0807

August 7, 2019

First import required modules

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import scipy.stats as stats
        from scipy.special import digamma
        from scipy.linalg import sqrtm
```

1 Regression Spline

Assume that the range of x is $[a, b]$. Let the point

$$a < \xi_1 < \cdots < \xi_K < b$$

be a partion of the interval $[a, b]$
$\{\xi_1, \ldots, \xi_K\}$ are called knots.

Then make the function which return the knot points

```
In [236]: def defineKnot(X,K=30):
              upper = max(X)
              lower = min(X)
              out = np.linspace(start=lower,stop=upper,num=K+2)[1:K+1]
              return(out)
```

2 Radial Basis Function

A RBF φ is a real valued function whose value depends only on the distance from origin. A real function $\varphi : [0, \infty) \to \mathbb{R}$ with a metric on space $\| \cdot \| : V \to [0, \infty)$ a function $\varphi_c = \varphi(\|\mathbf{x} - \mathbf{c}\|)$ is said to be a radial kernel centered at c. A radial function and the associated radial kernels are said to be radial basis function

we use radial basis functions defined by

$$\mathbf{b}(u) = \left\{ u, \left| \frac{u - \tau_1}{c} \right|^3, \cdots, \left| \frac{u - \tau_K}{c} \right|^3 \right\}$$

where c is sample standard deviation

Then we can make the function which retrun the basis

```
In [237]: def b(u,tau,sd):
              lst = []
              #lst.append(np.ones(len(u)))
              #lst.append(u)
              for i in tau:
                  lst.append(abs((u-i)/sd)**3)
              out = np.array(lst)
              return(out)
```

Nonparametric linear model can be represented as

$$Y = \mathbf{b}(X)\boldsymbol{\beta} + \varepsilon$$

where $Y \in \mathbb{R}^{n\times 1}$, $X \in \mathbb{R}^{n\times 1}$ and $\varepsilon \sim N(0, \tau^{-1})$

3 Make toy data

Let

$$y = \sum_{l=1}^{4} f_l(X_l) + e$$

$$\begin{aligned} f_1(x) &= 3exp(-30(x-0.3)^2) + exp(-50(x-0.7)^2) \\ f_2(x) &= sin(2\pi x) \\ f_3(x) &= x \\ f_4(x) &= 0 \end{aligned}$$

Plotting true distribution of Y is

```
In [238]: def f(x):
              #out = np.sin(2*np.pi*x)
              out = 3*np.exp(-30*(x-0.2)**2) + np.exp(-50*(x-0.7)**2)
              return(out)
```

```
In [239]: def f1(x):
              #out = np.sin(2*np.pi*x)
              out = 3*np.exp(-30*(x-0.2)**2) + np.exp(-50*(x-0.7)**2)
              return(out)
          def f2(x):
              out = np.sin(2*np.pi*x)
              #out = 3*np.exp(-30*(x-0.2)**2) + np.exp(-50*(x-0.7)**2)
              return(out)
          def f3(x):
              #out = np.sin(2*np.pi*x)
              #out = 3*np.exp(-30*(x-0.2)**2) + np.exp(-50*(x-0.7)**2)
              out = x
              return(out)
          def f4(x):
              #out = np.sin(2*np.pi*x)
```

```
        #out = 3*np.exp(-30*(x-0.2)**2) + np.exp(-50*(x-0.7)**2)
        out = 0*x
        return(out)
```

```
In [240]: grid_x = np.linspace(0,1,1000)
          grid_y = f(grid_x)
```

```
In [316]: #plt.plot(grid_x, grid_y, 'k')
          #plt.ylim(lim)
          #plt.show()
```

make the simulation function which make the obs with error $N(0, 0.5)$

```
In [242]: def mkToy(n=800,tau = 0.5):
              np.random.seed(4428)
              x = np.random.uniform(size = n)
              e = np.random.normal(0,np.sqrt(0.5), size= n)
              y = f(x) + e
              #out = np.column_stack([x,y])
              return(x,y)
```

```
In [243]: def mkToys(n=800,tau = 0.5):
              np.random.seed(4428)
              x1 = np.random.uniform(size = n)
              x2 = np.random.uniform(size = n)
              x3 = np.random.uniform(size = n)
              x4 = np.random.uniform(size = n)
              e = np.random.normal(0,np.sqrt(0.5), size= n)
              y = f1(x1) + f2(x2) + f3(x3) + f4(x4)+ e
              #out = np.column_stack([x,y])
              return(x1,x2,x3,x4,y)
```

Plotting the distribution of simulated data

$$y = f_1(X_1) + f_2(X_2) + f_3(X_3) + f_4(X_4) + \varepsilon$$

where $\varepsilon \sim N(0, 0.5)$

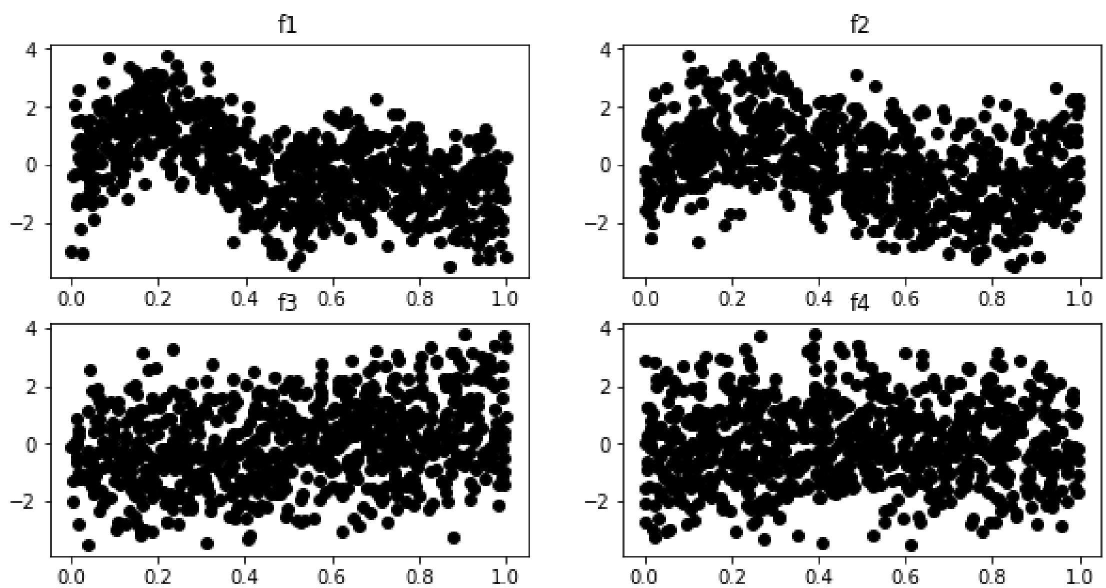
```
In [244]: #x, y = mkToys()
          #y= y-y.mean()
```

$$\tilde{y} = y - \bar{y} = b_1(X_1)\beta_1 + b_2(X_2)\beta_2 + b_3(X_3)\beta_3 + b_4(X_4)\beta_4 + \varepsilon$$

```
In [245]: x1,x2,x3,x4,y = mkToys()
          y= y-y.mean()
```

```
In [246]: plt.figure(figsize=(10,5))
```

```
plt.subplot(221)
plt.title('f1')
plt.plot(x1,y,'ko')

plt.subplot(222)
plt.title('f2')
plt.plot(x2,y,'ko')

plt.subplot(223)
plt.title('f3')
plt.plot(x3,y,'ko')

plt.subplot(224)
plt.title('f4')
plt.plot(x4,y,'ko')

#plt.ylim(lim)
plt.show()
```



Calculate the standard deviation of observed data and define the knot and make design matrix

```
In [247]: sd = np.std(x)
          knot = defineKnot(x)
          d_x = b(x,knot,sd).T
```

```
In [248]: sd1 = np.std(x1)
          knot1 = defineKnot(x1)
          d_x1 = b(x1,knot1,sd1).T
```

```
sd2 = np.std(x2)
knot2 = defineKnot(x2)
d_x2 = b(x2,knot2,sd2).T

sd3 = np.std(x3)
knot3 = defineKnot(x3)
d_x3 = b(x3,knot3,sd3).T

sd4 = np.std(x4)
knot4 = defineKnot(x4)
d_x4 = b(x4,knot4,sd4).T
```

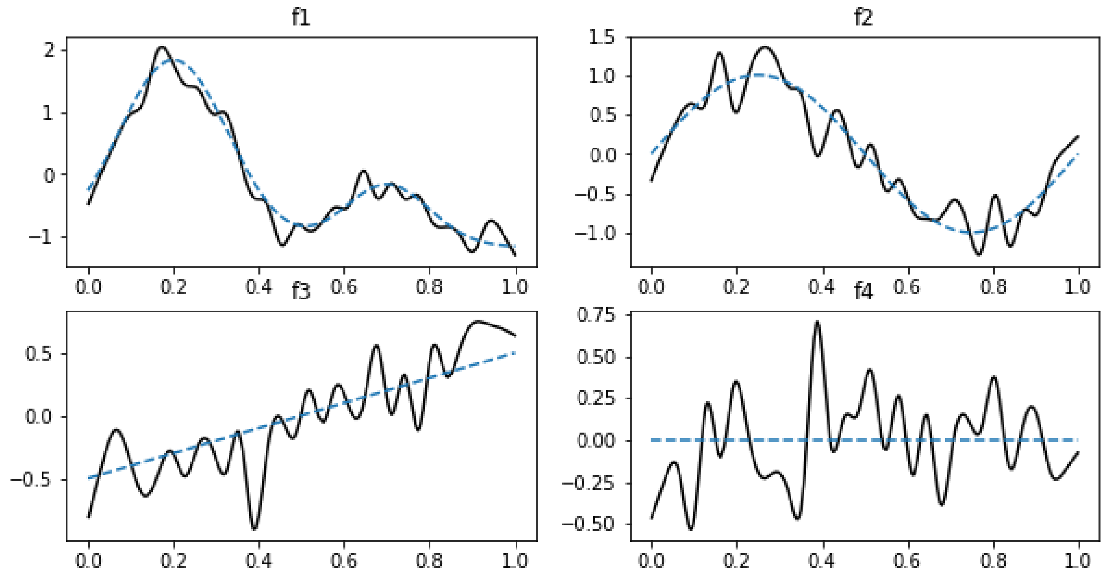
4 LSE method

plotting the fitted value

```
In [284]: def lplot(x,d_x):
              fitted  = d_x.dot(np.linalg.inv(d_x.T.dot(d_x))).dot(d_x.T).dot(y)
              plot_m = np.array(sorted(np.array([x,fitted]).T,key=lambda x: x[0]))
              plt.plot(plot_m[:,0],plot_m[:,1],'k',grid_x, grid_y-grid_y.mean(), '--')

          def lplotb(x,d_x,m):
              fitted  = d_x.dot(m)
              plot_m = np.array(sorted(np.array([x,fitted]).T,key=lambda x: x[0]))
              plt.plot(plot_m[:,0],plot_m[:,1],'k',grid_x, grid_y-grid_y.mean(), '--')
```

```
In [250]: plt.figure(figsize=(10,5))
          grid_x = np.linspace(0,1,1000)

          grid_y = f1(grid_x)
          plt.subplot(221)
          plt.title('f1')
          lplot(x1,d_x1)

          grid_y = f2(grid_x)
          plt.subplot(222)
          plt.title('f2')
          lplot(x2,d_x2)

          grid_y = f3(grid_x)
          plt.subplot(223)
          plt.title('f3')
          lplot(x3,d_x3)

          grid_y = f4(grid_x)
          plt.subplot(224)
          plt.title('f4')
          lplot(x4,d_x4)
```

```
plt.show()
```



Blue dashed line is true function and solid line is LSE estimated function

5 MFVB method

setting prior as

$$
\begin{aligned}
p(Y|\tau,\beta) &\sim N(X\beta, \tau^{-1} \cdot I_N) \\
p(\beta_i|\gamma_i) &\sim^{ind} N(0, \gamma^{-1}) \text{ for } i = 1, \dots p \\
p(\gamma) &\sim Gamma(a, b) \\
p(\tau) &\sim Gamma(c, d)
\end{aligned}
$$

By Baye's rule

$$p(\tau, \gamma, \beta|Y) \propto p(Y|\tau, \beta)p(\beta|\gamma)p(\tau)p(\gamma)$$

Then variational distribution is

$$p(\tau, \gamma, \mu|Y) \approx q(\tau, \gamma, \mu) = q_1(\tau)q_2(\gamma)q_3(\mu)$$

we can maximize ELBO by coordinate descent algorithm

$$
\begin{aligned}
q_1^*(\tau) &= E_{q_2,q_3}[p(\tau, \gamma, \beta|Y)] \propto E_{q_2,q_3}[p(Y|\tau, \beta)p(\tau)] \\
q_2^*(\gamma) &= E_{q_1,q_3}[p(\tau, \gamma, \beta|Y)] \propto E_{q_1,q_3}[p(\beta|\gamma)p(\gamma)] \\
q_3^*(\beta) &= E_{q_1,q_2}[p(\tau, \gamma, \beta|Y)] \propto E_{q_1,q_2}[p(Y|\tau, \beta)p(\beta|\gamma)]
\end{aligned}
$$

Then

$$q_1^* \sim Gamma\left(c + \frac{N+1}{2}, d + \frac{1}{2}\left\{Y'Y - E_{q3}[\beta'](X'Y)\right\} + tr\left[X(var_{q3}[\beta] + E_{q3}[\beta]E_{q3}[\beta'])X'\right]\right)$$

$$q_2^* \sim \prod_{i=1}^{p} Gamma(a + \frac{1}{2}, b + \frac{1}{2}\{var_{q3}[\beta]_{i,i} + E_{q3}[\beta_i]^2\})$$

$$q_3^* \sim N\left(E_{q1}[\tau]\Sigma X'Y, \left(diag(E_{q2}[\gamma]) + E_{q1}[\tau]X'X\right)^{-1} = \Sigma\right)$$

In [251]:
```
def product(a):
    n = len(a)
    out = np.zeros([n,n])
    for i in range(n):
        for j in range(n):
            out[i,j] = a[i]*a[j]
    return(out)
```

In [252]:
```
def mfvb(X,y,max_iter=100):

    N,p = X.shape
    a ,b, c, d = [10**(-7)]*4
    a_tilde = np.repeat(a + 0.5, p)
    b_tilde = np.repeat(b,p)
    c_tilde = c + (N+1)/2
    d_tilde = d

    mu_coeffs = np.repeat(0,p)
    sigma_coeffs = np.diag(np.repeat(1,p))

    for i in range(max_iter):
        expected_coeffs = mu_coeffs
        double_expected_coeffs = sigma_coeffs + product(mu_coeffs)
        diagonal_sigma = np.diag(sigma_coeffs)
        expected_alpha = np.array(list(map(lambda x : a_tilde[x]/b_tilde[x] , np.arang
        log_expected_alpha = np.array(list(map(lambda x : digamma(a_tilde[x])-np.log(b
        expected_tau = c_tilde / d_tilde
        log_expected_tau = digamma(c_tilde)-np.log(d_tilde)
        sigma_coeffs = np.linalg.inv(np.diag(expected_alpha)+expected_tau*(X.T.dot(X))
        mu_coeffs =  expected_tau*sigma_coeffs.dot(X.T.dot(y))
        b_tilde = np.array(list(map(lambda x : (diagonal_sigma[x]+mu_coeffs[x]**2)/2 +
        d_tilde = d+0.5*(y.T.dot(y)) - expected_coeffs.T.dot((X.T.dot(y)))+ 0.5*sum(np
    return mu_coeffs,sigma_coeffs
```

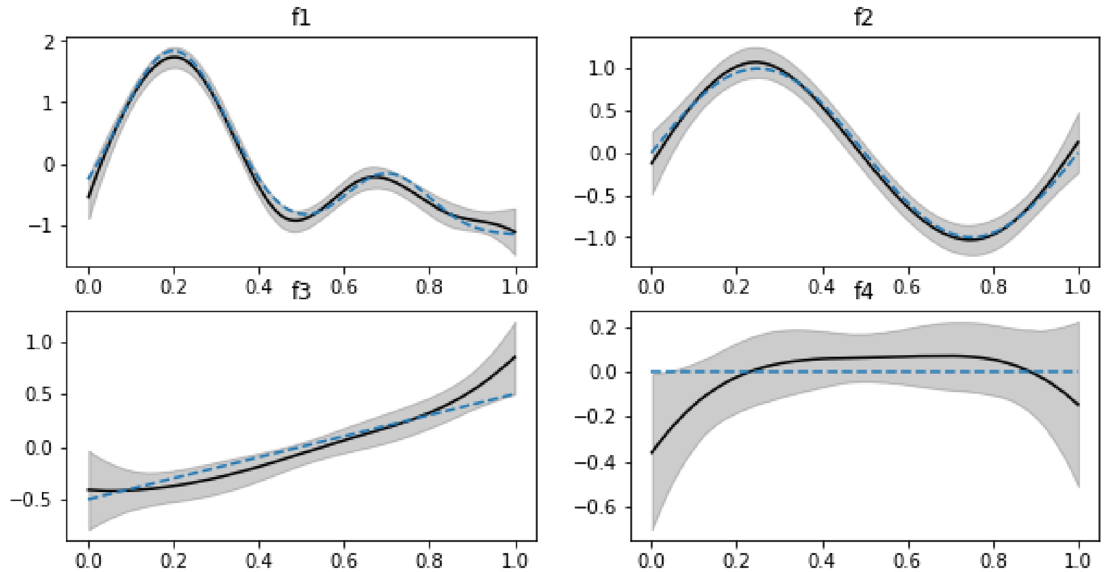
In [253]:
```
m1,c1 = mfvb(d_x1,y)
m2,c2 = mfvb(d_x2,y)
m3,c3 = mfvb(d_x3,y)
m4,c4 = mfvb(d_x4,y)
```

In [254]:
```
def ci95(m,c,n=1000):
    np.random.seed(4428)
```

```
    sampled_coef = np.random.multivariate_normal(m,c,size=n)
    y_grid = np.array([d_x.dot(b) for b in sampled_coef])
    quantile = np.array([np.sort(x)[[int(n*0.025),int(n*0.5),int(n*0.975)]] for x in y
    xq = np.array(sorted(np.array([x,quantile[:,0],quantile[:,1],quantile[:,2]]).T,key
    plt.fill_between(xq[:,0], xq[:,1],xq[:,3], color =(0,0,0,0.2))
    y = f(grid_x)- f(grid_x).mean()
    plt.plot(xq[:,0],xq[:,2],'k',grid_x, y, '--')
    #plt.plot(x_grid,y_grid[10],'k',x_grid, f(x_grid), '--')
    #plt.ylim(-1.5,3.5)
    plt.show()


def ci95s(m,c,y,n=10000):
    np.random.seed(4428)
    sampled_coef = np.random.multivariate_normal(m,c,size=n)
    y_grid = np.array([d_x.dot(b) for b in sampled_coef])
    quantile = np.array([np.sort(x)[[int(n*0.025),int(n*0.5),int(n*0.975)]] for x in y
    xq = np.array(sorted(np.array([x,quantile[:,0],quantile[:,1],quantile[:,2]]).T,key
    plt.fill_between(xq[:,0], xq[:,1],xq[:,3], color =(0,0,0,0.2))
    plt.plot(xq[:,0],xq[:,2],'k',grid_x, y, '--')
    #plt.plot(x_grid,y_grid[10],'k',x_grid, f(x_grid), '--')
    #plt.ylim(-1.5,3.5)
    #plt.show()
```

In [255]:
```
grid_x = np.linspace(0,1,1000)
y1 = f1(grid_x)- f1(grid_x).mean()
y2 = f2(grid_x)- f2(grid_x).mean()
y3 = f3(grid_x)- f3(grid_x).mean()
y4 = f4(grid_x)- f4(grid_x).mean()

plt.figure(figsize=(10,5))


plt.subplot(221)
plt.title('f1')
ci95s(m1,c1,y1)

plt.subplot(222)
plt.title('f2')
ci95s(m2,c2,y2)

plt.subplot(223)
plt.title('f3')
ci95s(m3,c3,y3)

plt.subplot(224)
plt.title('f4')
ci95s(m4,c4,y4)
```

```
plt.show()
```




```
In [256]: #ci95(m,c,n=1000)
```

Dashed line is True function, solid line is median estimator and gray filled area is 95% confidence interval

6 MFVB method with variable selection

Variable selection model is

$$Y = X\Gamma\beta + \epsilon, \quad \epsilon \sim N(0, \sigma^2 I)$$

where

$$\begin{aligned} Y|\beta, \sigma^2, \gamma &\sim N(X\Gamma\beta, \sigma^2 I) \\ \sigma^2 &\sim Inverse - Gamma(A, B) \\ \beta_j &\sim N(0, \sigma^2_\beta) \\ \gamma_j &\sim Bernoulli(\rho) \end{aligned}$$

```
In [257]: '''
          n = 100
          x1 = np.random.uniform(size = n)
          x2 = np.random.uniform(size = n)
          x3 = np.random.uniform(size = n)
          x4 = np.random.uniform(size = n)
          x5 = np.random.uniform(size = n)
```

```
x6 = np.random.uniform(size = n)

X = np.array([x1,x2,x3,x4,x5,x6]).T
Beta_true = np.array([0.02,0.03,0.4,1,0,0])
y2 = X.dot(Beta_true) + np.random.normal(size=n)
y2 = y2- y2.mean()
N,p = X.shape
'''
```

Out[257]: '\nn = 100\nx1 = np.random.uniform(size = n)\nx2 = np.random.uniform(size = n)\nx3 = n

In [258]:
```
def expit(x):
    if x < 100:
        return(np.exp(x)/(1+np.exp(x)))
    else:
        return(1)
```

In [309]:
```
def vselect(X,y,maxiter=100,rho = 0.45):
    N,p = X.shape
    sigmab = 1
    A = 10**(-7)
    B = 10**(-7)
    tau = 1
    w = np.repeat(0.5,p)
    lamb= np.log(rho/(1-rho))
    t = 0
    for iteration in range(maxiter):
        test= False
        W = np.diag(w)
        omega = product(w) + W.dot(np.eye(p)-W)
        sigma = np.linalg.inv(tau*np.multiply(X.T.dot(X),omega)+ (1/sigmab) * np.eye(p
        mu = tau*sigma.dot(W.dot(X.T.dot(y)))

        s = B + 0.5*(np.linalg.norm(y)**2 -2*y.T.dot(X).dot(W).dot(mu) + np.trace(np.m
        tau = (A+N/2)/s

        wstar = w.copy()
        eta = np.zeros(p)
        for j in range(p):
            eta[j] = lamb - 0.5*tau *(mu[j]**2 + sigma[j,j])*np.linalg.norm(X[:,j])**2
            wstar[j] = expit(eta[j])
        w = wstar
        #print(np.array(eta).round(2))
        #print(np.array(wstar).round(2))
    return(mu,sigma,w)
```

In [310]:
```
ms1,cs1,ws1 = vselect(d_x1,y)
ms2,cs2,ws2 = vselect(d_x2,y)
```

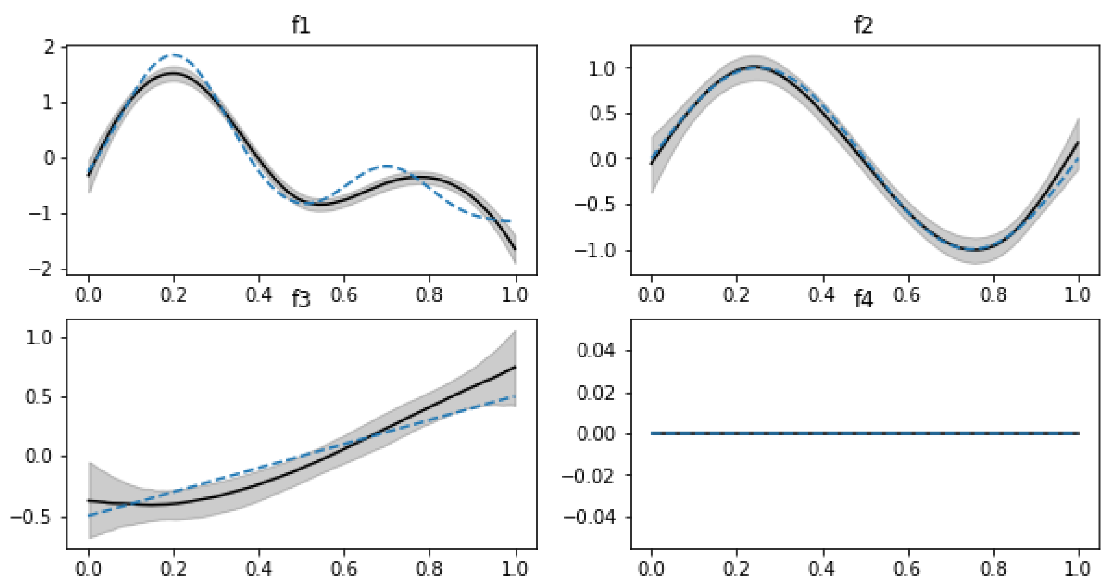
```
ms3,cs3,ws3 = vselect(d_x3,y)
ms4,cs4,ws4 = vselect(d_x4,y)
```

In [311]:
```
def cil95(m,c,w,n=100):
    np.random.seed(4428)
    #smaple_gam = np.random.r
    sampled_coef = np.random.multivariate_normal(m,c,size=n)
    lst = []
    for j in w2:
        lst.append(np.random.binomial(1,j))
    w2ar = np.array(lst)

    y_grid = np.array([d_x.dot(np.diag(w2ar)).dot(b) for b in sampled_coef])
    quantile = np.array([np.sort(x)[[int(n*0.05),int(n*0.5),int(n*0.95)]] for x in y_g
    xq = np.array(sorted(np.array([x,quantile[:,0],quantile[:,1],quantile[:,2]]).T,key
    plt.fill_between(xq[:,0], xq[:,1],xq[:,3], color =(0,0,0,0.2))
    y = f(grid_x)- f(grid_x).mean()
    plt.plot(xq[:,0],xq[:,2],'k',grid_x, y, '--')
    #plt.plot(x_grid,y_grid[10],'k',x_grid, f(x_grid), '--')
    #plt.ylim(-1.5,3.5)
    plt.show()

def cil95s(m,c,w,x,d_x,y,n=1000):
    np.random.seed(4428)
    #smaple_gam = np.random.r
    sampled_coef = np.random.multivariate_normal(m,c,size=n)
    lst = []
    for j in w:
        lst.append(np.random.binomial(1,j))
    w2ar = np.array(lst)

    y_grid = np.array([d_x.dot(np.diag(w2ar)).dot(b) for b in sampled_coef])
    quantile = np.array([np.sort(x)[[int(n*0.05),int(n*0.5),int(n*0.95)]] for x in y_g
    xq = np.array(sorted(np.array([x,quantile[:,0],quantile[:,1],quantile[:,2]]).T,key
    plt.fill_between(xq[:,0], xq[:,1],xq[:,3], color =(0,0,0,0.2))
    #y = f(grid_x)- f(grid_x).mean()
    plt.plot(xq[:,0],xq[:,2],'k',grid_x, y, '--')
    #plt.plot(x_grid,y_grid[10],'k',x_grid, f(x_grid), '--')
    #plt.ylim(-1.5,3.5)
    #plt.show()
```

Spike and slab variable selection plot

In [312]:
```
grid_x = np.linspace(0,1,1000)
y1 = f1(grid_x)- f1(grid_x).mean()
y2 = f2(grid_x)- f2(grid_x).mean()
y3 = f3(grid_x)- f3(grid_x).mean()
y4 = f4(grid_x)- f4(grid_x).mean()
```

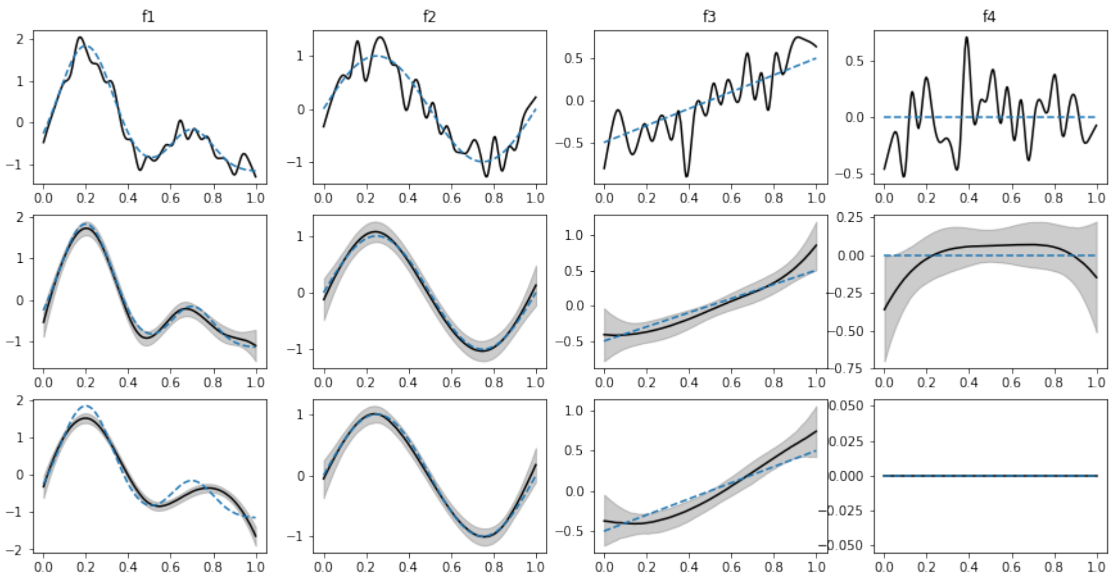
```
plt.figure(figsize=(10,5))


plt.subplot(221)
plt.title('f1')
cil95s(ms1,cs1,ws1,x1,d_x1,y1)

plt.subplot(222)
plt.title('f2')
cil95s(ms2,cs2,ws2,x2,d_x2,y2)

plt.subplot(223)
plt.title('f3')
cil95s(ms3,cs3,ws3,x3,d_x3,y3)

plt.subplot(224)
plt.title('f4')
cil95s(ms4,cs4,ws4,x4,d_x4,y4)


plt.show()
```




```
In [318]: y1 = f1(grid_x)- f1(grid_x).mean()
          y2 = f2(grid_x)- f2(grid_x).mean()
          y3 = f3(grid_x)- f3(grid_x).mean()
          y4 = f4(grid_x)- f4(grid_x).mean()
```

```
plt.figure(figsize=(15,7.5))
grid_x = np.linspace(0,1,1000)


grid_y = f1(grid_x)
plt.subplot(341)
plt.title('f1')
lplot(x1,d_x1)

grid_y = f2(grid_x)
plt.subplot(342)
plt.title('f2')
lplot(x2,d_x2)

grid_y = f3(grid_x)
plt.subplot(343)
plt.title('f3')
lplot(x3,d_x3)

grid_y = f4(grid_x)
plt.subplot(344)
plt.title('f4')
lplot(x4,d_x4)

plt.subplot(345)
#plt.title('f1')
ci95s(m1,c1,y1)

plt.subplot(346)
#plt.title('f2')
ci95s(m2,c2,y2)

plt.subplot(347)
#plt.title('f3')
ci95s(m3,c3,y3)

plt.subplot(348)
#plt.title('f4')
ci95s(m4,c4,y4)

plt.subplot(349)
#plt.title('f1')
cil95s(ms1,cs1,ws1,x1,d_x1,y1)

plt.subplot(3,4,10)
#plt.title('f2')
cil95s(ms2,cs2,ws2,x2,d_x2,y2)
```

```
plt.subplot(3,4,11)
#plt.title('f3')
cil95s(ms3,cs3,ws3,x3,d_x3,y3)

plt.subplot(3,4,12)
#plt.title('f4')
cil95s(ms4,cs4,ws4,x4,d_x4,y4)



plt.show()
```



$$y = \sum_{l=1}^{4} f_l(X_l) + e$$

$$f_1(x) = 3exp(-30(x-0.3)^2) + exp(-50(x-0.7)^2)$$
$$f_2(x) = sin(2\pi x)$$
$$f_3(x) = x$$
$$f_4(x) = 0$$

파란색 점선은 각 함수의 참값을 나타내고 검은색 실선은 median estimator 회색 음영부분은 95% 신뢰구간을 표현합니다
위에서 부터 각각 LSE, Variational inference, Variational Inference with spike and slab prior 입니다
변수 선택을 한 마지막 모형에서 함수가 0인 부분 을 잘 표현하는 것을 확인 할 수 있었습니다