Diagnosis of mechanical rotor faults in drones using functional gaussian mixture classifier

B. Bartoszewski, K. Jarzyna

Necessary imports

In [ ]:
```
import numpy as np
from numpy.random import normal
import matplotlib.pyplot as plt
import matplotlib as mpl
from cmdstanpy import CmdStanModel
import arviz as az
import pandas as pd
import scipy.stats as stats
from PIL import Image
from IPython.display import display

from DA_tools.DA_tools import ribbon_plot
from DA_tools.FDA_data_prepare import create_spline_matrix
from DA_tools.FDA_prepare_model import prepare_data, get_results
from DA_tools.DA_contraction_z_score import get_z_contr
from DA_tools.DA_colors import *

plt.style.context("seaborn-white")
mpl.rcParams["figure.dpi"] = 200
acc_healthy = pd.read_csv('data_preprocesed/acc_healthy_samples.csv')
acc_healthy_data = np.array([acc_healthy[col].values for col in acc_healt

def disp_image(path,size=4,rotate=True):
    img = Image.open(path).convert("RGB")
    if rotate:
        img = img.rotate(-90,expand=True)
    display(img.resize([int(x/size) for x in img.size]))
```

Problem formulation [0-5 pts]:

- is the problem clearly stated [1 pt]
- what is the point of creating model, are potential use cases defined [1 pt]

- where do data comes from, what does it containt [1 pt]
- DAG has been drawn [1 pt]
- confoundings (pipe, fork, collider) were described [1 pt]

Detection of Drone Propeller Damage

A program has been developed for detecting damage to drone propellers, based on data from the accelerometer and gyroscope. The system analyzes changes in the drone's vibrations and movements, allowing for early detection of anomalies indicating mechanical damage to the propellers. This enables quick corrective actions, minimizing the risk of further damage, and improving the safety and reliability of drone operations.

Potential use

By analyzing vibration patterns, the system can quickly identify anomalies that may suggest damage or wear of components. When unusual vibrations are detected, the system immediately alerts the operator, enabling prompt actions such as controlled landing for inspection and potential replacement of damaged parts.
Using this program can reduce the risk of multirotor flights and decrease the operational cost of drones by mitigating the potential destruction of the entire drone or surrounding infrastructure.

Collecting Data

The data was collected by our own. To gather data, flights were conducted using an eight-rotor drone. Initially, flights were carried out with all propellers in good condition, followed by series of flights with one damaged propeller measuring 1.5 cm in length. Subsequently, the damaged propeller was replaced with one measuring 3 cm in length, and the series of tests continued.

A healthy drone used for collecting data

In [ ]:
```
disp_image('photos/IMG_3984.JPEG')
```

One propeller cut 1.5 cm

In [ ]:
```
disp_image('photos/1_5cm.jpg',size=1,rotate=False)
```

One propeller cut 3 cm

In [ ]:
```
disp_image('photos/3cm.jpg',size=1,rotate=False)
```

The system used in the drone is Ardupilot. It is an open-source autopilot designed for controlling, among other things, unmanned aerial vehicles.

One of the important features of ArduPilot is the ability to record flight data onto an external SD card. The capability of collecting flight logs was utilized, where the

appropriate logging parameters needed to be selected before the flight to gather data from the flights.

Set logging parameters

DAG

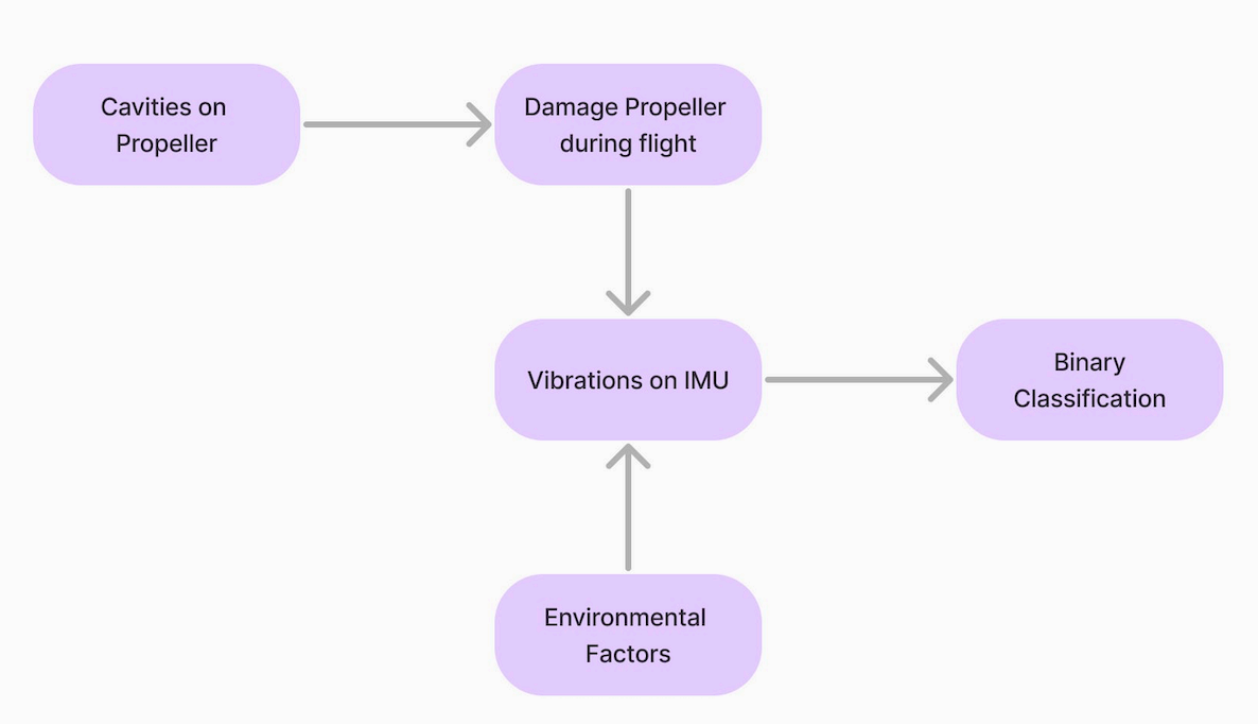
Several dependencies are represented in the model, which will influence the created model:

- Cavities on Propeller
- Damage Propeller during flight
- Vibrations on IMU
- Environmental Factors

We assume that due to damage to the carbon fiber on the propeller, the propeller may tear during flight, especially during the ascent phase where the forces acting on the drone are the greatest. Due to the uneven distribution of forces on the propeller during engine operation, vibrations will occur, causing one arm to oscillate, which can also be observed on the IMU. The second factor that directly affects the vibrations observable on the IMU are environmental factors such as wind and frame vibrations. The output of our model will be a binary classification indicating whether the model is damaged or not.
Output:

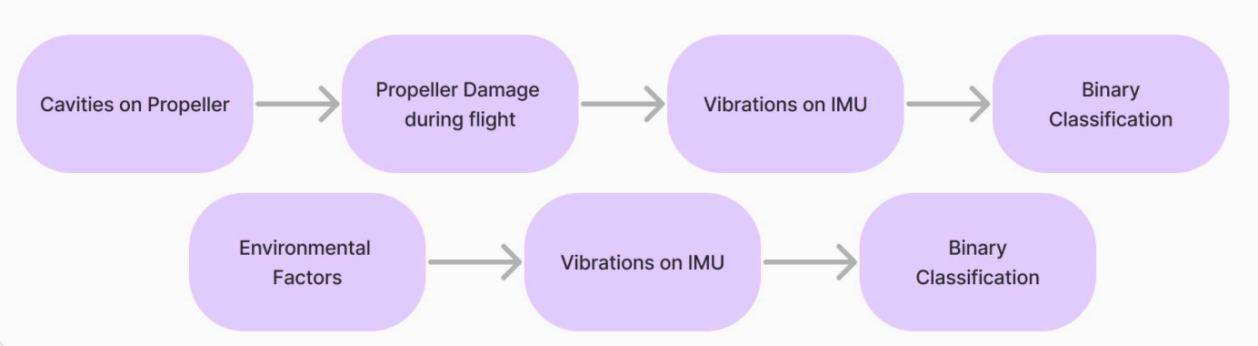
- Binary Classification

In [ ]:
```
disp_image('photos/DAG.jpg',size=1,rotate=False)
```



Pipes from DAG

In [ ]:
```
disp_image('photos/pipes_dag.jpg',size=1,rotate=False)
```



Data preprocessing [0-2 pts]:

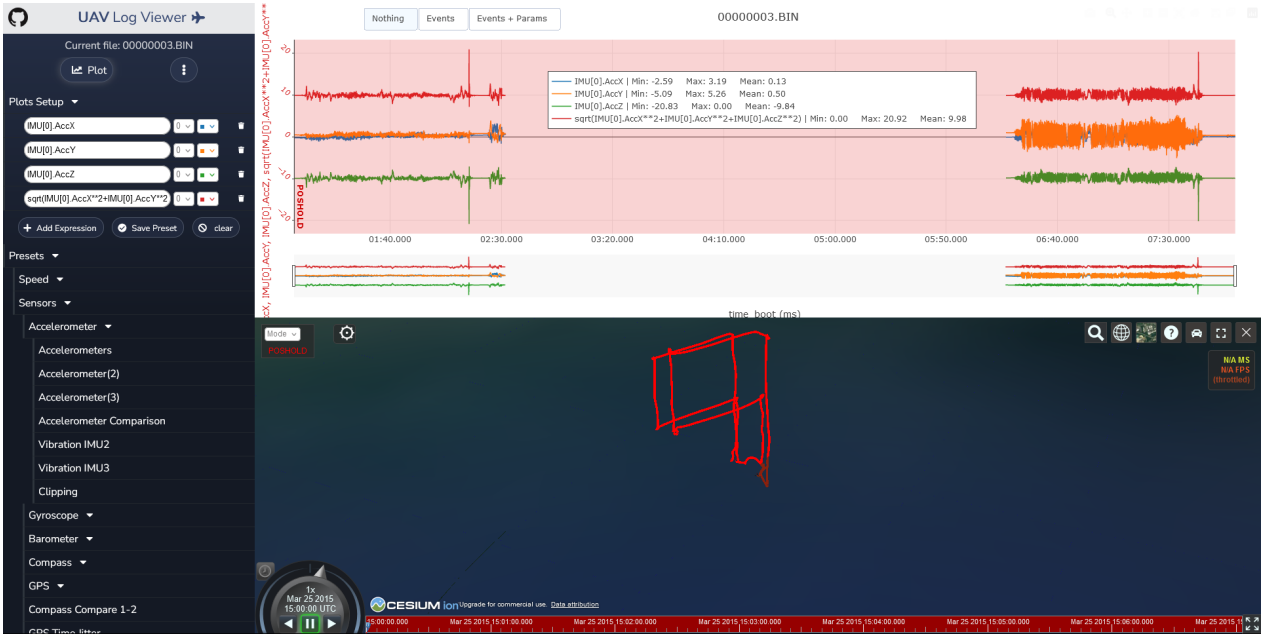
- is preprocessing step clearly described [1 pt]
- reasoning and types of actions taken on the dataset have been described [1 pt]

Data preprocessing

After collecting data, files from the `logs_from_flights` folder were replayed in the UAV Log Viewer program to export data from the gyroscope and accelerometer. This program was used because it allowed for a clear visualization of the drone's flight trajectory and ensured that the exported data corresponded to the appropriate flights with the relevant damages.

Data visualization in UAV Log Viewer

In [ ]:
```
disp_image("photos/uav_logs.png",size=1,rotate=False)
```



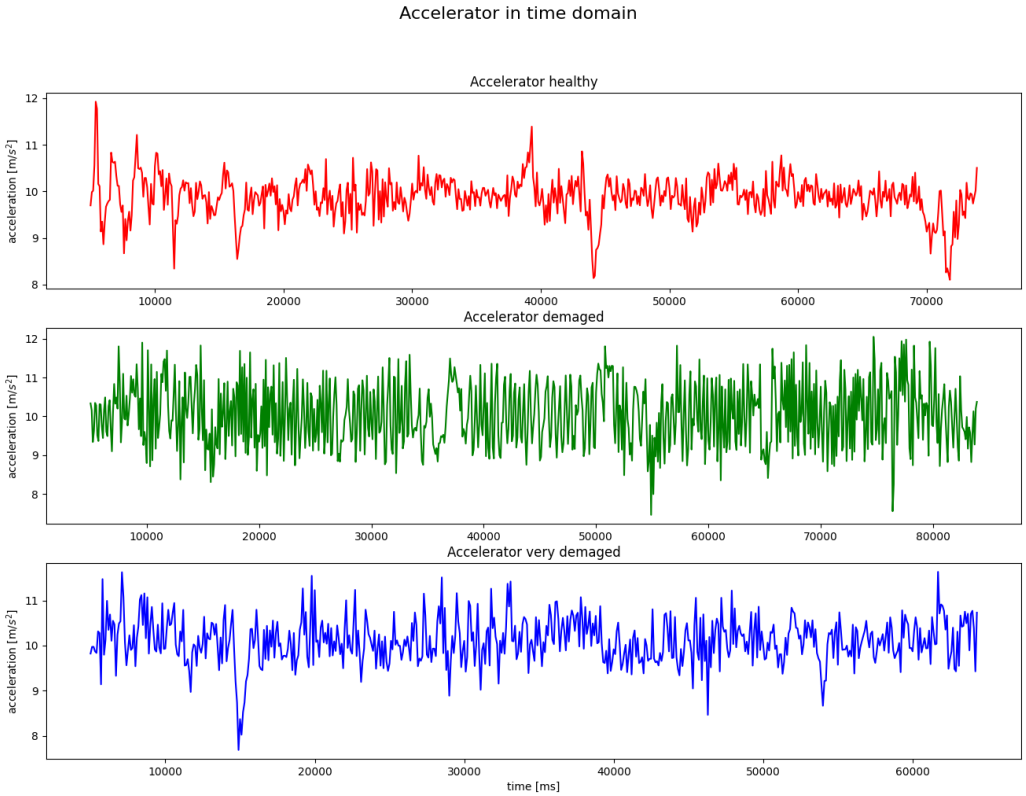
In file preprocessing.ipynb was made preprocessing from data whitch was generated in UAV Log Viewer. We saved thee types of data:

- accelerometer data in the time domain
- gyroscope data in the frequency domain
- gyroscope data in the frequency domain with accumulation

At the beginning, Euclidean normalization of the data was performed to reduce the dimensions to one.

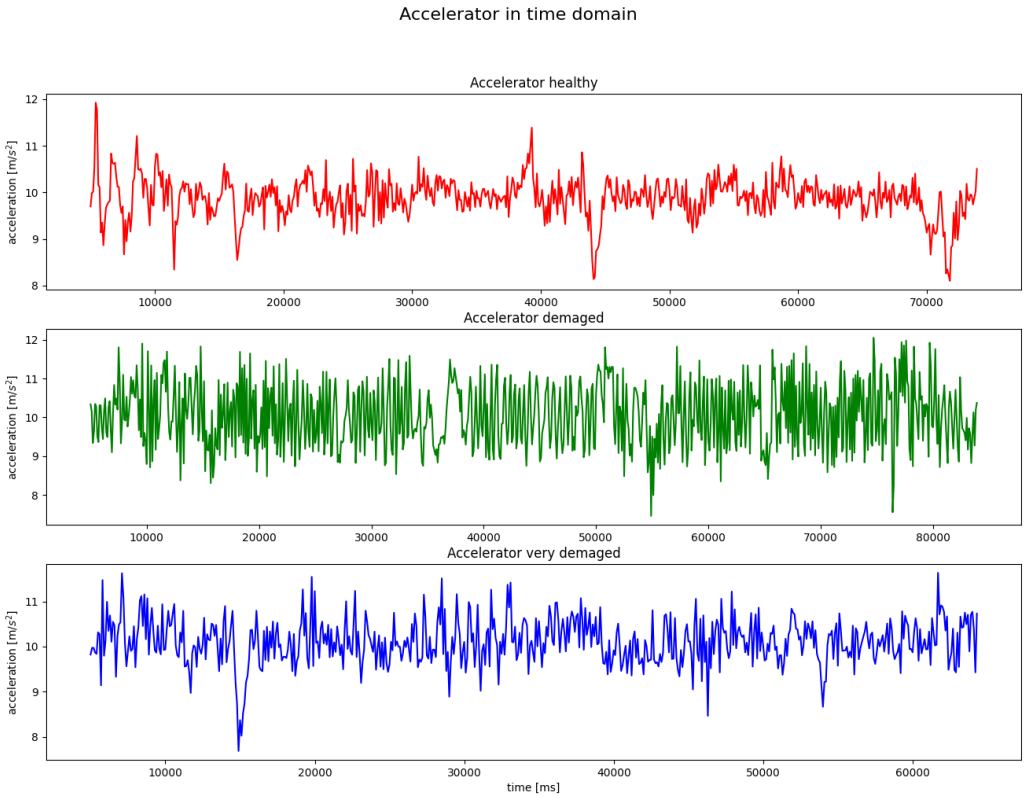
$$\|\mathbf{acc}\|_2 = \sqrt{acc_x^2 + acc_y^2 + acc_z^2}$$

In [ ]:
```
disp_image('photos/data_from_acc_1.png',size=1,rotate=False)
```



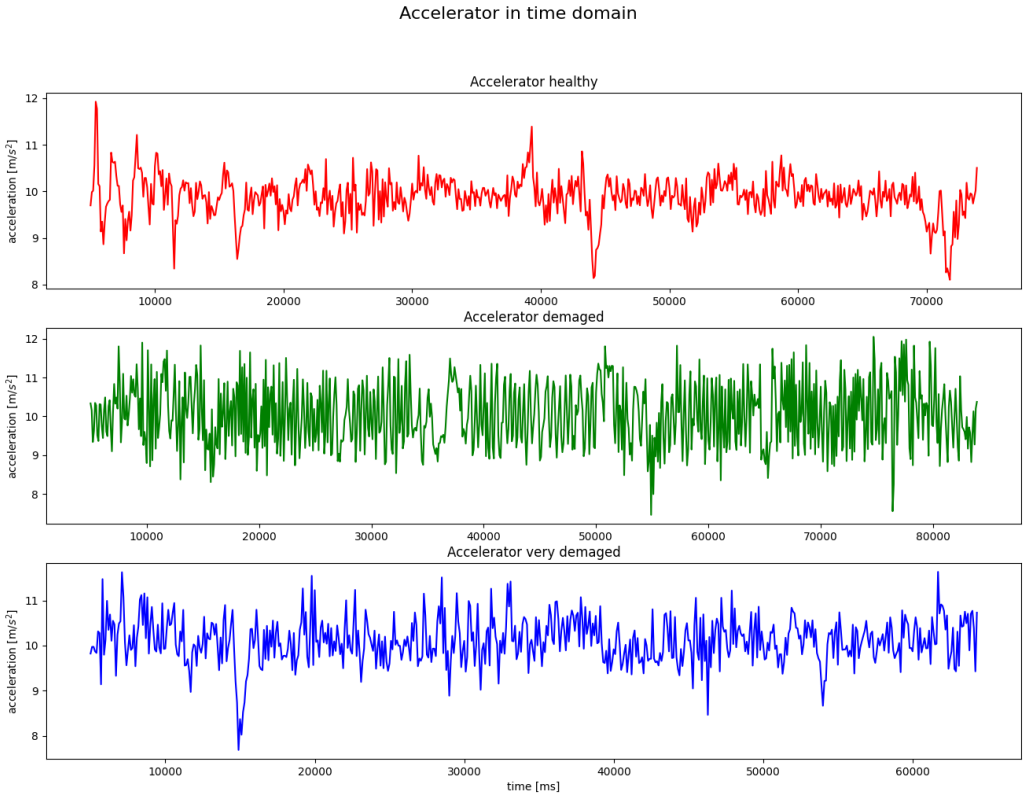
The same operation was performed for the gyroscope.

In [ ]:
```
disp_image('photos/data_from_acc_1.png',size=1,rotate=False)
```



The data was subjected to the removal of unit jumps that occur during takeoff and landing, which could reduce the model's effectiveness. Additionally, gyroscope data was subjected to Fourier transformation to examine how vibrations behave at different frequencies.

In [ ]:
```
disp_image('photos/data_from_acc_1.png',size=1,rotate=False)
```



Before exporting the data to the model, the flight data was divided into segments every 1.5 seconds. This aims to match the input data to the models, where the first and second models have a sliding window, and the third model aggregates the data.

Model [0-4 pts]:

- are models sufficiently described (what are formulas, what are parameters, what data are required) [1 pt]
- are difference between two models explained [1 pt]
- is the difference in the models justified (e.g. does adding aditional parameter makes sense?) [1 pt]
- are two different models specified [1 pt]

We decided to use Bayesian Mixture Model as our binary classifier. We assume that a given measurement, y, represented using B-spline basis can be drawn from one of

M data generating processes (which represents our fault classes), each described by unique set of parameters. To create this model, we need to build the likelihood function and then use it to develop the posterior distribution.

Data generating process is given by

$$y \sim Normal(\mu, \sigma)$$

$$\mu = \sum_{m=1}^{M} \beta_m \phi_m(t)$$

where y is sampled signal with uncertanity of normal distribution given by σ. Functions $\phi_m(t)$ are B-splines on assigned M knot grid. μ represents transformed parameters of the model, as it corresponds to the mean of fitted distribution of each class. β_m are terms regulating each spline values, all are descirbed by normal distribution

$$\beta_m \sim Normal(\mu_b, \sigma_b)$$

Our two model differs in number of used splines to represent signals, the more we use the better we can approximate signal with respect to local phenomenas however we are also greatly increasing computational time and risk overfitting. First model uses 15 splines and second 27. We decided on this numbers because initial experiments yielded promising results with low number of basis functions, however increasing them one by one was slowly improving correct classifications.

Priors [0-4 pts]:

- Have prior predictive checks been done for parameters (are parameters simulated from priors make sense) [1 pt]
- How prior parameters were selected [1 pt]
- Have prior predictive checks been done for measurements (are measurements simulated from priors make sense) [1 pt]
- Is it explained why particular priors for parameters were selected [1 pt]

We have chosen following priors for both of our models:

$$\sigma \sim Exponential(1)$$

$$\beta \sim Normal(0, 1)$$

For β, we selected a weakly informative prior - uniform distribution since we didn't have a strong rationale for choosing a different one due to nature of representing

signal as a set of basis functions. For σ, we opted for an exponential prior because its thicker tail provides greater flexibility. Since it's binary classifier number of classes M is equal to 2 (damaged, healthy).

With priors chosen we proceed to prior predictive analysis for first model, it's worth noting that we calculate prior only for one class since it's the exact same process for both classes.

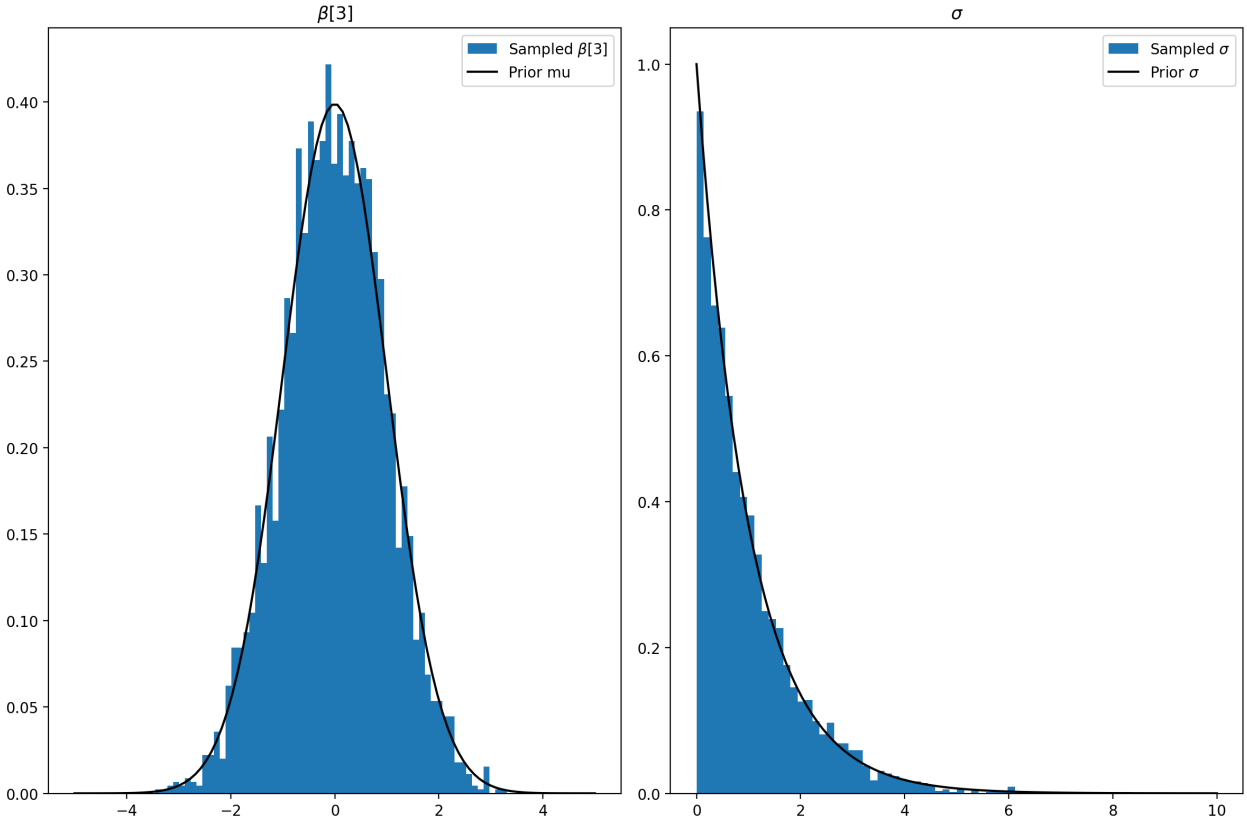
In [ ]:
```
prior_model_1 = CmdStanModel(stan_file='stan/prior_check.stan')

num_knots = 13
N = 150
times = np.linspace(0,N*10,N)
knot_list = np.quantile(times,np.linspace(0,1,num_knots))
B0 = create_spline_matrix(N, times, 3, num_knots)

data_ppc = {
    "N": N,
    "K": num_knots+2,
    "X": B0
}
ppc_samples = prior_model_1.sample(data=data_ppc)
```

```
22:32:59 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

22:32:59 - cmdstanpy - INFO - CmdStan done processing.
```

Since model uses 15 different β plotting all of them is rather confusing than helpful, that's why we decided to show only one β and σ

In [ ]:
```
beta_pred= ppc_samples.stan_variable('betas')[:,2]
sigmas_pred = ppc_samples.stan_variable('sigma')[:]
fig,axes = plt.subplots(1,2,figsize = (12,8), tight_layout=True)

x = np.linspace(-5, 5, 100)
y = stats.norm.pdf(x=x,loc=0,scale=1)
axes[0].hist(beta_pred, bins=60,label=r'Sampled $\beta[3]$',density=True)
axes[0].set_title(r'$\beta[3]$')
axes[0].plot(x,y,label = 'Prior mu',color='k')
axes[0].legend()

x = np.linspace(0,10,100)
y = stats.expon.pdf(x=x,scale=1)
axes[1].hist(sigmas_pred, bins=60,label=r'Sampled $\sigma$',density=True)
axes[1].set_title(r'$\sigma$')
axes[1].plot(x,y,label=r'Prior $\sigma$',color='k')
axes[1].legend()
```

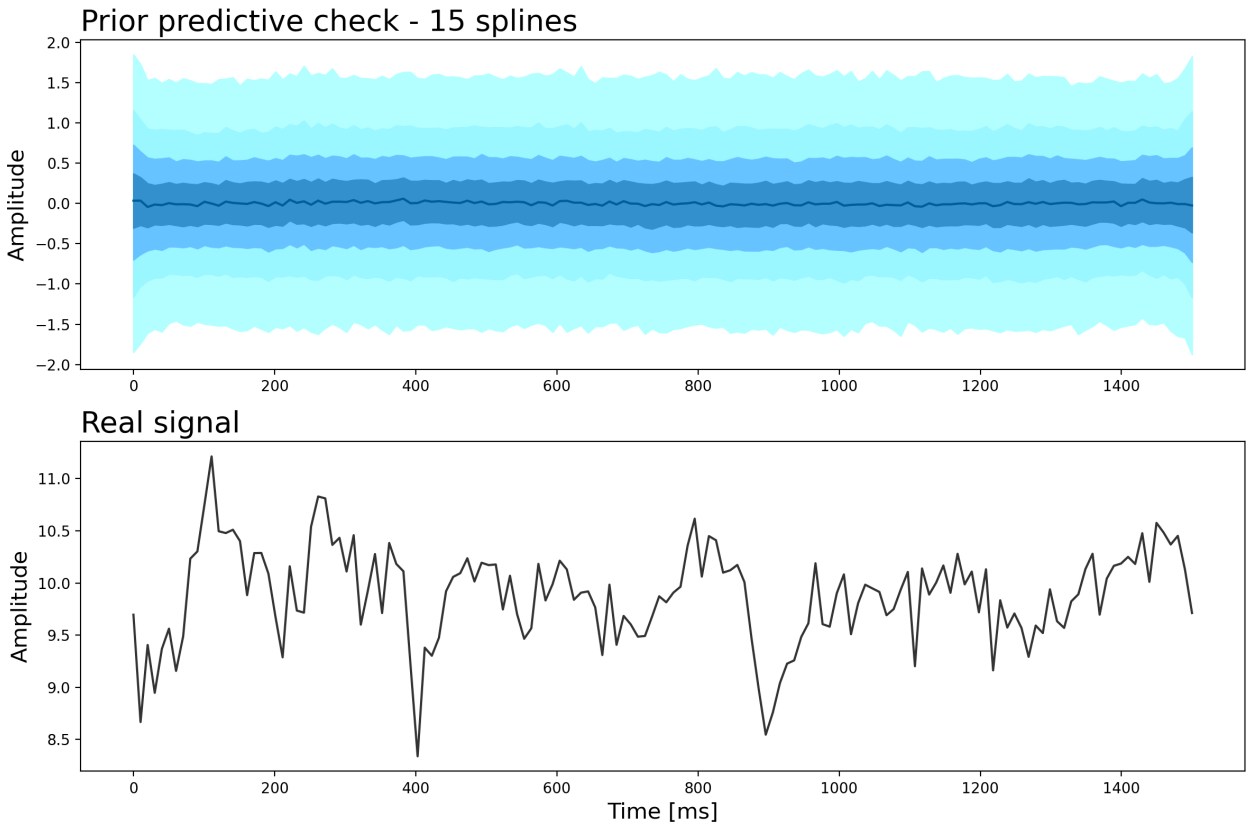
Out[ ]: <matplotlib.legend.Legend at 0x171736cf0>



We present samples generated by our model by using ribbon plot on which we can see 90th percentile of generated samples and for comparison we are showing in black real signal from healthy drone. As we can see our model is capable of generating shape of a signal.

In [ ]:
```
x = np.linspace(0,1500,N)
fig,axes = plt.subplots(2,1,figsize = (12,8), tight_layout=True)
pred = ppc_samples.stan_variable('y_pred')
axes[0] = ribbon_plot(x,pred,axes[0],supress_warning=True,probs = [10, 20
axes[1].set_xlabel('Time [ms]',fontsize = 15)
axes[0].set_ylabel('Amplitude',fontsize = 15)
axes[1].set_ylabel('Amplitude',fontsize = 15)
axes[0].set_title('Prior predictive check - 15 splines', loc= 'left',font
axes[1].plot(x,acc_healthy_data[1], color = 'k',alpha=0.8)
axes[1].set_title('Real signal', loc= 'left',fontsize = 20)
```

Out[ ]: Text(0.0, 1.0, 'Real signal')



And analogically for second model

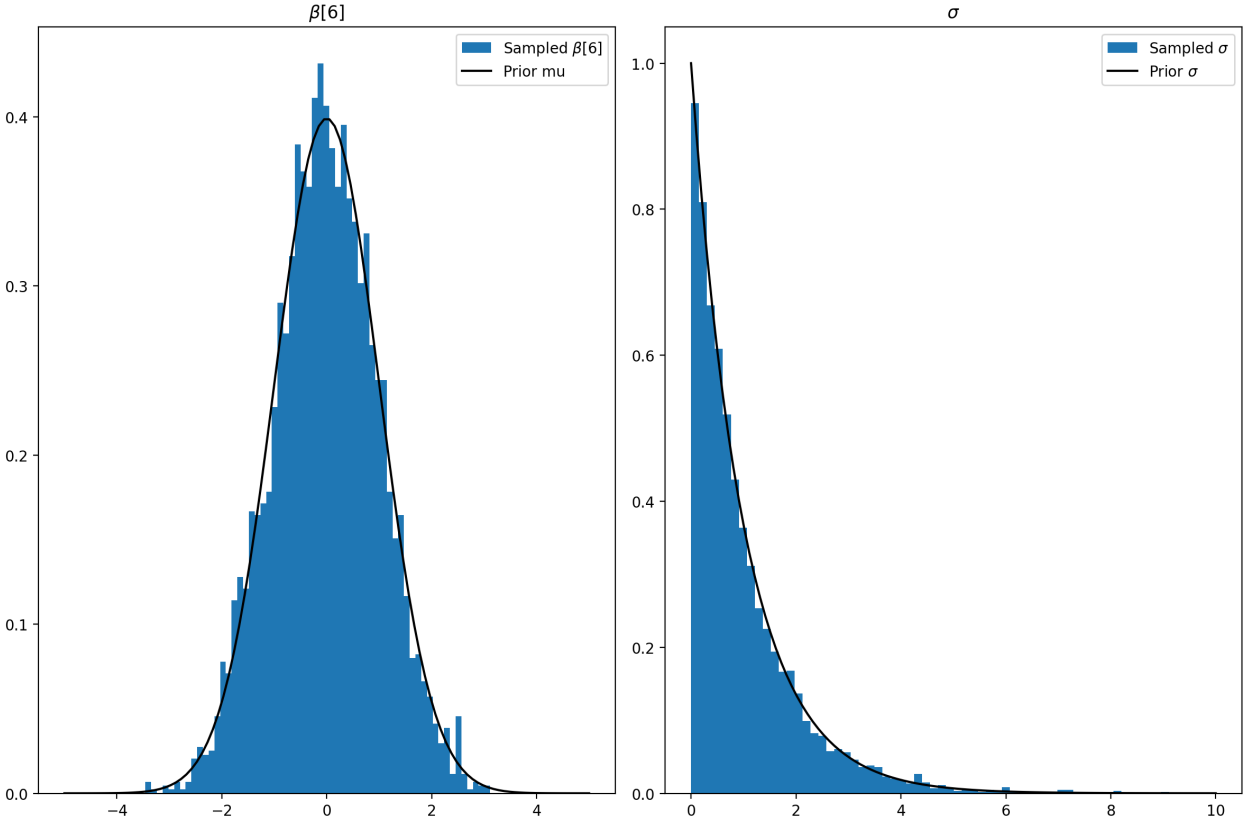
In [ ]:
```
prior_model_2 = CmdStanModel(stan_file='stan/prior_check.stan')

num_knots = 25
N = 150
times = np.linspace(0,N*10,N)
knot_list = np.quantile(times,np.linspace(0,1,num_knots))
B0 = create_spline_matrix(N, times, 3, num_knots)

data_ppc = {
    "N": N,
    "K": num_knots+2,
    "X": B0
}
ppc_samples_2 = prior_model_2.sample(data=data_ppc)
```

```
22:33:01 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

22:33:02 - cmdstanpy - INFO - CmdStan done processing.
```

In [ ]:
```
beta_pred= ppc_samples_2.stan_variable('betas')[:,5]
sigmas_pred = ppc_samples_2.stan_variable('sigma')[:]
fig,axes = plt.subplots(1,2,figsize = (12,8), tight_layout=True)
```

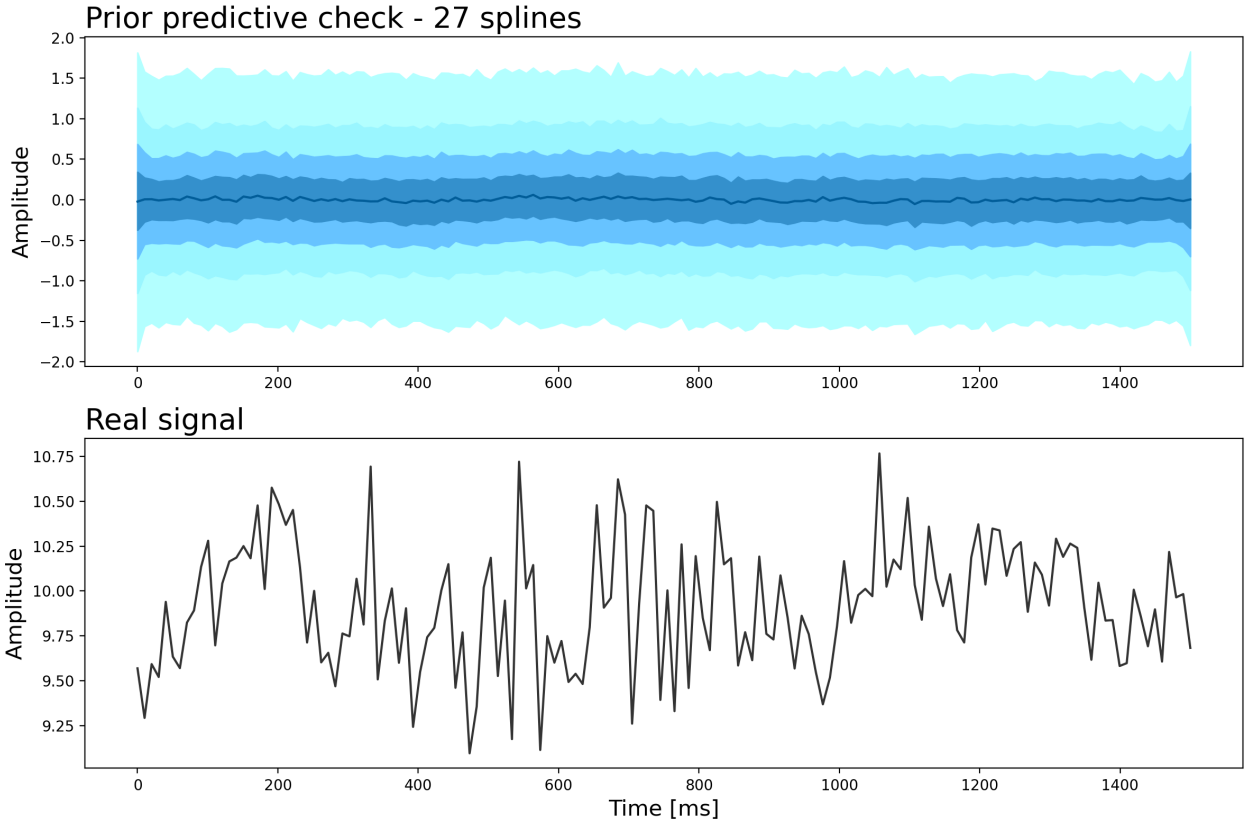
```
x = np.linspace(-5, 5, 100)
y = stats.norm.pdf(x=x,loc=0,scale=1)
axes[0].hist(beta_pred, bins=60,label=r'Sampled $\beta[6]$',density=True)
axes[0].set_title(r'$\beta[6]$')
axes[0].plot(x,y,label = 'Prior mu',color='k')
axes[0].legend()

x = np.linspace(0,10,100)
y = stats.expon.pdf(x=x,scale=1)
axes[1].hist(sigmas_pred, bins=60,label=r'Sampled $\sigma$',density=True)
axes[1].set_title(r'$\sigma$')
axes[1].plot(x,y,label=r'Prior $\sigma$',color='k')
axes[1].legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x1712e02f0>



In [ ]:
```
x = np.linspace(0,1500,N)
fig,axes = plt.subplots(2,1,figsize = (12,8), tight_layout=True)
pred = ppc_samples_2.stan_variable('y_pred')
axes[0] = ribbon_plot(x,pred,axes[0],supress_warning=True,probs = [10, 20
axes[1].set_xlabel('Time [ms]',fontsize = 15)
axes[0].set_ylabel('Amplitude',fontsize = 15)
axes[1].set_ylabel('Amplitude',fontsize = 15)
axes[0].set_title('Prior predictive check - 27 splines', loc= 'left',font
axes[1].plot(x,acc_healthy_data[6], color = 'k',alpha=0.8)
axes[1].set_title('Real signal', loc= 'left',fontsize = 20)
```

Out[ ]: Text(0.0, 1.0, 'Real signal')



We aquired very simmilar results from both models, which was expceted since their main difference can be seen on local level of function.

Posterior analysis (model 1) [0-4 pts]:

- have parameter marginal disrtibutions been analyzed (histograms of individual parametes plus summaries, are they diffuse or concentrated, what can we say about values) [1 pt]
- were there any issues with the sampling? if there were what kind of ideas for mitigation were used [1 pt]
- are the data consistent with posterior predictive samples and is it sufficiently commented (if they are not then is the justification provided)
- are the samples from posterior predictive distribution analyzed [1 pt]

Loading ang organizing data

In [ ]:
```
acc_healthy = pd.read_csv('data_preprocesed/acc_healthy_samples.csv')
acc_damaged = pd.read_csv('data_preprocesed/acc_damaged_samples.csv')
acc_very_damaged = pd.read_csv('data_preprocesed/acc_very_damaged_samples

acc_healthy_data = np.array([acc_healthy[col].values for col in acc_healt
acc_damaged_data = np.array([acc_damaged[col].values for col in acc_damag
acc_v_damaged_data = np.array([acc_very_damaged[col].values for col in ac
```

```
acc = [acc_healthy_data,acc_damaged_data,acc_v_damaged_data]
```

As posterior predicitve check we provide data generative model that takes 6 of our samples from signle class as input and based on them it generates signals, again we focus only on single class since both data generating processes are exacly the same. There were no issues during sampling.

In [ ]:
```
model_posterior = CmdStanModel(stan_file="stan/posterior_check.stan")
I = 6
num_knots = 13
N = 150
times = np.linspace(0,N*10,N)
knot_list = np.quantile(times,np.linspace(0,1,num_knots))
B0 = create_spline_matrix(N, times, 3, num_knots)
sampling_order = np.random.permutation([*range(len(acc[0]))])
y = acc[0][sampling_order][:I]

data_ppc = {
    "I": I,
    "N": N,
    "K": num_knots+2,
    "X": B0,
    "y": y.T
}

samples_posterior = model_posterior.sample(data=data_ppc)
print(samples_posterior.diagnose())
```

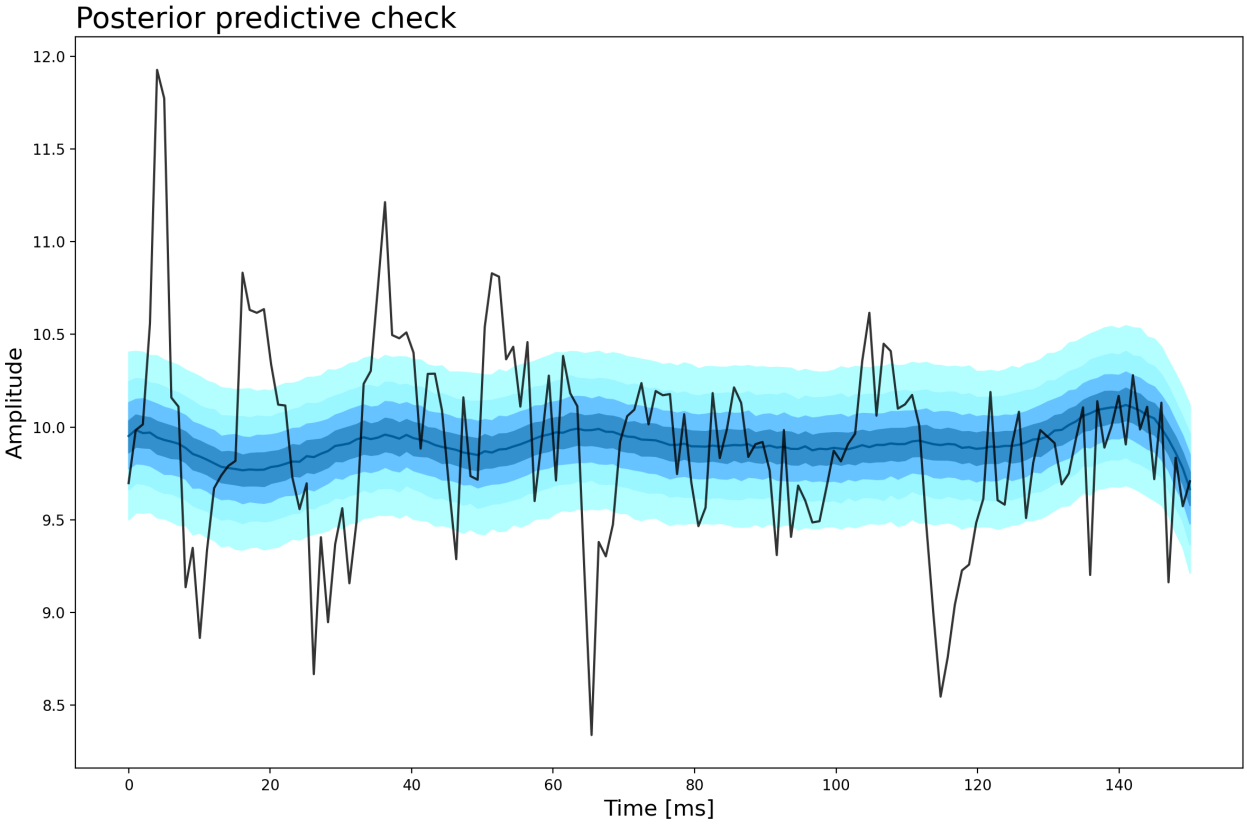
```
22:33:03 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

22:33:04 - cmdstanpy - INFO - CmdStan done processing.
22:33:04 - cmdstanpy - WARNING - Non-fatal error during sampling:
Exception: normal_lpdf: Scale parameter is 0, but must be positive! (in '/
Users/kacperjarzyna/Desktop/studia/DATA_ANALYTICS_DRONE/drones_prediction/
stan/posterior_check.stan', line 22, column 4 to column 30)
Consider re-running with show_console=True if the above output is unclear!
```

```
Processing csv files: /var/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp
472ljshx/posterior_checkjbr3o8ck/posterior_check-20240613223303_1.csv, /va
r/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp472ljshx/posterior_checkj
br3o8ck/posterior_check-20240613223303_2.csv, /var/folders/69/dws6xwp11mgd
nm5sb8sr1kph0000gn/T/tmp472ljshx/posterior_checkjbr3o8ck/posterior_check-2
0240613223303_3.csv, /var/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp4
72ljshx/posterior_checkjbr3o8ck/posterior_check-20240613223303_4.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

We present generated samples on ribbon plot with real singal in black. As we can see with 15 basis function we capture mean of signal quite well and roughly estimate local phenomenas.

In [ ]:
```
x = np.linspace(0,150,N)
fig = plt.figure(figsize=(12, 8), tight_layout = True)
axes = plt.subplot(1, 1 ,1)
y_pred= samples_posterior.stan_variable('y_hat')[:,:]
axes = ribbon_plot(x,y_pred,axes,supress_warning=True,probs = [10, 20, 30
axes.set_xlabel('Time [ms]',fontsize = 15)
axes.set_ylabel('Amplitude',fontsize = 15)
axes.set_title('Posterior predictive check', loc= 'left',fontsize = 20)
axes.plot(x,acc_healthy_data[0], color = 'k',alpha=0.8)
```

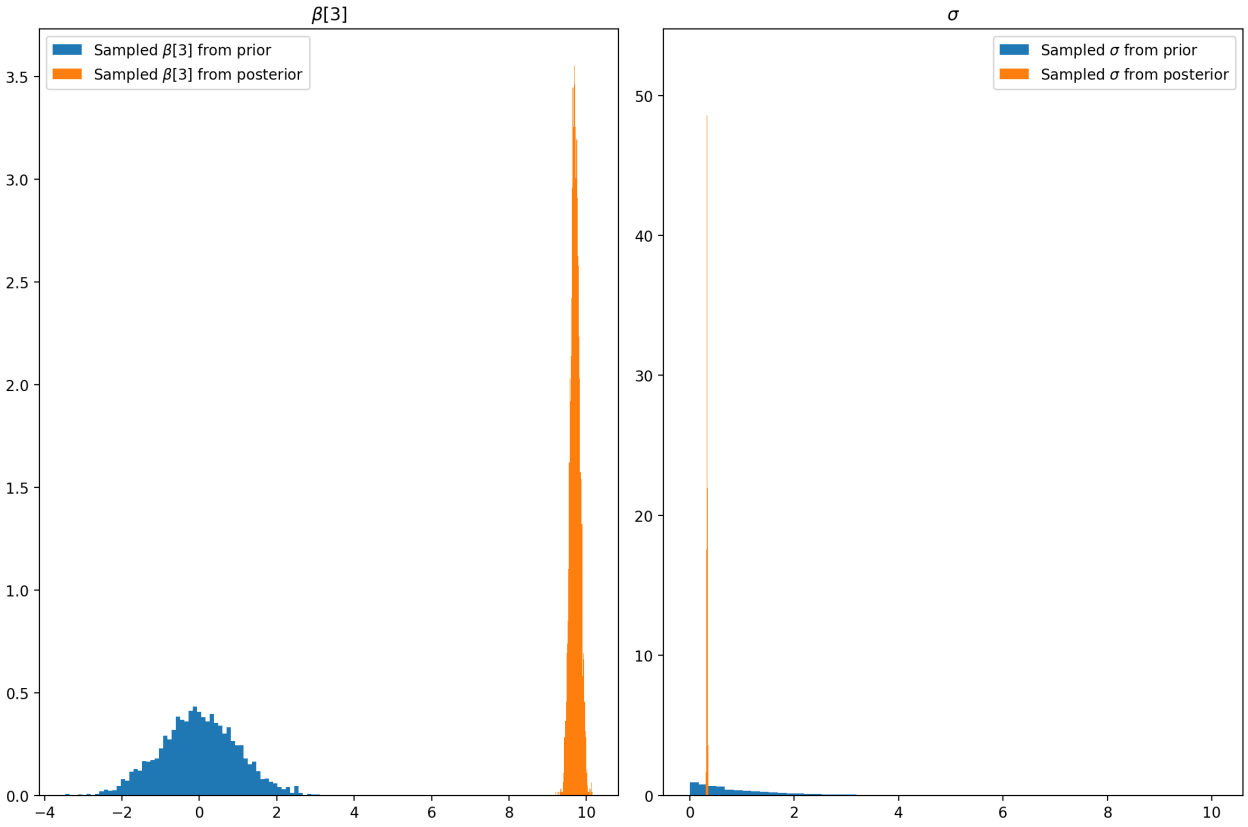
Out[ ]:
```
[<matplotlib.lines.Line2D at 0x171af9e80>]
```



On histogram we present how our parameters contracted after fitting, again showing all 15 β would be confusing so we limit ourselfs to only one.

In [ ]:
```
sigma = samples_posterior.stan_variable('sigma')
sigma_pred = samples_posterior.stan_variable('sigma_sim_hat')

betas = samples_posterior.stan_variable('betas')[:,2]
betas_pred = samples_posterior.stan_variable('betas_sim_hat')[:,2]


fig,axes = plt.subplots(1,2,figsize = (12,8), tight_layout=True)

x = np.linspace(-5, 5, 100)
y = stats.norm.pdf(x=x,loc=0,scale=1)
axes[0].hist(beta_pred, bins=60,label=r'Sampled $\beta[3]$ from prior',de
axes[0].set_title(r'$\beta[3]$')
axes[0].hist(betas, bins=60,label=r'Sampled $\beta[3]$ from posterior',de
axes[0].legend()

x = np.linspace(0,10,100)
y = stats.expon.pdf(x=x,scale=1)
axes[1].hist(sigma_pred, bins=60,label=r'Sampled $\sigma$ from prior',den
axes[1].set_title(r'$\sigma$')
axes[1].hist(sigma, bins=60,label=r'Sampled $\sigma$ from posterior',dens
axes[1].legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x171ad7ce0>
```



We also calculate posterior contraction given as

$$c[f \mid \tilde{y}] = 1 - \frac{\mathbb{V}_{\text{post}}[f \mid \tilde{y}]}{\mathbb{V}_{\text{prior}}[f \mid \tilde{y}]}$$

where $\mathbb{V}$ stands for variance, measures how much the data from an observation update our understanding of the function f. If the posterior contraction is close to zero, it means the data add little new information beyond what the prior model already provides. If it is close to one, the data are much more informative compared to the prior model. Results are showing that we were able to inform our model greatly.

In [ ]:
```
_, contraction = get_z_contr(results=samples_posterior,num_knots=15)
betas_names = ['Sigma'] + [fr'Beta {x+1}' for x in range(len(contraction[
contr = [[value] for value in contraction[0]]
df_betas = pd.DataFrame(contr,betas_names,columns=['Contraction value'])
df_betas
```

Out[ ]:

	Contraction value
Sigma	0.999939
Beta 1	0.989775
Beta 2	0.985475
Beta 3	0.986002
Beta 4	0.991135
Beta 5	0.992439
Beta 6	0.993030
Beta 7	0.993262
Beta 8	0.993043
Beta 9	0.992916
Beta 10	0.993052
Beta 11	0.992489
Beta 12	0.991304
Beta 13	0.986173
Beta 14	0.985236
Beta 15	0.989751

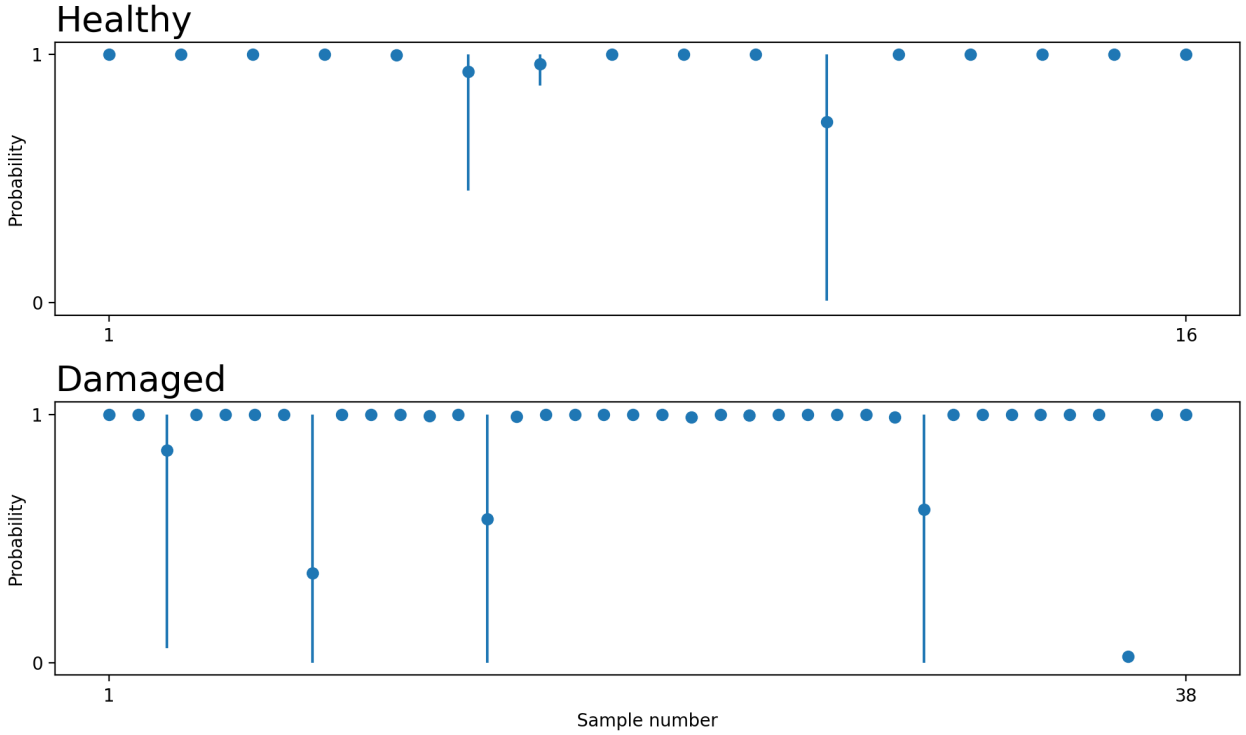
Additionally we provide example results of our classifier. The plot displays probabilistic classifications for a set of samples categorized into two classes: healthy and damaged. Each plot shows the probability distribution of each sample belonging to its respective class across test run. Blue dot represents the mean values, when blue bar presents 95% confidence interval. As we can see model predicted healthy signals quite well except one sample where confidence interval is quite broad. In damaged class we can see a bit more uncertanity and one wrong classifitaion. We can use information about broad confidence interval as a warining that our drone might be worth inspecting but it's not necessary to stop mission.

In [ ]:
```
num_knots = 13
seed = 26042024
data, labels, IT, IL, total,B0,knot_list = prepare_data(acc,frequencies=N
                                                        num_knots=num_kno
model = CmdStanModel(stan_file='stan/mix.stan')
hit_rate = get_results(model=model,data=data,seed=seed,labels=labels,IT=I
print(f'Accuracy of single run {hit_rate}%')
```

```
22:33:08 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
```

```
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status
```

```
22:33:10 - cmdstanpy - INFO - CmdStan done processing.
Accuracy of single run 0.9074074074074074%
```



During tests we noted a group of "problematic" samples in both data groups which classification results strongly correlates with samples chosen for training. To not let this influence classifier score we decided to perform 100 independent experiments each with randomly chosen training data, and our final score is statistic of said experiments. As we can see we managed to get 90,2 $\pm$ 4,7% accuracy on data from accelerometr.

In [ ]:
```
## Estimated time ~ 5 minutes

# num_knots = 13
# res = []
# for i in range(100):
#     data, labels, IT, IL, total ,B0, knot_list = prepare_data(acc, freq
#     res.append(get_results(model=model, data=data, labels=labels, IT=IT
# np.savetxt('results/acc_15.csv', res, delimiter=',', fmt='%f')
```

In [ ]:
```
results = np.genfromtxt('results/acc_15.csv', delimiter=',')
mean = np.mean(results)
st_dev = np.std(results)
print(f'Mean accuracy: {np.round(mean,3)*100}% \nStandard deviation: {np.
```

```
Mean accuracy: 90.2%
Standard deviation: 4.7%
```

Posterior analysis (model 2) [0-4 pts]:

- have parameter marginal disrtibutions been analyzed (histograms of individual parametes plus summaries, are they diffuse or concentrated, what can we say about values) [1 pt]
- were there any issues with the sampling? if there were what kind of ideas for mitigation were used [1 pt]
- are the samples from posterior predictive distribution analyzed [1 pt]
- are the data consistent with posterior predictive samples and is it sufficiently commented (if they are not then is the justification provided)

We perform exactly the same steps as for model 1

In [ ]:
```
model_posterior_2 = CmdStanModel(stan_file="stan/posterior_check.stan")
I = 6
num_knots = 25
N = 150
times = np.linspace(0,N*10,N)
knot_list = np.quantile(times,np.linspace(0,1,num_knots))
B0 = create_spline_matrix(N, times, 3, num_knots)
sampling_order = np.random.permutation([*range(len(acc[0]))])
y = acc[0][sampling_order][:I]

data_ppc = {
    "I": I,
    "N": N,
    "K": num_knots+2,
    "X": B0,
    "y": y.T
}

samples_posterior_2 = model_posterior_2.sample(data=data_ppc)
print(samples_posterior_2.diagnose())
```

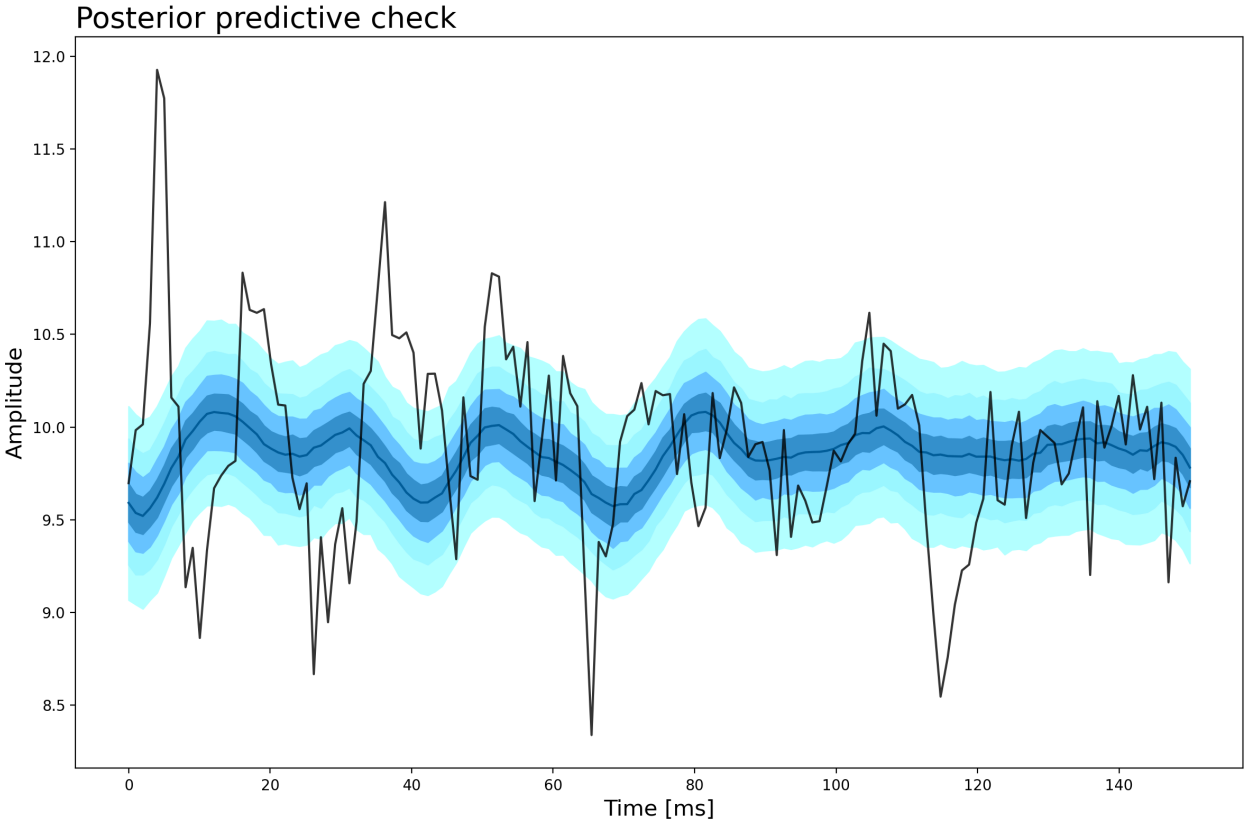
```
22:33:11 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

22:33:12 - cmdstanpy - INFO - CmdStan done processing.
```

```
Processing csv files: /var/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp
472ljshx/posterior_checktskn3bl5/posterior_check-20240613223311_1.csv, /va
r/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp472ljshx/posterior_checkt
skn3bl5/posterior_check-20240613223311_2.csv, /var/folders/69/dws6xwp11mgd
nm5sb8sr1kph0000gn/T/tmp472ljshx/posterior_checktskn3bl5/posterior_check-2
0240613223311_3.csv, /var/folders/69/dws6xwp11mgdnm5sb8sr1kph0000gn/T/tmp4
72ljshx/posterior_checktskn3bl5/posterior_check-20240613223311_4.csv

Checking sampler transitions treedepth.
Treedepth satisfactory for all transitions.

Checking sampler transitions for divergences.
No divergent transitions found.

Checking E-BFMI - sampler transitions HMC potential energy.
E-BFMI satisfactory.

Effective sample size satisfactory.

Split R-hat values satisfactory all parameters.

Processing complete, no problems detected.
```

It is shown that increasing number of basis function allows model to capture local phenomenas with greater precision

In [ ]:
```
x = np.linspace(0,150,N)
fig = plt.figure(figsize=(12, 8), tight_layout = True)
axes = plt.subplot(1, 1 ,1)
y_pred= samples_posterior_2.stan_variable('y_hat')[:,:]
axes = ribbon_plot(x,y_pred,axes,supress_warning=True,probs = [10, 20, 30
axes.set_xlabel('Time [ms]',fontsize = 15)
axes.set_ylabel('Amplitude',fontsize = 15)
axes.set_title('Posterior predictive check', loc= 'left',fontsize = 20)
axes.plot(x,acc_healthy_data[0], color = 'k',alpha=0.8)
```

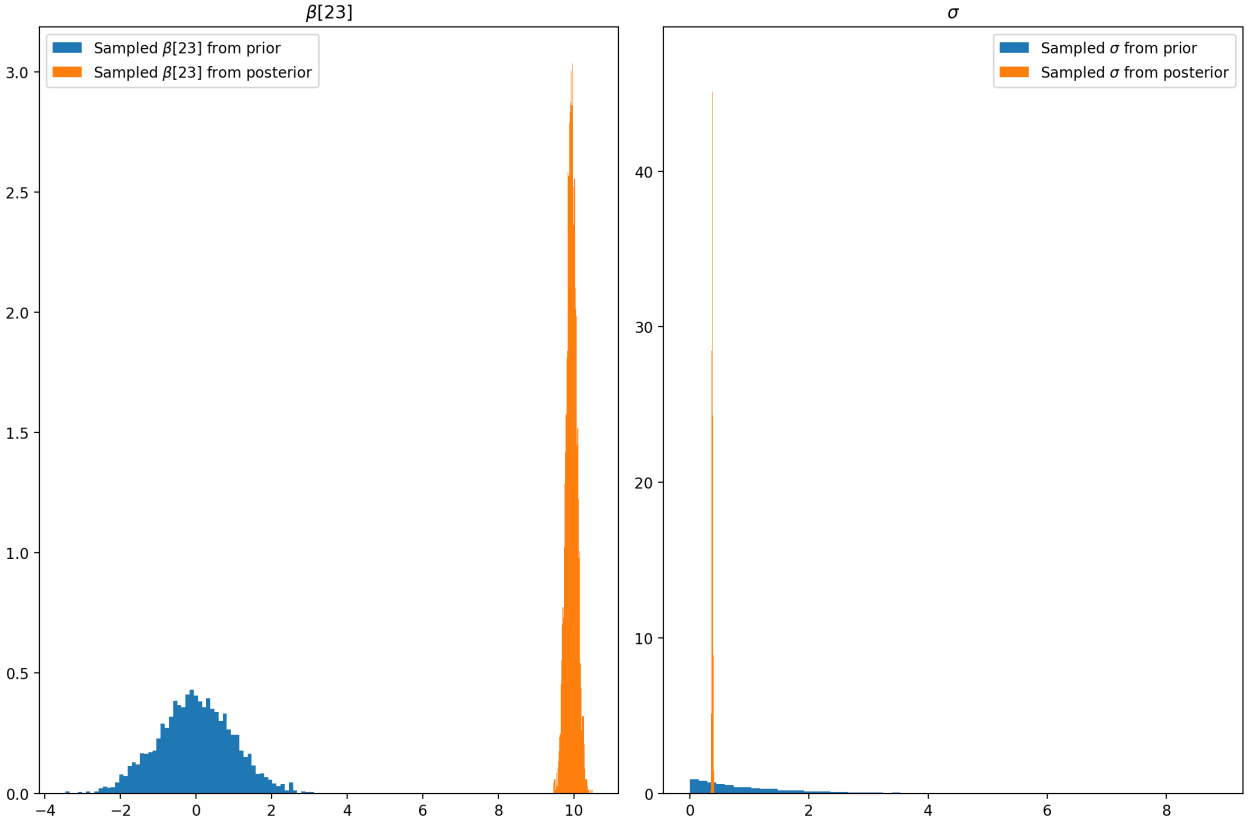
Out[ ]:
```
[<matplotlib.lines.Line2D at 0x171fb2b70>]
```



Simmilarly we aquired satysfying results of contraction for our parameters

In [ ]:
```
sigma = samples_posterior_2.stan_variable('sigma')
sigma_pred = samples_posterior_2.stan_variable('sigma_sim_hat')

betas = samples_posterior_2.stan_variable('betas')[:,22]
betas_pred = samples_posterior_2.stan_variable('betas_sim_hat')[:,22]


fig,axes = plt.subplots(1,2,figsize = (12,8), tight_layout=True)

x = np.linspace(-5, 5, 100)
y = stats.norm.pdf(x=x,loc=0,scale=1)
axes[0].hist(beta_pred, bins=60,label=r'Sampled $\beta[23]$ from prior',d
axes[0].set_title(r'$\beta[23]$')
axes[0].hist(betas, bins=60,label=r'Sampled $\beta[23]$ from posterior',d
axes[0].legend()

x = np.linspace(0,10,100)
y = stats.expon.pdf(x=x,scale=1)
axes[1].hist(sigma_pred, bins=60,label=r'Sampled $\sigma$ from prior',den
axes[1].set_title(r'$\sigma$')
axes[1].hist(sigma, bins=60,label=r'Sampled $\sigma$ from posterior',dens
axes[1].legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x171feb830>
```



In [ ]:
```
_, contraction = get_z_contr(results=samples_posterior_2,num_knots=27)
betas_names = ['Sigma'] + [fr'Beta {x+1}' for x in range(len(contraction[
contr = [[value] for value in contraction[0]]
df_betas = pd.DataFrame(contr,betas_names,columns=['Contraction value'])
df_betas
```

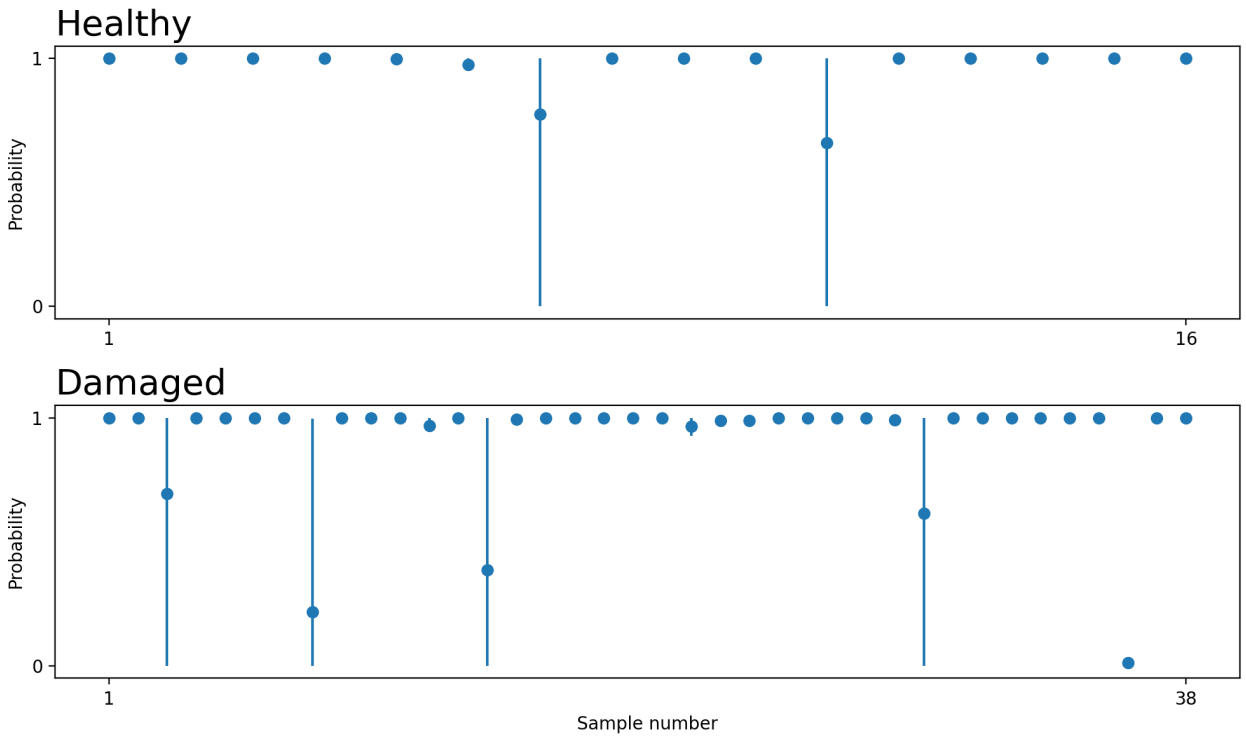
Out[ ]:

	Contraction value
Sigma	0.999913
Beta 1	0.980339
Beta 2	0.965035
Beta 3	0.965344
Beta 4	0.977307
Beta 5	0.980214
Beta 6	0.980704
Beta 7	0.981721
Beta 8	0.981562
Beta 9	0.981454
Beta 10	0.980976
Beta 11	0.981332
Beta 12	0.981768
Beta 13	0.981906
Beta 14	0.981374
Beta 15	0.981482
Beta 16	0.981665
Beta 17	0.982100
Beta 18	0.982027
Beta 19	0.981799
Beta 20	0.981631
Beta 21	0.981724
Beta 22	0.980815
Beta 23	0.980413
Beta 24	0.977477
Beta 25	0.965420
Beta 26	0.964572
Beta 27	0.981021

Example of classification

In [ ]:
```
num_knots = 25
seed = 26042024
data, labels, IT, IL, total,B0,knot_list = prepare_data(acc,frequencies=N
                                                        num_knots=num_kno
model_2 = CmdStanModel(stan_file='stan/mix.stan')
hit_rate = get_results(model=model_2,data=data,seed=seed,labels=labels,IT
print(f'Accuracy of single run {hit_rate}%')
```

```
22:33:16 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status
chain 3 |          | 00:00 Status
chain 4 |          | 00:00 Status

22:33:19 - cmdstanpy - INFO - CmdStan done processing.
Accuracy of single run 0.8703703703703703%
```



Accuracy declined to 87,6 ± 4,9%, which was suprising to us, seeing that signals generated from model resembled reality with greater precision.

In [ ]:
```
## Estimated time ~ 6 minutes

# num_knots = 25
# res = []
# for i in range(100):
#     data, labels, IT, IL, total ,B0, knot_list = prepare_data(acc, freq
#     res.append(get_results(model=model, data=data, labels=labels, IT=IT
# np.savetxt('results/acc_27.csv', res, delimiter=',', fmt='%f')
```

In [ ]:
```
results = np.genfromtxt('results/acc_27.csv', delimiter=',')
mean = np.mean(results)
```

```
st_dev = np.std(results)
print(f'Mean accuracy: {np.round(mean,3)*100}% \nStandard deviation: {np.
```

```
Mean accuracy: 87.6%
Standard deviation: 4.9%
```

Model comparison [0-4 pts]:

- Have models been compared using information criteria [1 pt]
- Have result for WAIC been discussed (is there a clear winner, or is there an overlap, were there any warnings) [1 pt]
- Have result for PSIS-LOO been discussed (is there a clear winner, or is there an overlap, were there any warnings) [1 pt]
- Was the model comparison discussed? Do authors agree with information criteria? Why in your opinion one model better than another [1 pt]

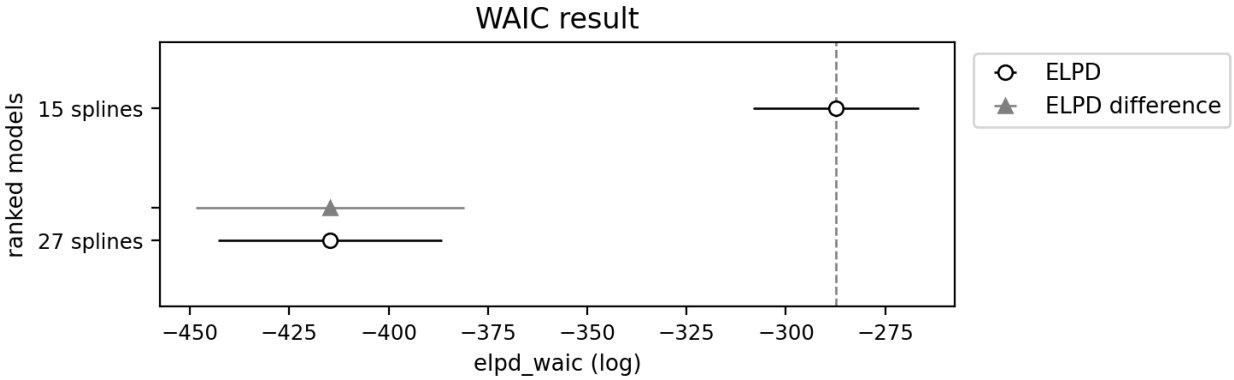
WAIC criterion

In [ ]:
```
splines_15_data = az.from_cmdstanpy(posterior=samples_posterior,log_likel
                          posterior_predictive='y_hat')

splines_27_data = az.from_cmdstanpy(posterior=samples_posterior_2,log_lik
                          posterior_predictive='y_hat')
```

In [ ]:
```
comparison_waic = az.compare({'15 splines': splines_15_data, '27 splines'
print(comparison_waic)
az.plot_compare(comparison_waic)
plt.title('WAIC result')
plt.show()
```

```
            rank    elpd_waic     p_waic   elpd_diff    weight         se
\
15 splines     0 -287.406866  12.614438    0.000000  0.694051  20.820937
27 splines     1 -414.730777  19.930794  127.323911  0.305949  28.148504

                dse  warning scale
15 splines   0.000000     True   log
27 splines  33.706454     True   log
```

```
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:1648: UserWarning: For one or more samples the poster
ior variance of the log predictive densities exceeds 0.4. This could be in
dication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:1648: UserWarning: For one or more samples the poster
ior variance of the log predictive densities exceeds 0.4. This could be in
dication of WAIC starting to fail.
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:309: FutureWarning: Setting an item of incompatible d
type is deprecated and will raise an error in a future version of pandas.
Value 'True' has dtype incompatible with float64, please explicitly cast t
o a compatible dtype first.
  df_comp.loc[val] = (
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:309: FutureWarning: Setting an item of incompatible d
type is deprecated and will raise an error in a future version of pandas.
Value 'log' has dtype incompatible with float64, please explicitly cast to
a compatible dtype first.
  df_comp.loc[val] = (
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/plots/backends/matplotlib/compareplot.py:87: FutureWarning: Series._
_getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame b
ehavior). To access a value by position, use `ser.iloc[pos]`
  scale = comp_df["scale"][0]
```

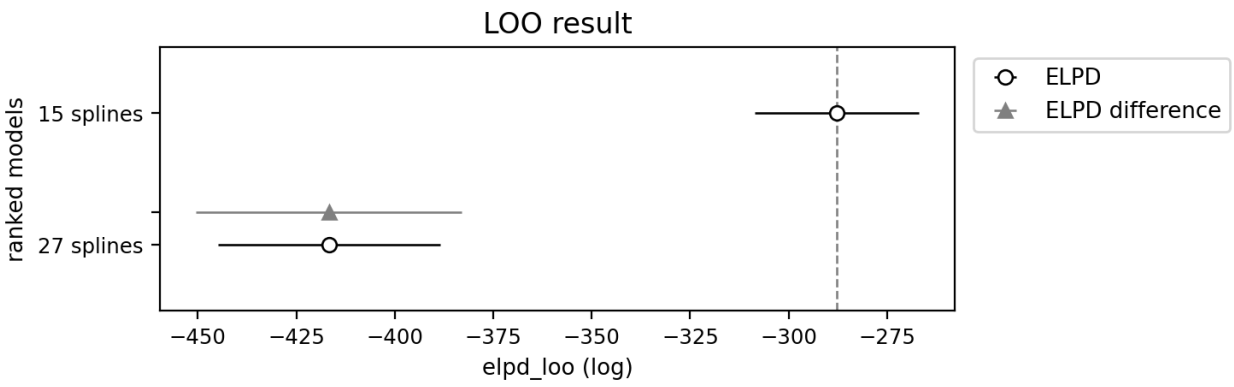


Model 2, which takes in more basis functions, was recognized as the best model, evidenced by its lowest rank and highest relative weight. This model also demonstrated a superior out-of-sample predictive fit. Additionally, Model 2 has a higher estimated effective number of parameters. Both models exhibited similar standard errors in their WAIC estimates. The standard error of the difference between the two models is small, indicating significant overlap. A warning occurred for both models, suggesting that the WAIC computation may not be reliable, likely due to the large number of parameters in each model.

In [ ]:
```
comparison_loo = az.compare({'15 splines': splines_15_data, '27 splines'
print(comparison_loo)
az.plot_compare(comparison_loo)
plt.title('LOO result')
plt.show()
```

```
            rank    elpd_loo      p_loo   elpd_diff    weight         se
\
15 splines     0 -287.827534  13.035105    0.000000  0.696603  20.829919
27 splines     1 -416.699548  21.899565  128.872015  0.303397  28.180564

                dse  warning scale
15 splines   0.000000    False   log
27 splines  33.732471     True   log
```

```
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:805: UserWarning: Estimated shape parameter of Pareto
distribution is greater than 0.7 for one or more samples. You should consi
der using a more robust model, this is because importance sampling is less
likely to work well if the marginal posterior and LOO posterior are very d
ifferent. This is more likely to happen with a non-robust model and highly
influential observations.
  warnings.warn(
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:309: FutureWarning: Setting an item of incompatible d
type is deprecated and will raise an error in a future version of pandas.
Value 'False' has dtype incompatible with float64, please explicitly cast
to a compatible dtype first.
  df_comp.loc[val] = (
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/stats/stats.py:309: FutureWarning: Setting an item of incompatible d
type is deprecated and will raise an error in a future version of pandas.
Value 'log' has dtype incompatible with float64, please explicitly cast to
a compatible dtype first.
  df_comp.loc[val] = (
/Users/kacperjarzyna/opt/anaconda3/envs/stan/lib/python3.12/site-packages/
arviz/plots/backends/matplotlib/compareplot.py:87: FutureWarning: Series._
_getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame b
ehavior). To access a value by position, use `ser.iloc[pos]`
  scale = comp_df["scale"][0]
```



Model 2, which takes in more basis functions, was recognized as the best model due

to its lowest rank and highest relative weight. This model also demonstrated superior out-of-sample predictive fit. Additionally, Model 2 has a higher estimated effective number of parameters. Both models exhibited similar standard errors in their LOO estimates. The standard error of the difference between the two models is small, indicating significant overlap.

A warning occurred for both models, suggesting that the LOO computation may not be reliable, likely due to the large number of parameters in each model.

Summary

For both information criterias we obtained similar results confirming what was expected - more complicated model performs better, as it can represent more accurately represent signals. However as observed better represantation doesn't mean better generalization, and can lead to overfitting.

Additional tests

Loading and organizing data rest of data

In [ ]:
```
gyro_healthy = pd.read_csv('data_preprocesed/gyro_healthy_samples.csv')
gyro_damaged = pd.read_csv('data_preprocesed/gyro_damaged_samples.csv')
gyro_very_damaged = pd.read_csv('data_preprocesed/gyro_very_damaged_sampl

gyro_agg_healthy = pd.read_csv('data_preprocesed/gyro_agg_healthy_samples
gyro_agg_damaged = pd.read_csv('data_preprocesed/gyro_agg_damaged_samples
gyro_agg_very_damaged = pd.read_csv('data_preprocesed/gyro_agg_very_damag

gyro_healthy_data = np.array([gyro_healthy[col].values for col in gyro_he
gyro_damaged_data = np.array([gyro_damaged[col].values for col in gyro_da
gyro_very_damaged_data = np.array([gyro_very_damaged[col].values for col
gyro = [gyro_healthy_data,gyro_damaged_data,gyro_very_damaged_data]

gyro_agg_healthy_data = np.array([gyro_agg_healthy[col].values for col in
gyro_agg_damaged_data = np.array([gyro_agg_damaged[col].values for col in
gyro_agg_very_damaged_data = np.array([gyro_agg_very_damaged[col].values
gyro_agg = [gyro_agg_healthy_data, gyro_agg_damaged_data, gyro_agg_very_d

frequencies = gyro_damaged['Frequencies']
```

In [ ]:
```
# # Estimated time ~ 42 minutes

# test_knots = [13,25]
# testing_data = [gyro,gyro_agg]
# for num_knots in test_knots:
```

```
#     for data_in,name in zip(testing_data,['gyro','gyro_agg']):
#         res = []
#         for i in range(100):
#             data, labels, IT, IL, total ,B0, knot_list = prepare_data(d
#             res.append(get_results(model=model, data=data, labels=label
#         np.savetxt(f'results/{name}_{num_knots+2}.csv', res, delimiter=
```

In [ ]:
```
import os
names_dict = {'acc_15.csv': 'Accelerator 15 splines',
              'acc_27.csv': 'Accelerator 27 splines',
              'gyro_15.csv': 'Giroscope 15 splines',
              'gyro_27.csv': 'Giroscope 27 splines',
              'gyro_agg_15.csv': 'Giroscope with aggregating window 15 sp
              'gyro_agg_27.csv': 'Giroscope with aggregating window 27 sp
              }
for data in os.listdir('results'):
    results = np.genfromtxt(f'results/{data}', delimiter=',')
    mean = np.mean(results)
    st_dev = np.std(results)
    print(names_dict.get(data))
    print(f'Mean accuracy: {np.round(mean,3)*100}% \nStandard deviation:
    print('\n')
```

```
Giroscope 27 splines
Mean accuracy: 78.4%
Standard deviation: 4.5%


Accelerator 27 splines
Mean accuracy: 87.6%
Standard deviation: 4.9%


Giroscope with aggregating window 27 splines
Mean accuracy: 94.19999999999999%
Standard deviation: 2.3%


Accelerator 15 splines
Mean accuracy: 90.2%
Standard deviation: 4.7%


Giroscope with aggregating window 15 splines
Mean accuracy: 93.5%
Standard deviation: 2.1999999999999997%


Giroscope 15 splines
Mean accuracy: 78.60000000000001%
Standard deviation: 4.5%
```

Signal from giroscope with aggregating window using 27 splines yielded the best results, however it also needs the most data preparation since we perform fft, also we didn't limit the size of window, so later signals were much more informative. For real test we plan to use data from accelerometer since it needs least amount of data processing and works in time domain instead of frequency while still giving promising accuracy.