ENM 531

Data-driven Modeling and Probabilistic Scientific Computing

Assignment 2

On varying N and M, we observe the following things:

- On reducing the value of N and keeping M constant, we observe the following:
  - For a value of N from 100 – 500, the fits are pretty good for a constant M of 5 and above. If N is reduced below 100, then the data points reduce and the fit becomes worse.
- On changing the value of M and keeping N constant, we see the following
  - The fourier basis is still a good fit when M goes up to 3 but lower than that even fourier basis is not a good fit
  - For M < 5, the monomial and legendre basis are also good fits but when M < 5 upto M = 2, only fourier gives a good fit to the data.

We can thus say that the fourier basis is the best basis to use for this data. This is justified as our data is a combination of exponential and trigonometric function.

HW3M8

February 7, 2019

```
In [47]: import numpy as np
         import matplotlib.pyplot as plt
         from pyDOE import lhs
         from scipy.special import legendre

In [25]: class BayesianLinearRegression:
           """
             Linear regression model: y = (w.T)*phi + epsilon
             w ~ N(0,beta^(-1)I)
             P(y|phi,w) ~ N(y|(w.T)*phi,alpha^(-1)I)
           """
           def __init__(self, phi, y, alpha = 1.0, beta = 1.0):

               self.X = phi
               self.y = y

               self.alpha = alpha
               self.beta = beta

               self.jitter = 1e-8


           def fit_MLE(self):
               phiTphi_inv = np.linalg.inv(np.matmul(self.X.T,self.X) + self.jitter)
               phiTy = np.matmul(self.X.T, self.y)

               w_MLE = np.matmul(phiTphi_inv,phiTy)

               self.w_MLE = w_MLE

               return w_MLE

           def fit_MAP(self):
               Lambda = np.matmul(self.X.T,self.X) + \
                 (self.beta/self.alpha)*np.eye(self.X.shape[1])
               Lambda_inv  = np.linalg.inv(Lambda)
               phiTy = np.matmul(self.X.T,self.y)
               mu = np.matmul(Lambda_inv,phiTy)
```

```
        self.w_MAP = mu
        self.Lambda_inv = Lambda_inv

        return mu, Lambda_inv

    def predictive_distribution(self,x_star):

        mean_star = np.matmul(x_star, self.w_MAP)
        var_star = 1/self.alpha + np.matmul(x_star, \
                          np.matmul(self.Lambda_inv, x_star.T))

        return mean_star, var_star
```

In [26]:
```
class BasisFunctions:

    def __init__(self,x,N,M):

        self.x = x
        self.N = N
        self.M = M

    def identity_basis(self):
        phi = self.x

        return phi

    def monomial_basis(self):

        phi = np.reshape(np.ones(self.N),self.x.shape)

        phi_mon = phi;

        for i in range(1,(self.M)+1):
            phi_mon = np.concatenate((phi_mon,self.x**i),axis = 1)

        return phi_mon

    def fourier_basis(self):
        #phi0 = np.reshape(np.zeros(self.N),self.x.shape)
        phi1 = np.reshape(np.ones(self.N),self.x.shape)

        #phi_fou = np.concatenate((phi0,phi1),axis = 1)
        phi_fou = phi1

        for i in range(1,(self.M)+1):
            psin = np.sin(i*np.pi*self.x)
            pcos = np.cos(i*np.pi*self.x)
```

```
                phi_fou = np.concatenate((phi_fou,psin,pcos),axis = 1)

            return phi_fou

        def legendre_basis(self):

            leg1 = legendre(0)
            phi_leg = leg1(self.x)

            for i in range(1,(self.M)+1):
                leg1 = legendre(i)
                phi_leg = np.concatenate((phi_leg,leg1(self.x)),axis = 1)

            return phi_leg
```

In [27]:
```
# number of data points
N = 500
```

In [28]:
```
# number of features
M = 7
```

In [29]:
```
# initializing gaussian noise
noise_mean = 0
noise_var = 0.5
noise = np.reshape(np.random.normal(noise_mean,noise_var,N),(500,1))
```

In [30]:
```
alpha = 5
beta = 0.1
```

In [31]:
```
# samples of x
x = 2*lhs(1,N)
```

In [32]:
```
# Corresponding y values
y = np.exp(x)*np.sin(2*np.pi*x) + noise
```

In [33]:
```
# Defining all the different basis sets
phi = BasisFunctions(x,N,M)
phi_id = np.reshape(phi.identity_basis(),[500,1])
phi_mon = phi.monomial_basis()
phi_fou = phi.fourier_basis()
phi_leg = phi.legendre_basis()
```

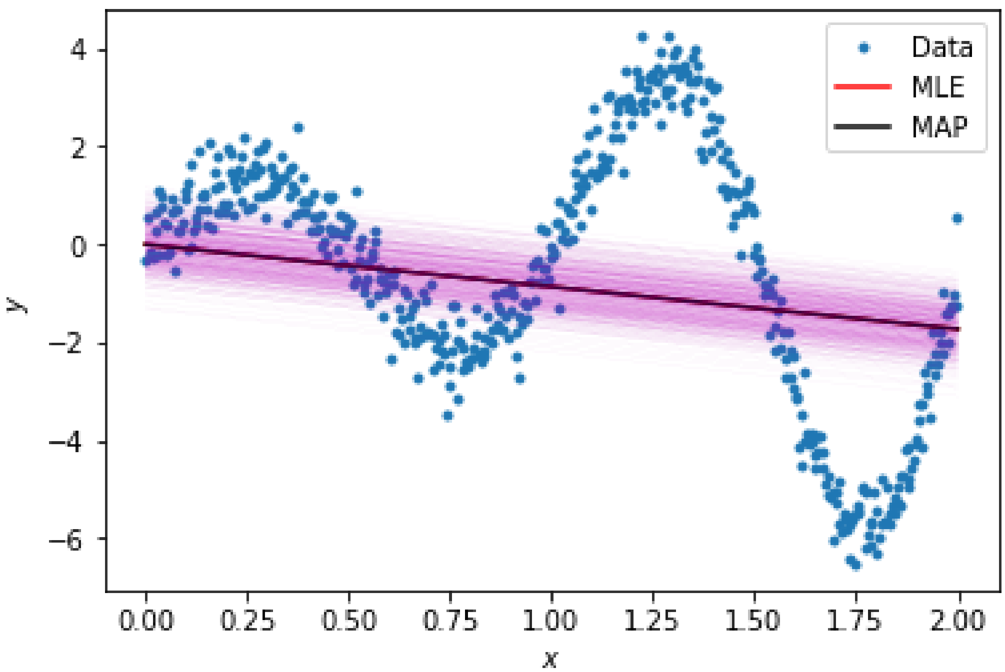
In [34]:
```
# Defining the models
blr_id = BayesianLinearRegression(phi_id,y,alpha,beta)
blr_mon = BayesianLinearRegression(phi_mon, y, alpha, beta)
blr_fou = BayesianLinearRegression(phi_fou,y,alpha,beta)
blr_leg = BayesianLinearRegression(phi_leg,y,alpha,beta)
```

In [35]:
```
# Fit identity MLE and MAP estimates for w
w_MLE_id = blr_id.fit_MLE()
w_MAP_id, Lambda_inv_id = blr_id.fit_MAP()
```

```
In [36]: # Fit monomial MLE and MAP estimates for w
         w_MLE_mon = blr_mon.fit_MLE()
         w_MAP_mon, Lambda_inv_mon = blr_mon.fit_MAP()

In [37]: # Fit fourier MLE and MAP estimates for w
         w_MLE_fou = blr_fou.fit_MLE()
         w_MAP_fou, Lambda_inv_fou = blr_fou.fit_MAP()

In [38]: # Fit legendre MLE and MAP estimates for w
         w_MLE_leg = blr_leg.fit_MLE()
         w_MAP_leg, Lambda_inv_leg = blr_leg.fit_MAP()

In [39]: # generating new samples for prediction
         X_star = np.linspace(0,2,N)[:,None]

In [40]: # Defining basis sets for prediction
         phi_star = BasisFunctions(X_star,N,M)
         phi_star_id = np.reshape(phi_star.identity_basis(),[N,1])
         phi_star_mon = phi_star.monomial_basis()
         phi_star_fou = phi_star.fourier_basis()
         phi_star_leg = phi_star.legendre_basis()

In [41]: # All the predicted values
         y_MLE_id = np.matmul(phi_star_id,w_MLE_id)
         y_MAP_id = np.matmul(phi_star_id,w_MAP_id)

         y_MLE_mon = np.matmul(phi_star_mon,w_MLE_mon)
         y_MAP_mon = np.matmul(phi_star_mon,w_MAP_mon)

         y_MLE_fou = np.matmul(phi_star_fou,w_MLE_fou)
         y_MAP_fou = np.matmul(phi_star_fou,w_MAP_fou)

         y_MLE_leg = np.matmul(phi_star_leg,w_MLE_leg)
         y_MAP_leg = np.matmul(phi_star_leg,w_MAP_leg)

In [42]: # Predictive destribution
         num_samples = 500
         mean_star_id, var_star_id = blr_id.predictive_distribution(phi_star_id)
         samples_id = np.random.multivariate_normal(mean_star_id.flatten(),\
                                                    var_star_id,num_samples)

         mean_star_mon, var_star_mon = blr_mon.predictive_distribution(phi_star_mon)
         samples_mon = np.random.multivariate_normal(mean_star_mon.flatten(),\
                                                     var_star_mon,num_samples)

         mean_star_fou, var_star_fou = blr_fou.predictive_distribution(phi_star_fou)
         samples_fou = np.random.multivariate_normal(mean_star_fou.flatten(),\
                                                     var_star_fou,num_samples)
```

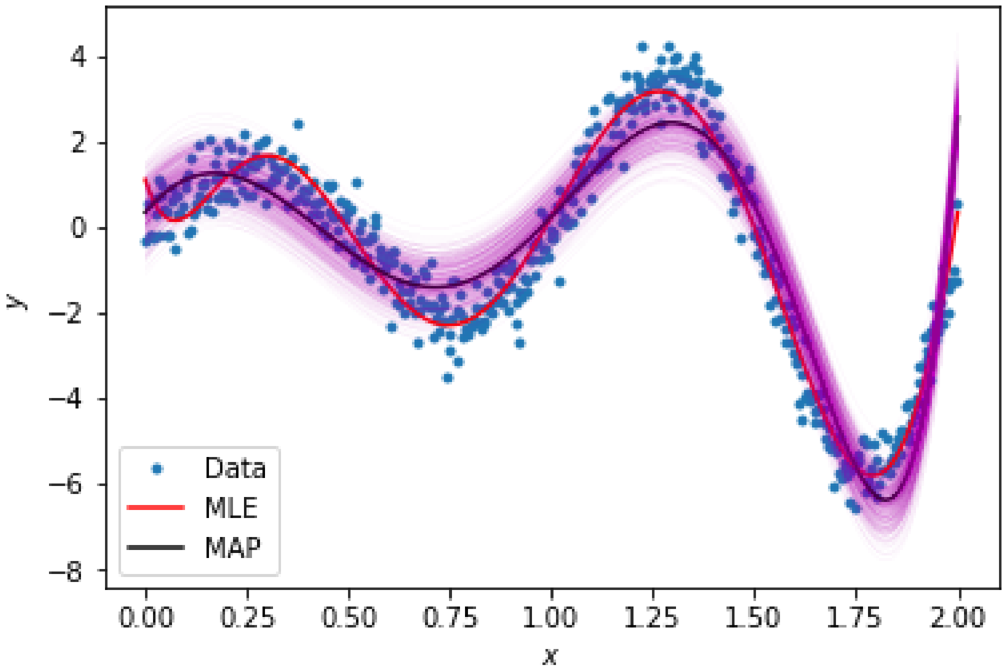
```
mean_star_leg, var_star_leg = blr_leg.predictive_distribution(phi_star_leg)
samples_leg = np.random.multivariate_normal(mean_star_leg.flatten(),\
                                            var_star_leg,num_samples)
```

In [43]:
```
# Plotting the data, fits and predictive distribution for identity basis

f1 = plt.figure()
plt.plot(x,y,'.', label = 'Data')
plt.plot(X_star,y_MLE_id,'r', label = 'MLE')
plt.plot(X_star,y_MAP_id,'k', label = 'MAP')
for i in range(0,num_samples):
    plt.plot(X_star, samples_id[i,:],'m',linewidth = 0.05,alpha = 0.4)
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

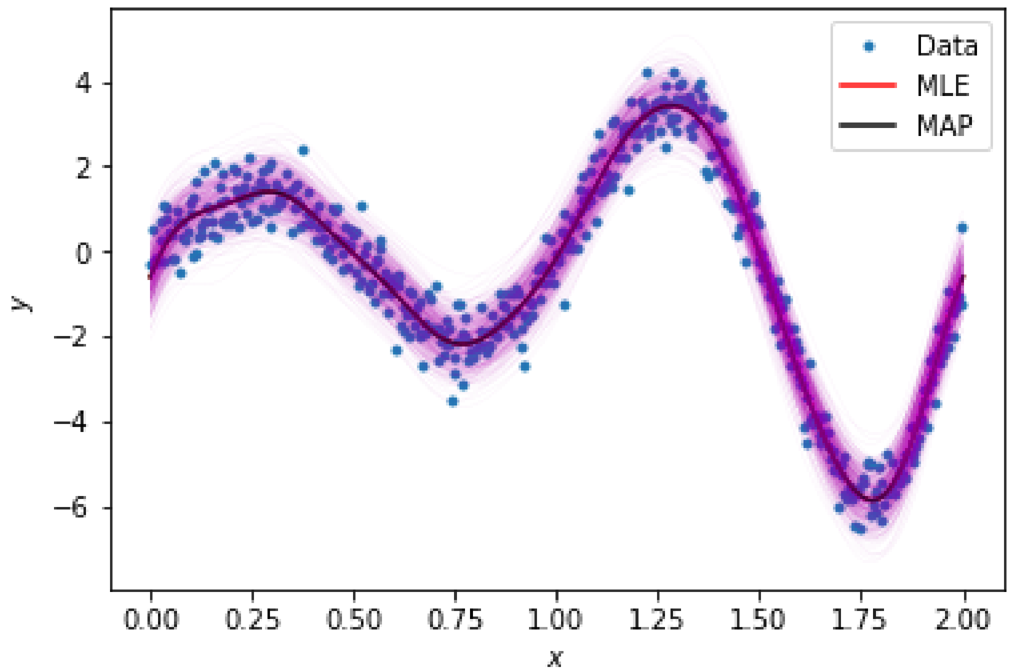


In [44]:
```
# Plotting the data, fits and predictive distribution for monomial basis

f2 = plt.figure()
plt.plot(x,y,'.', label = 'Data')
plt.plot(X_star,y_MLE_mon,'r', label = 'MLE')
plt.plot(X_star,y_MAP_mon,'k', label = 'MAP')
for i in range(0,num_samples):
    plt.plot(X_star, samples_mon[i,:],'m',linewidth = 0.05,alpha = 0.4)
```

```
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```

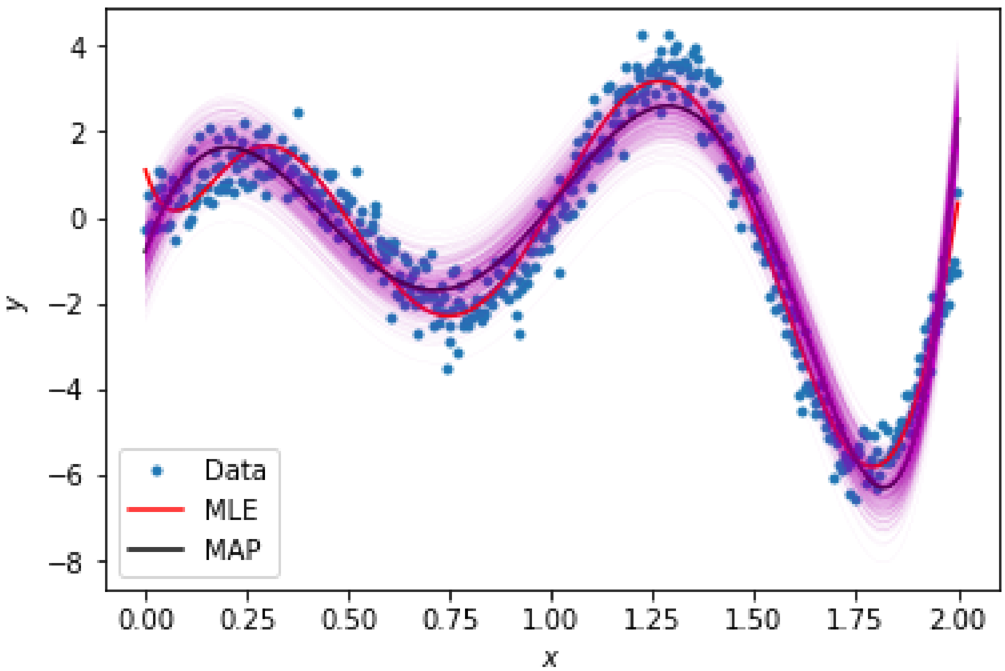



```
In [45]: # Plotting the data, fits and predictive distribution for fourier basis

         f3 = plt.figure()
         plt.plot(x,y,'.', label = 'Data')
         plt.plot(X_star,y_MAP_fou,'r', label = 'MLE')
         plt.plot(X_star,y_MLE_fou,'k', label = 'MAP')
         for i in range(0,num_samples):
             plt.plot(X_star, samples_fou[i,:],'m',linewidth = 0.05,alpha = 0.4)
         plt.legend()
         plt.xlabel('$x$')
         plt.ylabel('$y$')
         plt.show()
```



In [46]: # Plotting the data, fits and predictive distribution for legendre basis

```
f4 = plt.figure()
plt.plot(x,y,'.', label = 'Data')
plt.plot(X_star,y_MLE_leg,'r', label = 'MLE')
plt.plot(X_star,y_MAP_leg,'k', label = 'MAP')
for i in range(0,num_samples):
    plt.plot(X_star,samples_leg[i,:],'m',linewidth = 0.05,alpha = 0.4)
plt.legend()
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```



4
2
0
−2
−4
−6
−8
y
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
x
Data
MLE
MAP

HW3M4

February 7, 2019

```
In [117]: import numpy as np
          import matplotlib.pyplot as plt
          from pyDOE import lhs
          from scipy.special import legendre

In [118]: class BayesianLinearRegression:
            """
              Linear regression model: y = (w.T)*phi + epsilon
              w ~ N(0,beta^(-1)I)
              P(y|phi,w) ~ N(y|(w.T)*phi,alpha^(-1)I)
            """
            def __init__(self, phi, y, alpha = 1.0, beta = 1.0):

                self.X = phi
                self.y = y

                self.alpha = alpha
                self.beta = beta

                self.jitter = 1e-8


            def fit_MLE(self):
                phiTphi_inv = np.linalg.inv(np.matmul(self.X.T,self.X) + self.jitter)
                phiTy = np.matmul(self.X.T, self.y)

                w_MLE = np.matmul(phiTphi_inv,phiTy)

                self.w_MLE = w_MLE

                return w_MLE

            def fit_MAP(self):
                Lambda = np.matmul(self.X.T,self.X) + (self.beta/self.alpha)*\
                                  np.eye(self.X.shape[1])
                Lambda_inv  = np.linalg.inv(Lambda)
                phiTy = np.matmul(self.X.T,self.y)
                mu = np.matmul(Lambda_inv,phiTy)
```

```
            self.w_MAP = mu
            self.Lambda_inv = Lambda_inv

            return mu, Lambda_inv

        def predictive_distribution(self,x_star):

            mean_star = np.matmul(x_star, self.w_MAP)
            var_star = 1/self.alpha + np.matmul(x_star, \
                                  np.matmul(self.Lambda_inv, x_star.T))

            return mean_star, var_star
```

In [119]:
```
class BasisFunctions:

    def __init__(self,x,N,M):

        self.x = x
        self.N = N
        self.M = M

    def identity_basis(self):
        phi = np.ones(self.N)

        return phi

    def monomial_basis(self):

        phi = np.reshape(np.ones(self.N),self.x.shape)

        phi_mon = phi;

        for i in range(1,(self.M)+1):
            phi_mon = np.concatenate((phi_mon,self.x**i),axis = 1)

        return phi_mon

    def fourier_basis(self):
        #phi0 = np.reshape(np.zeros(self.N),self.x.shape)
        phi1 = np.reshape(np.ones(self.N),self.x.shape)

        #phi_fou = np.concatenate((phi0,phi1),axis = 1)
        phi_fou = phi1

        for i in range(1,(self.M)+1):
            psin = np.sin(i*np.pi*self.x)
            pcos = np.cos(i*np.pi*self.x)
```

```
                phi_fou = np.concatenate((phi_fou,psin,pcos),axis = 1)

            return phi_fou

        def legendre_basis(self):

            leg1 = legendre(0)
            phi_leg = leg1(self.x)

            for i in range(1,(self.M)+1):
                leg1 = legendre(i)
                phi_leg = np.concatenate((phi_leg,leg1(self.x)),axis = 1)

            return phi_leg
```

In [120]:
```
# number of data points
N = 500
```

In [121]:
```
# number of features
M = 4
```

In [122]:
```
# initializing gaussian noise
noise_mean = 0
noise_var = 0.5
noise = np.reshape(np.random.normal(noise_mean,noise_var,N),(500,1))
```

In [123]:
```
alpha = 5
beta = 0.1
```

In [124]:
```
# samples of x
x = 2*lhs(1,N)
```

In [125]:
```
# Corresponding y values
y = np.exp(x)*np.sin(2*np.pi*x) + noise
```

In [126]:
```
# Defining all the different basis sets
phi = BasisFunctions(x,N,M)
phi_id = np.reshape(phi.identity_basis(),[500,1])
phi_mon = phi.monomial_basis()
phi_fou = phi.fourier_basis()
phi_leg = phi.legendre_basis()
```

In [127]:
```
# Defining the models
blr_id = BayesianLinearRegression(phi_id,y,alpha,beta)
blr_mon = BayesianLinearRegression(phi_mon, y, alpha, beta)
blr_fou = BayesianLinearRegression(phi_fou,y,alpha,beta)
blr_leg = BayesianLinearRegression(phi_leg,y,alpha,beta)
```

In [128]:
```
# Fit identity MLE and MAP estimates for w
w_MLE_id = blr_id.fit_MLE()
w_MAP_id, Lambda_inv_id = blr_id.fit_MAP()
```

In [129]:
```
# Fit monomial MLE and MAP estimates for w
w_MLE_mon = blr_mon.fit_MLE()
w_MAP_mon, Lambda_inv_mon = blr_mon.fit_MAP()
```

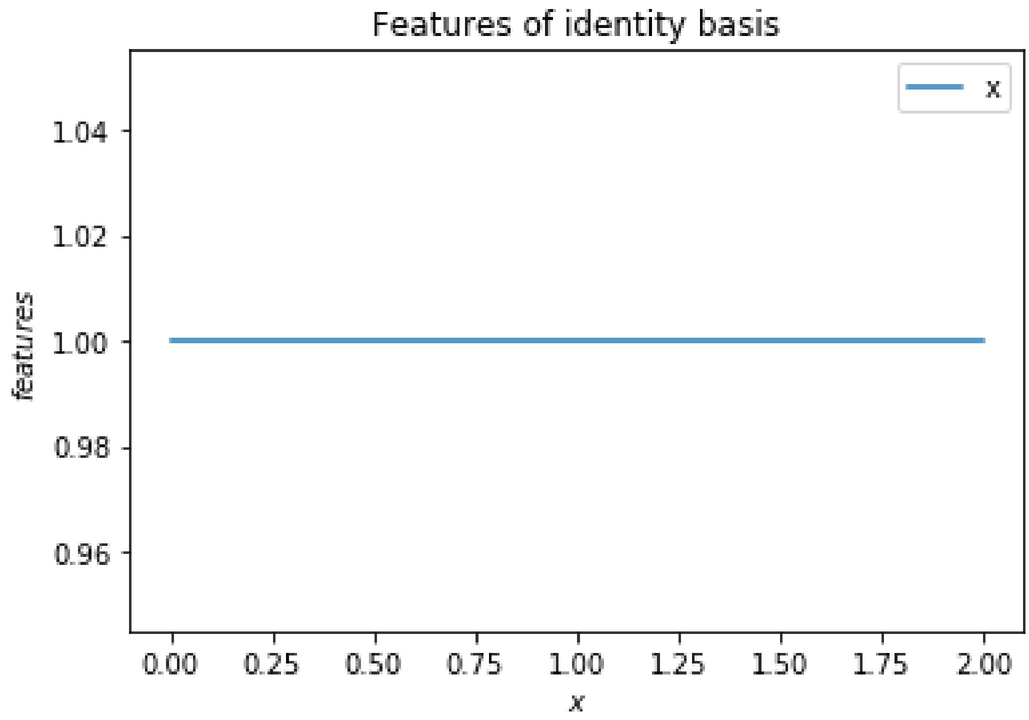
In [130]:
```
# Fit fourier MLE and MAP estimates for w
w_MLE_fou = blr_fou.fit_MLE()
w_MAP_fou, Lambda_inv_fou = blr_fou.fit_MAP()
```

In [131]:
```
# Fit legendre MLE and MAP estimates for w
w_MLE_leg = blr_leg.fit_MLE()
w_MAP_leg, Lambda_inv_leg = blr_leg.fit_MAP()
```

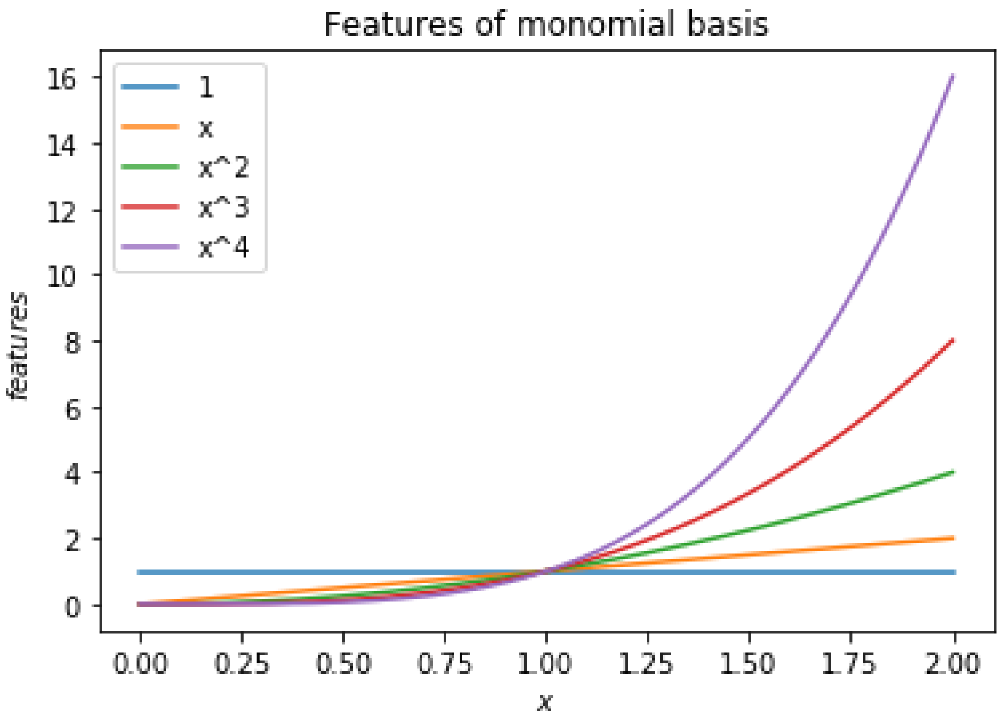
In [132]:
```
# generating new samples for prediction
X_star = np.linspace(0,2,N)[:,None]
```

In [133]:
```
# Defining basis sets for prediction
phi_star = BasisFunctions(X_star,N,M)
phi_star_id = np.reshape(phi_star.identity_basis(),[N,1])
phi_star_mon = phi_star.monomial_basis()
phi_star_fou = phi_star.fourier_basis()
phi_star_leg = phi_star.legendre_basis()
```

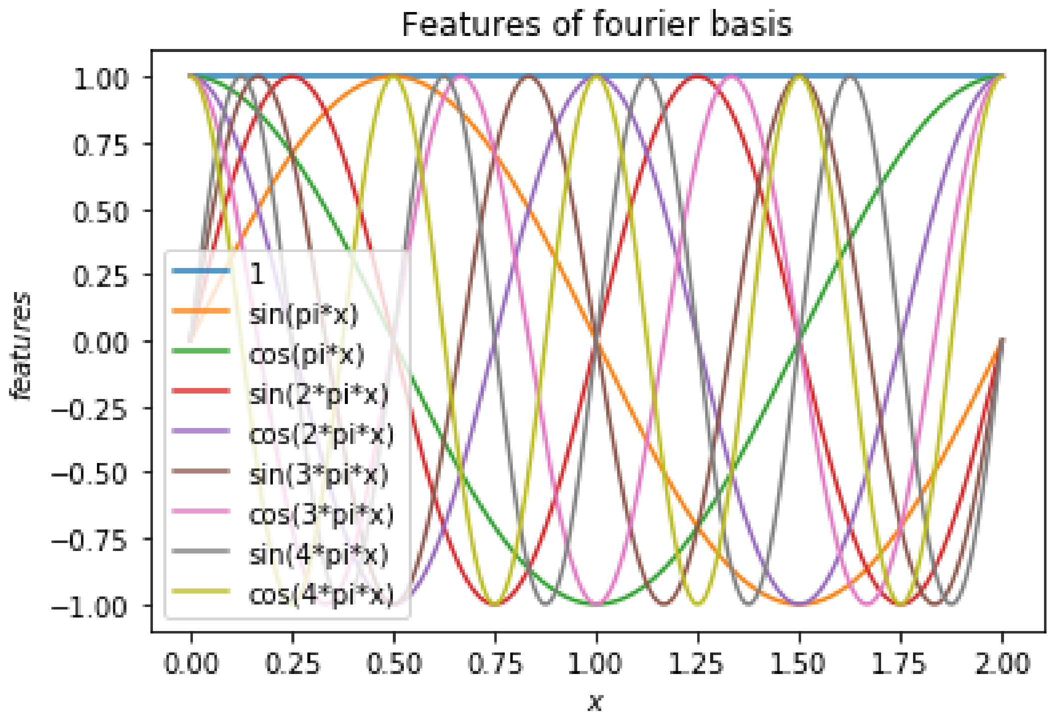
In [134]:
```
# Plotting the features for identity basis

f1 = plt.figure()
plt.plot(X_star,phi_star_id)
plt.legend('x')
plt.title('Features of identity basis')
plt.xlabel('$x$')
plt.ylabel('$features$')
plt.show()
```



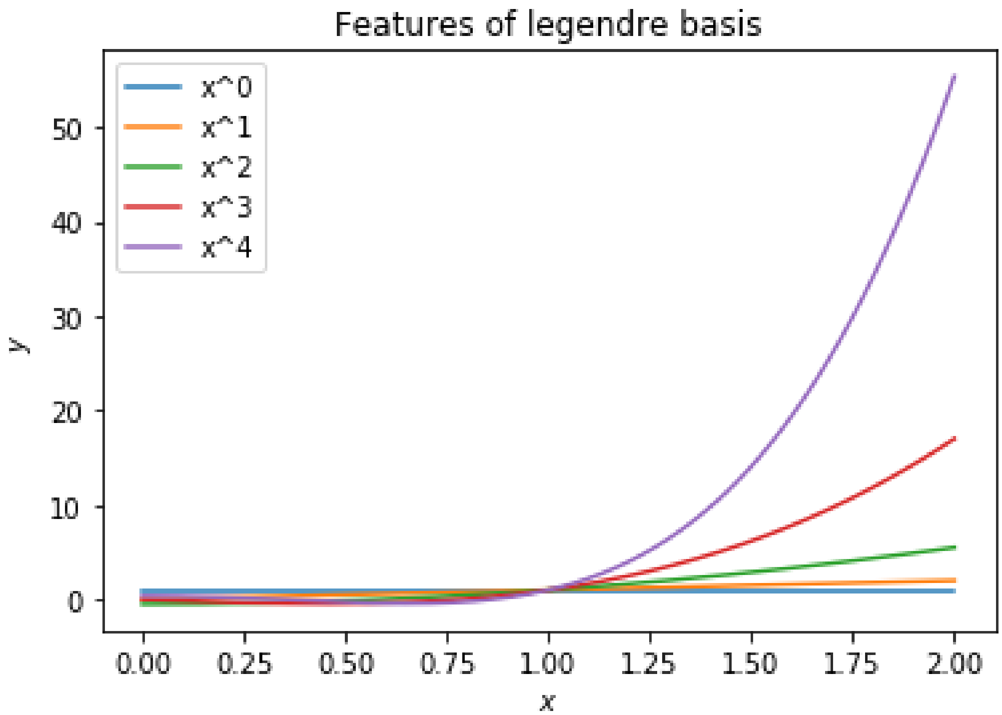
In [135]:
```
# Plotting the features for monomial basis

f2 = plt.figure()
plt.plot(X_star,phi_star_mon)
plt.legend(('1','x','x^2','x^3','x^4'))
plt.title('Features of monomial basis')
plt.xlabel('$x$')
plt.ylabel('$features$')
plt.show()
```



In [136]:
```
# Plotting the data, fits and predictive distribution for fourier basis

f3 = plt.figure()
plt.plot(X_star,phi_star_fou)
plt.legend(('1','sin(pi*x)','cos(pi*x)','sin(2*pi*x)','cos(2*pi*x)','sin(3*pi*x)','co
plt.title('Features of fourier basis')
plt.xlabel('$x$')
plt.ylabel('$features$')
plt.show()
```



In [137]:
```
# Plotting the data, fits and predictive distribution for legendre basis

f4 = plt.figure()
plt.plot(X_star,phi_star_leg)
plt.legend(('x^0','x^1','x^2','x^3','x^4'))
plt.title('Features of legendre basis')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```


Features of legendre basis
x^0
x^1
x^2
x^3
x^4
y
x
0
10
20
30
40
50
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00