HW 26-27

Imports

In [1]:
```
import jax
import jax.numpy as jnp
import numpy as np
import pandas as pd
# import PIL
import scipy
# import sympy as sp
from matplotlib import pyplot as plt
from scipy.optimize import minimize
```

HW 26

A classical benchmark problem for nonlinear regression algorithms is to determine the reaction coefficients for the thermal isomerization of α-pinene. The model takes the form of a set of five, coupled linear differential equation:

$$\begin{aligned}
\dot{x}_1 &= -(\theta_1 + \theta_2)x_1 \\
\dot{x}_2 &= \theta_1 x_1 \\
\dot{x}_3 &= \theta_2 x_1 - (\theta_3 + \theta_4)x_3 + \theta_5 x_5 \\
\dot{x}_4 &= \theta_3 x_3 \\
\dot{x}_5 &= \theta_4 x_3 - \theta_5 x_5
\end{aligned}$$

a. Write the model equations in the form $\dot{\boldsymbol{x}} = A(\theta)\boldsymbol{x}$ where the matrix $A(\theta) \in \mathbb{R}^{5\times 5}$ depends on the parameters θ and is independent of $\boldsymbol{x}$.

In [2]:
```
def A_theta(th1: float, th2: float, th3: float, th4: float, th5: float) -> np.ndarray:
    out = np.array([
        [-(th1+th2) , 0, 0          , 0, 0      ],
        [th1        , 0, 0          , 0, 0      ],
        [th2        , 0, -(th3+th4) , 0, th5    ],
        [0          , 0, th3        , 0, 0      ],
        [0          , 0, th4        , 0, -th5   ]
    ])
    return out

def x_dot(thet: np.ndarray, x: np.ndarray) -> np.ndarray:
    return A_theta(*thet)*x
```

b. The formal solution to the this differential equation is $\boldsymbol{x}(t) = e^{A(\theta)t}\boldsymbol{x}(0)$ where $e^{A(\theta)t} \in \mathbb{R}^{5\times 5}$ refers to the *matrix exponential function*. The exponential of a matrix is defined in terms of a power series in analogy to a scalar exponential,

$$e^A = \sum_{n=0}^{\infty} \frac{A^n}{n!},$$

and nearly all numerical linear algebra libraries have efficient implementations of this function. Find a function in your environment that implements the matrix exponential and write a routine that takes three arguments: $t \in \mathbb{R}^n$, $\boldsymbol{x}(0) \in \mathbb{R}^5$ and $\theta \in \mathbb{R}^5$. It should return a 5 by n matrix whose i^{th} column correspond to $\boldsymbol{x}(t_i) = e^{A(\theta)t_i}\boldsymbol{x}(0)$.

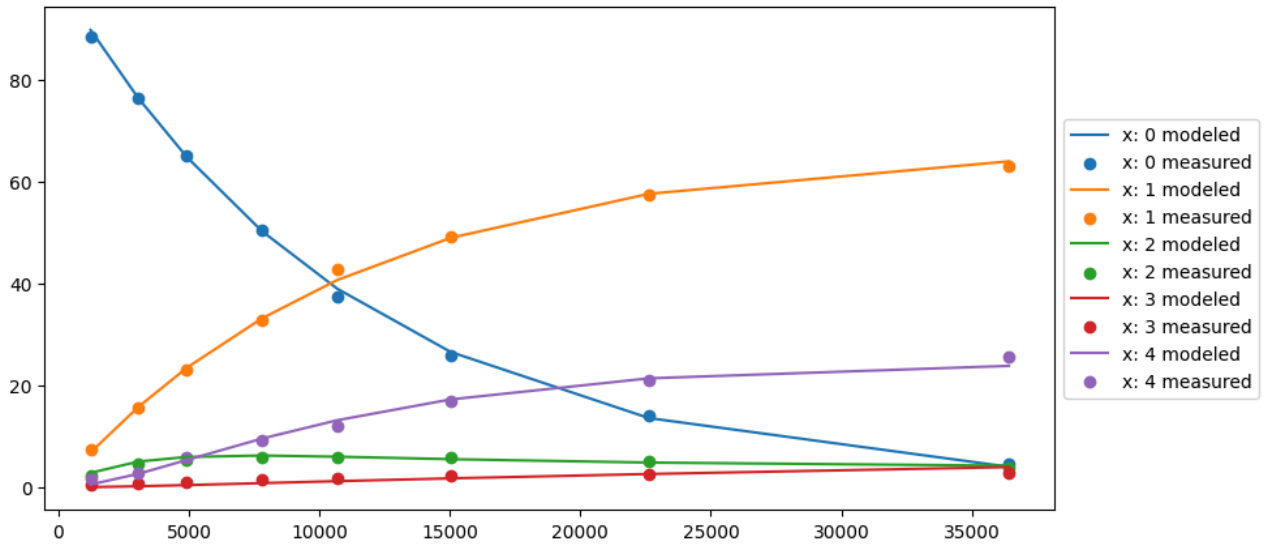
In [3]:
```
def concentrations(t: np.ndarray, x_0: np.ndarray, thet: np.ndarray) -> np.ndarray:
    out = np.zeros((5, t.shape[0]))
    for i, ti in enumerate(t.flatten()):
        model_map = x_dot(thet, ti)
        exp_m = scipy.linalg.expm(model_map)
        con_t = np.matmul(exp_m, x_0).flatten() # concentration at t
        out[:, i] = con_t
    return out
```

c. The files *iad_x.txt*, *iad_t.txt*, *iad_x0.txt* contain data, time points, and initial conditions respectively for this model. Formulate and solve a learning problem from this data using the function you wrote in part b. Report the numerical values for your parameters estimates and plot the curves $\hat{\boldsymbol{x}}(t)$ for your learned model.

In [4]:
```
iad_x = pd.read_csv("iad-x.txt", sep=r'\s+', header=None).values.T # data (8, 5) -> (5, 8)
iad_t = pd.read_csv("iad-t.txt", sep=r'\s+', header=None).values # time points (8, 1)
iad_x0 = pd.read_csv("iad-x0.txt", sep=r'\s+', header=None).values # initial conditions (5, 1)

def cost(thet, t, x0, y):
    y_hat = concentrations(t, x0, thet)
    c = 0
    for i in range(t.flatten().shape[0]):
        y_i = y[:, i]
        y_hat_i = y_hat[:, i]
        diff = y_i-y_hat_i
        c += np.linalg.norm(diff, ord=2)
    return c

thet_hat = minimize(cost, np.array([0.0001, 0.0001, 0.0001, 0.0001, 0.0001]), args=(iad_t, iad_x0, iad_x),
x_hat_t = concentrations(iad_t, iad_x0, thet_hat.x)

print(f"theta_hat: {thet_hat.x}")

fig, ax = plt.subplots(figsize=(10, 5))

for i in range(5):
    c_hat = x_hat_t[i]
    c_meas = iad_x[i]
    plt.plot(iad_t, c_hat, label=f"x: {i} modeled")
    plt.scatter(iad_t, c_meas, label=f"x: {i} measured")

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5)) # this line from ChatGPT
```

```
theta_hat: [5.87922721e-05 2.94517042e-05 2.18517358e-05 2.98084919e-04
 5.08000318e-05]
```

Out[4]: <matplotlib.legend.Legend at 0x118186cf0>



HW 27

In this problem you will practice generalizing the analysis of the FIM to nonlinear cases involving heteroscedastic error. The file `NonLinearPoints.txt` has as its first column the samples of the independent variable, the samples of the second column as the samples of the dependent variable, and the uncertainty in the measurements as the third column. Treat the third column as σ_i for each of your data points. Assume that your model is:

$$e^{-\theta_1 x}\sin(\theta_2 x)$$

and that your cost function is one half of the square of the l_2 norm (i.e. $\frac{1}{2}\sum_i (y_i - \hat{y}_i)^2$, where y_i is your data point and $\hat{y}_i$ is your prediction).

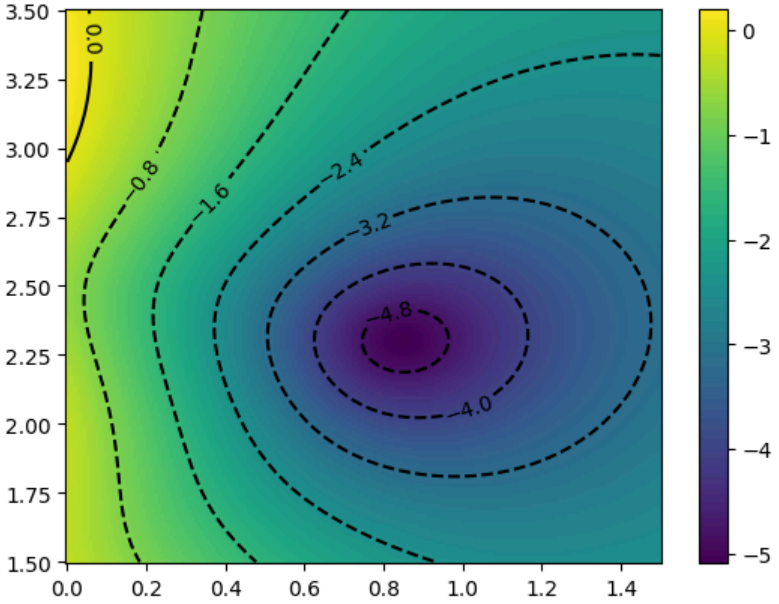
In [5]:
```
data = pd.read_csv("nonlinearpoints.txt", sep=r'\s+', header=None).values
x=jnp.array(data[:, 0])
y=jnp.array(data[:, 1])
u=jnp.array(data[:, 2])

def modl(x, th):
    return jnp.exp(-th[0]*x)*jnp.sin(th[1]*x)

def cost(x, y, th):
    y_hat = modl(x, th)
    l2 = jnp.linalg.norm((y-y_hat))
    c = (l2**2)/2
    return c
```

(a) Generate a contour plot depicting the cost surface of your model if you do not consider the heteroscedastic errors.

In [6]:
```
# adapting from HW 21a and 24b and for heatmap, etc.

th_1 = jnp.linspace(0, 1.5, 200)
th_2 = jnp.linspace(1.5, 3.5, 200)

m = np.zeros((3, 200, 200))
for i, t1 in enumerate(th_1):
    for j, t2 in enumerate(th_2):
        c = cost(x, y, jnp.array([t1, t2]))
        log_cost = np.log(c)
        m[:, i, j] = np.array([t1, t2, log_cost])
```

In [7]:
```
# plot
pcm = plt.pcolormesh(*m)
cont_set = plt.contour(*m, colors="black")
plt.clabel(cont_set)
plt.colorbar(pcm)
```

Out[7]:
```
<matplotlib.colorbar.Colorbar at 0x11835e660>
```



(b) Plot an ellipse defined using the eigenvalues and eigenvectors of the FIM evaluated at the minimum over the cost surface. Scale the ellipse to show that the local curvature of the cost surface around the minimum matches that defined by the FIM. You may use the following procedure:

- Compute the eigen-decomposition of the FIM:
- Assume the semi-axis lengths of the ellipse are given by

$$a_i = \sqrt{\frac{c}{\lambda_i}},$$

where λ_i are the eigenvalues of the FIM.

- Generate points on the unit circle
- Transform these circle points into ellipse coordinates using the following formula:

$$\boldsymbol{\theta}(t) = \hat{\boldsymbol{\theta}} + \mathbf{V} \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix} \mathbf{u}(t).$$

Where $\hat{\boldsymbol{\theta}}$ is the minimum of your cost surface, $\mathbf{V}$ is the matrix whose columns are the eigenvectors of the FIM, and $\mathbf{u}(t)$ are the points on the unit circle.

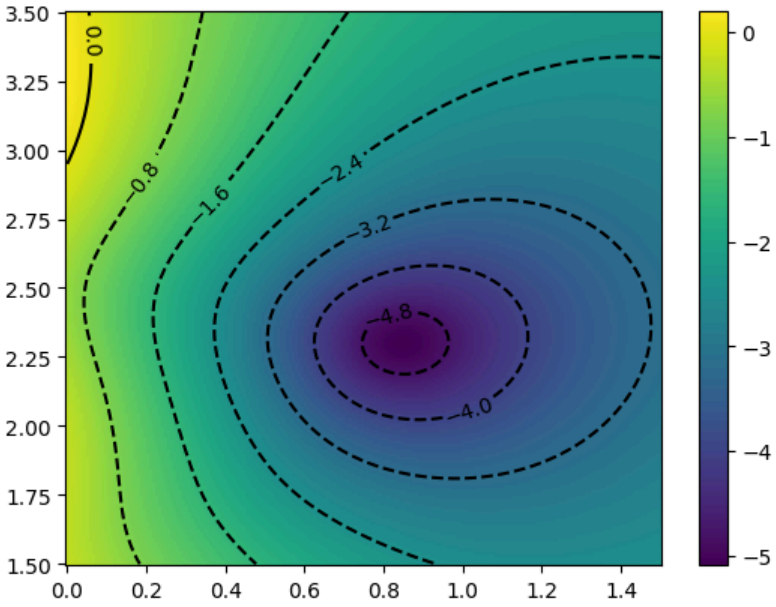
- Plot $\boldsymbol{\theta}(t)$.

In [8]:
```
# first find the min of the cost surface
minim = np.array([np.inf, np.inf, np.inf])
for i, t1 in enumerate(th_1):
    for j, t2 in enumerate(th_2):
        log_cost = m[2, i, j]
        if log_cost < minim[2]:
            minim = m[:, i, j]
min_th1, min_th2, min_c = minim

# SVD and eigen-decomposition of FIM (from HW 24)
J = jax.jacrev(modl, argnums=1)
ja = J(x, minim[0:2])
FIM = jnp.matmul(ja.T, ja)
eigvals, eigvecs = jnp.linalg.eig(FIM)

# axis lengths
axis_lengths = jnp.array([jnp.sqrt(min_c/lam) for lam in eigvals])

# unit circle and transform
unit_circle = np.array([[np.sin(x_rad), np.cos(x_rad)] for x_rad in np.linspace(0, 2*np.pi, 50)]) # 50x2 :
transformed_circle_points = (min_c + jnp.matmul(jnp.matmul(eigvecs, jnp.diag(axis_lengths)), unit_circle.T
```

```
# My eigenvalues were complex numbers, so downstream calculations didn't work
```

In [9]:
```
# plot
pcm = plt.pcolormesh(*m)
cont_set = plt.contour(*m, colors="black")
# plt.plot(transformed_circle_points[:, 0], transformed_circle_points[:, 1], color="red")
plt.clabel(cont_set)
plt.colorbar(pcm)
```

Out[9]:
```
<matplotlib.colorbar.Colorbar at 0x1193d7750>
```

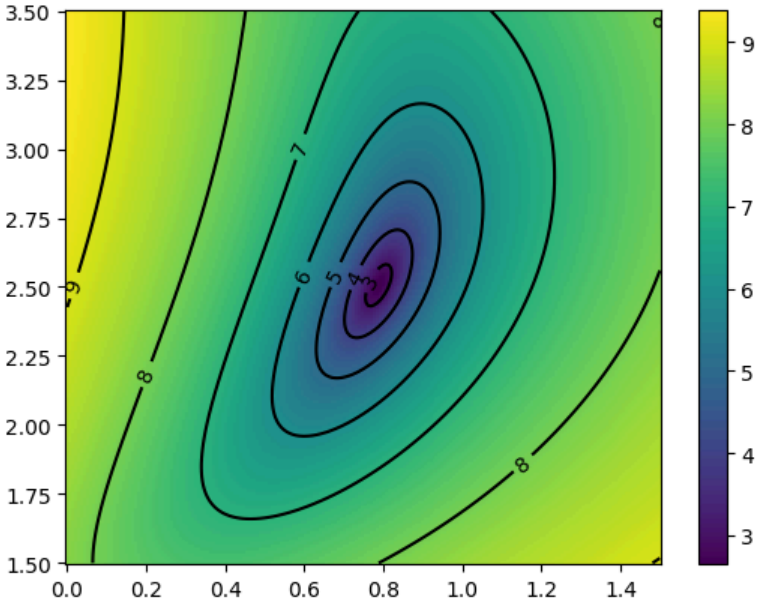


(c) Generate a second contour plot depicting the cost surface of your model if you **do** consider the heteroscedastic errors. In this case, your cost will be:

$$\frac{1}{2}\sum_i \frac{1}{\sigma_i^2}(y_i - \hat{y}_i)^2$$

In [10]:
```
def cost(x, y, th):
    y_hat = modl(x, th)
    l2 = jnp.linalg.norm((1/(u**2))*(y-y_hat))
    c = (l2**2)/2
    return c

# adapting from HW 21a and 24b and for heatmap, etc.

th_1 = jnp.linspace(0, 1.5, 200)
th_2 = jnp.linspace(1.5, 3.5, 200)

m = np.zeros((3, 200, 200))
for i, t1 in enumerate(th_1):
    for j, t2 in enumerate(th_2):
        c = cost(x, y, jnp.array([t1, t2]))
        log_cost = np.log(c)
        m[:, i, j] = np.array([t1, t2, log_cost])
# plot
pcm = plt.pcolormesh(*m)
cont_set = plt.contour(*m, colors="black")
plt.clabel(cont_set)
plt.colorbar(pcm)
```

Out[10]:
```
<matplotlib.colorbar.Colorbar at 0x119701950>
```



(d) Repeat part b using the weighted FIM: $J'SJ$, where S is the matrix whose diagonal elements are $\frac{1}{\sigma_i^2}$.

In [11]:
```
# first find the min of the cost surface
minim = np.array([np.inf, np.inf, np.inf])
for i, t1 in enumerate(th_1):
    for j, t2 in enumerate(th_2):
        log_cost = m[2, i, j]
        if log_cost < minim[2]:
            minim = m[:, i, j]
min_th1, min_th2, min_c = minim

# SVD and eigen-decomposition of FIM (from HW 24)
J = jax.jacrev(modl, argnums=1)
ja = J(x, minim[0:2])
S = jnp.diag(1/(u**2)) # we add this line
FIM = jnp.matmul(jnp.matmul(ja.T, S), ja) # and change this line
eigvals, eigvecs = jnp.linalg.eig(FIM)

# axis lengths
axis_lengths = jnp.array([jnp.sqrt(min_c/lam) for lam in eigvals])

# unit circle and transform
unit_circle = np.array([[np.sin(x_rad), np.cos(x_rad)] for x_rad in np.linspace(0, 2*np.pi, 50)]) # 50x2 :
transformed_circle_points = (min_c + jnp.matmul(jnp.matmul(eigvecs, jnp.diag(axis_lengths)), unit_circle.T

# My eigenvalues were complex numbers, so downstream calculations didn't work
```

(e) In your own words, how does the FIM's ability to describe cost basins generalize to the nonlinear case?

In [13]:
```
# The fim's ability to describe cost basisn generalizes to the nonlinear case by incorporating a relationsl
# of warp of the cost surface between parameters.
```

Acknowledgment

Work in this repository and with associated assignments and projects may be adapted or copied from similar files used in my prior academic and industry work (e.g., using a LaTeX file or Dockerfile as a starting point). Those files and any other work in this repository may have been developed with the help of LLM's like ChatGPT. For example, to provide context, answer questions, refine writing, understand function call syntax, and assist with repetitive tasks. In these cases, deliverables and

associated work reflect my best efforts to optimize my learning and demonstrate my capacity, while using available resources and LLM's to facilitate the process.

ChatGPT Conversation