Order-restricted equality of proportions

```
import os

if "KERAS_BACKEND" not in os.environ:
    # set this to "torch", "tensorflow", or "jax"
    os.environ["KERAS_BACKEND"] = "jax"

import matplotlib.pyplot as plt
import numpy as np
import bayesflow as bf
import keras
```

```
INFO:bayesflow:Using backend 'jax'
```

Compared to the previous example, now we will add another model to the mix, a model that suggests that $\theta_2 > \theta_1$:

$$\begin{aligned} s_1 &\sim \text{Binomial}(\theta_1, n_1) \\ s_2 &\sim \text{Binomial}(\theta_2, n_2) \\ (\theta_1, \theta_2) &\sim \text{Uniform}(0, 1)^2, \theta_2 > \theta_1 \end{aligned} \tag{1}$$

Simulator

We will also amortize over different sample sizes for each group.

```
def context():
    n = np.random.randint(1000, 10_000, size=2)
    return dict(n = n)

def prior_null():
    theta = np.random.beta(a=1, b=1)
```

```
    return dict(theta = np.array([theta, theta]))

def prior_alternative():
    theta = np.random.beta(a=1, b=1, size=2)
    return dict(theta = theta)

def prior_restricted():
    theta = np.random.beta(a=1, b=1, size=2)
    theta = np.sort(theta)

    return dict(theta=theta)

def likelihood(theta, n):
    s = np.random.binomial(n=n, p=theta)
    return dict(s=s)

simulator_null = bf.make_simulator([context, prior_null, likelihood])
simulator_alternative = bf.make_simulator([context, prior_alternative, likelihood])
simulator_restricted = bf.make_simulator([context, prior_restricted, likelihood])
simulator = bf.simulators.ModelComparisonSimulator(
    simulators=[simulator_null, simulator_alternative, simulator_restricted],
    use_mixed_batches=True)
```

Approximator

```
adapter = (
    bf.Adapter()
    .concatenate(['n', 's'], into="classifier_conditions")
    .drop('theta')
    )
```

```
classifier_network = keras.Sequential([
    keras.layers.Dense(32, activation="gelu")
    for _ in range(8)
])
approximator = bf.approximators.ModelComparisonApproximator(
    num_models=3,
    classifier_network=classifier_network,
    adapter=adapter)
```

Training

```
epochs = 50
num_batches = 100
batch_size = 512

learning_rate = keras.optimizers.schedules.CosineDecay(5e-4, decay_steps=epochs*num_batches)
optimizer = keras.optimizers.Adam(learning_rate=learning_rate, clipnorm=1.0)
approximator.compile(optimizer=optimizer)
```

```
history = approximator.fit(
    epochs=epochs,
    num_batches=num_batches,
    batch_size=batch_size,
    simulator=simulator,
    adapter=adapter
)
```

```
f=bf.diagnostics.plots.loss(history=history)
```

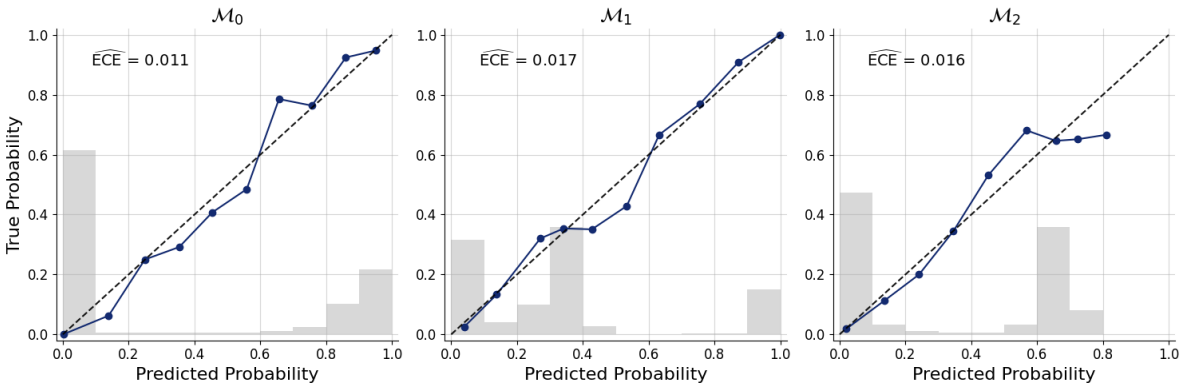
Validation

```
test_data=simulator.sample(5_000)
```

```
true_models=test_data["model_indices"]
pred_models=approximator.predict(conditions=test_data)
```

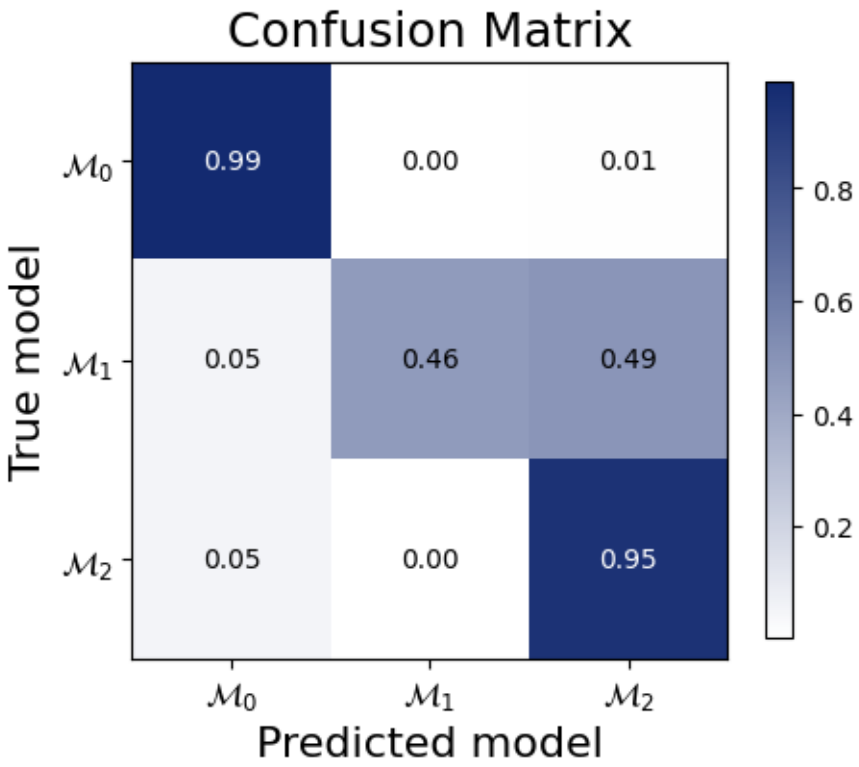
```
f=bf.diagnostics.plots.mc_calibration(
    pred_models=pred_models,
    true_models=true_models,
    model_names=[r"$\mathcal{M}_0$",r"$\mathcal{M}_1$", r"$\mathcal{M}_2$"],
)
```

```
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
```




```
f=bf.diagnostics.plots.mc_confusion_matrix(
    pred_models=pred_models,
    true_models=true_models,
    model_names=[r"$\mathcal{M}_0$",r"$\mathcal{M}_1$", r"$\mathcal{M}_2$"],
    normalize="true"
)
```

```
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
INFO:matplotlib.mathtext:Substituting symbol M from STIXNonUnicode
```



Inference

```
inference_data = dict(n = np.array([[5416, 9072]]), s = np.array([[424, 777]]))
```

```
pred_models = approximator.predict(conditions=inference_data)[0]
pred_models
```

```
array([0.84351516, 0.06271459, 0.0937703 ], dtype=float32)
```

Lee, M. D., & Wagenmakers, E.-J. (2013). *Bayesian Cognitive Modeling: A Practical Course*. Cambridge University Press.