

# Tutorial-Copy1

November 24, 2021

## 1 Tutorial for the Bayesian machine scientist

This tutorial illustrates how to program a Bayesian machine scientist, using the code provided here. The tutorial assumes general knowledge of Python programming. We start by importing all necessary Python modules:

```
[1]: import sys
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

import matplotlib.pyplot as plt
from copy import deepcopy
from ipywidgets import IntProgress
from IPython.display import display, display_latex

sys.path.append('./')
sys.path.append('./Prior/')
from mcmc import *
from parallel import *
from fit_prior import read_prior_par
```

### 1.1 Loading and preparing the data

We then load the data. In this particular case, we load the salmon stocks data. The features (independent variables) are loaded into a Pandas `DataFrame` named `x`, whereas the target (dependent) variable is loaded into a Pandas `Series` named `y`. Data should **always** be loaded in these formats to avoid problems.

```
[2]: XLABS = [
    'x1',
    'x2'
]
def f1(x):
    return 2.5+x*np.sin(x)
def f2(x1,x2):
    return x1*(1+x2)*np.cos(x2)
```

```

x1=np.arange(-2.5,2.5,0.5)
x2=np.arange(-2.5,2.5,0.5)
x=pd.DataFrame(columns={'x1','x2'})
for a in x1:
    for b in x2:
        x=x.append({'x1':a,'x2':b},ignore_index=True)

y = pd.Series(f2(x['x1'],x['x2']))#+np.random.normal(0,0.1,size=len(x.x1)))
x,y

```

```

[2]: (   x1  x2
0 -2.5 -2.5
1 -2.5 -2.0
2 -2.5 -1.5
3 -2.5 -1.0
4 -2.5 -0.5
.. ... ..
95  2.0  0.0
96  2.0  0.5
97  2.0  1.0
98  2.0  1.5
99  2.0  2.0

[100 rows x 2 columns],
0    -3.004289
1    -1.040367
2     0.088422
3    -0.000000
4    -1.096978
...
95     2.000000
96     2.632748
97     2.161209
98     0.353686
99    -2.496881
Length: 100, dtype: float64)

```

## 1.2 Initializing the Bayesian machine scientist

We start by initializing the machine scientist. This involves three steps: - **Reading the prior hyperparameters.** The values of the hyperparameters depend on the number of variables `nv` and parameters `np` considered during the search. Many combinations of `nv` and `np` have hyperparameters calculated in the `Prior` directory. Otherwise, the hyperparameters should be fit. - **Setting the “temperatures” for the parallel tempering.** If you don’t know what parallel tempering is, you can read it in the Methods section of the paper, or just leave it as is in the code. In general, more temperatures (here 20) lead to better sampling of the expression space (we use a maximum of 100 different temperatures) - **Initializing the (parallel) scientist.**

```
[3]: # Read the hyperparameters for the prior
prior_par = read_prior_par('./Prior/final_prior_param_sq.named_equations.nv2.
↳np2.2016-09-09 18:49:43.038278.dat')

# Set the temperatures for the parallel tempering
Ts = [1] + [1.04**k for k in range(1, 20)]

# Initialize the parallel machine scientist
pms = Parallel(
    Ts,
    variables=XLABS,
    parameters=['a%d' % i for i in range(2)],
    x=x, y=y,
    prior_par=prior_par,
)
```

### 1.3 Sampling expressions with the Bayesian machine scientist

We are now ready to start sampling expressions with the Bayesian machine scientist, using MCMC. In its simplest form, one just needs to run the `mcmc_step()` and the `tree_swap()` methods as many times as necessary. `mcmc_step()` performs an MCMC update at each of the temperatures of the parallel tempering, whereas `tree_swap()` attempts to swap the expressions at two consecutive temperatures.

```
[4]: %%time
best_description_lengths, lowest_md1, best_model = [], np.inf, None
display_latex(best_model)
for run in range(5):
    # Initialize the parallel machine scientist
    pms = Parallel(
        Ts,
        variables=XLABS,
        parameters=['a%d' % i for i in range(2)],
        x=x, y=y,
        prior_par=prior_par,
    )
    # Number of MCMC steps
    nstep = 5000

    # Draw a progress bar to keep track of the MCMC progress
    f = IntProgress(min=0, max=nstep, description='Running equilibration:') #_
    ↳instantiate the bar
    #display(f)

    # MCMC
    for i in range(nstep):
        # MCMC update
```

```

pms.mcmc_step() # MCMC step within each T
pms.tree_swap() # Attempt to swap two randomly selected consecutive
↳ temps
    # Update the progress bar
    f.value += 1
    # Number of MCMC steps
nstep = 5000

# Draw a progress bar to keep track of the MCMC progress
f = IntProgress(min=0, max=nstep, description='Running production:') #
↳ instantiate the bar
    #display(f)

# MCMC
description_lengths, mdl, mdl_model = [], np.inf, None
for i in range(nstep):
    # MCMC update
    pms.mcmc_step() # MCMC step within each T
    pms.tree_swap() # Attempt to swap two randomly selected consecutive
↳ temps
        # Add the description length to the trace
        description_lengths.append(pms.t1.E)
        # Check if this is the MDL expression so far
        if pms.t1.E < mdl:
            mdl, mdl_model = pms.t1.E, deepcopy(pms.t1)
        # Update the progress bar
        f.value += 1
    if best_model==None:
        best_description_lengths,lowest_mdl,best_model=description_lengths,mdl,
↳ deepcopy(mdl_model)
        if mdl<lowest_mdl:
            best_description_lengths=deepcopy(description_lengths)
            lowest_mdl=deepcopy(mdl)
            best_model=deepcopy(mdl_model)
print(f"Run {run+1}")
print(mdl, mdl_model)

```

Run 1

```

-9461.58351976943 (((x1 * cos(abs(x2))) * (abs(_a1_) + ((sqrt(x2) ** 2) *
_a1_))) + (_a0_ ** 3))

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳ verbose)
    731         try:                        # We've seen this canonical before!

```

```
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733             except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'x1*sin(I*x2+exp(abs(x2*exp(c1))))'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

Run 2

```
-37230.6769631012 (((x1 * (((_a1_ ** _a1_) + x2) + _a1_) + _a1_)) * cos(x2)) +
((_a1_ + (_a1_ * (x2 * ((_a1_ ** x1) * exp((_a0_ / x1)))))) * _a0_))
```

Run 3

```
-3546.52516291065 (cos(-(x2 * (_a1_ + _a1_))) * (((abs(_a0_) * x1) / _a0_) *
((_a0_ ** _a1_) + x2)))
```

```
-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731             try:                # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733             except KeyError: # Never seen this canonical formula before:
```

```
KeyError: '-c1*x1*(exp(exp(x2))+I*pi)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

```
-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731             try:                # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733             except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'c1*(c1+2*x1)*(exp(exp(x2))+I*pi)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

```
-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731         try:                # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'c1*x1*(exp(exp(x2))+I*pi)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

Run 4

```
-3790.53701739186 ((((_a0_ * x1) * (x2 + _a1_)) * log(_a0_)) * (_a1_ * cos(x2)))
```

```
-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731         try:                # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'c1*cos(c1+c2+x2)+c1+c2+x1*(x2+I)*cos(-c1+2*c2+x2)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

```

KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳ verbose)
    731         try:                        # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳ representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:

KeyError: 'c1*cos(x2)+c1+c2+x1*(x2+I)*cos(c1*x1+2*c2+x2)'

```

During handling of the above exception, another exception occurred:

```

TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳ according to the rule 'safe'

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳ verbose)
    731         try:                        # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳ representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:

KeyError: 'c1*cos(x2)+c1+c2+x1*(x2+I)*cos(c2*x2+c2+x2)'

```

During handling of the above exception, another exception occurred:

```

TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳ according to the rule 'safe'

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳ verbose)
    731         try:                        # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳ representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:

KeyError: 'c1*cos(x2)+c1+c2+x1*(x2+I)*cos(x2)'

```

During handling of the above exception, another exception occurred:

```

TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731         try:                # We've seen this canonical before!
--> 732         rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:

KeyError: 'c1+c2*cos(x2)+c2+x1*(x2+I)*cos(x2)'

```

During handling of the above exception, another exception occurred:

```

TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731         try:                # We've seen this canonical before!
--> 732         rep, rep_energy, rep_par_values = self.
↳representative[canonical]
    733         except KeyError: # Never seen this canonical formula before:

KeyError: 'c1+c2*cos(x2)+c2+x1*(x2+sqrt(2)*I/2)*cos(x2)'

```

During handling of the above exception, another exception occurred:

```

TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'

```

```

-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
    731         try:                # We've seen this canonical before!
--> 732         rep, rep_energy, rep_par_values = self.
↳representative[canonical]

```



```
733         except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'c1*cos(x2)+x1*(x2+I)*cos(x2)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

```
-----
KeyError                                Traceback (most recent call last)
~/Documents/rguimera-machine-scientist/mcmc.py in update_representative(self,
↳verbose)
```

```
731         try:                                # We've seen this canonical before!
--> 732             rep, rep_energy, rep_par_values = self.
↳representative[canonical]
733         except KeyError: # Never seen this canonical formula before:
```

```
KeyError: 'c1*cos(x2)+x1*(x2+I)*cos(c2+x2)'
```

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent call last)
TypeError: Cannot cast array data from dtype('complex128') to dtype('float64')
↳according to the rule 'safe'
```

Run 5

```
-37272.7993788333 (((x2 + _a1_) + -((_a1_ / _a0_))) * (x1 * cos(x2)))
```

CPU times: user 1h 25min 4s, sys: 19.1 s, total: 1h 25min 23s

Wall time: 1h 24min 15s

So let's take a look at the objects we stored. Here is the best model sampled by the machine scientist:

```
[5]: print('Best model:\t', best_model)
      print('Desc. length:\t', lowest_md1)
```

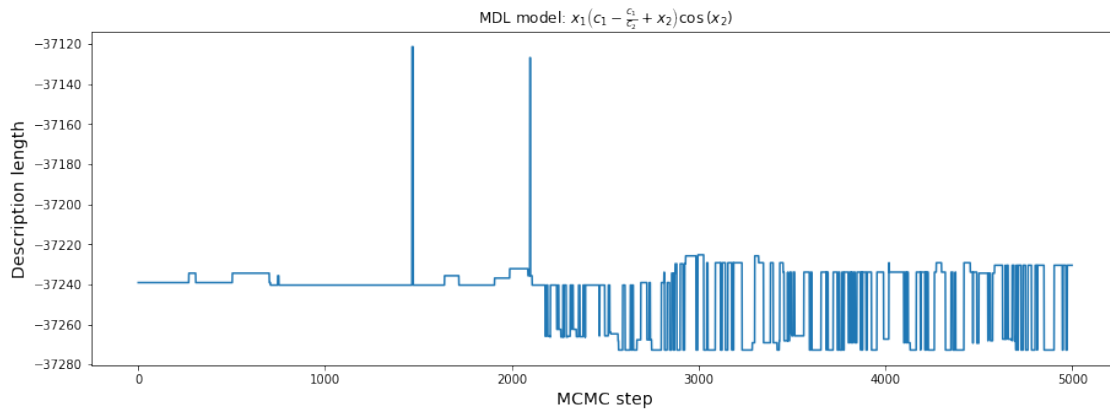
```
Best model:      (((x2 + _a1_) + -((_a1_ / _a0_))) * (x1 * cos(x2)))
```

```
Desc. length:    -37272.7993788333
```

And here is the trace of the description length:

```
[6]: plt.figure(figsize=(15, 5))
      plt.plot(best_description_lengths)
      plt.xlabel('MCMC step', fontsize=14)
      plt.ylabel('Description length', fontsize=14)
      plt.title('MDL model:  $\%s\%$ ' % best_model.latex())
```

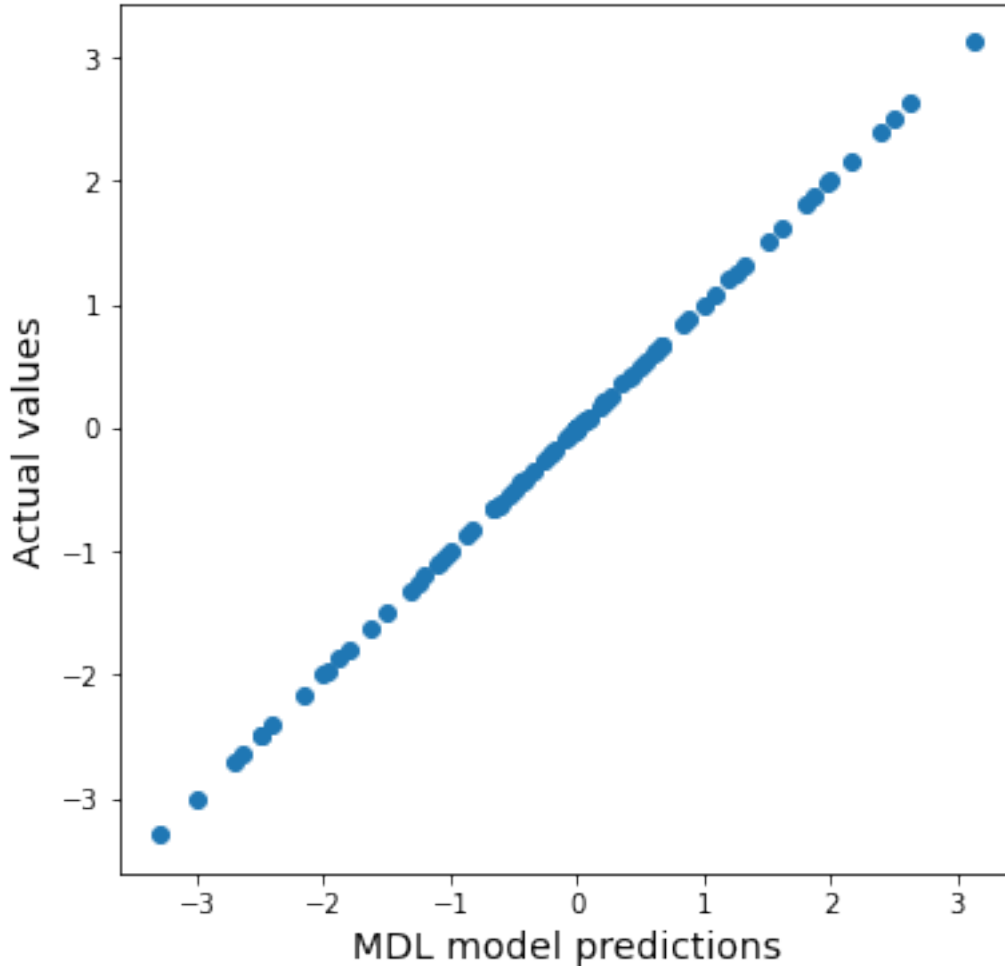
```
plt.show()
```



## 1.4 Making predictions with the Bayesian machine scientist

Finally, we typically want to make predictions with models. In this regard, the interface of the machine scientist is similar to those in Scikit Learn: to make a prediction we call the `predict(x)` method, with an argument that has the same format as the training `x`, that is, a Pandas `DataFrame` with the exact same columns.

```
[7]: plt.figure(figsize=(6, 6))
plt.scatter(best_model.predict(x), y)
#plt.plot((-6, 0), (-6, 0))
plt.xlabel('MDL model predictions', fontsize=14)
plt.ylabel('Actual values', fontsize=14)
plt.show()
```



### 1.5 Further refinements

The examples above are only intended to illustrate how a basic MCMC would be implemented. In practice, there are other considerations that we kept in mind in all the experiments reported in the manuscript, and that anyone using the code should too: - **Equilibration**: One should not start sampling until the MCMC has converged to the stationary distribution. Although determining when a sample is in equilibrium, a necessary condition is that the description length is not increasing or, more typically, decreasing. The trace of the description length should be flat (except for fluctuations) before we start collecting samples. - **Thinning**: MCMC samples should be thinned, so only one in, say, 100 samples are kept for the trace. Otherwise, one is getting highly correlated samples, which may lead to, for example, erroneous estimates of confidence intervals. - **Getting trapped**: Despite the parallel tempering, the MCMC can get trapped in local minima of the description length. For this, we typically keep track of the number of steps since the last `tree_swap()` move was accepted for each temperature. If a particular temperature has *not* accepted swaps in a long time, then we anneal the whole system, that is, we increase all temperatures and decrease them slowly back to equilibrium so as to escape the local minima. Using several restarts of the MCMC and comparing the results is also a convenient check. - **Memory issues**: By default, the

machine scientist keeps a cache of all visited models, so as to avoid duplicates of previously considered models, as well as to speed up the process of obtaining the maximum likelihood estimators of the model parameters. For long MCMC chains this becomes memory intensive, so it may be convenient to periodically clean this cache (or, at least, old models in this cache) by reinitializing the `fit_pat` and representative attributes of the `Parallel` instance.

```
[8]: %%javascript
      IPython.notebook.save_notebook()
```

<IPython.core.display.Javascript object>

```
[9]: save=input()
      if save=='T':
          import os
          from datetime import datetime
          os.system("jupyter nbconvert --output-dir='./data/Tests/nd2pdf/' --output_
          ↳ 'date_{}.pdf' --to pdf Tutorial-Copy1.ipynb".format(datetime.now()))
```

T

```
[NbConvertApp] Converting notebook Tutorial-Copy1.ipynb to pdf
[NbConvertApp] Support files will be in date_2021-11-19 10:05:18.473159_files/
[NbConvertApp] Making directory ./date_2021-11-19 10:05:18.473159_files
[NbConvertApp] Making directory ./date_2021-11-19 10:05:18.473159_files
[NbConvertApp] Writing 59339 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 110816 bytes to data/Tests/nd2pdf/date_2021-11-19
10:05:18.473159.pdf

"""
if i%2000==0:
    print('Delete cache',len(pms.trees['1'].fit_par),len(pms.trees['1'].representative))
    pms.trees['1'].fit_par={}
    pms.trees['1'].representative={}

print('Delete                cache',len(pms.trees['1'].fit_par),len(pms.trees['1'].representative))
print(pms.trees['1'].fit_par) pms.trees['1'].fit_par={} pms.trees['1'].representative={} """
```