-02
 1.59824999e-02 1.59824999e-02 1.59824999e-02 1.59824999e-02
 1.59824999e-02 1.59824999e-02 1.59824999e-02 1.59824999e-02
 1.56832071e-02 1.52173651e-02 1.44632656e-02 1.42804211e-02
 1.31683349e-02 1.21803616e-02 1.12636615e-02 9.74479133e-03
 8.50066376e-03 6.77766520e-03 1.40842474e-03 2.23459254e-14
 2.36590378e-14 2.37157727e-14 2.50258127e-14 2.18866223e-14
 2.44614489e-14 2.31364935e-14 2.16639910e-14 2.37501244e-14
```

```
 2.50177896e-14 2.09323956e-14 2.27130044e-14 2.30426620e-14
 2.29557955e-14 2.37568718e-14]
```

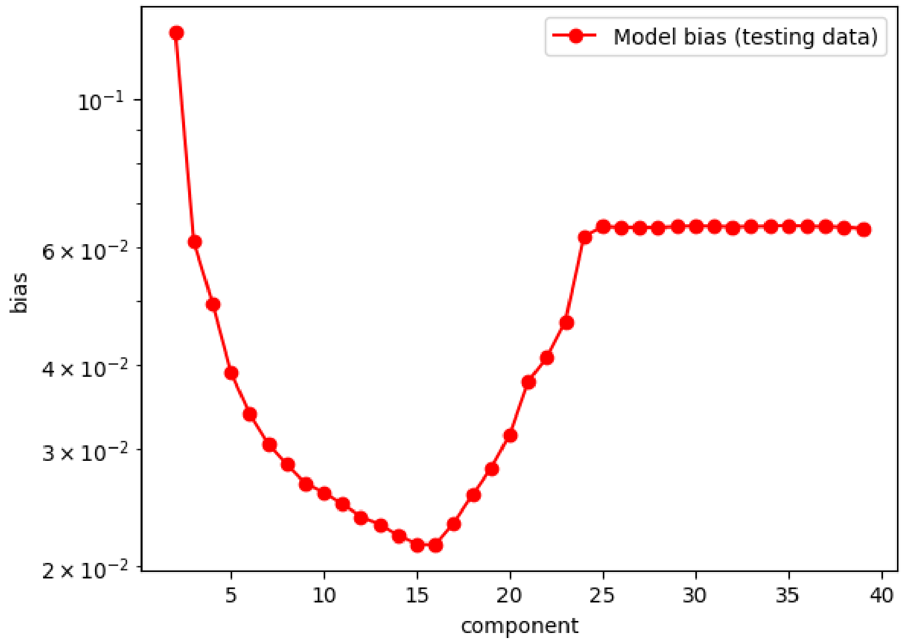


In [44]:

```
plt.xlabel('component')
plt.ylabel('bias')
plt.plot(components, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[44]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

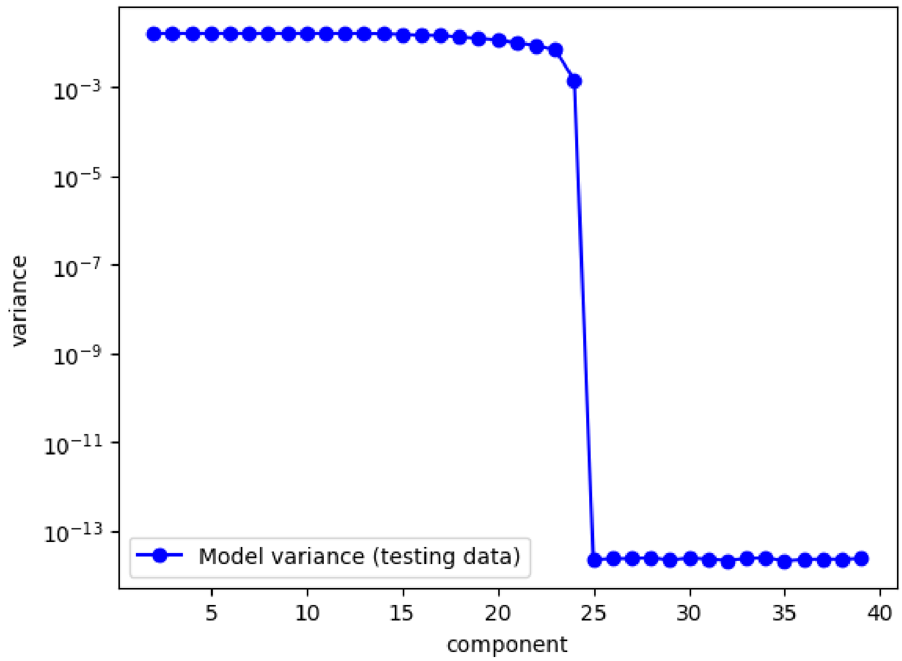


In [45]:

```
plt.xlabel('component')
plt.ylabel('variance')
plt.plot(components, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show
```

Out[45]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [46]:

```
best_ncomponents = components[np.argmin(mean_bias)]
print(best_ncomponents)

index_min = np.argmin(mean_bias)
print(index_min)
```

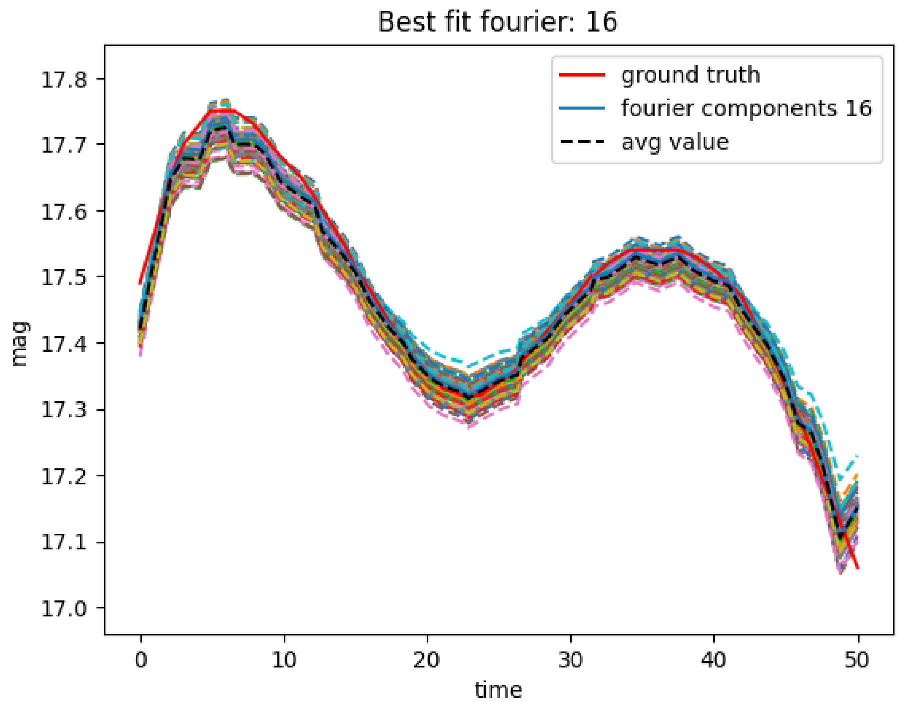
```
16
14
```

In [47]:

```
#DATA_PATH_NOISE2 = ''
Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()

y_pred_all = []

MSE_list_train = []
MSE_list_test = []

plt.title( f'Best fit fourier: {best_ncomponents}')
plt.ylim(Y_test.min() - 0.1, Y_test.max() + 0.1)

for i in range(0,100):
```

```
    y_i = y[:, i]
    # ---------------train
    # Calculate the Fourier transform
    Y_train_fft = fft(y_i)
    # Select Fourier components
    n_components = best_ncomponents
    Y_train_fft_filtered = np.zeros_like(Y_train_fft)
    Y_train_fft_filtered[:n_components] = Y_train_fft[:n_components]
    n_samples = y_i.size
    # Build the design matrix
    X_train_fourier = np.abs(np.fft.ifft(Y_train_fft_filtered).real)
    # Fit the linear regression model
    model = LinearRegression()
    model.fit(X_train_fourier[:, None], y_i)
    # ---------------test
    # Calculate the Fourier transform
    Y_test_fft = fft(Y_test.squeeze())
    # # Select Fourier components
    Y_test_fft_filtered = np.zeros_like(Y_test_fft)
    Y_test_fft_filtered[:n_components] = Y_test_fft[:n_components]
    # Build the design matrix
    X_test_fourier = np.abs(np.fft.ifft(Y_test_fft_filtered).real)
    # Predictions
    Y_pred_test = model.predict(X_test_fourier[:, None])
    Y_pred_train = model.predict(X_train_fourier[:,None])
    y_pred_all.append(Y_pred_test)

    # Calculate MSE on training set
    mse_train = mean_squared_error(y_i, Y_pred_train)
    MSE_list_train.append(mse_train)

    # Calculate MSE on test set
    mse_test = mean_squared_error(Y_test, Y_pred_test)
    MSE_list_test.append(mse_test)

    y_pred_all.append(Y_pred_test)

    plt.plot(X_test, Y_pred_test, linestyle='--')

pred_mean = np.mean(y_pred_all, axis=0)

plt.plot(X_test, Y_test, 'r', label = 'ground truth')
plt.plot(X_test, Y_pred_test,label = f'fourier components {best_ncomponents}')
plt.plot(X_test, pred_mean,'--k', label = 'avg value' )
plt.xlabel('time')
plt.ylabel('mag')
```

```
plt.legend(loc="best")
plt.show()

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
bias:  0.021550595508744327
variance 0.014463265597318848
```

In [48]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)


MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.0014485494377859663
MSE_test:  0.000855686788903582
```

Kernel methods¶ In [49]:

```
X = df_noise2[0]  #time
```

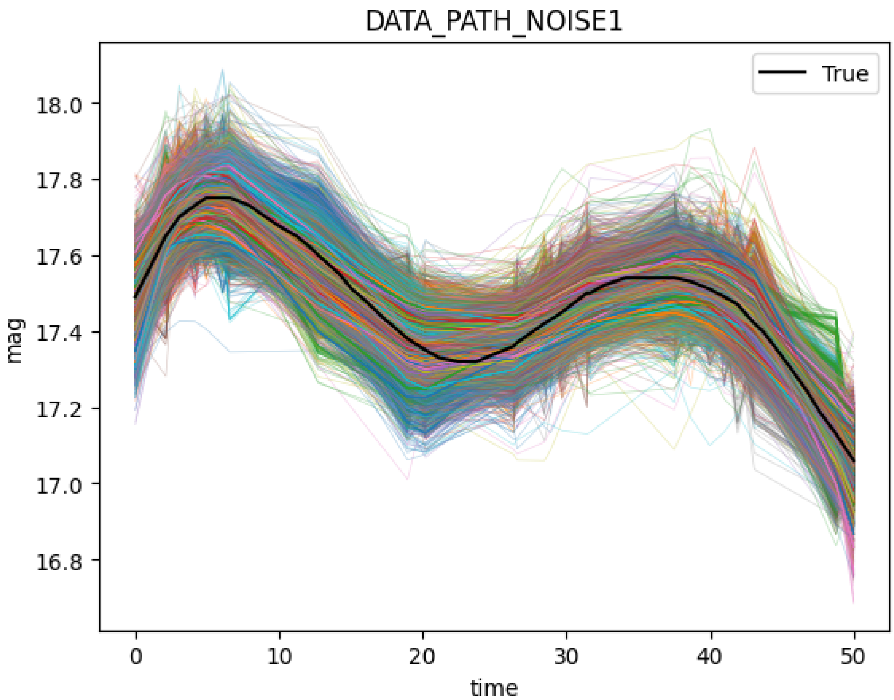
```
x = X.values.reshape(-1,1) #X[:, np.newaxis]
Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()

X_test = df_true[0]
Y_test = df_true[1]
x_test = X_test.values.reshape(-1,1) #x_test = X_test[:, np.newaxis]
y_test = Y_test.values.reshape(-1,1) #y_test = Y_test[:, np.newaxis]

in_Nsigma = 5

sigmas = list(range(in_Nsigma,200))

mean_bias = np.zeros(len(sigmas))
mean_variance = np.zeros(len(sigmas))

for j,sigma in enumerate(sigmas):
  bias = []
  y_pred_all = []
  for i in range(0,100):
    y_i = y[:, i]
    ones = []
    for k in range(0,Y.size):
      ones.append(1)

    #Do an array of ones
    ones1 = np.array(ones)

    #make de function K1 where
    #ex = data time
    #n = number of points
    #c = Centers of gaussians
    #k = number of kernels
    #d = kernels width
    def K1(ex,n,c,km,d):
      matrix = [[0 for _ in range(n)] for _ in range(km)]
      for i in range(0,km):
        for j in range (0,n):
          matrix[i][j] = m.exp(-(abs(ex[j]-c[i])**2/(d[j]**2))) #the kernel function
      return matrix

    #First Step: make the Gran_Matrix
    Gram_matrix = K1(X, X.size, X, X.size, ones1*sigma)
    #This function returns a Matrix instead an array
    Gram_matrix_M = np.asarray(Gram_matrix)
```

```
    #The pseudo inverse of the matrix wiht the function np.linalg.pinv
    #gettin the H matrix
    pinvGram_matrix_M = np.linalg.pinv(np.transpose(Gram_matrix_M))
    #In this point we can calculate the alpha becase we have the pseudo-inverse of the matri
    alpha = pinvGram_matrix_M.dot(y_i)

    #getting H from Alpha
    alphaT = np.transpose(alpha)
    #Remove axes of length one from alphaTD.
    alphaTD = np.squeeze(alphaT)

    #make the kernel method
    h = alphaTD.dot(Gram_matrix_M)

    #Remove axes of length one from h.
    hArray = np.squeeze(np.asarray(np.transpose(h)))

    #Metrics
    MSE_train = mean_squared_error(y_i,np.transpose(h))

    y_pred_all.append(hArray)
    #bias
    Y_test2 = Y_test[:25]
    bias.append(abs(Y_test2 - y_pred_all[i]))
    plt.plot(x, hArray,linewidth = 0.5, alpha = 0.3)
  # bias
  pred_mean = np.mean(bias, axis=0)
  mean_bias[j] = np.mean(pred_mean)

  # variance
  pred_variance = np.std(y_pred_all, axis=0)
  mean_variance[j] = np.mean(pred_variance)

#Plotting
plt.plot(x_test,y_test, color='k', label="True")

print("Mean Bias:")
print(mean_bias)
print()
print("Mean Variance:")
print(mean_variance)

plt.xlabel("time")
plt.ylabel("mag")
plt.legend(loc="best")
plt.title("DATA_PATH_NOISE1")
```

```
plt.show()
print(Y_test.shape)
print(hArray.shape)
```

```
Mean Bias:
[0.14916399 0.14916176 0.14891541 0.14836703 0.15051726 0.14778229
 0.14641765 0.14560266 0.14689153 0.15408288 0.15109249 0.14349138
 0.14380668 0.16289111 0.1478448  0.14214772 0.1436894  0.15095882
 0.14151712 0.14040474 0.140108   0.14386138 0.14049077 0.14036898
 0.14054452 0.13955111 0.1564727  0.1395443  0.1391404  0.14047392
 0.14226009 0.14315371 0.15466093 0.13873671 0.13878344 0.13870525
 0.13905365 0.13870802 0.13901585 0.13911316 0.13999445 0.14042122
 0.14401051 0.13755674 0.13761798 0.13734572 0.13763461 0.13752628
 0.13722268 0.13747679 0.1377762  0.13760431 0.13680323 0.14098319
 0.1424891  0.13947406 0.15918503 0.13700099 0.13696157 0.13703936
 0.13695551 0.13701373 0.13664241 0.13701378 0.13659549 0.13713615
 0.13674392 0.13720267 0.13600774 0.13693881 0.13533302 0.13816321
 0.13806637 0.13870477 0.1415371  0.14253672 0.1407797  0.13614225
 0.13611424 0.13619832 0.13614663 0.13629833 0.1361552  0.13617447
 0.13626743 0.13612509 0.13601275 0.13637292 0.13622398 0.13603926
 0.13632893 0.13622376 0.13581318 0.13591311 0.1364196  0.13594867
 0.13586625 0.13658599 0.13521126 0.13590235 0.13676528 0.13719055
 0.13575294 0.14047369 0.13685298 0.13647597 0.13684479 0.13835399
 0.13751887 0.13801613 0.1474597  0.16307463 0.13558263 0.13557771
 0.1355657  0.13555316 0.1354923  0.13553335 0.13552893 0.13552198
 0.13552131 0.13557313 0.13562097 0.13558047 0.13563214 0.13558047
 0.13556421 0.13557446 0.13559734 0.13553035 0.13543361 0.1353661
 0.13567684 0.13560899 0.13544234 0.13563618 0.13569135 0.13521235
 0.13544032 0.13561249 0.13520766 0.13531147 0.13525828 0.13572962
 0.1356965  0.13604988 0.13494244 0.13580389 0.1358816  0.13700471
 0.13571774 0.13598924 0.13479373 0.13596371 0.13627604 0.13654238
 0.13532045 0.13662905 0.13582202 0.13523941 0.13809721 0.13606968
 0.13503231 0.13744125 0.1349165  0.1353891  0.13769073 0.13728916
 0.13526335 0.13548808 0.14112159 0.13892531 0.14499565 0.13629612
 0.13722503 0.1465335  0.14509306 0.13749699 0.13805683 0.13907699
 0.14601277 0.13410022 0.13409461 0.13408876 0.13409222 0.13410067
 0.13410143 0.13410162 0.1340858  0.13409768 0.13409114 0.13408549
 0.13408744 0.13411986 0.1340955 ]

Mean Variance:
[0.08277134 0.08277627 0.08411498 0.08115432 0.07968443 0.0806095
 0.08781843 0.0730986  0.07123579 0.09150605 0.08955497 0.06622762
 0.06936519 0.12019694 0.06774161 0.0621755  0.06338523 0.07546547
 0.05985534 0.06013472 0.06196001 0.07443342 0.05750195 0.05743236
 0.05795266 0.05901016 0.06861162 0.05475209 0.05500521 0.0569063
 0.06016395 0.06883434 0.0997435  0.05191139 0.05198902 0.0520183
```

```
0.05212524 0.05195245 0.05275261 0.0551795  0.05625624 0.06181845
0.07710489 0.04956715 0.04953005 0.0496227  0.04962118 0.04955802
0.04970143 0.04960125 0.04990537 0.0508647  0.050745   0.05237265
0.05387513 0.05901056 0.0610409  0.04678229 0.04671765 0.04669429
0.04676067 0.04682459 0.04670864 0.04675245 0.04686191 0.04717573
0.04699368 0.04749332 0.04761793 0.04751429 0.04775655 0.04975389
0.05137112 0.05421684 0.05533403 0.06735697 0.06810348 0.04400634
0.0439653  0.04400575 0.04400488 0.04399386 0.04394894 0.04401818
0.04396224 0.04400029 0.04404825 0.04404605 0.04400862 0.04404732
0.04422914 0.04408201 0.04409343 0.04434043 0.04417079 0.04385439
0.04442731 0.04460313 0.04450333 0.04516421 0.04510139 0.04489974
0.04523557 0.04584656 0.04679107 0.04700834 0.0488607  0.04894805
0.05080725 0.05538207 0.05451897 0.05590043 0.04163215 0.04162758
0.0416302  0.04163518 0.04162317 0.04164178 0.04160469 0.04162448
0.04162263 0.0416307  0.04164077 0.04162847 0.04164472 0.0416356
0.04164457 0.04154485 0.04158905 0.04170392 0.04158618 0.04152891
0.04161695 0.04172644 0.04163608 0.04158385 0.04161057 0.04166414
0.04164852 0.04166485 0.04180004 0.04153844 0.04173688 0.04172948
0.04171184 0.04162529 0.04163238 0.04206052 0.04185066 0.0420357
0.0418883  0.04199242 0.04192142 0.04139173 0.04180645 0.04182189
0.04199594 0.04245411 0.0423953  0.04248937 0.04239528 0.04309683
0.04281495 0.04263015 0.0435908  0.044969   0.04377797 0.04616786
0.04424271 0.04539093 0.04680685 0.04766887 0.04617161 0.04522276
0.05040455 0.04657331 0.05285281 0.0531796  0.05361612 0.05926994
0.05810551 0.03920715 0.0392018  0.03920755 0.03920115 0.03920175
0.03920319 0.03920681 0.03920138 0.0392045  0.03919862 0.03920053
0.03920606 0.0392007  0.03919683]
```



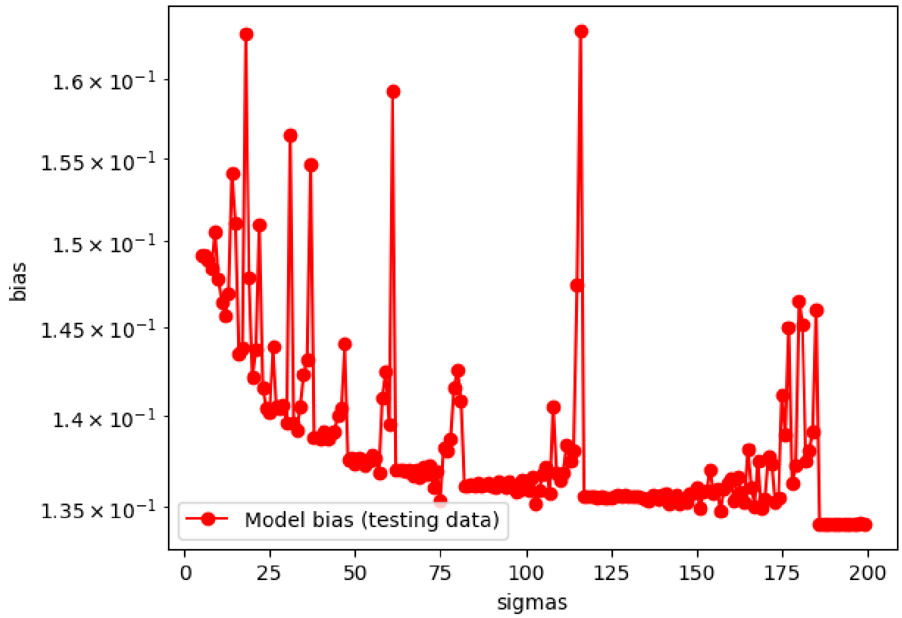

```
(50,)
(25,)
```

In [50]:

```
plt.xlabel('sigmas')
plt.ylabel('bias')
plt.plot(sigmas, mean_bias,'-ro', label = 'Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
# plt.xticks(range(in_Nsigma,len(sigmas)))
plt.show
```

Out[50]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

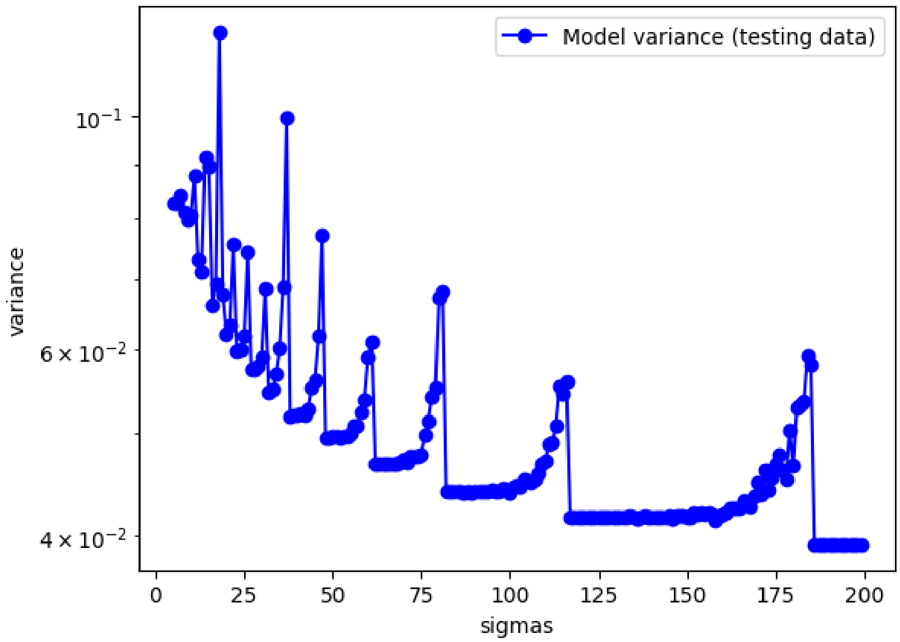


In [51]:

```
plt.xlabel('sigmas')
plt.ylabel('variance')
plt.plot(sigmas, mean_variance,'-bo', label = 'Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
# plt.xticks(range(in_Nsigma,len(sigmas)))
plt.show
```

Out[51]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [52]:

```
best_sigma = sigmas[np.argmin(mean_bias)]
print(best_sigma)

index_min = np.argmin(mean_bias)
print(index_min)
```

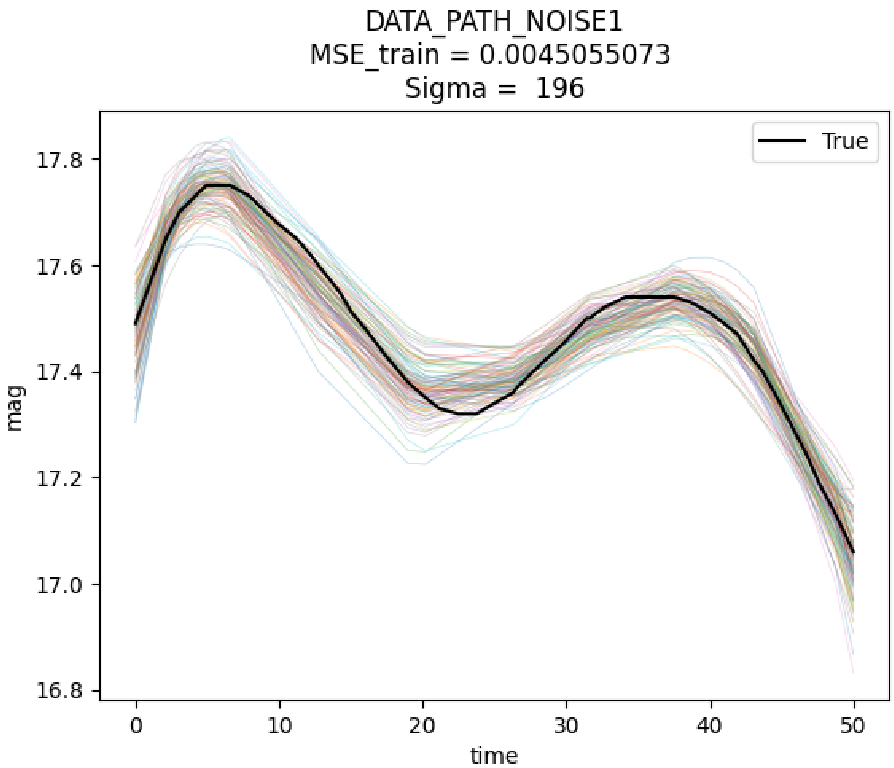
```
196
191
```

In [53]:

```
X = df_noise2[0]  #time
x = X.values.reshape(-1,1) #X[:, np.newaxis]
Y = df_noise2.iloc[:,1:101]
y = Y.to_numpy()

X_test = df_true[0]
Y_test = df_true[1]
x_test = X_test.values.reshape(-1,1) #x_test = X_test[:, np.newaxis]
y_test = Y_test.values.reshape(-1,1) #y_test = Y_test[:, np.newaxis]

sigma=best_sigma

MSE_train_list = []
```

```
MSE_test_list = []

for i in range(0,100):
    y_i = y[:, i]
    ones = []
    for k in range(0,Y.size):
      ones.append(1)

    #Do an array of ones
    ones1 = np.array(ones)

    #make de function K1 where
    #ex = data time
    #n = number of points
    #c = Centers of gaussians
    #k = number of kernels
    #d = kernels width
    def K1(ex,n,c,km,d):
      matrix = [[0 for _ in range(n)] for _ in range(km)]
      for i in range(0,km):
        for j in range (0,n):
          matrix[i][j] = m.exp(-(abs(ex[j]-c[i])**2/(d[j]**2))) #the kernel function
      return matrix

    #First Step: make the Gran_Matrix
    Gram_matrix = K1(X, X.size, X, X.size, ones1*sigma)
    #This function returns a Matrix instead an array
    Gram_matrix_M = np.asarray(Gram_matrix)

    #The pseudo inverse of the matrix wiht the function np.linalg.pinv
    #gettin the H matrix
    pinvGram_matrix_M = np.linalg.pinv(np.transpose(Gram_matrix_M))
    #In this point we can calculate the alpha becase we have the pseudo-inverse of the matri
    alpha = pinvGram_matrix_M.dot(y_i)

    #getting H from Alpha
    alphaT = np.transpose(alpha)
    #Remove axes of length one from alphaTD.
    alphaTD = np.squeeze(alphaT)

    #make the kernel method
    h = alphaTD.dot(Gram_matrix_M)

    #Remove axes of length one from h.
    hArray = np.squeeze(np.asarray(np.transpose(h)))
```

```
    #Metrics
    MSE_train = mean_squared_error(y_i,np.transpose(h))


    #---------------------------
    Gram_matrix_test = K1(X_test, X_test.size, X, X.size, ones1 * sigma)

    # Calculate predictions for test data
    h_test = alphaTD.dot(Gram_matrix_test)

    # Calculate MSE for test data
    MSE_test = mean_squared_error(y_test, np.transpose(h_test))

    MSE_list_train.append(MSE_train)
    MSE_list_test.append(MSE_test)

    plt.plot(x, hArray,linewidth = 0.5, alpha = 0.3)

#Plotting
plt.plot(x_test,y_test, color='k', label="True")
plt.xlabel("time")
plt.ylabel("mag")
plt.legend(loc="best")
plt.title("DATA_PATH_NOISE1\nMSE_train = {:.8} \nSigma =  {:}".format(MSE_train,sigma))
plt.show()
print(Y_test.shape)
print(hArray.shape)

print('bias: ', mean_bias[index_min])
print('variance', mean_variance[index_min])
```




```
(50,)
(25,)
bias:  0.13408548596692113
variance 0.03920053391968567
```

In [54]:

```
MSE_train = np.mean(MSE_list_train)
print('MSE_train: ',MSE_train)

MSE_test = np.mean(MSE_list_test)
print('MSE_test: ',MSE_test)
```

```
MSE_train:  0.003448622531903397
MSE_test:  0.0015359072407190632
```

Table: Dataset: DS-5-1-GAP-5-1-N-3¶

Regression	MSE training	MSE Testing (ground truth)	Bias
Polynomial (degree = 5)	0.005457059069296167	0.002235361707444709	0.0364346191622859
Splines (degree = 5)	0.01311374761653855	0.00822981194506003	0.0332781883387352

Regression	MSE training	MSE Testing (ground truth)	Bias
Fourier(n_components = 16)	0.0014485494377859663	0.000855686788903582	0.0215505955087443
Kernel method(sigma = 59)	0.0032074350483016247	0.002130008622080765	0.136522

GRNN¶

In [8]:

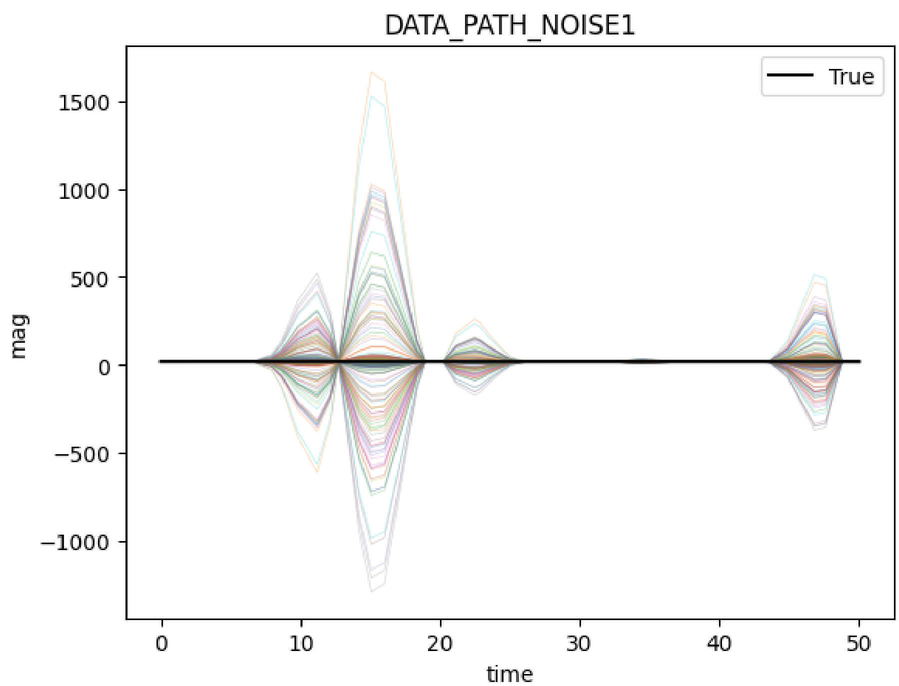
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import math as m
```

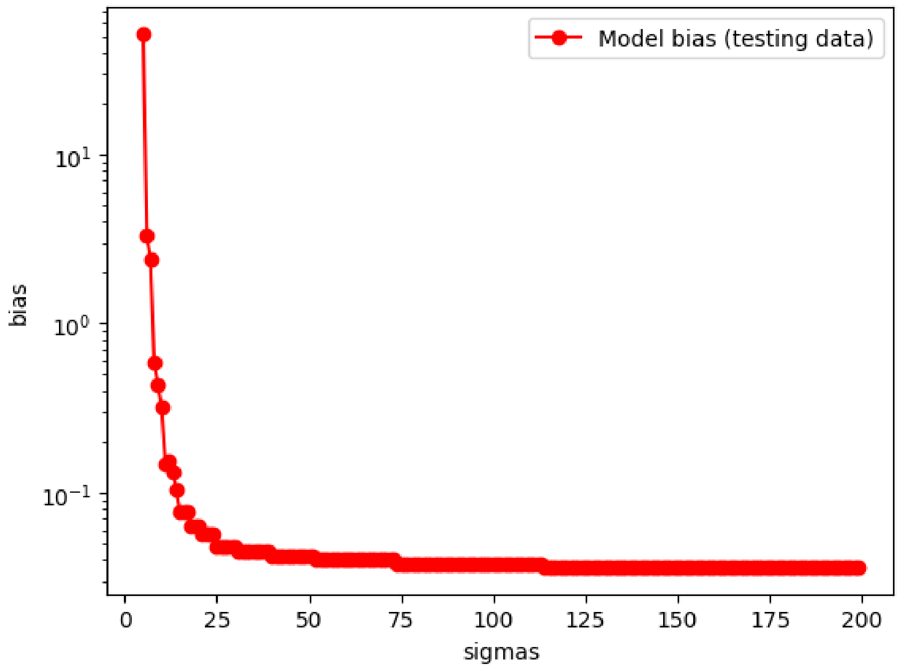
In [9]:

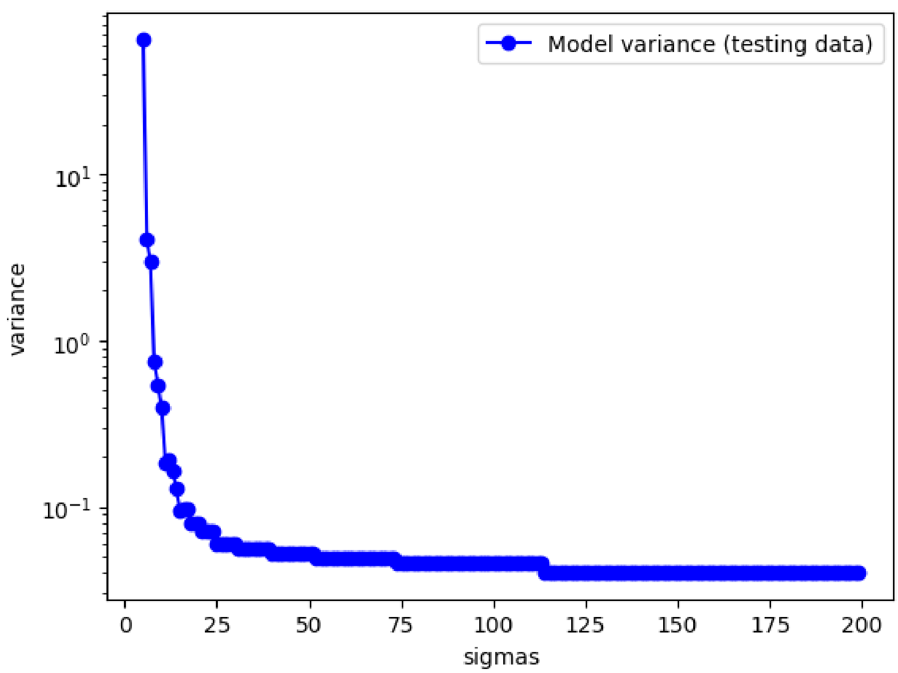
```
def GRNN(X_train, y_train, X_test, sigma):
    # Calculate the Gram matrix
    Gram_matrix = np.exp(-((X_train[:, None] - X_train) ** 2) / (2 * (sigma ** 2)))

    # Calculate the alpha coefficients
    alpha = np.linalg.lstsq(Gram_matrix, y_train, rcond=None)[0]

    # Calculate predictions for training data
    h_train = Gram_matrix.dot(alpha)

    # Calculate predictions for test data
    Gram_matrix_test = np.exp(-((X_test[:, None] - X_train) ** 2) / (2 * (sigma ** 2)))
    h_test = Gram_matrix_test.dot(alpha)

    return h_train, h_test

# Assuming df_noise2 and df_true are your dataframes

X_train = df_noise2[0].values
y_train = df_noise2.iloc[:, 1:101].values
X_test = df_true[0].values
y_test = df_true[1].values

in_Nsigma = 5
sigmas = list(range(in_Nsigma, 200))

mean_bias = np.zeros(len(sigmas))
mean_variance = np.zeros(len(sigmas))
MSE_list_train = []
MSE_list_test = []
```

```
for j, sigma in enumerate(sigmas):
    bias = []
    y_pred_all = []
    for i in range(0, 100):
        # Use GRNN for prediction
        h_train, h_test = GRNN(X_train, y_train[:, i], X_test, sigma)

        # Metrics
        MSE_train = mean_squared_error(y_train[:, i], h_train)
        MSE_test = mean_squared_error(y_test, h_test)
        MSE_list_train.append(MSE_train)
        MSE_list_test.append(MSE_test)

        y_pred_all.append(h_test)
        # Bias
        bias.append(abs(y_test - h_test))
        plt.plot(X_test, h_test, linewidth=0.5, alpha=0.3)

    # Bias
    pred_mean = np.mean(bias, axis=0)
    mean_bias[j] = np.mean(pred_mean)

    # Variance
    pred_variance = np.std(y_pred_all, axis=0)
    mean_variance[j] = np.mean(pred_variance)

# Plotting
plt.plot(X_test, y_test, color='k', label="True")
plt.xlabel("time")
plt.ylabel("mag")
plt.legend(loc="best")
plt.title("DATA_PATH_NOISE1")
plt.show()

# Plotting bias
plt.xlabel('sigmas')
plt.ylabel('bias')
plt.plot(sigmas, mean_bias, '-ro', label='Model bias (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show()

# Plotting variance
plt.xlabel('sigmas')
plt.ylabel('variance')
```

```
plt.plot(sigmas, mean_variance, '-bo', label='Model variance (testing data)')
plt.legend(loc='best')
plt.yscale('log')
plt.show()

best_sigma = sigmas[np.argmin(mean_bias)]

# Evaluating with the best sigma
index_min = np.argmin(mean_bias)
print("Best Sigma:", best_sigma)
print("Bias:", mean_bias[index_min])
print("Variance:", mean_variance[index_min])
print("MSE_train:", np.mean(MSE_list_train))
print("MSE_test:", np.mean(MSE_list_test))
```








```
Best Sigma: 114
```

```
Bias: 0.03618458315201611
Variance: 0.04031529327775219
MSE_train: 0.004858962633963207
MSE_test: 110.88202271681308
```

In [ ]: