Problem 1

This week in lecture we discussed regression techniques utilising least squares. We first started off with linear and nonlinear regression. We showed how both of these problems can be formulated similarly with the minimization problem of $||y - Ax||^2$. The matrix A essentially encodes the structure of the solution as a linear combination of basis functions. The minimization problem itself can have many solutions based on the rank of A. Under such conditions the problem can be converted to a ridge regression problem, where a penalty is established on the norm of the decision variable, giving us the ability to tune our solution for accuracy vs magnitude. Using the Representer Theorem, least squares can even be used to solve regression problems in infinte dimensional Hilbert space.

Least squares regression techniques are an intuitive and powerful technique of fitting functions to data. It gives us the representation to come up with functions as linear combinations of basis functions as well as giving us the framework to solve for the coefficients of the basis functions. In my research I have used nonlinear least squares regression for system identification. In that setting, we wanted to know the masses and inertias for a robot's manipulator and each row in our A matrix consisted of how the masses and inertias are linearly related to the center of mass. The data points we used consisted of the pose information and center of mass reading.

```
import sys
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.io as sio

#mpl.style.use('seaborn')

mat_filename = "hw4p2_data.mat"
data_samples = sio.loadmat(mat_filename)

udata = data_samples['udata']
ydata = data_samples['ydata']
sdata = udata[0]
tdata = udata[1]

def part_a():
    print("Part a")
    print("""The matrix A can be computed where each row represents a prediction
for y, i.e. ym approx Am * alpha = fm. fm itself can be represented with the
following vector equation, which is a linear combination of the following basis
functions and alpha as the vector of coefficients:
fm = [sm**2, tm**2, sm*tm, sm, tm, 1] * [a1, a2, a3, a4, a5, a6]^T
   = [sm**2, tm**2, sm*tm, sm, tm, 1] * alpha
   = Am * alpha
Thus A[m,:] = [sm**2, tm**2, sm*tm, sm, tm, 1]
""")

    M = len(ydata)
    A = np.ones(shape=(M, 6))

    for m in range(M):
        s = sdata[m]
        t = tdata[m]
        A[m,:] = [s**2, t**2, s*t, s, t, 1]

    return A

A = part_a()

def part_b():
    print("Part b")
    A_rank = np.linalg.matrix_rank(A)
    print("rank of A: " + str(A_rank) + " => full rank (rank = N)")

    alpha = np.linalg.inv(np.transpose(A) @ A) @ np.transpose(A) @ ydata
    print("")
    print("alpha_hat")
    print(alpha)
    return alpha

alpha = part_b()

def plot_contour(alpha):
    fig = plt.figure()
    fig.suptitle("Countour plot of f_hat(s,t)")
    ax = fig.add_subplot(111)

    s_vec = np.linspace(0, 1, num=1000)
    t_vec = np.linspace(0, 1, num=1000)

    s_mat, t_mat = np.meshgrid(s_vec, t_vec)

    f_hat_mat = alpha[0] * s_mat**2 + alpha[1] * t_mat**2 + \
```

```
            alpha[2] * s_mat * t_mat + alpha[3] * s_mat + alpha[4] * t_mat + \
            alpha[5]

    cset1 = ax.contourf(s_mat, t_mat, f_hat_mat, levels=50)
    cset = ax.contour(s_mat, t_mat, f_hat_mat, cset1.levels, colors='k')

    fig.colorbar(cset1, ax=ax)

    ax.set_xlabel("s")
    ax.set_ylabel("t")

    plt.show()

def part_c(alpha):
    plot_contour(alpha)

part_c(alpha)
```

Part a
The matrix A can be computed where each row represents a prediction for y, i.e. ym approx Am * alpha = fm. fm itself can be represented with the following vector equation, which is a linear combination of the following basis functions and alpha as the vector of coefficients:

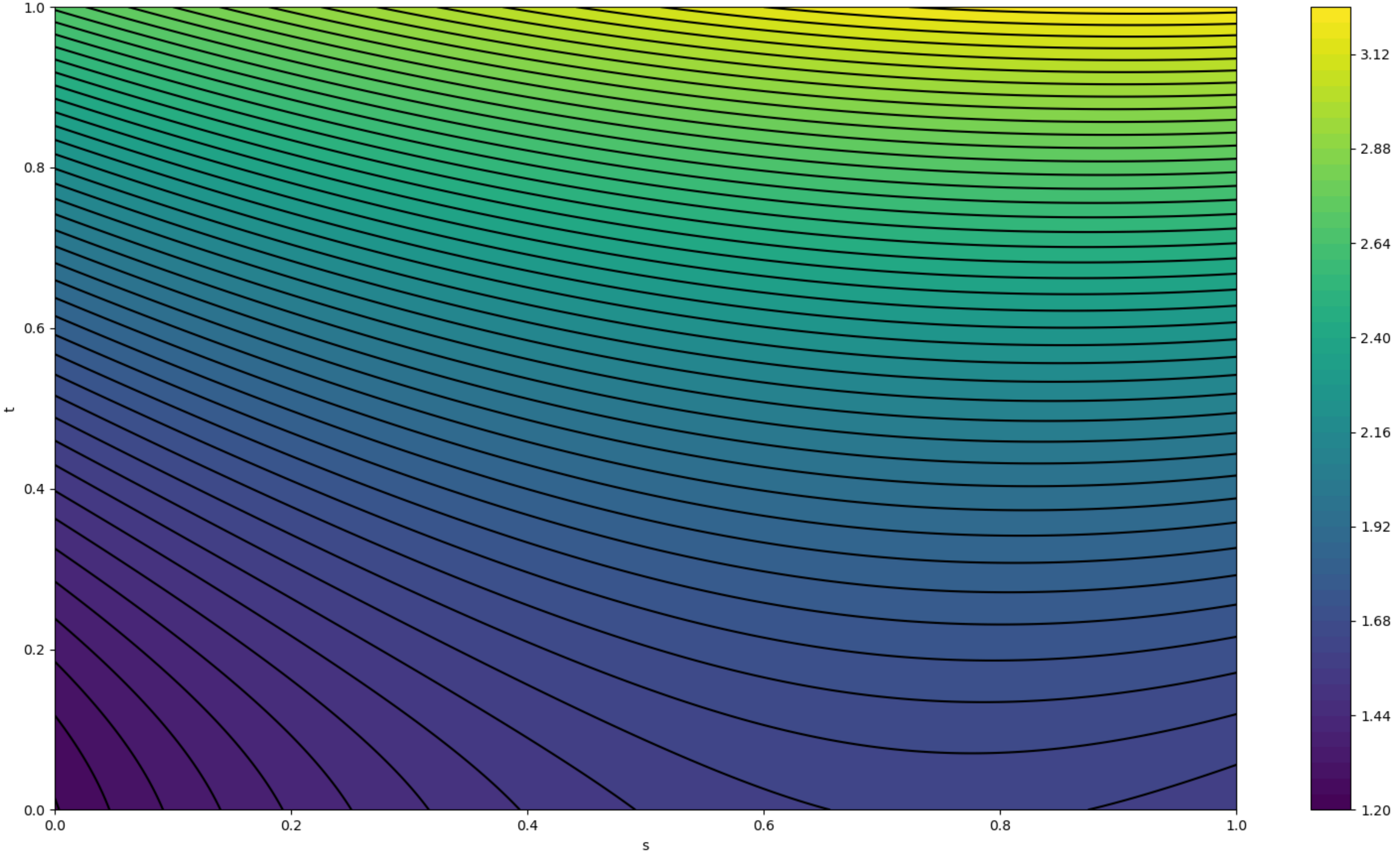
```
fm = [sm**2, tm**2, sm*tm, sm, tm, 1] * [a1, a2, a3, a4, a5, a6]^T
   = [sm**2, tm**2, sm*tm, sm, tm, 1] * alpha
   = Am * alpha
Thus A[m,:] = [sm**2, tm**2, sm*tm, sm, tm, 1]
```

Part b
rank of A: 6 => full rank (rank = N)

alpha_hat

```
[[-0.63387231]
 [ 1.22817315]
 [ 0.19346972]
 [ 0.97008806]
 [ 0.22554988]
 [ 1.23638021]]
```

Countour plot of f_hat(s,t)



3. $f(s,t)=\alpha_1 s^2+\alpha_2 t^2+\alpha_3 st+\alpha_4 s+\alpha_5 t+\alpha_6$

a) $\nabla f(s,t)=G_{s,t}\alpha \quad G_{s,t}=?$

$$\nabla f(s,t)=\begin{bmatrix}\frac{\partial f}{\partial s}\\ \frac{\partial f}{\partial t}\end{bmatrix}=\begin{bmatrix}2\alpha_1 s+0\alpha_2+\alpha_3 t+\alpha_4+0\alpha_5+0\alpha_6\\ 0\alpha_1+2\alpha_2 t+\alpha_3 s+0\alpha_4+\alpha_5+0\alpha_6\end{bmatrix}=\boxed{\underbrace{\begin{bmatrix}2s&0&t&1&0&0\\0&2t&s&0&1&0\end{bmatrix}}_{G_{s,t}}}\alpha$$

b) $\|\nabla f(s,t)\|_2^2=\alpha^T H_{s,t}\alpha \quad H_{s,t}=?$

$$\|G_{s,t}\alpha\|_2^2=(G_{s,t}\alpha)^T(G_{s,t}\alpha)=\alpha^T G_{s,t}^T G_{s,t}\alpha$$

$$H_{s,t}=G_{s,t}^T G_{s,t}=\begin{bmatrix}2s&0\\0&2t\\t&s\\1&0\\0&1\\0&0\end{bmatrix}\begin{bmatrix}2s&0&t&1&0&0\\0&2t&s&0&1&0\end{bmatrix}=\boxed{\underbrace{\begin{bmatrix}4s^2&0&2st&2s&0&0\\0&4t^2&2st&0&2t&0\\2st&2st&t^2+s^2&t&s&0\\2s&0&t&1&0&0\\0&2t&s&0&1&0\\0&0&0&0&0&0\end{bmatrix}}_{H_{s,t}}}$$

$$\mathrm{Null}(H_{s,t})=\mathrm{Null}(G_{s,t}^T G_{s,t})=\mathrm{Null}(G_{s,t})$$

The nullspace of $G_{s,t}$ for all s and t consists of α s.t. $\alpha_{1-5}=0$ and α_6 can be any value.

Row space of $G_{s,t}$ = $a_1\begin{bmatrix}2s\\0\\t\\1\\0\\0\end{bmatrix}+b_1\begin{bmatrix}0\\2t\\s\\0\\1\\0\end{bmatrix}$ $\quad \mathrm{Null}(G_{s,t})=\mathrm{Row}(G_{s,t})^{\perp}$

$$=c_1\begin{bmatrix}0\\0\\0\\0\\0\\1\end{bmatrix}\ \forall s,t$$

Thus functions f, where $\alpha_{1-5}=0$ and α_6 is free are in the nullspace of $H_{s,t}$.

$f=\alpha_1 s^2+\alpha_2 t^2+\alpha_3 st+\alpha_4 s+\alpha_5 t+\alpha_6$

$\boxed{f=\alpha_6}$

These are constant valued functions.

3c) Compute the matrix

$$Q = \int_0^1\int_0^1 H_{s,t}\, ds\, dt$$

$$= \int_0^1\int_0^1 \begin{bmatrix} 4s^2 & 0 & 2st & 2s & 0 & 0 \\ 0 & 4t^2 & 2st & 0 & 2t & 0 \\ 2st & 2st & t^2+s^2 & t & s & 0 \\ 2s & 0 & t & 1 & 0 & 0 \\ 0 & 2t & s & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} ds\, dt = \int_0^1 \begin{bmatrix} 4/3 & 0 & t & 1 & 0 & 0 \\ 0 & 4t^2 & t & 0 & 2t & 0 \\ t & t & t^2+\frac{1}{3} & t & 1/2 & 0 \\ 1 & 0 & t & 1 & 0 & 0 \\ 0 & 2t & 1/2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} dt$$

$$\frac{t^3}{3} + \frac{1}{3}t \Big|_0^1 = \frac{1}{3} + \frac{1}{3}$$

$$= \boxed{\begin{bmatrix} 4/3 & 0 & 1/2 & 1 & 0 & 0 \\ 0 & 4/3 & 1/2 & 0 & 1 & 0 \\ 1/2 & 1/2 & 2/3 & 1/2 & 1/2 & 0 \\ 1 & 0 & 1/2 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}} = Q$$

d)

$$\underset{f \in P_2}{\text{minimize}} \sum_{m=1}^{M} (y_m - f(s_m, t_m))^2 + \delta \int_0^1\int_0^1 \|\nabla f(s,t)\|_2^2\, ds\, dt$$

$$\Rightarrow \underset{\alpha \in \mathbb{R}^6}{\text{minimize}} \|y - A\alpha\|_2^2 + \delta \int_0^1\int_0^1 \alpha^T H_{s,t}\, \alpha\, ds\, dt$$

$$\Rightarrow \underset{\alpha \in \mathbb{R}^6}{\text{minimize}} \|y - A\alpha\|_2^2 + \delta\, \alpha^T \int_0^1\int_0^1 H_{s,t}\, ds\, dt\, \alpha$$

$$\Rightarrow \underset{\alpha \in \mathbb{R}^6}{\text{minimize}} \|y - A\alpha\|_2^2 + \delta\, \alpha^T Q \alpha$$

$$\nabla_\alpha(\|y - A\alpha\|_2^2 + \delta \alpha^T Q \alpha) = \nabla(\cancel{\|y\|_2^2} - 2y^T A\alpha + \|A\alpha\|_2^2 + \delta \alpha^T Q \alpha)$$

$$= -2y^T A + 2A^T A\alpha + 2\delta Q\alpha$$

$$= -2A^T y + 2(A^T A + \delta Q)\alpha$$

$$\Rightarrow \quad 0 = -\cancel{2}A^T y + \cancel{2}(A^T A + \delta Q)\alpha$$

$(A^T A + \delta Q)\alpha = A^T y$ ← optimality conditions.

$$\boxed{\alpha = (A^T A + \delta Q)^{-1} A^T y}$$

The regularizer is penalizing high values of Q, which corresponds to high values of the gradient of f, $\nabla f(s,t)$. Since it is penalizing $\nabla f(s,t)$, for large δ we expect to see functions that do not have high $\nabla f(s,t)$, i.e. functions that do not oscillate a lot and are smooth.

See code for part e)

4.

$$\underset{f \in T}{\text{minimize}} \sum_{m=1}^{M} |y_m - f(t_m)|^2$$

Does having an orthonormal basis for T help you computationally in this setting?

$$\underset{f \in T}{\text{minimize}} \sum_{m=1}^{M} |y_m - f(t_m)|^2 \qquad f(t) = \sum_{n=1}^{N} x_n \psi_n(t)$$

where $\psi_n(t)$ are elements in an orthonormal basis, $\{\psi_1(t), \ldots, \psi_N(t)\}$, for T

$$\Rightarrow \underset{x \in \mathbb{R}^N}{\text{minimize}} \|y - Ax\|_2^2 \qquad \text{where } A = \begin{bmatrix} \psi_1(t_1) & \cdots & \psi_N(t_1) \\ & & \vdots \\ \psi_1(t_m) & \cdots & \psi_N(t_m) \end{bmatrix}$$

The solution to this problem obeys the following equation:

$$\nabla(\|y - Ax\|_2^2) = \nabla(\|y\|_2^2 - 2y^T Ax + \|Ax\|_2^2) = 0$$

$$\Rightarrow 0 = -2y^T A + 2A^T A x \Rightarrow A^T y = A^T A x$$

$$x = (A^T A)^{-1} A^T y$$

$$A^T A = \underset{N \times M}{\begin{bmatrix} \psi_1(t_1) & \cdots & \psi_1(t_M) \\ \vdots & \ddots & \vdots \\ \psi_N(t_1) & \cdots & \psi_N(t_M) \end{bmatrix}} \underset{M \times N}{\begin{bmatrix} \psi_1(t_1) & \cdots & \psi_N(t_1) \\ \vdots & \ddots & \vdots \\ \psi_1(t_M) & \cdots & \psi_N(t_M) \end{bmatrix}} = \underset{N \times N}{\begin{bmatrix} \sum_{m=1}^{M} \psi_1(t_m)\psi_1(t_m) & \cdots & \sum_{m=1}^{M} \psi_N(t_m)\psi_1(t_m) \\ \vdots & \ddots & \vdots \\ \sum_{m=1}^{M} \psi_1(t_m)\psi_N(t_m) & \cdots & \sum_{m=1}^{M} \psi_N(t_m)\psi_N(t_m) \end{bmatrix}}$$

Since $\psi_n(t)$ are elements of an orthonormal basis, we know that

$$\langle \psi_i(t), \psi_j(t) \rangle = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad \text{where } \langle \psi_i(t), \psi_j(t) \rangle = \int_{-\infty}^{\infty} \psi_i(t)\psi_j(t)\,dt$$

However, noting that the entries are discrete sums of $\psi_i(t_m) \cdot \psi_j(t_m)$ instead of an integral, we cannot leverage the inner product of orthonormal basis's to simplify the elements and computation of $A^T A$. It is interesting to note that if our data, t_m, is dense enough, the sums can be approximated as the continuous integrals.

5. Let A be a $M \times N$ matrix with $\text{rank}(A) < N$. We have seen in this case that the least squares problem

$$\underset{x \in \mathbb{R}^N}{\text{minimize}} \ \|y - Ax\|_2^2 \quad (2)$$

has an infinite number of solutions. We have also seen, however, that the regularized least squares problem

$$\underset{x \in \mathbb{R}^N}{\text{minimize}} \ \|y - Ax\|_2^2 + \delta\|x\|_2^2 \quad (3)$$

has a unique solution for every $\delta > 0$. In this problem, we will show that as $\delta \to 0$ the regularized solution goes to the minimum norm solution of

$$\underset{x \in \mathbb{R}^N}{\text{minimize}} \ \|x\|_2^2 \ \text{subject to} \ A^T A x = A^T y \quad (4)$$

a) Start by showing that if $x_1 \in \text{Row}(A)$ and $x_2 \in \text{Row}(A)$ then $A^T A x_1 \neq A^T A x_2$ unless $x_1 = x_2$.

Showing $A^T A x_1 \neq A^T A x_2$ unless $x_1 = x_2$ is equivalent to showing

$$A^T A x_1 = A^T A x_2 \ \text{unless} \ x_1 \neq x_2$$

$$A^T A x_1 - A^T A x_2 = 0$$

$$A^T A (x_1 - x_2) = 0$$

Thus $x_1 - x_2 \in \text{Null}(A^T A)$

From Technical details in Notes: $(\text{Null}(A^T A) = \text{Null}(A))$

$$x_1 - x_2 \in \text{Null}(A)$$

Since x_1 and x_2 are in $\text{Row}(A)$ any linear combination is also in $\text{Row}(A)$

Thus, $x_1 - x_2 \in \text{Row}(A)$

For $x_1 - x_2$ to be in both $\text{Row}(A)$ and $\text{Null}(A)$, $x_1 - x_2$ must be 0.

$$x_1 - x_2 = 0 \Rightarrow x_1 = x_2$$

Therefore $A^T A x_1 = A^T A x_2$ if $x_1 = x_2$ and $A^T A x_1 \neq A^T A x_2$ otherwise.

b) Argue that the solution to (4) is always unique.

Let us assume there are two solutions, x_1, x_2 to the following equation:

$$A^T A x_1 = A^T y$$

$$A^T A x_2 = A^T y$$

$\Rightarrow A^T A x_1 = A^T A x_2$ $\qquad$ $x_1, x_2 \in \text{Row}(A^T A)$

$x_1, x_2 \in \text{Null}(A^T A)^{\perp} \rightarrow$ From Technical Details

$x_1, x_2 \in \text{Null}(A)^{\perp}$

$x_1, x_2 \in \text{Row}(A)$

Using our results from part a)

$A^T A x_1 = A^T A x_2$ if $x_1 = x_2$ therefore the solution is a unique x.

5c) Show that $\lim_{n\to\infty} \|x^* - \hat{x}_n\|_2 = 0$

$A^TAx^* = A^Ty$

$(A^TA + \delta I)\hat{x}_n = A^Ty$

$\Rightarrow A^TAx^* = A^TA\hat{x}_n + \delta I\hat{x}_n$

$A^TAx^* - A^TA\hat{x}_n - \delta\hat{x}_n = 0$

$A^TA(x^* - \hat{x}_n) = \delta\hat{x}_n$

$\|A^TA(x^* - \hat{x}_n)\| = \|\delta\hat{x}_n\|$ $\quad$ $\|A^TA(x_1 - x_2)\|_2 \geq C\|x_1 - x_2\|_2$

$\|A^TA(x^* - \hat{x}_n)\| \geq C\|x^* - \hat{x}_n\|$

$\Rightarrow \|\delta\hat{x}_n\| \geq C\|x^* - \hat{x}_n\|$

$\|x^* - \hat{x}_n\| \leq \frac{\delta}{C}\|\hat{x}_n\|$ $\quad$ $\delta = \frac{1}{n}$

$\|x^* - \hat{x}_n\| \leq \frac{1}{nC}\|\hat{x}_n\|$

$\|x^* - \hat{x}_n\| \leq \frac{1}{nC}\|x^*\|$

$\|y - Ax^*\|^2 \leq \|y - A\hat{x}_n\|^2$ Since x^* is a minimizer of $\|y-Ax\|^2$

$\|y - Ax^*\|^2 - \|y - A\hat{x}_n\|^2 \leq 0$ because $\|y-Ax\|^2$ is an optimality condition

$\|y - A\hat{x}_n\|^2 + \delta\|\hat{x}_n\|^2 \leq \|y - Ax^*\|^2 + \delta\|x^*\|^2$ Since $\hat{x}$ is a minimizer of $\|y-Ax\|^2 + \delta\|x\|$

$\delta\|\hat{x}_n\|^2 \leq \|y - Ax^*\|^2 - \|y - A\hat{x}_n\|^2 + \delta\|x^*\|^2$

$\delta\|\hat{x}_n\|^2 \leq \delta\|x^*\|^2$

$\|\hat{x}_n\|^2 \leq \|x^*\|^2$

$\|\hat{x}_n\| \leq \|x^*\|$

$\lim_{n\to\infty} \|x^* - \hat{x}_n\| \geq 0$ since $\|\cdot\| \geq 0$

$\lim_{n\to\infty} \|x^* - \hat{x}_n\| \leq \lim_{n\to\infty} \underbrace{\frac{1}{nC}}_{\to 0}\|x^*\|$

$= 0$

$\lim_{n\to\infty} \|x^* - \hat{x}_n\| \leq 0$

Thus $\lim_{n\to\infty} \|x^* - \hat{x}_n\| = 0$

6.

$h(t) = \begin{cases} e^{-t}, & t \geq 0 \\ 0, & t < 0 \end{cases}$

Set $\tau_1 = 1/3, \tau_2 = 1/2, \tau_3 = 3/2, \tau_4 = 2$ and $y_1 = 4, y_2 = 5, y_3 = 1, y_4 = -2$ and define

$a_m(t) = h(t - \tau_m)$

Solve $\underset{f \in L_2(\mathbb{R})}{\text{minimize}} \sum_{m=1}^{4} |y_m - \langle f, a_m \rangle|^2 + \delta \|f\|^2$ $\qquad \langle f, g \rangle = \int_{-\infty}^{\infty} f(t)\, g(t)\, dt$

for $\delta = 10^{-3}$ $\qquad \|f\|^2 = \int_{-\infty}^{\infty} |f(t)|^2 dt$

(Representer Theorem)

From the notes, we know the solution is given in the following form:

$$\hat{f} = \sum_{m=1}^{M} \hat{a}_m a_m$$

where $\hat{a} = (K + \delta I)^{-1} y$ and $K = \begin{bmatrix} \langle a_1, a_1 \rangle_s & \cdots & \langle a_M, a_1 \rangle_s \\ \vdots & \ddots & \vdots \\ \langle a_1, a_M \rangle_s & & \langle a_M, a_M \rangle_s \end{bmatrix}$

where $\langle a_i, a_j \rangle_s = \int_{-\infty}^{\infty} a_i(t)\, a_j(t)\, dt$

See the code for the plot.

```
import sys
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.io as sio

mat_filename = "hw4p2_data.mat"
data_samples = sio.loadmat(mat_filename)

udata = data_samples['udata']
ydata = data_samples['ydata']
sdata = udata[0]
tdata = udata[1]

def plot_contour(alpha, delta):
    fig = plt.figure()
    fig.suptitle("Countour plot of f_hat(s,t) with delta = " + str(delta))
    ax = fig.add_subplot(111)

    s_vec = np.linspace(0, 1, num=1000)
    t_vec = np.linspace(0, 1, num=1000)

    s_mat, t_mat = np.meshgrid(s_vec, t_vec)

    f_hat_mat = alpha[0] * s_mat**2 + alpha[1] * t_mat**2 + \
            alpha[2] * s_mat * t_mat + alpha[3] * s_mat + alpha[4] * t_mat + \
            alpha[5]

    cset1 = ax.contourf(s_mat, t_mat, f_hat_mat, levels=50)
    cset = ax.contour(s_mat, t_mat, f_hat_mat, cset1.levels, colors='k')

    fig.colorbar(cset1, ax=ax)

    ax.set_xlabel("s")
    ax.set_ylabel("t")

    plt.show()

def part_e():
    M = len(ydata)
    A = np.ones(shape=(M, 6))

    for m in range(M):
        s = sdata[m]
        t = tdata[m]
        A[m,:] = [s**2, t**2, s*t, s, t, 1]

    Q = np.ones(shape=(6, 6))
    Q[0,:] = [4/3,   0, 1/2,   1,   0,   0]
    Q[1,:] = [  0, 4/3, 1/2,   0,   1,   0]
    Q[2,:] = [1/2, 1/2, 2/3, 1/2, 1/2,   0]
    Q[3,:] = [  1,   0, 1/2,   1,   0,   0]
    Q[4,:] = [  0,   1, 1/2,   0,   1,   0]
    Q[5,:] = [  0,   0,   0,   0,   0,   0]

    deltas = [1e-3, 1e0, 1e3]

    alphas = [np.linalg.inv(np.transpose(A) @ A + delta * Q) @ \
            np.transpose(A) @ ydata for delta in deltas]

    for alpha, delta in zip(alphas, deltas):
        plot_contour(alpha, delta)

    print("Part e")
```

```
    print("""
delta = 1e-3 is an interesting value since at this value, the
delta essentially negates the penalty term on Q, i.e. the gradient of
f. Thus the solution is not affected by the penalty term and gives a "rougher"
solution, which a steeper contour plot.
On the other hand at a higher delta value like 1e3, the objective
function is dominated by the penalty term and thus the solution is one which
has a smaller gradient, i.e. one that is flat/smooth. The values of f
themselves don't change much over the interval of interest.
delta = 1e0 is another interesting value because this is where neither the
penalty term nor the first term has complete dominance on the solution, but
rather there is a noticeable contribution between both terms. Values smaller
than 1e0 result in solutions similar to 1e-3 and values greater than 1e0 result
in solutions similar to 1e3. Here similarity refers to the shape of the
contours.
""")

part_e()
```

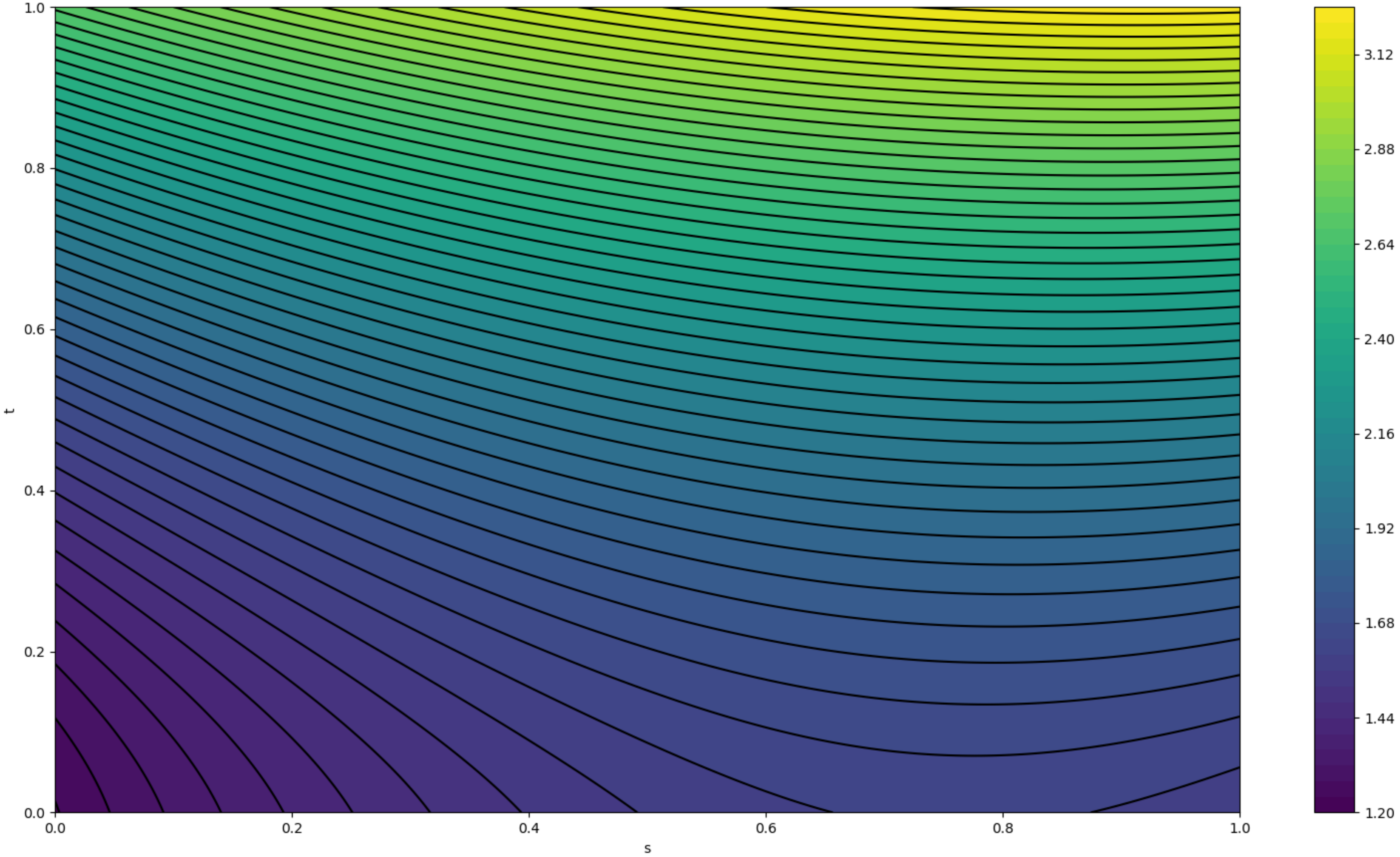
Part e

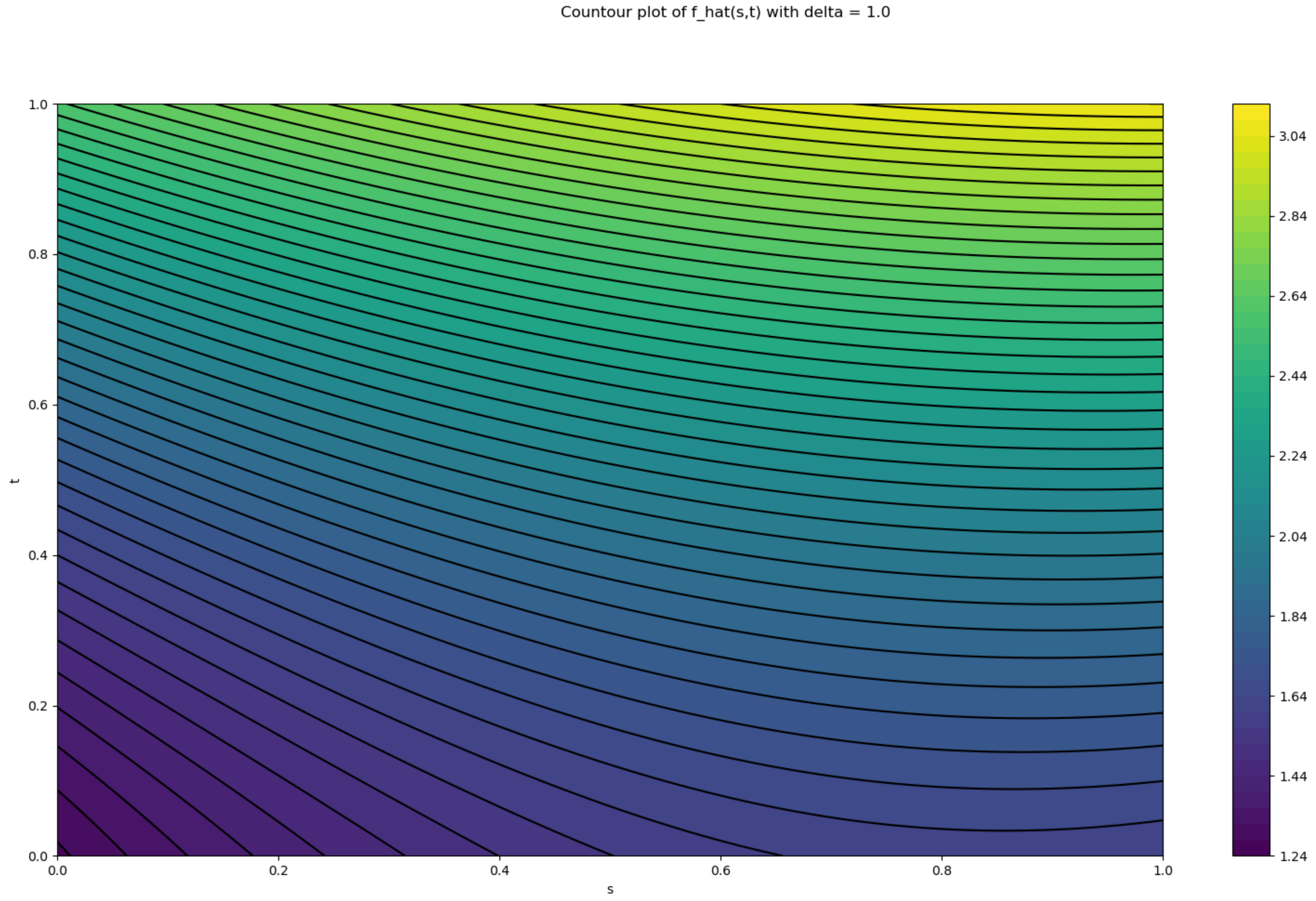
delta = 1e-3 is an interesting value since at this value, the delta essentially negates the penalty term on Q, i.e. the gradient of f. Thus the solution is not affected by the penalty term and gives a "rougher" solution, which a steeper contour plot.

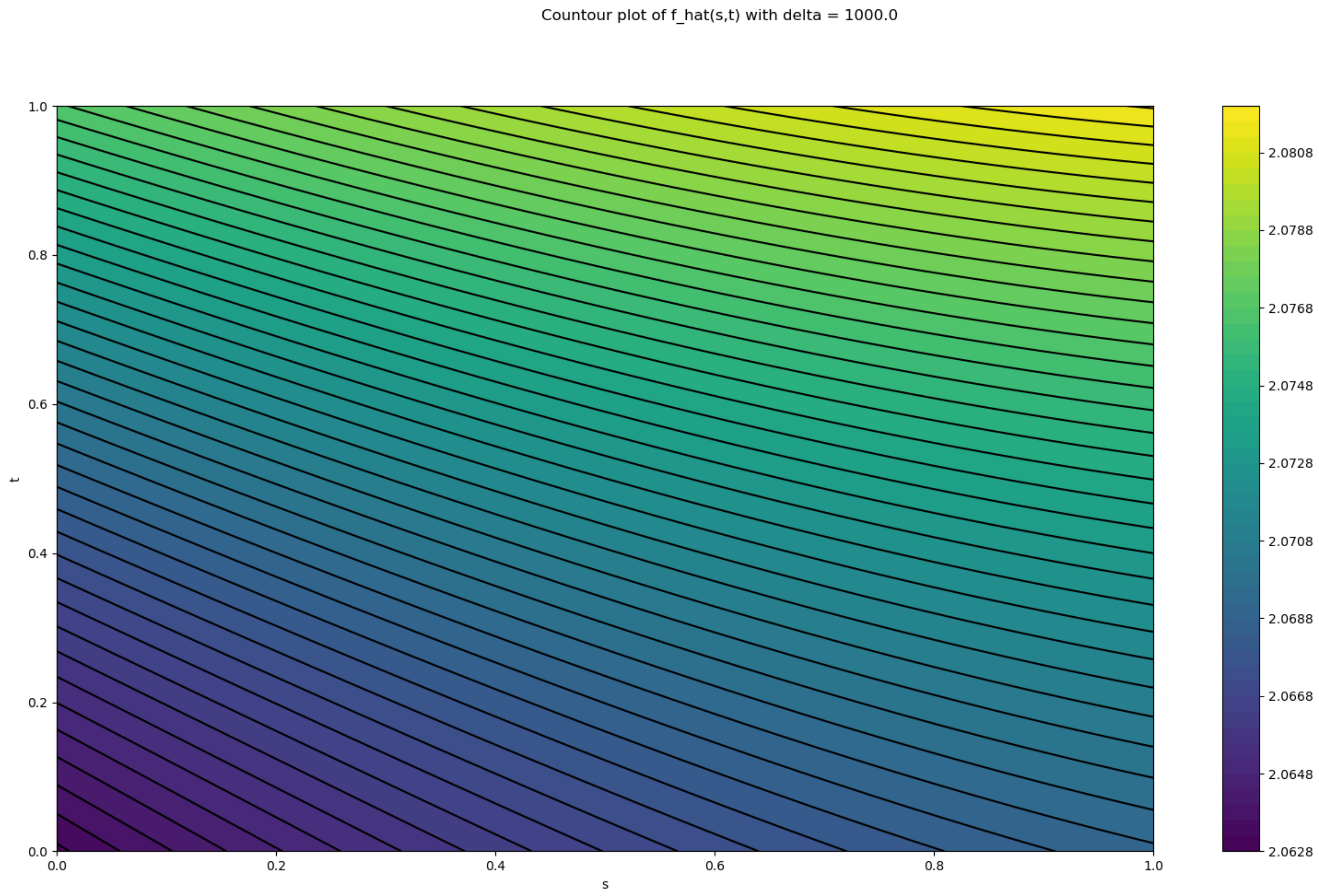
On the other hand at a higher delta value like 1e3, the objective function is dominated by the penalty term and thus the solution is one which has a smaller gradient, i.e. one that is flat/smooth. The values of f themselves don't change much over the interval of interest.

delta = 1e0 is another interesting value because this is where neither the penalty term nor the first term has complete dominance on the solution, but rather there is a noticeable contribution between both terms. Values smaller than 1e0 result in solutions similar to 1e-3 and values greater than 1e0 result in solutions similar to 1e3. Here similarity refers to the shape of the contours.

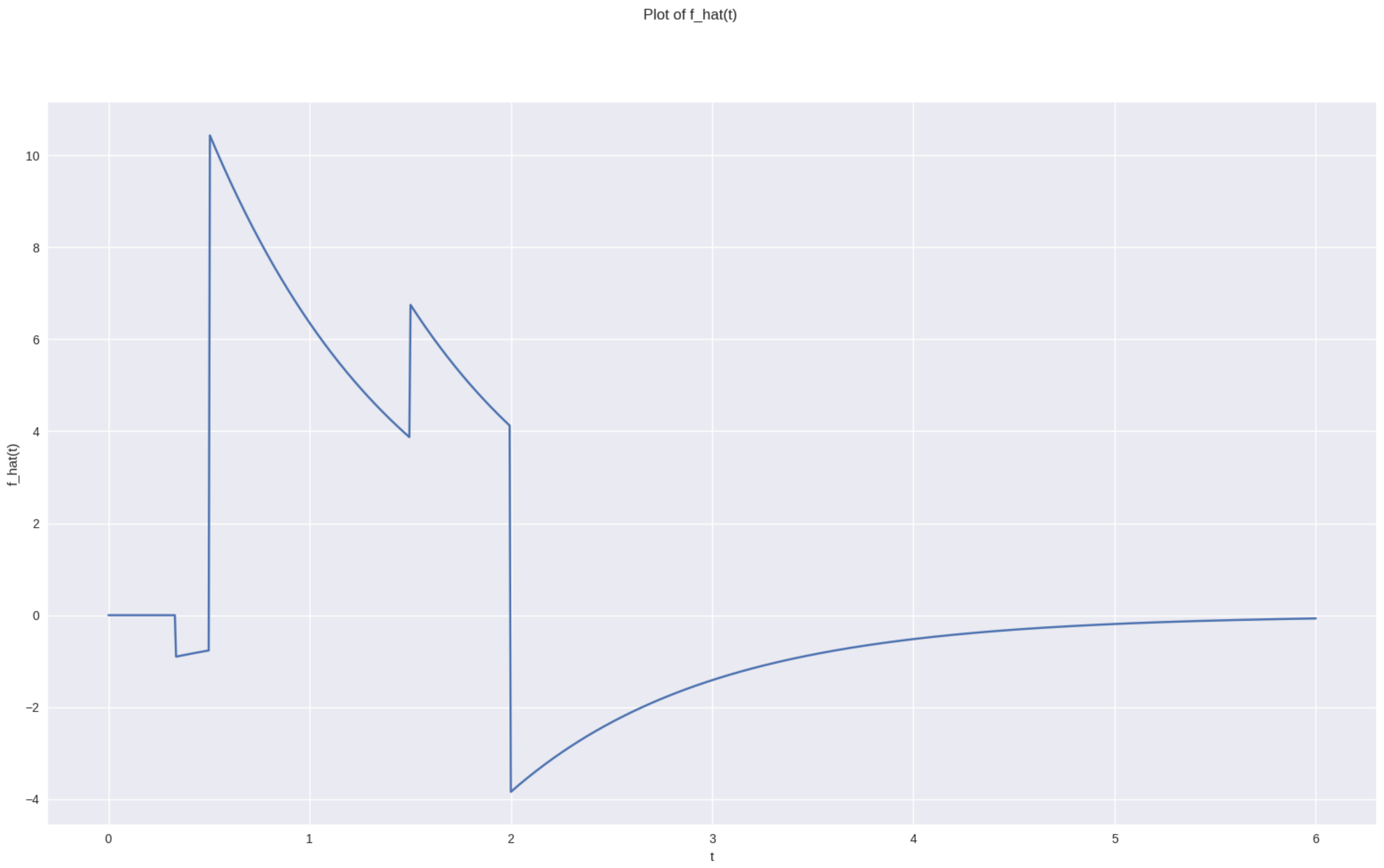
Countour plot of f_hat(s,t) with delta = 0.001




Countour plot of f_hat(s,t) with delta = 1.0
t
s
0.0
0.2
0.4
0.6
0.8
1.0
1.24
1.44
1.64
1.84
2.04
2.24
2.44
2.64
2.84
3.04


Countour plot of f_hat(s,t) with delta = 1000.0
t
s
0.0
0.2
0.4
0.6
0.8
1.0
2.0808
2.0788
2.0768
2.0748
2.0728
2.0708
2.0688
2.0668
2.0648
2.0628

```python
import sys
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.io as sio
import scipy.integrate as integrate

mpl.style.use('seaborn')

h = lambda z: np.exp(-z) if z >= 0 else 0

tau_vec = [1/3, 1/2, 3/2, 2]
y_vec = [4, 5, 1, -2]

def plot(alpha_hat):
    fig = plt.figure()
    fig.suptitle("Plot of f_hat(t)")
    ax = fig.add_subplot(111)

    t_vec = np.linspace(0, 6, 1000)
    f_vec = [sum([alpha_hat[i] * h(t - tau_vec[i]) for i in range(0,
len(alpha_hat))]) for t in t_vec]

    ax.plot(t_vec, f_vec)

    ax.set_xlabel("t")
    ax.set_ylabel("f_hat(t)")

    plt.show()

def prob6():
    M = len(y_vec)

    K = np.ones(shape=(M,M))

    aj_ai = lambda z: h(z - tau_vec[j]) * h(z - tau_vec[i])

    for i in range(0, M):
        for j in range(0, M):
            K[i,j] = integrate.quad(aj_ai, -np.inf, np.inf)[0]

    delta = 10e-3
    alpha_hat = np.linalg.inv(K + delta * np.identity(M)) @ y_vec

    plot(alpha_hat)

prob6()
```


Plot of f_hat(t)
10
8
6
4
2
0
−2
−4
f_hat(t)
0
1
2
3
4
5
6
t