PGM1_clean

December 29, 2016

1 Experiment 1

```
In [ ]: import numpy as np
        import pymc
        import json
        import matplotlib.pyplot as plt
        import matplotlib as mpl
        %matplotlib inline
        plt.style.use("seaborn-colorblind")

        import sys, os
        sys.path.append("C:/Users/lebobcrash/Documents/GitHub/pynoddy/")
        sys.path.append("../pynoddy/")
        import pynoddy.experiment
        import pynoddy.history
        import pynoddy.output

        import seaborn as sns
        sns.set_style("white")
        sns.set_context("paper", font_scale=1.3)
```

```
In [ ]: initial_history = "init.his" # set initial noddy history file
        cs = 10 # cube size
        ws = 900 # well site (x-coord)
```

Create prior pymc Distributions for the fold amplitude and wavelength:

```
In [ ]: fold_amplitude = pymc.Normal("fold_amplitude", 350., 1./np.square(25.))
        fold_wavelength = pymc.Normal("fold_wavelength", 2500., 1./np.square(140.))
```

```
In [ ]: fold_amplitude2 = pymc.Normal("fold_amplitude", 350., 1./np.square(25.))
        fold_wavelength2 = pymc.Normal("fold_wavelength", 2500., 1./np.square(140.))
```

Create the deterministic pymc function, which takes the prior distributions to create the model:

```
In [ ]: ex_pymc = pynoddy.experiment.Experiment(initial_history)
```

1.1 PYMC Functions

1.1.1 Deterministic Modelling Function

```
In [ ]: @pymc.deterministic
        def pynoddy_model(value=0,
                          fold_amplitude=fold_amplitude,
                          fold_wavelength=fold_wavelength,
```

```
                    ex=ex_pymc, ws=ws, cs=cs):
        # set fold event properties to prior parameter draws:
        ex.events[2].properties["Amplitude"] = fold_amplitude
        ex.events[2].properties['Wavelength'] = fold_wavelength
        # drillhole extraction (1-D model for faster simulation)
        well = ex.get_drillhole_data(ws,0)
        return well
```

1.1.2 Stochastic Likelihood Function

Create the stochastic likelihood function for the z-position of layer 2:

```
In [ ]: @pymc.stochastic
        def like_layer2_height(value=0, well=pynoddy_model):
            if len(np.where(well==2)[0]) < 1:
                layer2_height = -9999
            else:
                layer2_height = np.where(well==2)[0][0]
            return pymc.normal_like(layer2_height, 550., 1./np.square(25.))
```

1.2 Simulation

```
In [ ]: iterations = 30000
```

1.2.1 Bayesian Inference

```
In [ ]: model = pymc.Model([fold_wavelength, layer_height,
                            pynoddy_model, like_layer2_height])
```

```
In [ ]: BI_RUN = pymc.MCMC(model, db="hdf5", name="BI_database_name")
        BI_RUN.sample(iter=iterations)
```

1.2.2 Monte Carlo Forward Simulation

```
In [ ]: model_mcfs = pymc.Model([layer_height2, fold_wavelength2,
                                 pynoddy_model])
```

```
In [ ]: MCFS_RUN = pymc.MCMC(model_mcfs, db="hdf5", name="MCFS_database_name")
        MCFS_RUN.sample(iter=iterations)
```

1.3 Results

```
In [ ]: fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(9,4), sharey=True)

        ax[0].set_xlim(1400,3200)
        ax[1].set_xlim(1400,3200)
        ax[0].set_xticks([1400,1850,2300,2750,3200])
        ax[1].set_xticks([1400,1850,2300,2750,3200])

        sns.regplot(MCFS_RUN.trace("fold_wavelength")[:],
                    MCFS_RUN.trace("fold_amplitude")[:],
                    ax=ax[0])

        sns.regplot(BI_RUN.trace("fold_wavelength")[:],
                    BI_RUN.trace("fold_amplitude")[:],
                    color="darkorange", ax=ax[1])
```

```
ax[0].set_title("Monte Carlo Forward Simulation")
ax[1].set_title("Bayesian Inference")
ax[0].set_ylabel("Fold Amplitude [m]")
ax[0].set_xlabel("Fold Wavelength [m]")
ax[1].set_xlabel("Fold Wavelength [m]")
plt.tight_layout()
```