CSC4020 Homework 1 Programming

Interpolation of Noisy Data

@Jingxin Jiang 117020119

In [1]:
```
import numpy as np
from scipy import sparse
import matplotlib.pyplot as plt
np.random.seed(1)
```

---

1. Gaussian Linear System

1.1. Function

$f_1(t) = \sin(t),\ f_2(t) = \sin(10t),\ t \in [0, 2\pi]$

In [2]:
```
# Step 1

f1 = lambda x: np.sin(x)    # function handle f1 = sin(t)
f2 = lambda x: np.sin(10 * x) # function handle f2 = sin(10t)
l = 0                       # left endpoint
r = 2 * np.pi                 # right endpoint
```

1.2. Sub-Intervals

$\mathbf{t} = [t_1, \ldots, t_D]$ where $t_j = jh,\ h = \frac{2\pi}{D},\ j = 1, \ldots, D$

In [3]:
```
# Step 2

D = 1000
t = np.linspace(l, r, D+1)[1:]
```

1.3. Noisy Data Points

$\varepsilon = [\varepsilon_1, \ldots, \varepsilon_N]$ where $\varepsilon_j \sim \mathcal{N}\left(0, \sigma^2\right),\ j = 1, \ldots, N$

In [4]:
```
# Step 3

N = 30
σ2 = 0.1
ε = np.random.normal(loc = 0, scale = np.sqrt(σ2), size = N)
```

1.4. Zero-One Selection Matrix

$$D\left\{\begin{bmatrix}\mathbf{I}_N \\ \mathbf{O}\end{bmatrix} \xrightarrow[\text{shuffle}]{\text{row}} \begin{bmatrix}\vdots \\ \vdots\end{bmatrix} \xrightarrow{\text{transpose}} [\cdots \cdots] = \mathbf{A}_{N\times D}\right.$$

In [5]:
```
# Step 4.1

I = np.eye(N)
O = np.zeros((D-N,N))
A = np.concatenate((I,O))
np.random.shuffle(A)
A = A.T
```

$\mathbf{x} = f(\mathbf{t}),\ \mathbf{y} = \mathbf{A}\mathbf{x} + \varepsilon$

In [6]:
```
# Step 4.2

x_f1 = f1(t)
y_f1 = A @ x_f1 + ε

x_f2 = f2(t)
y_f2 = A @ x_f2 + ε
```

1.5. Prior Precision λ & Tridiagonal Matrix $\mathbf{L}$

$$\lambda_1 = 30,\ \lambda_2 = 5,\ L = \begin{bmatrix} 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{(D-2)\times D}$$

In [7]:
```
# Step 5

λ1 = 30
λ2 = 5
L = sparse.diags([1,-2,1],[0,1,2],(D-2,D)).toarray()
```

1.6. Posterior Mean, Posterior Variance and 95% Credibility Interval

1.6.1. Prior Dist.

$p(\mathbf{x}) = \mathcal{N}\left(\mathbf{x}|\mathbf{0}, \left(\lambda^2\mathbf{L}^T\mathbf{L}\right)^{-1}\right)$

In [8]:
```
# Step 6.1

priorMu_λ1 = np.zeros(D)
priorPrec_λ1 = λ1**2 * L.T@L
priorSigma_λ1 = np.linalg.pinv(priorPrec_λ1)

priorMu_λ2 = np.zeros(D)
priorPrec_λ2 = λ2**2 * L.T@L
priorSigma_λ2 = np.linalg.pinv(priorPrec_λ2)
```

1.6.2. Observed Dist.

$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}\left(\mathbf{y}|\mathbf{A}\mathbf{x}, \sigma^2\mathbf{I}\right)$

In [9]:
```
# Step 6.2

obsMu_f1 = A @ x_f1
obsMu_f2 = A @ x_f2
obsSigma = σ2 * np.eye(N)
obsPrec = np.linalg.pinv(obsSigma)
```

1.6.3. Posterior Dist.

$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}\left(\mathbf{x}|\mu_{x|y}, \mathbf{\Sigma}_{x|y}\right)$ where:

$$\mathbf{\Sigma}_{x|y}^{-1} = \mathbf{\Sigma}_x^{-1} + \mathbf{A}^T\mathbf{\Sigma}_y^{-1}\mathbf{A} = \lambda^2\mathbf{L}^T\mathbf{L} + \frac{1}{\sigma^2}\mathbf{A}^T\mathbf{A} = \lambda^2\mathbf{L}^T\mathbf{L} + \frac{1}{\sigma^2}diag(\mathbf{I}_N, \mathbf{0})$$

$$\mu_{x|y} = \mathbf{\Sigma}_{x|y}\left[\mathbf{A}^T\mathbf{\Sigma}_y^{-1}\mathbf{y} + \mathbf{\Sigma}_x^{-1}\mu_x\right] = \frac{1}{\sigma^2}\mathbf{\Sigma}_{x|y}\mathbf{A}^T\mathbf{y}$$

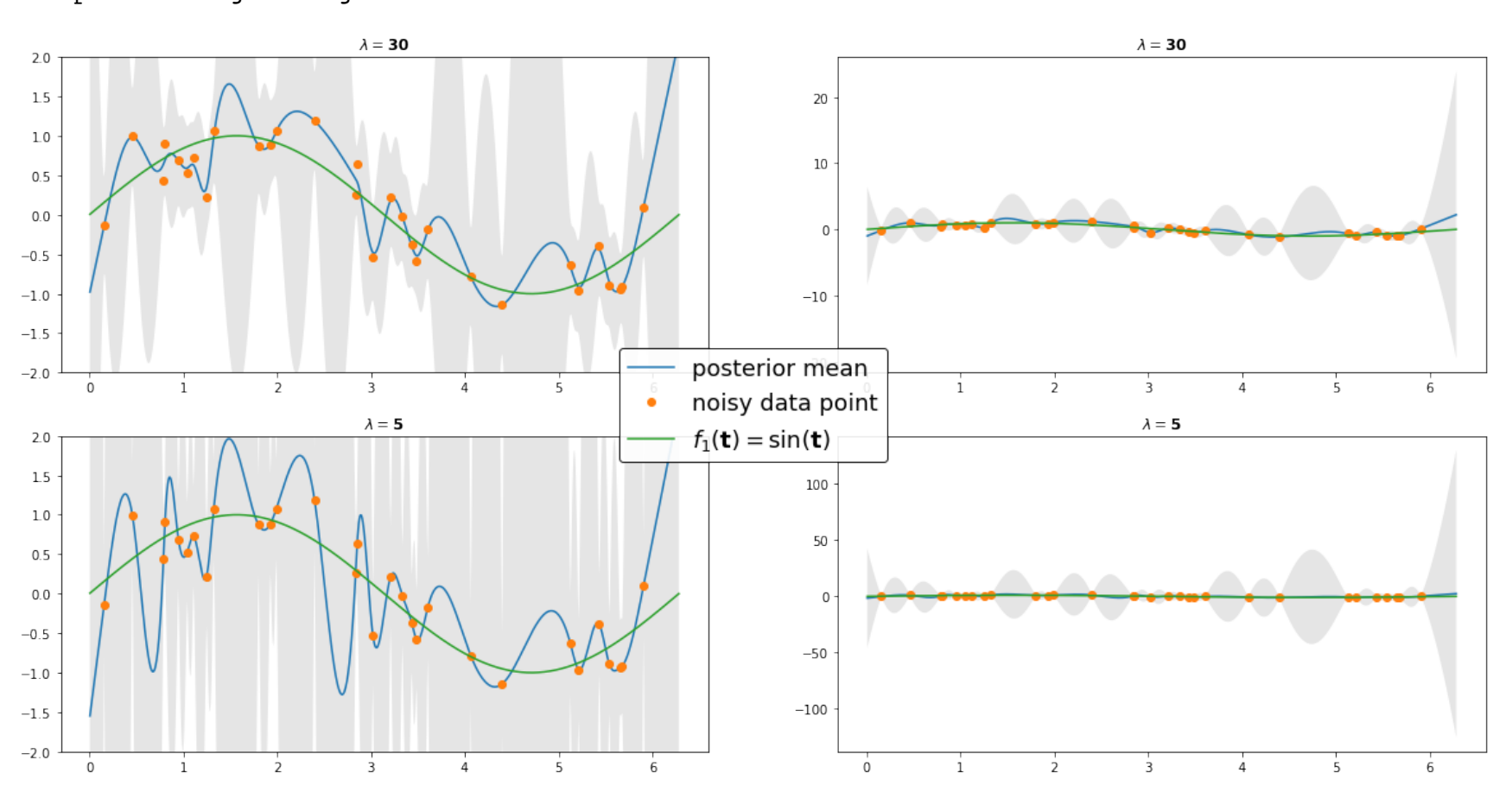
In [10]:
```
# Step 6.3

# λ1 = 30
postPrec_λ1 = priorPrec_λ1 + A.T@obsPrec@A
postSigma_λ1 = np.linalg.pinv(postPrec_λ1)
postMu_λ1_f1 = postSigma_λ1 @ (A.T@obsPrec@y_f1 + priorPrec_λ1@priorMu_λ1) # f1(x) = sin(x)
postMu_λ1_f2 = postSigma_λ1 @ (A.T@obsPrec@y_f2 + priorPrec_λ1@priorMu_λ1) # f2(x) = sin(10x)

# λ2 = 5
postPrec_λ2 = priorPrec_λ2 + A.T@obsPrec@A
postSigma_λ2 = np.linalg.pinv(postPrec_λ2)
postMu_λ2_f1 = postSigma_λ2 @ (A.T@obsPrec@y_f1 + priorPrec_λ2@priorMu_λ2) # f1(x) = sin(x)
postMu_λ2_f2 = postSigma_λ2 @ (A.T@obsPrec@y_f2 + priorPrec_λ2@priorMu_λ2) # f2(x) = sin(10x)
```

1.7. Plot

1.7.1 $f_1(t) = \sin(t)$

In [11]:
```
#f1(x) = sin(x)
fig1 = plt.figure(figsize = (20,10))

# λ1 = 30
ax11 = plt.subplot(221)
ax11.set_ylim([-2,2])
ax11.plot(t, postMu_λ1_f1)
ax11.plot(A@t, y_f1, 'o')
ax11.plot(t, f1(t))
ax11.fill_between(t,
                  postMu_λ1_f1 - 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  postMu_λ1_f1 + 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax11.set_title(r'$\lambda={\bf 30}$')

ax12 = plt.subplot(222)
ax12.plot(t, postMu_λ1_f1)
ax12.plot(A@t, y_f1, 'o')
ax12.plot(t, f1(t))
ax12.fill_between(t,
                  postMu_λ1_f1 - 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  postMu_λ1_f1 + 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax12.set_title(r'$\lambda={\bf 30}$')

# λ2 = 5
ax21 = plt.subplot(223)
ax21.set_ylim([-2,2])
ax21.plot(t, postMu_λ2_f1, label = 'posterior mean')
ax21.plot(A@t, y_f1, 'o', label = 'noisy data point')
ax21.plot(t, f1(t), label = r'$f_1({\bf t})=\sin({\bf t})$')
ax21.fill_between(t,
                  postMu_λ2_f1 - 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  postMu_λ2_f1 + 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax21.set_title(r'$\lambda={\bf 5}$')

ax22 = plt.subplot(224)
ax22.plot(t, postMu_λ2_f1)
ax22.plot(A@t, y_f1, 'o')
ax22.plot(t,f1(t))
ax22.fill_between(t,
                  postMu_λ2_f1 - 1.96*np.sqrt(np.diag(postSigma_λ2)),
                  postMu_λ2_f1 + 1.96*np.sqrt(np.diag(postSigma_λ2)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax22.set_title(r'$\lambda={\bf 5}$')

handles, labels = ax21.get_legend_handles_labels()
fig1.legend(handles,labels,
            loc = 'center',
            fontsize = 18,
            facecolor = 'w',f
            ramealpha = 0.9,
            edgecolor = 'k')
```

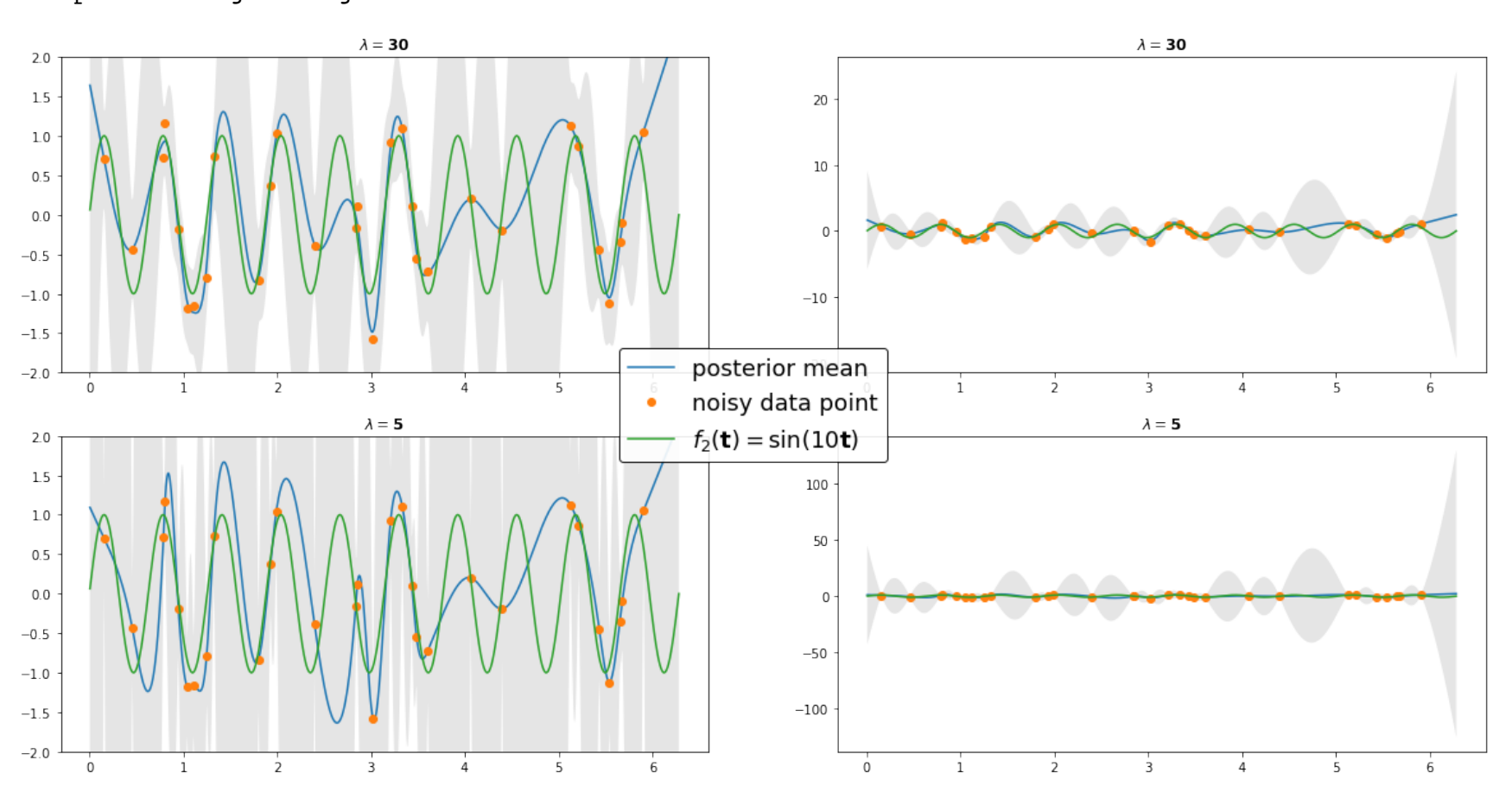
Out[11]: `<matplotlib.legend.Legend at 0x11804eb38>`



1.7.2 $f_2(t) = \sin(10t)$

In [12]:
```
#f2(x) = sin(10x)
fig2 = plt.figure(figsize = (20,10))

# λ1 = 30
ax11 = plt.subplot(221)
ax11.set_ylim([-2,2])
ax11.plot(t, postMu_λ1_f2)
ax11.plot(A@t, y_f2, 'o')
ax11.plot(t, f2(t))
ax11.fill_between(t,
                  postMu_λ1_f2 - 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  postMu_λ1_f2 + 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax11.set_title(r'$\lambda={\bf 30}$')

ax12 = plt.subplot(222)
ax12.plot(t, postMu_λ1_f2)
ax12.plot(A@t, y_f2, 'o')
ax12.plot(t, f2(t))
ax12.fill_between(t,
                  postMu_λ1_f2 - 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  postMu_λ1_f2 + 1.96 * np.sqrt(np.diag(postSigma_λ1)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax12.set_title(r'$\lambda={\bf 30}$')

# λ2 = 5
ax21 = plt.subplot(223)
ax21.set_ylim([-2,2])
ax21.plot(t, postMu_λ2_f2, label = 'posterior mean')
ax21.plot(A@t, y_f2, 'o', label = 'noisy data point')
ax21.plot(t, f2(t), label = r'$f_2({\bf t})=\sin(10{\bf t})$')
ax21.fill_between(t,
                  postMu_λ2_f2 - 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  postMu_λ2_f2 + 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax21.set_title(r'$\lambda={\bf 5}$')

ax22 = plt.subplot(224)
ax22.plot(t, postMu_λ2_f2)
ax22.plot(A@t, y_f2, 'o')
ax22.plot(t, f2(t))
ax22.fill_between(t,
                  postMu_λ2_f2 - 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  postMu_λ2_f2 + 1.96 * np.sqrt(np.diag(postSigma_λ2)),
                  facecolor = 'grey',
                  alpha = 0.2)
ax22.set_title(r'$\lambda={\bf 5}$')

handles, labels = ax21.get_legend_handles_labels()
fig2.legend(handles, labels,
            loc='center',
            fontsize = 18,
            facecolor = 'w',
            framealpha = 0.9,
            edgecolor = 'k')
```

Out[12]: `<matplotlib.legend.Legend at 0x1220a5320>`



---

2. Ridge Regression

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \left(w_0 + \mathbf{w}^T\mathbf{x}_i\right)\right)^2 + \lambda\|\mathbf{w}\|_2^2$$

2.1. Paras

$d \in \{2, 4, 6, 8\},\ \lambda \in \{0.01, 0.1, 1, 10\}$

In [13]:
```
ds = [2, 4, 6, 8]
λs = [0.01, 0.1, 1, 10]
```

2.2. Fit

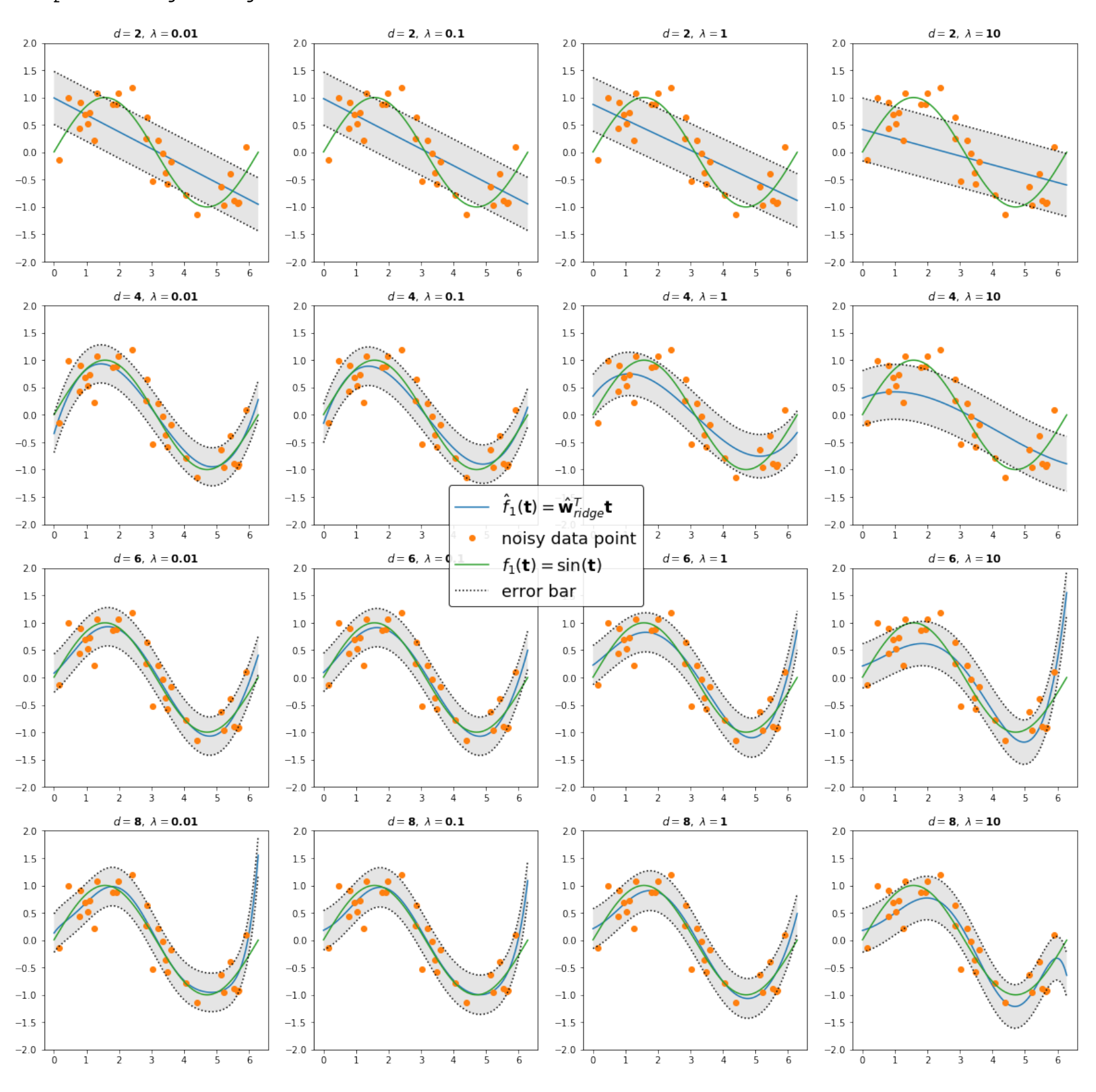
$\mathbf{T} = [\mathbf{t}^0, \mathbf{t}^1, \ldots, \mathbf{t}^{d-1}],\ \mathbf{X} = \mathbf{A}\mathbf{T}$

$\hat{\mathbf{w}}_{ridge} = \left(\lambda\mathbf{I}_d + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$

2.2.1 $f_1(t) = \sin(t)$

In [14]:
```
#f1(x) = sin(x)

fig1 = plt.figure(figsize = (20,20))

i = 1
for d in ds:
    T = np.power(np.tile(t,(d,1)).T,np.arange(d))
    X = A @ T
    for λ in λs:
        w = np.linalg.pinv(λ*np.eye(d)+X.T@X)@X.T@y_f1
        σ = np.sqrt((y_f1-X@w)@(y_f1-X@w)/(N-d))
        ax = plt.subplot(4,4,i)
        ax.set_ylim(-2,2)
        ax.set_title(r'$d={\bf'+str(d)+'},\ \lambda ={\\bf'+str(λ)+'}$')
        ax.plot(t, T@w, label = r'$\hat f_1({\bf t})=\hat{\bf w}_{ridge}^T{\bf t}$')
        ax.plot(A@t, y_f1, 'o', label = 'noisy data point')
        ax.plot(t, f1(t), label = r'$f_1({\bf t})=\sin({\bf t})$')
        ax.plot(t, T@w-σ, 'k:', label = 'error bar')
        ax.plot(t, T@w+σ, 'k:')
        ax.fill_between(t, T@w-σ, T@w+σ, facecolor = 'grey', alpha = 0.2)
        i += 1

handles, labels = ax.get_legend_handles_labels()
fig1.legend(handles, labels,
            loc = 'center',
            fontsize = 18,
            facecolor = 'w',
            framealpha = 0.9,
            edgecolor = 'k')
```

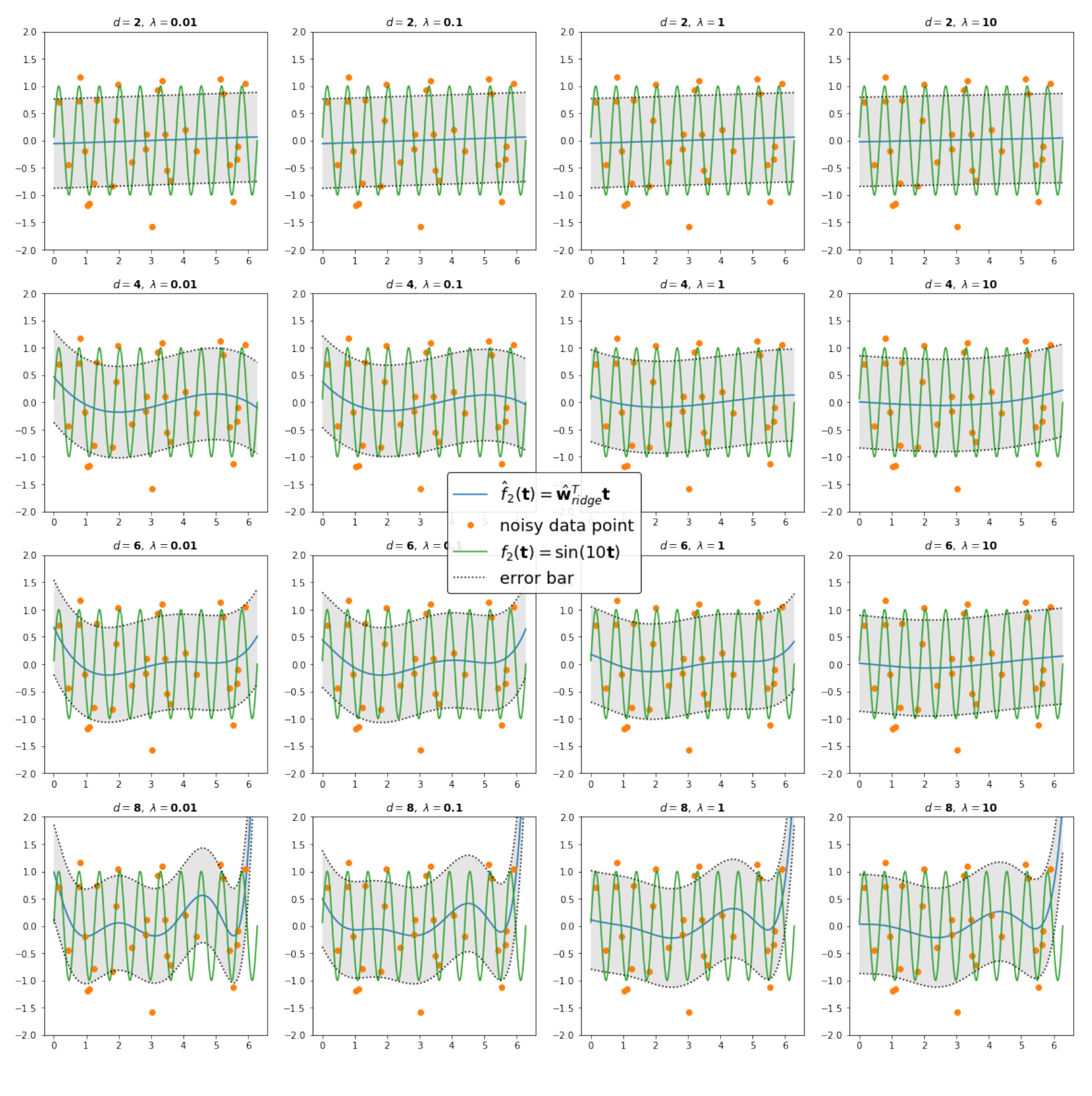
Out[14]: `<matplotlib.legend.Legend at 0x121d1ff60>`



2.2.2 $f_2(t) = \sin(10t)$

In [15]:
```
#f2(x) = sin(10x)

fig2 = plt.figure(figsize = (20,20))

i = 1
for d in ds:
    T = np.power(np.tile(t,(d,1)).T, np.arange(d))
    X = A @ T
    for λ in λs:
        w = np.linalg.pinv(λ*np.eye(d)+X.T@X)@X.T@y_f2
        σ = np.sqrt((y_f2-X@w)@(y_f2-X@w)/(N-d))
        ax = plt.subplot(4,4,i)
        ax.set_ylim(-2,2)
        ax.set_title(r'$d={\bf'+str(d)+'},\ \lambda ={\\bf'+str(λ)+'}$')
        ax.plot(t,T@w, label = r'$\hat f_2({\bf t})=\hat{\bf w}_{ridge}^T{\bf t}$')
        ax.plot(A@t, y_f2, 'o', label = 'noisy data point')
        ax.plot(t, f2(t), label = r'$f_2({\bf t})=\sin(10{\bf t})$')
        ax.plot(t, T@w-σ, 'k:', label = 'error bar')
        ax.plot(t, T@w+σ, 'k:')
        ax.fill_between(t, T@w-σ, T@w+σ, facecolor = 'grey', alpha = 0.2)
        i += 1

handles, labels = ax.get_legend_handles_labels()
fig2.legend(handles, labels,
            loc = 'center',
            fontsize = 18,
            facecolor = 'w',
            framealpha = 0.9,
            edgecolor = 'k')
```

Out[15]: `<matplotlib.legend.Legend at 0x12348b6a0>`



---

3. Summary

3.1. Gaussian Linear System

The prior precision term λ controls how much we think the function will vary, a large $\lambda = 30$ corresponds to a belief that the function is more smooth, a samll $\lambda = 5$ corresponds to a belief that the function is quite wiggly.

By comparing the ploting result of $\lambda = 30$ and $\lambda = 5$ for both $f(t) = \sin(t)$ and $f(t) = \sin(10t)$, we find indeed that when λ is small, the curve is more wiggly and have larger variance and when λ become larger, the function become more smooth and the posterior variance is also smaller.

3.2. Ridge Regression

From the ploting result, comparing the result horizontally, we find that, as the complexity penalty term λ get lager, the regression curve gets smoother; comparing the result vertically, we find that, as the degree d of polynomial gets larger, the model become more complex and the fitting curve becomes more wiggly.

When fitting the function $f(t) = \sin(10t)$, we find that, the low-polynomial model can no longer fits the function well as there are more turning points in the interval.

3.3. Comparison

Given the same number of observation, comparing the `Gaussian Linear System` and `Ridge Regression` , we find that, by assuming the underlying relationship, the regression model can fits the true relationship better when the assumed model is closer to the true model.

However, the regression model is less robust to differnt variations of the model. For example, in section 2.2.2 we find that the low-degree polynomial regression fit the function poorly when there are more turning points in the interval, but in section 1.7.2 the Bayesian approach still works.

The End