WM_pymc3_model

October 10, 2021

1 Hierarchical model of duration reproduction under cognitive load

[ ]:
```
#%%
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymc3 as pm
#!pip install arviz
import arviz as az
import os
import datetime as dt
print('Last Updated On: ', dt.datetime.now())
```

```
Last Updated On:  2021-10-10 10:29:06.189176
```

1.1 Import raw files

There are four experiments stored in 4 csv files in subfolder `data`.

[ ]:
```
# %% read raw data
cwd = os.getcwd()
raw = [pd.read_csv(cwd+'/../data/Exp' + str(x) + '.csv') for x in range(1,5)]
raw[1].describe()
```

[ ]:
```
            WMSize  ShortLong     DurLevel     TPresent          NT  \
count  5760.000000     5760.0  5760.000000  5760.000000  5760.00000
mean      3.000000        1.0     3.000000     1.500000   180.50000
std       1.633135        0.0     1.414336     0.500043   103.93167
min       1.000000        1.0     1.000000     1.000000     1.00000
25%       1.000000        1.0     2.000000     1.000000    90.75000
50%       3.000000        1.0     3.000000     1.500000   180.50000
75%       5.000000        1.0     4.000000     2.000000   270.25000
max       5.000000        1.0     5.000000     2.000000   360.00000

              NSub       curDur       repDur         WMRP        valid  \
```

```
count  5760.000000  5760.000000  5760.000000  5760.000000  5760.000000
mean      8.500000     1.100000     1.043487     1.501562     0.995139
std       4.610172     0.424301     0.366769     0.500041     0.069558
min       1.000000     0.500000     0.021941     1.000000     0.000000
25%       4.750000     0.800000     0.790022     1.000000     1.000000
50%       8.500000     1.100000     1.013977     2.000000     1.000000
75%      12.250000     1.400000     1.269794     2.000000     1.000000
max      16.000000     1.700000     3.893956     2.000000     1.000000

         stdRepDur
count  5760.000000
mean      0.276324
std       0.106302
min       0.089569
25%       0.210917
50%       0.251763
75%       0.317479
max       0.759867
```

1.2 Quickly visualize the mean data

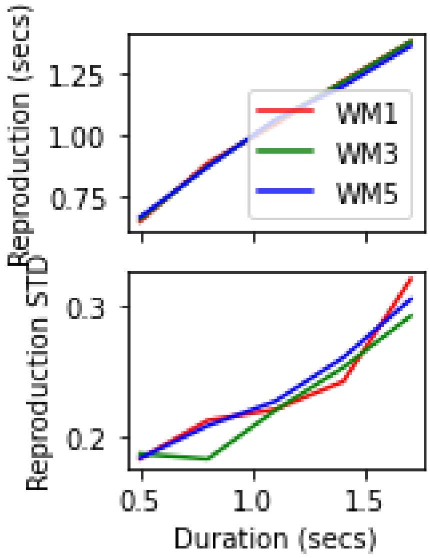
[ ]:

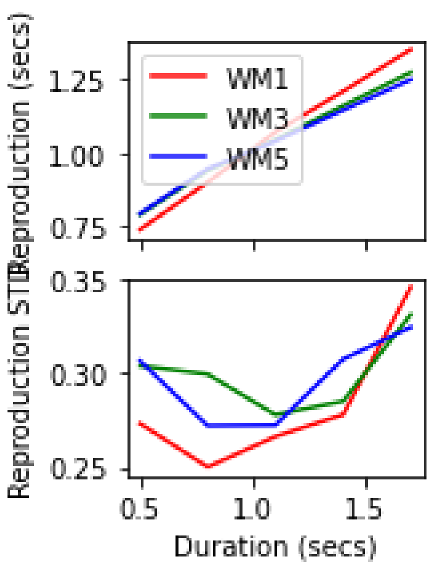
```
%matplotlib inline
def getMeans(dat, withFigure  = False):
    dat = dat.query("valid == 1 & repDur > 0.25 & repDur < 3.4")
    mdat = dat.groupby(['WMSize','curDur']).agg(
        {'repDur':['mean','std'] }).reset_index()
    #dat.pivot_table('repDur', 'curDur', 'WMSize').plot()
    # check the log distribution
    #dat.repDur.plot.hist(bins = 50)
    #plt.figure()
    #log_rep = np.log(dat.repDur)
    #log_rep.plot.hist(bins = 50)
    #
    #dat.query('curDur == 1.1').repDur.plot.hist(bins = 50)
    if withFigure:
        colors = 'rgb'
        fig, axs = plt.subplots(2, sharex = True, figsize = (2,3))
        for m in range(3):
            cur = mdat[mdat.WMSize == 1 + 2*m]
            axs[0].plot(cur.curDur, cur.repDur['mean'],colors[m])
            axs[1].plot(cur.curDur, cur.repDur['std'], colors[m])
        axs[0].legend(['WM1',"WM3","WM5"])
        axs[0].set_ylabel('Reproduction (secs)')
        axs[1].set_ylabel('Reproduction STD')
        axs[1].set_xlabel('Duration (secs)')
        plt.show(fig)
```

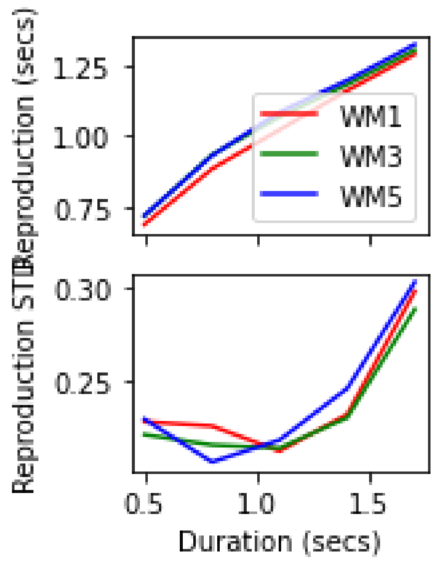
```
        return mdat, fig
```

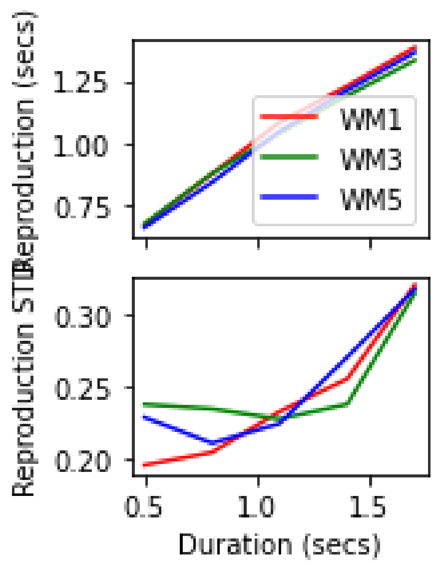
Experiment 1 Mean reproduction and standard deviation.

```
[ ]: mdats = [getMeans(raw[i], withFigure = True) for i in range(0,4)]
```









1.3 Model Framework

1. Sensory measure

$$S = ln(D) + \epsilon$$

where $S \sim N(\mu_s, \sigma_s^2)$

Influence of cognitive load M:

$$\mu_{wm} = \ln(D) - k_s \cdot M$$
$$\sigma_{wm}^2 = \sigma_s^2 + l_s \cdot M$$

2. Memory mixing

$$\mu_{post} = w_p \mu_p + (1 - w_p) u_{wm}$$

$$w_p = \frac{1/\sigma_p^2}{1/\sigma_p^2 + 1/\sigma_{wm}^2}$$

3. Duration Reproduction

$$\mu_r = e^{\mu_{post} + k_r M + \sigma_{post}^2}$$

$$\sigma_r^2 = |e^{\sigma_{post}^2 - 1}| \cdot e^{2(\mu_{post} + k_r M) + \sigma_{post}^2}$$

$$\sigma_{obs}^2 = \sigma_r^2 + \sigma_0^2 / D$$

For partial pooling, we need to store parameters for individual participants. So it is convenient to use `NSub` if it is continuous. So first we check this. In four experiments, it seems they are all tagged as 1 to 16.

[ ]:
```
[raw[i].NSub.unique() for i in range(0,4)]
```

[ ]:
```
[array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]),
 array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]),
 array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16]),
 array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16])]
```

1.4 Constrains on hierachical models.

Given that the first experiment, working memory load is after duration task, so we contrain this with no memory influence on the production and reproduction phases. That is, k_s, l_s, k_r all set to 0.

Experiment 2 manipulated the cognitive load in the production phase, so k_r sets to 0.

Experiment 3 manipulated the cognitive load in the reproduction phase, so k_s, l_s set to 0.

To implement such constrains, we use variable `constrain` with three elements to control. For example, Experiment 1 will be `constrain = [0,0,0]`.

[ ]:
```python
# set global constrain and dat before define models.
constrain = [0,0,0]
dat = raw[0]
```

[ ]:
```python
# %% partial pooling model)

def findMAP(dat, constrain):
    #prepare data
    dat = dat.query("valid == 1 & repDur > 0.25 & repDur < 3.4")
    subid = dat.NSub - 1 # starting from index 0
    nsub = len(dat.NSub.unique())  # number of subject
    wm_idx = np.intc((dat.WMSize.values-1)/2)
    durs = dat.curDur.to_numpy()
    repDur = dat.repDur
    lnDur = np.log(durs)
#    niter = 1000
    # define model
    with pm.Model() as model:
        # sensory measurement
        sig_s = pm.HalfNormal('sig_s',1., shape = nsub) # noise of the sensory␣
↪measurement
        if constrain[0] == 1:
            k_s = pm.HalfNormal('k_s',1, shape = nsub) # working memory coeff.␣
↪on ticks
        else:
            k_s = np.zeros(nsub)
        if constrain[1] == 1:
            l_s = pm.HalfNormal('l_s',1, shape = nsub) # working memory impacts␣
↪on variance
        else:
            l_s = np.zeros(nsub)
        # sensory measurement with log encoding + ticks loss by memory task
        D_s = lnDur - k_s[subid] * wm_idx
        sig_sm = sig_s[subid] + l_s[subid] * wm_idx # variance influenced by␣
↪memory tasks
        # prior (internal log encoding)
        mu_p = pm.Normal('mu_p', 0, sigma = 1,  shape = nsub) # in log space
        sig_p = pm.HalfNormal('sig_p', 1, shape = nsub) # in log-space

        if constrain[2] == 1:
            k_r = pm.HalfNormal('k_r', 1, shape = nsub) # working memory␣
↪influence on reproduction
        else:
            k_r = np.zeros(nsub)
        sig_n = pm.HalfNormal('sig_n', 1., shape = nsub) #pm.Bound( pm.
↪HalfNormal, lower = 0.15)('sig_n', 5.) # constant decision /motor noise
        # integration
```

```
        w_p = sig_sm*sig_sm / (sig_p[subid]*sig_p[subid] + sig_sm*sig_sm)
        u_x = (1-w_p)*D_s + w_p * mu_p[subid]
        sig_x2 = sig_sm*sig_sm*sig_p[subid]*sig_p[subid]/(sig_sm*sig_sm +
↪sig_p[subid]*sig_p[subid])

        # reproduction
        # reproduced duration
       u_r = np.exp(u_x + k_r[subid] * wm_idx + sig_x2/2) # reproduced duration
↪with corrupted from memory task
        #reproduced sigmas
        sig_r = np.sqrt((np.exp(sig_x2)-1)*np.exp(2*(u_x + k_r[subid] * wm_idx)
↪+sig_x2 )  +
            sig_n[subid]*sig_n[subid] /durs )

        # Data likelihood
        resp_like = pm.Normal('resp_like', mu = u_r, sigma = sig_r, observed =
↪repDur)

    # use defined model to find MAP estimation
    pMap = pm.find_MAP(model=model)
    #step = pm.Metropolis() # Have a choice of samplers
    #trace = pm.sample(niter, step, start, random_seed=123, progressbar=True)
    return  pMap
```

1.5 Results

First, find MAP estimations...

1.5.1 Experiment 1

[ ]:
```
par1 = findMAP(raw[0], [0,0,0])
par1
```

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

[ ]:
```
{'sig_s_log__': array([-1.11059913, -1.46814586, -1.59361687, -1.324336  ,
-1.9985538 ,
        -1.28770156, -1.06130114, -1.00083248, -1.82932455, -1.17116837,
        -1.3203011 , -1.99391276, -1.73510186, -1.72536309, -1.38357997,
        -1.38252838]),
 'mu_p': array([-0.09633093,  0.1111805 , -0.00192529, -0.04646086, -0.07134838,
        -0.16272582, -0.37433608, -0.02229598,  0.26657169, -0.04467419,
         0.00193831, -0.61239433,  0.15873427,  0.07717257, -0.03351871,
```

```
        -0.11909716]),
 'sig_p_log__': array([-1.65542764, -1.47937026, -1.65559368, -1.57881482,
-1.26800085,
        -1.53859173, -1.18626388, -0.89478734, -1.41697207, -1.21390423,
        -1.16841205, -0.84967163, -1.26518612, -0.51374081, -1.06600275,
        -1.05561488]),
 'sig_n_log__': array([-2.49228385, -2.35931193, -2.4807592 , -1.86665601,
-7.09189706,
        -3.44697185, -2.28334974, -2.86171932, -2.24815674, -3.10307166,
        -3.29319159, -2.50952788, -2.88693029, -2.70129816, -2.31905133,
        -2.71508885]),
 'sig_s': array([0.32936157, 0.2303522 , 0.20318937, 0.26597951, 0.13553115,
        0.2759042 , 0.34600532, 0.36757332, 0.16052196, 0.31000453,
        0.26705488, 0.13616161, 0.17638223, 0.17810837, 0.25067952,
        0.25094327]),
 'sig_p': array([0.19101035, 0.22778109, 0.19097864, 0.20621936, 0.28139361,
        0.21468322, 0.30536   , 0.4086945 , 0.24244702, 0.29703532,
        0.31086018, 0.42755531, 0.28218677, 0.59825343, 0.34438235,
        0.3479784 ]),
 'sig_n': array([0.08272083, 0.09448521, 0.08367967, 0.15463991, 0.00083182,
        0.03184191, 0.10194215, 0.05717038, 0.10559368, 0.04491104,
        0.03713514, 0.08130662, 0.05574708, 0.06711833, 0.09836686,
        0.06619907])}
```

Define a prediction function for all experiments.

[ ]:
```
# see the goodness of fit
def mapPrediction(dat, par1, constrain):
    dat = dat.query("valid == 1 & repDur > 0.25 & repDur < 3.4")
    subid = dat.NSub - 1 # starting from index 0
    wm_idx = np.intc((dat.WMSize.values-1)/2)
    durs = dat.curDur.to_numpy()
    lnDur = np.log(durs)

    # sensory measure
    if constrain[0] == 1:
        u_wm = lnDur - par1['k_s'][subid] * wm_idx
    else:
        u_wm = lnDur
    if constrain[1] == 1:
        s_wm = par1['sig_s'][subid]  + par1['l_s'][subid] * wm_idx
    else:
        s_wm = par1['sig_s'][subid]

    #weight of prior
    w_p = s_wm*s_wm / (par1['sig_p'][subid]*par1['sig_p'][subid] + \
        s_wm*s_wm)
```

```
    # integration
    u_post = (1-w_p)*u_wm + w_p * par1['mu_p'][subid]
    s_post = s_wm*s_wm*par1['sig_p'][subid]*par1['sig_p'][subid]/(s_wm*s_wm + \
        par1['sig_p'][subid]*par1['sig_p'][subid])

    # reproduction
    if constrain[2] == 1:
        u_r = np.exp(u_post + par1['k_r'][subid] * wm_idx + s_post/2) #␣
↪reproduced duration with corrupted from memory task
    else:
        u_r = np.exp(u_post + s_post/2) # reproduced duration with corrupted␣
↪from memory task

    sig_r = np.sqrt((np.exp(s_post)-1)*np.exp(2*(u_post) +s_post )  +␣
↪par1['sig_n'][subid]*par1['sig_n'][subid] /durs )

    dat.loc[:,'predDur'] = u_r
    dat.loc[:,'predSig'] = sig_r
    mdat = dat.groupby(['NSub','curDur','WMSize']).agg(
        {'repDur':['mean','std'],'predDur':'mean', 'predSig' : 'mean'}
        ).droplevel(axis=1, level=0).reset_index()
    # flaten column names, making names ambigous, rename them
    mdat.columns = ['Nsub', 'curDur','WMSize','mRep', 'sdRep', 'mPred','sdPred']
    return mdat
```

Now we do average for individual participants behavioral results and predictions.

[ ]:
```
# get predictions
mdat1 = mapPrediction(raw[0],par1, [0,0,0])
mdat1.head()
```

```
/Users/strongway/opt/miniconda3/envs/pymc3_env/lib/python3.9/site-
packages/pandas/core/indexing.py:1667: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = value
```

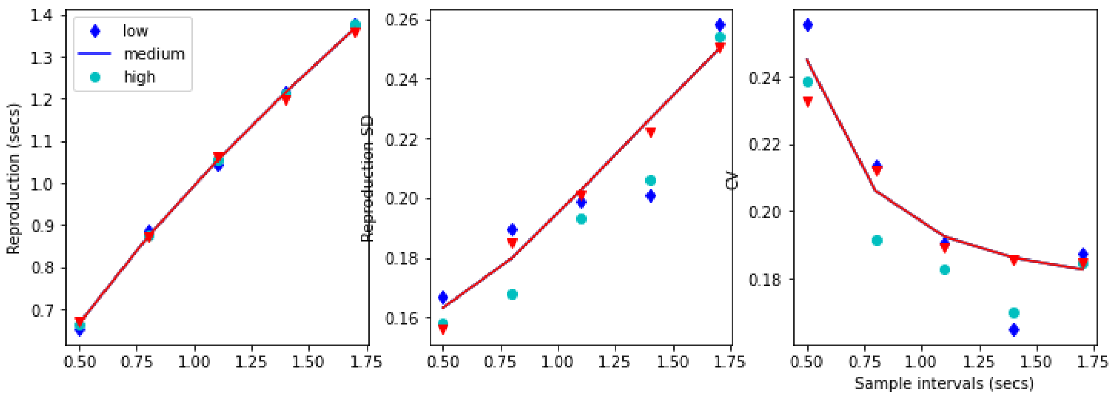
[ ]:

	Nsub	curDur	WMSize	mRep	sdRep	mPred	sdPred
0	1	0.5	1	0.723200	0.166440	0.792242	0.176232
1	1	0.5	3	0.825337	0.203701	0.792242	0.176232
2	1	0.5	5	0.806658	0.161117	0.792242	0.176232
3	1	0.8	1	0.908631	0.132347	0.891726	0.174822
4	1	0.8	3	0.930464	0.196054	0.891726	0.174822

Now we obtain grand means for plotting…

```
[ ]: # define a plotting function
     #Visualize
     def plotPred(mmdat):
         markers = 'dov'
         colors = 'bcr'
         fig, axs = plt.subplots(1, 3, figsize = (12,4))
         for m in range(3):
             cur = mmdat[mmdat.WMSize == 1 + 2*m]
             axs[0].plot(cur.curDur, cur.mRep,markers[m]+colors[m])
             axs[0].plot(cur.curDur, cur.mPred,colors[m])
             axs[1].plot(cur.curDur, cur.sdRep,markers[m]+colors[m])
             axs[1].plot(cur.curDur, cur.sdPred,colors[m])
             axs[2].plot(cur.curDur, cur.sdRep/cur.mRep,markers[m]+colors[m])
             axs[2].plot(cur.curDur, cur.sdPred/cur.mPred,colors[m])
         axs[0].legend(['low',"medium","high"])
         axs[0].set_ylabel('Reproduction (secs)')
         axs[1].set_ylabel('Reproduction SD')
         axs[2].set_ylabel('CV')
         axs[2].set_xlabel('Sample intervals (secs)')
         plt.show(fig)
```

```
[ ]: #grand average
     mmdat1 = mdat1.groupby(['curDur','WMSize']).agg(
         {'mRep':'mean', 'sdRep':'mean','mPred':'mean','sdPred':'mean'}).
      ↪reset_index()
     plotPred(mmdat1)
```



Do the same for Exps. 2, 3, and 4.

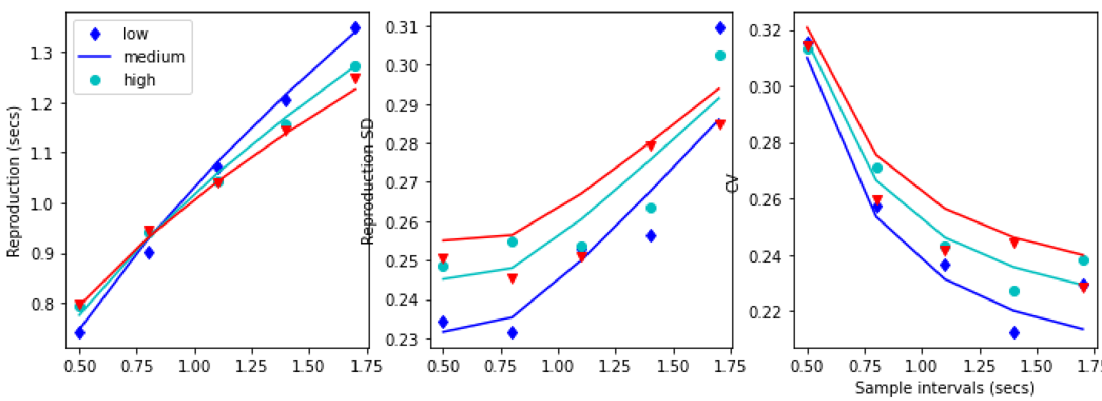
1.5.2 Experiment 2

```
[ ]: par2 = findMAP(raw[1], [1,1,0])
     mdat2 = mapPrediction(raw[1],par2, [1,1,0])
     mmdat2 = mdat2.groupby(['curDur','WMSize']).agg(
         {'mRep':'mean', 'sdRep':'mean','mPred':'mean','sdPred':'mean'}).
      ↪reset_index()
     plotPred(mmdat2)
```

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

```
/Users/strongway/opt/miniconda3/envs/pymc3_env/lib/python3.9/site-
packages/pandas/core/indexing.py:1667: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = value
```



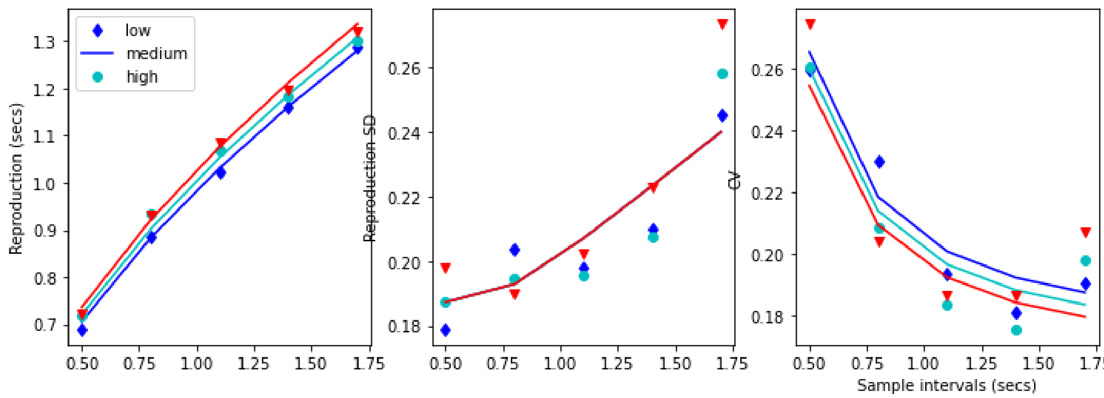
1.5.3 Experiment 3

```
[ ]: par3 = findMAP(raw[2], [0,0,1])
     mdat3 = mapPrediction(raw[2],par3, [0,0,1])
     mmdat3 = mdat3.groupby(['curDur','WMSize']).agg(
         {'mRep':'mean', 'sdRep':'mean','mPred':'mean','sdPred':'mean'}).
      ↪reset_index()
     plotPred(mmdat3)
```

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

```
/Users/strongway/opt/miniconda3/envs/pymc3_env/lib/python3.9/site-
packages/pandas/core/indexing.py:1667: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = value
```



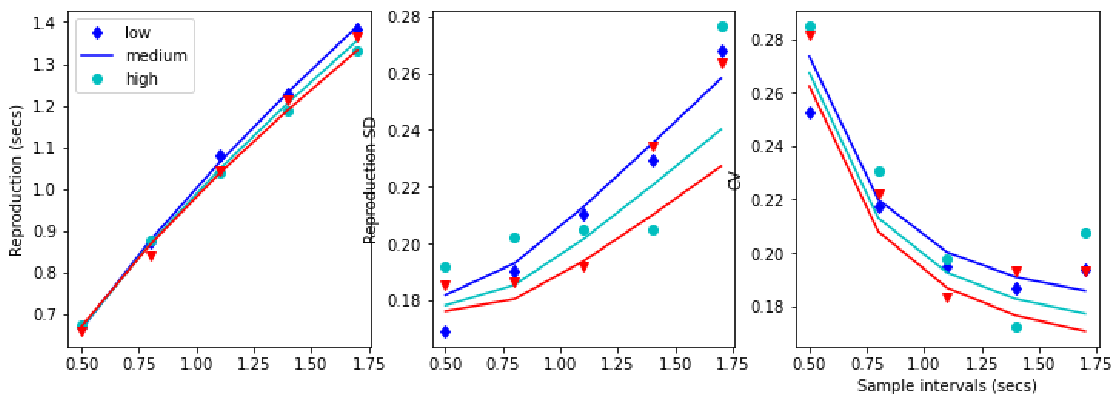
1.5.4 Experiment 4

```
[ ]: par4 = findMAP(raw[3], [1,1,1])
mdat4 = mapPrediction(raw[3],par4, [1,1,1])
mmdat4 = mdat4.groupby(['curDur','WMSize']).agg(
    {'mRep':'mean', 'sdRep':'mean','mPred':'mean','sdPred':'mean'}).
 ↪reset_index()
plotPred(mmdat4)
```

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

```
/Users/strongway/opt/miniconda3/envs/pymc3_env/lib/python3.9/site-
packages/pandas/core/indexing.py:1667: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[key] = value
```



1.6 Save parameters and predictions

Now we save parameters and predictions to csv files for further R statistical analyses.

[ ]:
```
pd.DataFrame(par1).to_csv(cwd+'/../data/'+'par_exp1.csv')
pd.DataFrame(par2).to_csv(cwd+'/../data/'+'par_exp2.csv')
pd.DataFrame(par3).to_csv(cwd+'/../data/'+'par_exp3.csv')
pd.DataFrame(par4).to_csv(cwd+'/../data/'+'par_exp4.csv')
mdat1.to_csv(cwd+'/../data/'+'mpred1.csv')
mdat2.to_csv(cwd+'/../data/'+'mpred2.csv')
mdat3.to_csv(cwd+'/../data/'+'mpred3.csv')
mdat4.to_csv(cwd+'/../data/'+'mpred4.csv')
```