CodingLab6_Root_Wendt_Weygoldt

June 13, 2023

Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2023

Names: Kathrin Root, Alexander Wendt, Patrick Weygoldt

1 Coding Lab 6

In this exercise we are going to fit a latent variable model (Poisson GPFA) to both toy data and real data from monkey primary visual cortex.

1.1 Preliminaries

1.1.1 1. Code

The toolbox we are going to use contains an implementation of the EM algorithm to fit the poisson-gpfa.

Assuming you `git clone https://github.com/mackelab/poisson-gpfa` to the notebooks/ directory and have the following directory structure:

```
data/
    nds_cl_6_data.mat
notebooks
    poisson-gpfa/
    CodingLab6.ipynb
matplotlib_style.txt
requirements.txt
```

then you can import the related functions via:

```
import sys
sys.path.append('./poisson-gpfa/')
sys.path.append('./poisson-gpfa/funs')

import funs.util as util
import funs.engine as engine
```

Change the paths if you have different directory structure. For the details of the algorithm, please refer to the thesis `hooram_thesis.pdf` from ILIAS.

1.1.2 2. Data

Download the data file `nds_cl_6_data.mat` from ILIAS and save it in a `data/` folder.

[1]:
```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt

# style
import seaborn as sns

# poisson-gpfa
import sys

sys.path.append("./poisson-gpfa/")
sys.path.append("./poisson-gpfa/funs")

import funs.util as util
import funs.engine as engine

%matplotlib inline

%load_ext jupyter_black

%load_ext watermark
%watermark --time --date --timezone --updated --python --iversions --watermark␣
  ↪-p sklearn
```

```
<IPython.core.display.HTML object>

Last updated: 2023-06-13 20:14:23CEST

Python implementation: CPython
Python version       : 3.11.3
IPython version      : 8.11.0

sklearn: 0.0.post1

matplotlib: 3.7.1
sys       : 3.11.3 (main, Apr  7 2023, 20:13:31) [Clang 14.0.0
(clang-1400.0.29.202)]
seaborn   : 0.12.2
scipy     : 1.10.1
numpy     : 1.24.3

Watermark: 2.3.1
```

[2]:
```
plt.style.use("../matplotlib_style.txt")
```

1.2 Task 1. Generate some toy data to test the poisson-GPFA code

We start by verifying our code on toy data. The cell below contains code to generate data for 30 neurons, 100 trials (1000 ms each) and 50ms bin size. The neurons' firing rate λ_k is assumed to be a constant d_k modulated by a one-dimensional latent state x, which is drawn from a Gaussian process:

$\lambda_k = \exp(c_k x + d_k)$

Each neuron's weight c_k is drawn randomly from a normal distribution and spike counts are sampled form a Poisson distribution with rate λ_k.

Your task is to fit a Poisson GPFA model with one latent variable to this data (see `engine.PPGPFAfit`).

Hint: You can use `util.dataset?`, `engine.PPGPFAfit?` or `util.initializeParams?` to find out more about the provided package.

Grading: 3 pts

[3]:
```
# ----------------------------------
# simulate a training set (0.5 pts)
# ---------------------------------
# Initialize random number generator
seed = 42
# Specify dataset & fitting parameters
neurons = 30
trials = 100
dt = 1000  # ms for each trial
bin_size = 50  # ms
latent_variabel = 1
trainigs_dataset = util.dataset(
    trialDur=dt,
    binSize=bin_size,
    numTrials=trials,
    ydim=neurons,
    xdim=latent_variabel,
    seed=seed,
)
```

```
+------------- Simulated Dataset Options -------------+
                                                    1 | Dimensionality of Latent
State
                                                   30 | Dimensionality of
Observed State (# neurons)
                                                 1000 | Duration of trials (ms):
                                                      50 | Size of bins (ms):
                                                     100 | Number of Trials
+-------------------------------------------------------+
Sampling trial 100 …
Average firing rate per neuron in this dataset: 3.470 Hz.
```

1.2.1 Fit the model

[4]:
```python
# ------------------------
# fit the model (0.5 pts)
# ------------------------

# Initialize parameters using Poisson-PCA
params = util.initializeParams(
    xdim=latent_variabel, ydim=neurons, experiment=trainigs_dataset
)

# choose sensible parameters and run fit
fitToy = engine.PPGPFAfit(trainigs_dataset, params, batchSize=100)
```

```
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options ---------------------+
                                                       1 | Dimensionality of Latent
State
                                                      30 | Dimensionality of
Observed State (# neurons)
                                                  Online | EM mode:
                                                      50 | Max EM iterations:
                                                 laplace | Inference Method
                                                  `diag` | Online Param Update
Method
                                                     100 | Batch size (trials):
+-------------------------------------------------------+
Iteration:  50 of  50, nPLL: = -262.0793
This dataset is a simulated dataset.
Processing performance against ground truth parameters…
```

[5]:
```python
# some useful functions
def allTrialsState(fit, p):
    """Reshape the latent signal and the spike counts"""
    x = np.zeros([p, 0])
    for i in range(len(fit.infRes["post_mean"])):
        x = np.concatenate((x, fit.infRes["post_mean"][i]), axis=1)
    return x


def allTrialsX(training_set):
    """Reshape the ground truth
    latent signal and the spike counts"""
    x_gt = np.array([])
    for i in range(len(training_set.data)):
        x_gt = np.concatenate((x_gt, training_set.data[i]["X"][0]), axis=0)
    return x_gt
```

1.2.2 Plot the ground truth vs. inferred model

Verify your fit by plotting both ground truth and inferred parameters for: 1. weights C 2. biases d 3. latent state x

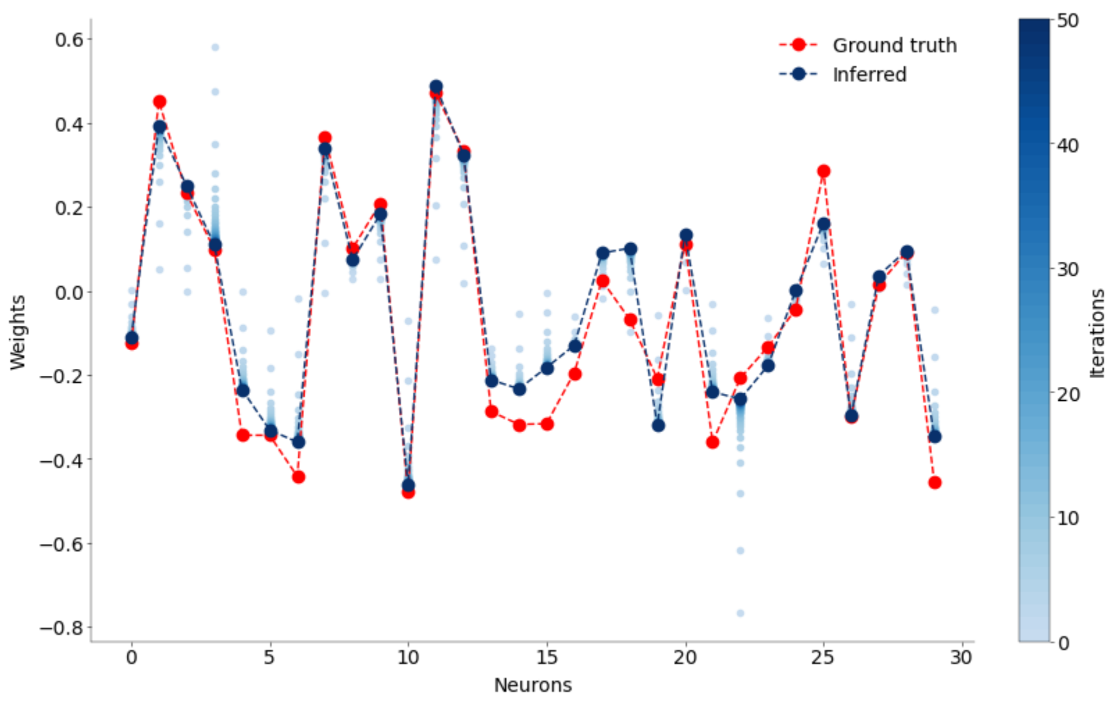
Note that the sign of fitted latent state and its weights are ambiguous (you can flip both without changing the model). Make sure you correct the sign for the plot if it does not match the ground truth.

[24]:
```
# ----------------------------------------------------
# Plot ground truth and inferred weights `C` (0.5 pts)
# ----------------------------------------------------

fig, ax = plt.subplots(figsize=(8, 5))
label = "Inferred over iterations"
# colormap of the inffered weights
colors = plt.cm.Blues(np.linspace(0.25, 1, len(fitToy.paramSeq[:-1])))

for iter in range(len(fitToy.paramSeq[:-1])):
    ax.scatter(
        np.arange(0, neurons),
        -fitToy.paramSeq[iter]["C"],
        color=colors[iter],
    )

ax.plot(
    np.arange(0, neurons),
    trainigs_dataset.params["C"],
    color="red",
    marker="o",
    linestyle="dashed",
    label="Ground truth",
)

ax.plot(
    np.arange(0, neurons),
    -fitToy.paramSeq[-1]["C"],
    color=colors[-1],
    marker="o",
    linestyle="dashed",
    label="Inferred",
)
```

```
# plot the colormap of the inferred weights
# change the arange to the number of iterations

ticks = np.arange(0, len(fitToy.paramSeq[:-1]) + 1, 10)


import matplotlib.colors as mcolors

cmap = mcolors.ListedColormap(colors)

# Create the colorbar
colorbar = plt.colorbar(
    plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=0,␣
 ↪vmax=len(colors))),
    ticks=np.arange(0, len(fitToy.paramSeq[:-1]) + 1, 10),
    label="Iterations",
)

colorbar.set_ticks(ticks)
ax.set_xlabel("Neurons")
ax.set_ylabel("Weights")
ax.legend(loc="upper right")


# add plot
# consider also plotting the optimal weights as a dotted line for reference
```

```
/var/folders/6f/4s63gb612m185fbsprljhfmh0000gn/T/ipykernel_11690/1407338814.py:4
5: MatplotlibDeprecationWarning: Unable to determine Axes to steal space for
Colorbar. Using gca(), but will raise in the future. Either provide the *cax*
argument to use as the Axes for the Colorbar, provide the *ax* argument to steal
space from it, or add *mappable* to an Axes.
  colorbar = plt.colorbar(
```

```
[24]: <matplotlib.legend.Legend at 0x167e52250>
```

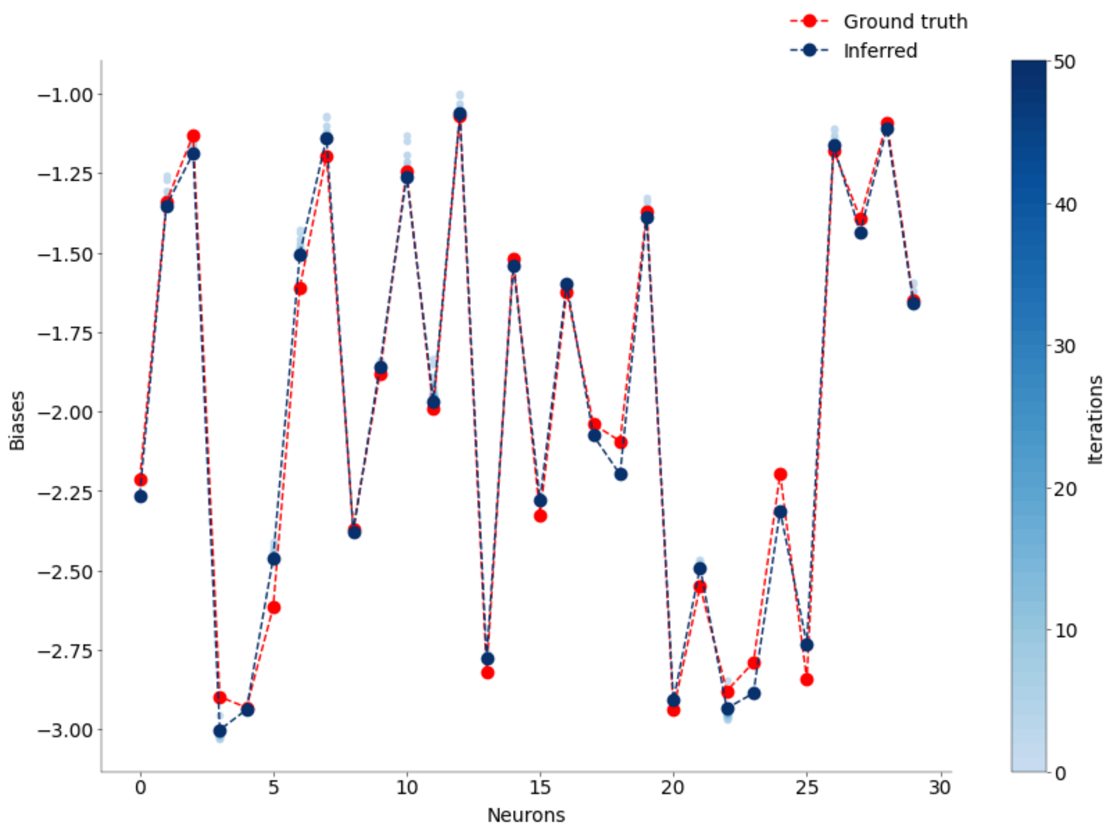


[23]:
```python
# ---------------------------------------------------
# Plot ground truth and inferred baises `d` (0.5 pts)
# ---------------------------------------------------

fig, ax = plt.subplots(figsize=(8, 6))
label = "Inferred over iterations"
# colormap of the inffered weights
colors = plt.cm.Blues(np.linspace(0.25, 1, len(fitToy.paramSeq[:-1])))

for iter in range(len(fitToy.paramSeq[:-1])):
    ax.scatter(
        np.arange(0, neurons),
        fitToy.paramSeq[iter]["d"],
        color=colors[iter],
    )

ax.plot(
    np.arange(0, neurons),
    trainigs_dataset.params["d"],
    color="red",
    marker="o",
    linestyle="dashed",
    label="Ground truth",
)
```

```
ax.plot(
    np.arange(0, neurons),
    fitToy.paramSeq[-1]["d"],
    color=colors[-1],
    marker="o",
    linestyle="dashed",
    label="Inferred",
)
# plot the colormap of the inferred weights
# change the arange to the number of iterations

ticks = np.arange(0, len(fitToy.paramSeq[:-1]) + 1, 10)

cmap = mcolors.ListedColormap(colors)

# Create the colorbar
colorbar = plt.colorbar(
    plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=0,␣
 ↪vmax=len(colors))),
    ticks=np.arange(0, len(fitToy.paramSeq[:-1]) + 1, 10),
    label="Iterations",
)

colorbar.set_ticks(ticks)
ax.set_xlabel("Neurons")
ax.set_ylabel("Biases")
ax.legend(loc=(0.8, 0.99))


# add plot
# consider also plotting the optimal weights as a dotted line for reference
```

```
/var/folders/6f/4s63gb612m185fbsprljhfmh0000gn/T/ipykernel_11690/2425399127.py:4
2: MatplotlibDeprecationWarning: Unable to determine Axes to steal space for
Colorbar. Using gca(), but will raise in the future. Either provide the *cax*
argument to use as the Axes for the Colorbar, provide the *ax* argument to steal
space from it, or add *mappable* to an Axes.
  colorbar = plt.colorbar(
```

```
[23]: <matplotlib.legend.Legend at 0x1679fa250>
```

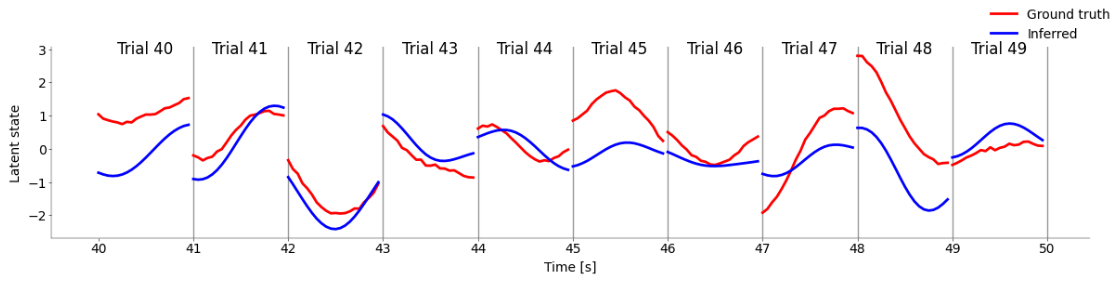


[21]:
```
# --------------------------------------------------------
# Plot ground truth and inferred latent states `x` (1pt)
# --------------------------------------------------------
# pick a few trials to plot
first_trial = 40
last_trial = 50



dt = 1000  # ms for each trial
bin_size = 50  # ms
plot_trials = np.arange(first_trial, last_trial)

fig, ax = plt.subplots(figsize=(12, 3))
padding_for_text = 0.3

for i in plot_trials:
    time_trial = np.arange(i * dt, (i + 1) * dt, bin_size) / 1000
    # because the sign of the latend states are arbitrary we need to flip them␣
 ↪for trial 48
    # and 40
    if i == 48:
        ax.plot(
            time_trial,
            trainigs_dataset.data[i]["X"][0],
            color="red",
            linewidth=2,
            label="Ground truth",
        )
        ax.plot(
            time_trial,
            fitToy.infRes["post_mean"][i][0],
            color="blue",
            linewidth=2,
            label="Inferred",
        )
        ax.axvline(time_trial[-1] + 0.05, ymin=-2, ymax=3, color="gray",␣
↪alpha=0.8)
        # trial number in on the top of the plot
        ax.text(
            time_trial[int(len(time_trial) / 2)] - padding_for_text,
            3,
            f"Trial {i}",
            verticalalignment="center",
            fontsize=12,
            label="Inferred",
        )

    elif i == 40:
        ax.plot(
            time_trial,
            trainigs_dataset.data[i]["X"][0],
            color="red",
            linewidth=2,
            label="Ground truth",
        )
        ax.plot(
            time_trial,
            fitToy.infRes["post_mean"][i][0],
            color="blue",
            linewidth=2,
            label="Inferred",
        )
        ax.axvline(time_trial[-1] + 0.05, ymin=-2, ymax=3, color="gray",␣
↪alpha=0.8)
        # trial number in on the top of the plot
        ax.text(
            time_trial[int(len(time_trial) / 2)] - padding_for_text,
            3,
            f"Trial {i}",
            verticalalignment="center",
            fontsize=12,
            label="Inferred",
        )

    else:
        ax.plot(
            time_trial,
            trainigs_dataset.data[i]["X"][0],
            color="red",
            linewidth=2,
            label="Ground truth",
        )
        ax.plot(
            time_trial,
            -fitToy.infRes["post_mean"][i][0],
            color="blue",
            linewidth=2,
            label="Inferred",
        )
        ax.axvline(time_trial[-1] + 0.05, ymin=-2, ymax=3, color="gray",␣
↪alpha=0.8)
        # trial number in on the top of the plot
        ax.text(
            time_trial[int(len(time_trial) / 2)] - padding_for_text,
            3,
            f"Trial {i}",
            verticalalignment="center",
            fontsize=12,
            label="Inferred",
        )

    if i == plot_trials[0]:
        ax.legend(loc=(0.9, 1.01))


ax.set_xlabel("Time [s]")
ax.set_ylabel("Latent state")
ax.set_xticks(np.arange(first_trial * dt, (last_trial) * dt + 1, 1000) / 1000)

# add plot
# plot only for a subset of trials
# consider seperating each trial by a vertical line
```

```
/var/folders/6f/4s63gb612m185fbsprljhfmh0000gn/T/ipykernel_11690/2744496561.py:9
```

```
9: UserWarning: Legend does not support handles for Text instances.
See: https://matplotlib.org/stable/tutorials/intermediate/legend_guide.html#impl
ementing-a-custom-legend-handler
  ax.legend(loc=(0.9, 1.01))
```

[21]:
```
[<matplotlib.axis.XTick at 0x1678b03d0>,
 <matplotlib.axis.XTick at 0x1678b3150>,
 <matplotlib.axis.XTick at 0x165b26750>,
 <matplotlib.axis.XTick at 0x16783b310>,
 <matplotlib.axis.XTick at 0x167824bd0>,
 <matplotlib.axis.XTick at 0x167857450>,
 <matplotlib.axis.XTick at 0x167865110>,
 <matplotlib.axis.XTick at 0x167838690>,
 <matplotlib.axis.XTick at 0x167866890>,
 <matplotlib.axis.XTick at 0x16786f710>,
 <matplotlib.axis.XTick at 0x167879b10>]
```



1.3 Task 2: Fit GPFA model to real data.

We now fit the model to real data and cross-validate ov

1.3.1 Load data

The cell below implements loading the data and encapsulates it into a class that matches the interface of the Poisson GPFA engine. You don't need to do anything here.

[9]:
```python
class EckerDataset:
    """Loosy class"""

    def __init__(
        self,
        path,
        subject_id=0,
        ydim=55,
        trialDur=2000,
        binSize=100,
        numTrials=100,
        ydimData=False,
        numTrData=True,
    ):
        T = int(trialDur / binSize)
        matdat = sio.loadmat(path)
        self.matdat = matdat
        data = []
        trial_durs = []
        for trial_id in range(numTrials):
            trial_time = matdat["spikeTimes"][:, trial_id][0]
            trial_big_time = np.min(trial_time)
            trial_end_time = np.max(trial_time)
            trial_durs.append(trial_end_time - trial_big_time)
        for trial_id in range(numTrials):
            Y = []
            spike_time = []
            data.append(
                {
                    "Y": matdat["spikeCounts"][:, :, trial_id],
                    "spike_time": matdat["spikeTimes"][:, trial_id],
                }
            )
        self.T = T
        self.trial_durs = trial_durs
        self.data = data
        self.trialDur = trialDur
        self.binSize = binSize
        self.numTrials = numTrials
        self.ydim = ydim
        util.dataset.getMeanAndVariance(self)
        util.dataset.getAvgFiringRate(self)
        util.dataset.getAllRaster(self)
```

[10]:
```
path = "../data/nds_cl_6_data.mat"
data = EckerDataset(path)
```

1.3.2 Fit Poisson GPFA models and perform model comparison

Split the data into 80 trials used for training and 20 trials held out for performing model comparison. On the training set, fit models using one to five latent variables. Compute the performance of each model on the held-out test set.

Hint: You can use the `crossValidation` function in the Poisson GPFA package.

Optional: The `crossValidation` function computes the mean-squared error on the test set, which is not ideal. The predictive log-likelihood under the Poisson model would be a better measure, which you are welcome to compute instead.

1.3.3 Derivation for log-likelihood

You can add your calculations in LaTeX here.

$p_\lambda(x_t) = ...$

$L(\lambda_k; x_1, ..., x_N) = ...$

$log(L) = l(\lambda_k; x_1, ..., x_N) = ...$

[11]:
```
# ------------------------------
# Perfom cross validation (1 pt)
# ------------------------------

# fit the model to the data
xdim = 1  # number of modulators
initParams = util.initializeParams(xdim, data.ydim, data)
fitBatch = engine.PPGPFAfit(data, initParams, batchSize=data.numTrials)
```

```
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options --------------------+
                                                    1 | Dimensionality of Latent
State
                                                   55 | Dimensionality of
Observed State (# neurons)
                                               Online | EM mode:
                                                   50 | Max EM iterations:
                                              laplace | Inference Method
                                               `diag` | Online Param Update
Method
                                                  100 | Batch size (trials):
+-----------------------------------------------------+
Iteration:  50 of  50, nPLL: = -382.1540
```

[12]:
```
# do the actual cross validation
maxXdim = 5
xval = util.crossValidation(
    data,
    numTrainingTrials=80,
    numTestTrials=20,
    maxXdim=maxXdim,
    learningMethod="diag",
    batchSize=80,
)
```

```
Assessing optimal latent dimensionality will take a long time.
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options --------------------+
                                                    1 | Dimensionality of Latent
State
```

```
                                                 55 | Dimensionality of
Observed State (# neurons)
                                             Online | EM mode:
                                                  3 | Max EM iterations:
                                            laplace | Inference Method
                                             `diag` | Online Param Update
Method
                                                 80 | Batch size (trials):
+--------------------------------------------------------+
Iteration:   3 of   3, nPLL: = -401.5516Performing leave-one-out cross
validation…
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options ---------------------+
                                                  2 | Dimensionality of Latent
State
                                                 55 | Dimensionality of
Observed State (# neurons)
                                             Online | EM mode:
                                                  3 | Max EM iterations:
                                            laplace | Inference Method
                                             `diag` | Online Param Update
Method
                                                 80 | Batch size (trials):
+--------------------------------------------------------+
Iteration:   3 of   3, nPLL: = -377.7591Performing leave-one-out cross
validation…
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options ---------------------+
                                                  3 | Dimensionality of Latent
State
                                                 55 | Dimensionality of
Observed State (# neurons)
                                             Online | EM mode:
                                                  3 | Max EM iterations:
                                            laplace | Inference Method
                                             `diag` | Online Param Update
Method
                                                 80 | Batch size (trials):
+--------------------------------------------------------+
Iteration:   3 of   3, nPLL: = -373.1875Performing leave-one-out cross
validation…
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options ---------------------+
                                                  4 | Dimensionality of Latent
State
                                                 55 | Dimensionality of
Observed State (# neurons)
                                             Online | EM mode:
```

```
                                                              3 | Max EM iterations:
                                                        laplace | Inference Method
                                                         `diag` | Online Param Update
Method
                                                             80 | Batch size (trials):
+-----------------------------------------------------+
Iteration:   3 of   3, nPLL: = -367.1550Performing leave-one-out cross
validation…
Initializing parameters with Poisson-PCA..
+-------------------- Fit Options --------------------+
                                                              5 | Dimensionality of Latent
State
                                                             55 | Dimensionality of
Observed State (# neurons)
                                                         Online | EM mode:
                                                              3 | Max EM iterations:
                                                        laplace | Inference Method
                                                         `diag` | Online Param Update
Method
                                                             80 | Batch size (trials):
+-----------------------------------------------------+
Iteration:   3 of   3, nPLL: = -357.6321Performing leave-one-out cross
validation…
```

1.3.4 Plot the test error

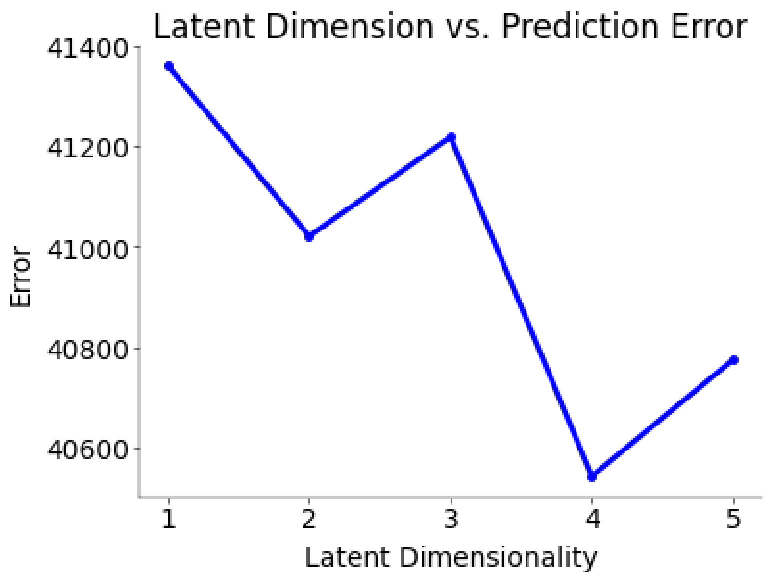
Make a plot of the test error for the five different models. As a baseline, please also include the test error of a model without a latent variable. This is essentially the mean-squared error of a constant rate model (or Poisson likelihood if you did the optional part above).

[13]:
```
#␣
 ↪------------------------------------------------------------------------------------------
# Compute and plot the test errors for the different latent variable models (0.
 ↪5 + 0.5 pts)
#␣
 ↪------------------------------------------------------------------------------------------

train_set, test_set = util.splitTrainingTestDataset(
    data, numTrainingTrials=80, numTestTrials=20
)

# compute baseline error
baseline_error = np.mean(test_set.all_raster - (np.mean(test_set.avgFR,␣
 ↪axis=0)) ** 2)
print(baseline_error)
```

```
-107.26683209917358
```

[14]:
```python
# Your plot here
fig, ax = plt.subplots(figsize=(4, 3))

# plot model error
plt.plot(np.arange(1, maxXdim + 1), xval.errs, "b.-", markersize=5, linewidth=2)
# plt.legend([xv.method],fontsize=9,framealpha=0.2)
plt.xlabel("Latent Dimensionality")
plt.ylabel("Error")
plt.title("Latent Dimension vs. Prediction Error")

# plot baseline
# ax.axhline(baseline_error, linestyle="--")
```

[14]:
```
Text(0.5, 1.0, 'Latent Dimension vs. Prediction Error')
```



1.4 Task 3. Visualization: population rasters and latent state. Use the model with a single latent state.

Create a raster plot where you show for each trial the spikes of all neurons as well as the trajectory of the latent state `x` (take care of the correct time axis). Sort the neurons by their weights `c_k`.

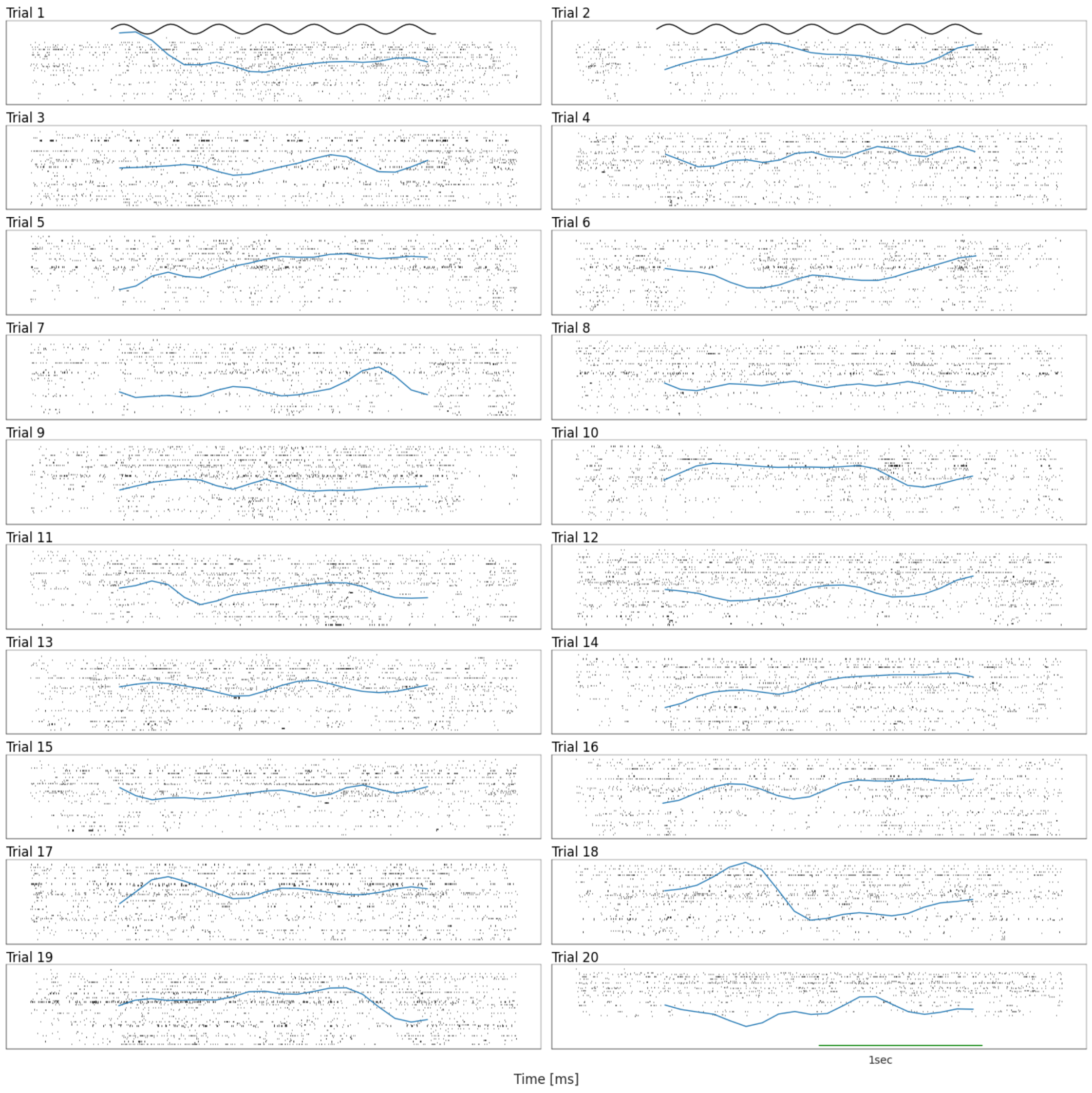
Plot only the first 20 trials.

Grading: 2 pts

[15]:
```python
from numpy import matlib

trialduration = 2000
binSize = 100
```

```
# Your plot here
fig, axs = plt.subplots(10, 2, figsize=(14, 14))
ts = np.linspace(0, trialduration, binSize)

#### stimulus
# amplitude
xa = 0.15
# sinusoidal stimulus
xs = 0.7 * xa * np.sin(ts / 1000 * 3.4 * 2 * np.pi) + xa

# sorted 'c'
index_sorted_c = np.argsort(fitBatch.paramSeq[-1]["C"][:, 0])


def squeezing(trial_data):
    return [np.squeeze(trial_data[i]) for i in range(len(trial_data))]


with sns.axes_style("ticks"):
    for ntrial, ax in enumerate(axs.flat):
        x = range(50, 2000, 100)  # assume binsize of 100ms

        # ------------------------
        # plot latent state (1 pt)
        # ------------------------
        # plot latent state on different y axis
        ax2 = ax.twinx()
        ax2.plot(x, fitBatch.infRes["post_mean"][ntrial][0])
        ax2.set_yticks(np.arange(-5, 5))
        ax2.set_yticklabels([])
        ax2.set_yticks([])

        # hint: can be plotted on top of the corresponding raster

        # sort neurons by weights
        neurons_trial = data.data[ntrial]["spike_time"][index_sorted_c]
        neruons_trial_squeezed = squeezing(neurons_trial)
        # ----------------------------------
        # plot raster for each neuron (1 pt)
        # ----------------------------------
        for neuron in range(len(neruons_trial_squeezed)):
            if neurons_trial[neuron].size == 1:
                continue
            if neurons_trial[neuron].size == 0:
                continue

            else:
                ax.eventplot(
                    neruons_trial_squeezed[neuron],
                    lineoffsets=neuron,
                    linelengths=0.6,
                    linewidths=0.6,
                    color="black",
                )
        ax.set_title("Trial " + str(ntrial + 1), loc="left")

        if ntrial == 0:
            ax.legend()
        if ntrial == 19:
            ax.plot([1000, 2000], [-30, -30], color="green")
            ax.text(1300, -50, "1sec")
        if ntrial < 2:
            ax.plot(ts, (xs * 40) + data.ydim, "k", color="black")

        ax.set_yticks([])
        ax.set_xticks([])
        fig.supxlabel("Time [ms]")
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note
that artists whose label start with an underscore are ignored when legend() is
called with no argument.
/var/folders/6f/4s63gb612m185fbsprljhfmh0000gn/T/ipykernel_11690/309882780.py:67
: UserWarning: color is redundantly defined by the 'color' keyword argument and
the fmt string "k" (-> color=(0.0, 0.0, 0.0, 1)). The keyword argument will take
precedence.
  ax.plot(ts, (xs * 40) + data.ydim, "k", color="black")
```



1.5 Task 4. Visualization of covariance matrix.

Plot (a) the covariance matrix of the observed data as well as its approximation using (b) one and (c) five latent variable(s). Use the analytical solution for the covariance matrix of the approximation*. Note that the solution is essentially the mean and covariance of the log-normal distribution.

$$\mu = \exp(\frac{1}{2}\ \mathrm{diag}(CC^T) + d)$$

$$\mathrm{Cov} = \mu\mu^T \odot \exp(CC^T) +\ \mathrm{diag}(\mu) - \mu\mu^T$$

*Krumin, M., and Shoham, S. (2009). Generation of Spike Trains with Controlled Auto- and Cross-Correlation Functions. Neural Computation 21, 1642–1664.

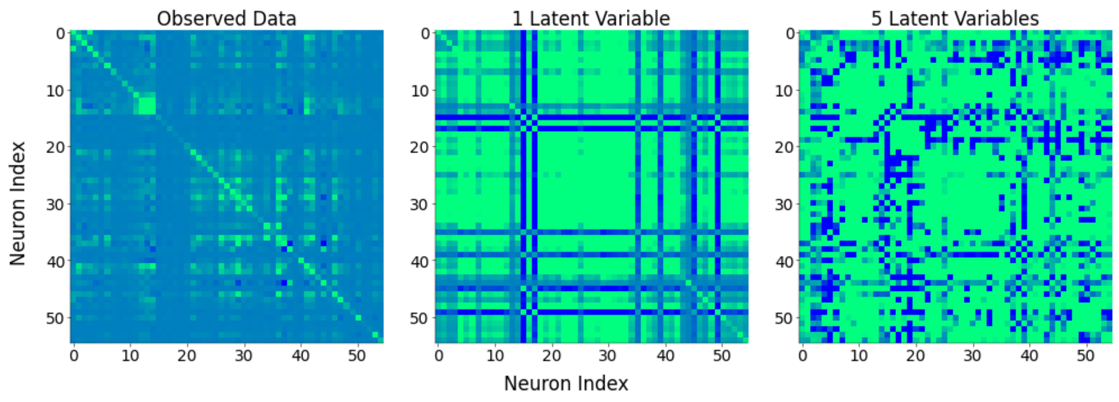
Grading: 3 pts

[20]:
```
# insert your code here

# --------------------------------------------------------------
# Complete the analytical solution for the covariance matrix of
# the approximation using the provide equations (2 pts)
# --------------------------------------------------------------


def cov(fit):
    # add your code here
    mu = np.exp(
        (
            np.diag(fit.paramSeq[-1]["C"] @ fit.paramSeq[-1]["C"].T)
            + fit.paramSeq[-1]["d"]
        )
        / 2
    )
    c = (
        (mu @ mu.T) * np.exp(fit.paramSeq[-1]["C"] @ fit.paramSeq[-1]["C"].T)
        + np.diag(mu)
        - (mu @ mu.T)
    )
    return c, mu


print(np.shape(xval.fits[0].paramSeq[-1]["C"]))

# --------------------------------------------------------------
# Plot the covariance matrix (1 pt) of
# (1) the observed data
# (2) its approximation using 1 latent variable
# (3) its approximation using 5 latent variable
# --------------------------------------------------------------

obs_corr = np.cov(data.all_raster)
opt_r1, mu1 = cov(xval.fits[0])
opt_r5, mu5 = cov(xval.fits[4])

vmin = -1
vmax = 1

fig, axs = plt.subplots(1, 3, figsize=(10, 3.5))
# add plot to visualize the differences in the covariance matrices

axs[0].imshow(obs_corr, vmin=vmin, vmax=vmax, cmap="winter")
```

```
axs[0].set_title("Observed Data")
axs[1].imshow(opt_r1, vmin=vmin, vmax=vmax, cmap="winter")

axs[1].set_title("1 Latent Variable")
axs[2].imshow(opt_r5, vmin=vmin, vmax=vmax, cmap="winter")

axs[2].set_title("5 Latent Variables")
fig.supxlabel("Neuron Index")
fig.supylabel("Neuron Index")
```

```
(55, 1)
```

[20]:
```
Text(0.02, 0.5, 'Neuron Index')
```


Observed Data
1 Latent Variable
5 Latent Variables
Neuron Index
Neuron Index