Homework 2: Bayesian Estimation & Linear Regression

SHREYA CHETAN PAWASKAR
pawaskas@uci.edu
12041645

Question 1:

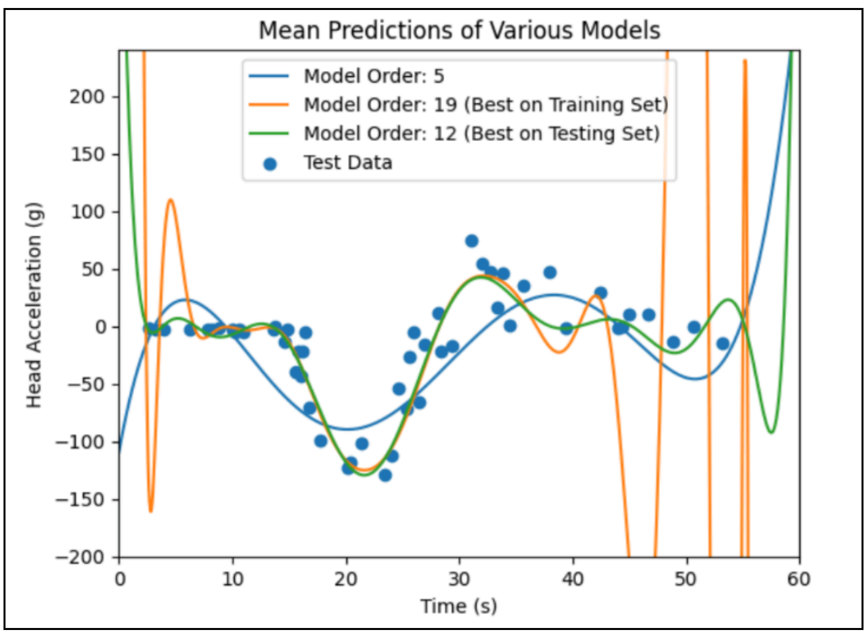
Code:

```
import numpy as np
import matplotlib.pyplot as plt
from basis_utils import basis_poly, basis_radial

data = np.load("motor.npy", allow_pickle=True).item()
X_train = data["Xtrain"]
Y_train = data["Ytrain"]
X_test = data["Xtest"]
Y_test = data["Ytest"]

# The 2 held-out points
X_held_out = np.asarray([57.6])
Y_held_out = np.asarray([10.7])

num_train_samples = X_train.shape[0]
num_test_samples = X_test.shape[0]

X_offset = (np.min(X_train) + np.max(X_train))/ 2.0
X_scale = (np.max(X_train) - np.min(X_train))/ 2.0
X_train_rescaled = (X_train - X_offset) / X_scale
X_test_rescaled = (X_test - X_offset) / X_scale
X_held_out_rescaled = (X_held_out - X_offset) / X_scale

time_grid = np.linspace(0, 60, 1000)
time_grid_rescaled = (time_grid - X_offset) / X_scale

L = ["Poly", "RBF"]

for i in L:
   plt.clf()
   plt.close()

   # Select basis function
   basis_function = None
   if i == "Poly":
       basis_function = basis_poly
       orders_list = list(range(20))
       example_index = 5
       print('Polynomial')
   else:
```

```
        basis_function = basis_radial
        orders_list = list(range(0, 26, 5))
        example_index = 1
        print('Radial Basis Function')

    alpha = 0
    beta = 1
    lambda_var = alpha/beta

    # Initialize arrays for storing results
    num_models = len(orders_list)
    wts_all = np.zeros((max(orders_list) + 1, num_models))
    func_all = np.zeros((time_grid.shape[0], num_models))
    train_error = np.zeros((num_models,))
    test_error = np.zeros((num_models,))
    held_out_error = np.zeros((num_models,))

    for iter in range(len(orders_list)):
        order = orders_list[iter]
        M = order + 1

        phi_train = basis_function(X_train_rescaled, order)
        phi_test = basis_function(X_test_rescaled, order)
        phi_grid = basis_function(time_grid_rescaled, order)
        phi_held_out = basis_function(X_held_out_rescaled, order)

        #least squares
        a = np.matmul(phi_train.T, phi_train) + (lambda_var * np.eye(M))
        b = np.matmul(phi_train.T, Y_train)
        wts = np.linalg.lstsq(a, b, rcond=-1)[0]

        wts_all[:M, iter] = wts
        func_all[:, iter] = np.matmul(phi_grid, wts)
        train_error[iter] = np.sqrt(np.mean((Y_train - np.matmul(phi_train, wts)) ** 2))
        test_error[iter] = np.sqrt(np.mean((Y_test - np.matmul(phi_test, wts)) ** 2))
        held_out_error[iter] = np.sqrt(np.mean((Y_held_out - np.matmul(phi_held_out, wts)) **
2))

    train_min = np.min(train_error)
    test_min = np.min(test_error)
    train_index = np.argmin(train_error)
    test_index = np.argmin(test_error)

    fig11 = plt.figure(1)
    plt.plot(time_grid, func_all[:, example_index],
             label=f"Model Order: {orders_list[example_index]}")
    plt.plot(time_grid, func_all[:, train_index],
             label=f"Model Order: {orders_list[train_index]} (Best on Training Set)")
    plt.plot(time_grid, func_all[:, test_index],
             label=f"Model Order: {orders_list[test_index]} (Best on Testing Set)")
    plt.scatter(X_test, Y_test, label="Test Data")
    plt.axis([0, 60, -200, 250])
    plt.xlabel('Time (s)')
```

```
plt.ylabel('Head Acceleration (g)')
plt.title('Mean Predictions of Various Models')
plt.legend()

#loss vs. model order
fig21 = plt.figure(2)
plt.plot(orders_list, train_error, '-o', label="Training Error")
plt.plot(orders_list, test_error, '-s', label="Testing Error")
plt.axis([0, max(orders_list), 0, 200])
plt.xlabel('Model Order')
plt.ylabel('Loss')
plt.title('Loss vs. Model Order')
plt.legend()
plt.show()

# loss on held-out point
print("Loss - best model on Training Set:", held_out_error[train_index])
print("Loss - best model on Testing Set:", held_out_error[test_index])
```

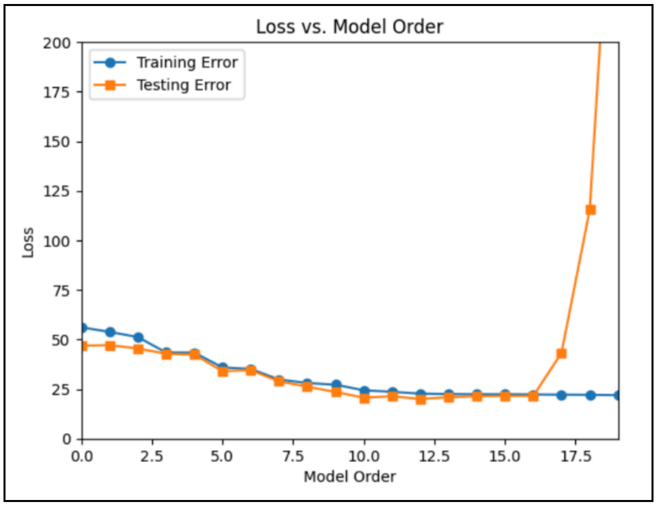
a) Polynomial basis functions: Plot of prediction functions for model order M = 5



---

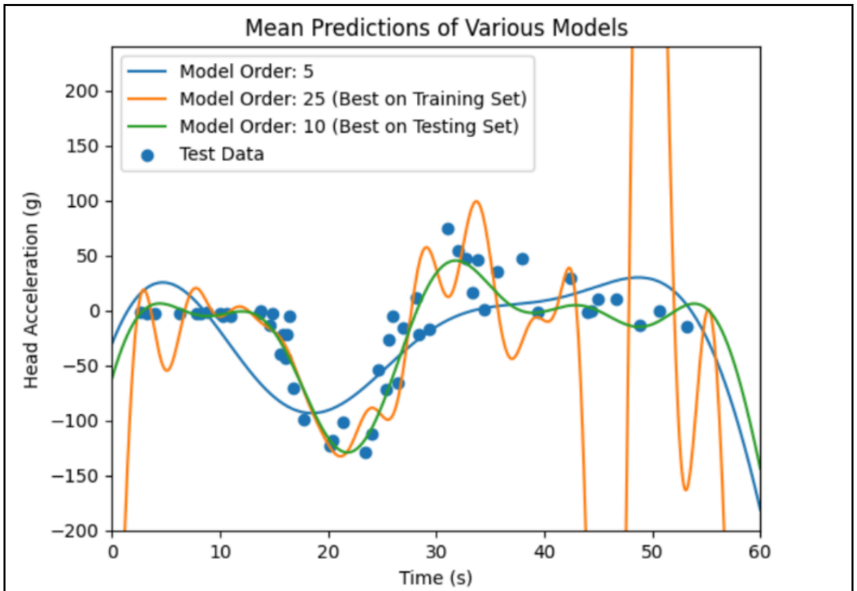
b) Training and test loss as a function of the model:

- *Training loss is smallest for M = 19*
- *Test loss is smallest for M = 12*



c) Radial basis function models of order L = 0, 5, 10, 15, 20, 25

- Training loss is smallest for M = 25
- Test loss is smallest for M = 10



d)

```
# loss on held-out point
print(" Loss - best model on Training Set:", held_out_error[train_index])
print(" Loss - best model on Testing Set:", held_out_error[test_index])
```

- For polynomial models:
  - The best polynomial model on training data (M = 19) has a loss over 264,000.
  - The best polynomial model on test data (M = 12) has a loss of 102.7.
- For RBF models:
  - The best RBF model on training data (M = 25) has a loss of 490.5.
  - The best RBF model on test data (M = 12) has a loss of 59.2.

- The time value at x = 57.6 occurs just milliseconds after the last training time, x = 55.4.
- This particular test point allows us to examine how the robustness of models varies depending on order & basis functions used.

- All predictions were significantly worse than the loss of 40.6.
- The higher-order polynomial functions can grow rapidly. This leads to unreliable predictions outside the range of training data. So we can see that the performance of the polynomial model with M = 19 on the held-out point is extremely poor.

Question 2:

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from basis_utils import basis_poly, basis_radial
```

```
data = np.load("motor.npy", allow_pickle=True).item()
X_train = data["Xtrain"]
Y_train = data["Ytrain"]
X_test = data["Xtest"]
Y_test = data["Ytest"]

X_held_out = np.asarray([57.6])
Y_held_out = np.asarray([10.7])
```

```
num_train_samples = X_train.shape[0]
num_test_samples = X_test.shape[0]

X_offset = (np.min(X_train) + np.max(X_train))/ 2.0
X_scale = (np.max(X_train) - np.min(X_train))/ 2.0
X_train_rescaled = (X_train - X_offset) / X_scale
X_test_rescaled = (X_test - X_offset) / X_scale
X_held_out_rescaled = (X_held_out - X_offset) / X_scale
```

```
time_grid = np.linspace(0, 60, 1000)
time_grid_rescaled = (time_grid - X_offset) / X_scale

L = ["Poly", "RBF"]
```

```
for i in L:
 # Do the actual basis function selection.
 basis_function = None
 if i == "Poly":
     basis_function = basis_poly
 else:
     basis_function = basis_radial

order = 50
 M = order + 1

 #basis funcs
 phi_train = basis_function(x_train_rescaled, order)
 phi_test = basis_function(x_test_rescaled, order)
 phi_grid = basis_function(x_grid_rescaled, order)
 phi_out = basis_function(x_out_rescaled, order)

 # Regularization weight
 alpha = np.logspace(-8, 8, 100)
```

```
beta = 0.0025 * np.ones(shape=alpha.shape)
lambda_var = alpha / beta

# L/MAP least squares fit for each model
shape1 = lambda_var.shape[0]
wts_all = np.zeros((order + 1, shape1))
func_all = np.zeros((x_grid.shape[0], shape1))
train_err = np.zeros((shape1,))
test_err = np.zeros((shape1,))
out_err = np.zeros((shape1,))

for id1 in range(shape1):
    a = np.matmul(phi_train.T, phi_train) + (lambda_var[id1] * np.eye(M))
    b = np.matmul(phi_train.T, y_train)

    wts = np.linalg.lstsq(a, b, rcond=-1)[0]

    wts_all[:M, id1] = wts
    func_all[:, id1] = np.matmul(phi_grid, wts)
    train_err[id1] = np.sqrt(np.mean((y_train - np.matmul(phi_train, wts)) ** 2))
    test_err[id1] = np.sqrt(np.mean((y_test - np.matmul(phi_test, wts)) ** 2))
    out_err[id1] = np.sqrt(np.mean((y_held_out - np.matmul(phi_out, wts)) ** 2))

train_min = np.min(train_err)
test_min = np.min(test_err)
train_index = np.argmin(train_err)
test_index = np.argmin(test_err)

figures_List = []
# Plot mean predictions
figures_List.append(plt.figure(1))
plt.plot(x_grid, func_all[:, train_index],
        label="alpha={:0.9f} (best on train)".format(alpha[train_index]))
plt.plot(x_grid, func_all[:, test_index],
        label="alpha={:0.9f} (best on test)".format(alpha[test_index]))
plt.scatter(x_test, y_test, color ="brown",marker='.', linewidth=linewidth)
plt.axis([0, 60, -200, 240])
plt.xlabel('Time')
plt.ylabel('Head Acceleration')
plt.legend()

#Training and test error vs. model order and held-out error
figures_List.append(plt.figure(2))
plt.semilogx(alpha, train_err, label="Training set error")
plt.semilogx(alpha, test_err,  label="Test set error")
plt.axis([np.min(alpha), np.max(alpha), 0, 100])
plt.xlabel('Alpha')
plt.ylabel('Loss')
plt.legend()

# Plot samples from Gaussian posterior
post_inv_covar = (beta[test_index] * np.matmul(phi_train.T, phi_train)) \
                + (alpha[test_index] * np.eye(M))
```

```
post_mean = np.linalg.lstsq(post_inv_covar, \
                            (beta[test_index] * np.matmul(phi_train.T, y_train)),
rcond=-1)[0]
post_inv_sqrt = np.linalg.cholesky(post_inv_covar)
post_covar = np.linalg.inv(post_inv_covar)
post_covar = 0.5 * (post_covar + post_covar.T)
num_samples = 10
func_samples = np.zeros((x_grid.shape[0], num_samples))
for i in range(num_samples):
    weight_samp = np.random.multivariate_normal(post_mean.T, post_covar).T
    func_samples[:, i] = np.matmul(phi_grid, weight_samp)

figures_List.append(plt.figure(3))
plt.plot(x_grid, func_samples)
plt.axis([0, 60, -200, 250])
plt.xlabel('Time')
plt.ylabel('Head Acceleration')
```

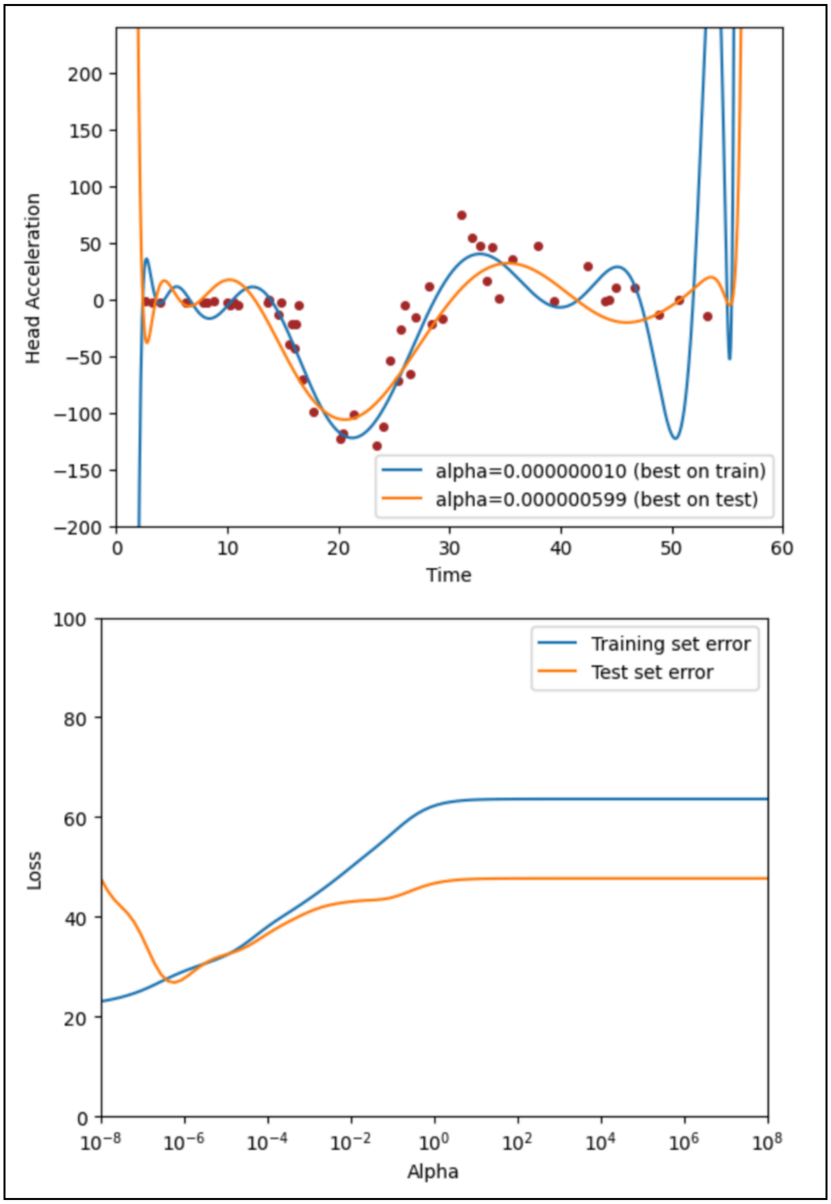
a)

1. In a Gaussian posterior, the mode and mean are the same, making the mean the MAP estimate for w.
2. For any model order M > 0, the mean and MAP estimate are unique.
3. However, with N = 40 training examples and M = 50, the ML estimate wouldn't be uniquely defined.
4. With more unknowns (polynomial coefficients) than equations (known mappings), there are infinitely many weight vectors that fit the data perfectly.
5. These weight vectors form an affine subspace.

---

b)

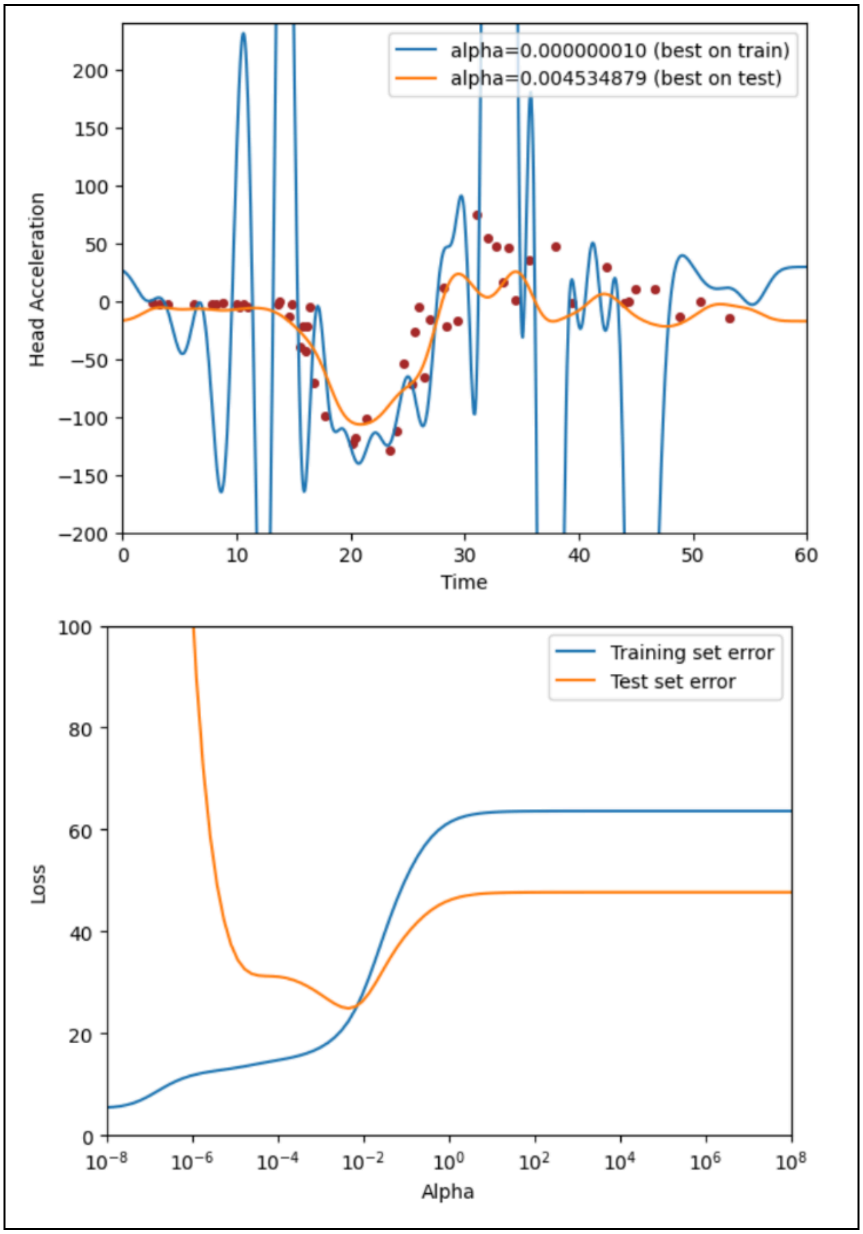
- *Training loss is smallest for alpha = 0.00000010*
- *Test loss is smallest for alpha = 0.000000599*



---

c)

- *Training loss is smallest for alpha = 0.000000010*
- *Test loss is smallest for alpha = 0.004534879*



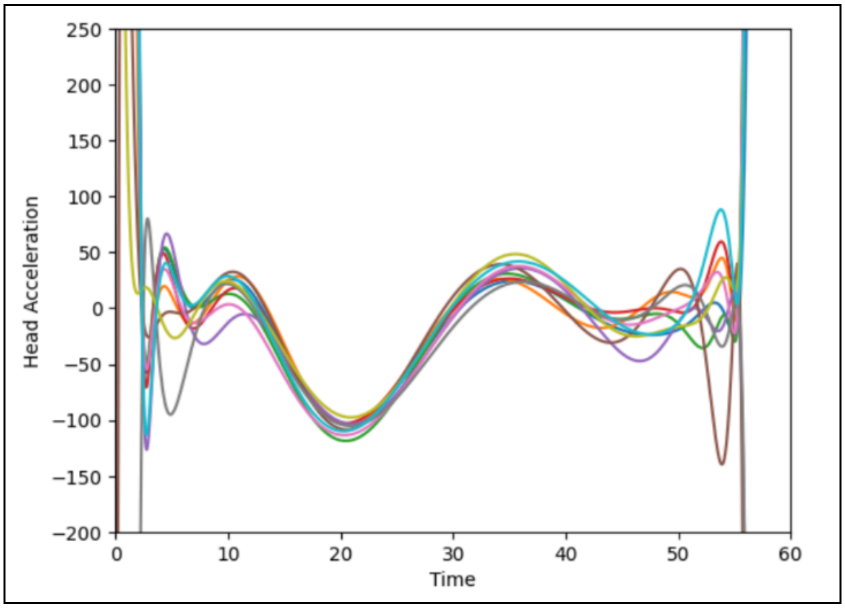
d)

- The posterior distribution for w is $w \sim \mathcal{N}(m_N, S_N)$, where m_N, S_N are given to us.
- According to the model, $f(x) = w^T \phi(x)$ denotes the model function.
- Given a fixed x but treating w as a Gaussian random variable, the distribution of $t = f(x)$ can be derived using standard properties of linear transforms of Gaussian random variables:

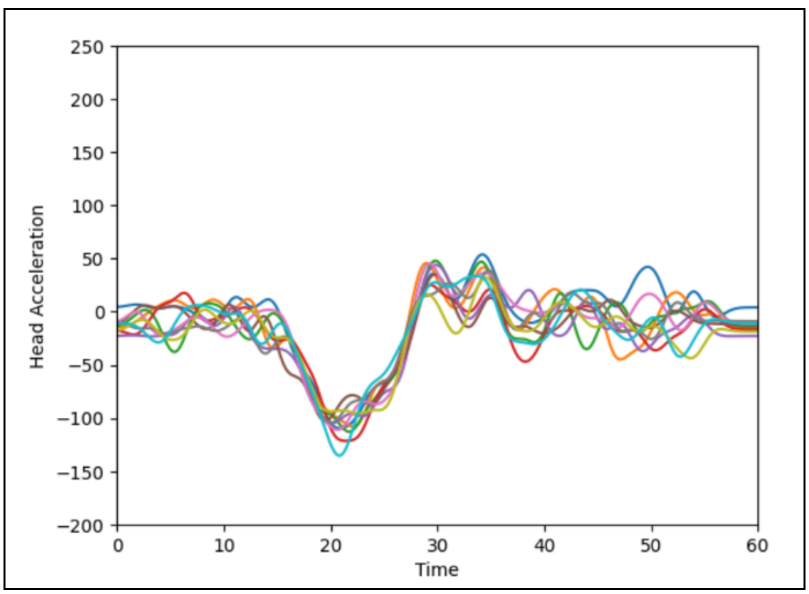
$$t \sim \mathcal{N}\left(m_N^T \phi(x), \phi(x)^T S_N \phi(x)\right)$$

- The graphs depict sample functions, generated by first sampling weights *w* from its posterior distribution and then setting $t = \Phi w$.

- This approach is equivalent to, but more efficient and numerically stable than, directly drawing $t \sim \mathcal{N}\left(\Phi m_N, \Phi S_N \Phi^T\right)$.



polynomial basis functions



Radial basis functions

---

e)

1. Polynomial prediction loss: over 7,265.
2. RBF prediction loss: only 27.4.
3. The polynomial posterior's huge variance outside the training data.
4. The large variance is related to the numerical instability of the ML estimator.