pyabc_tutorial_solutions

August 26, 2022

1 pyABC Tutorial

This is a tutorial for the python package **pyABC** for **likelihood-free parameter inference using approximate Bayesian computation**. See: * https://github.com/ICB-DCM/pyABC for the code * https://pyabc.readthedocs.io/en/latest/ for the documentation and examples * https://pyabc.readthedocs.io/en/latest/cite.html for the papers * https://github.com/yannikschaelte/pres_pyabc_inverse_2022 for presentation slides and further material

1.1 Install

1.1.1 Python

This step can be skipped if you already have python>=3.8 (check via `python3 --version`). Download the conda package manager via the mini distribution Miniconda, see the instructions. Specifically, for Linux, execute

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p miniconda3
rm Miniconda3-latest-Linux-x86_64.sh
```

1.1.2 Custom environment

We will work in a fresh local environment:

```
eval "$(miniconda3/bin/conda shell.bash hook)"
conda create -y -n invenv python
```

1.1.3 (Re-)Activate environment

To activate the local environment, in each new shell session execute:

```
eval "$(miniconda3/bin/conda shell.bash hook)"
conda activate invenv
```

To verify your installation: `which pip` should now point to your newly created environment.

1.1.4 Install dependencies

We install pyABC, as well as the interactive jupyter platform:

```
pip install jupyterlab pyabc
```

To open a jupyter notebook (e.g. this one):

```
jupyter lab [DIR]
```

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pyabc

pyabc.settings.set_figure_params('pyabc')  # for beautified plots
```

1.2 Toy example

As toy example, we consider a model $y \sim \mathcal{N}(\mu, \sigma^2)$ with fixed variance $\sigma^2 = 0.5^2$ and unknown univariate mean μ. We have observed data $y_{\text{obs}} = 3.5$, and assume prior knowledge that the true man is in the interval $\mu \in [1, 10]$.

1.2.1 Own ABC

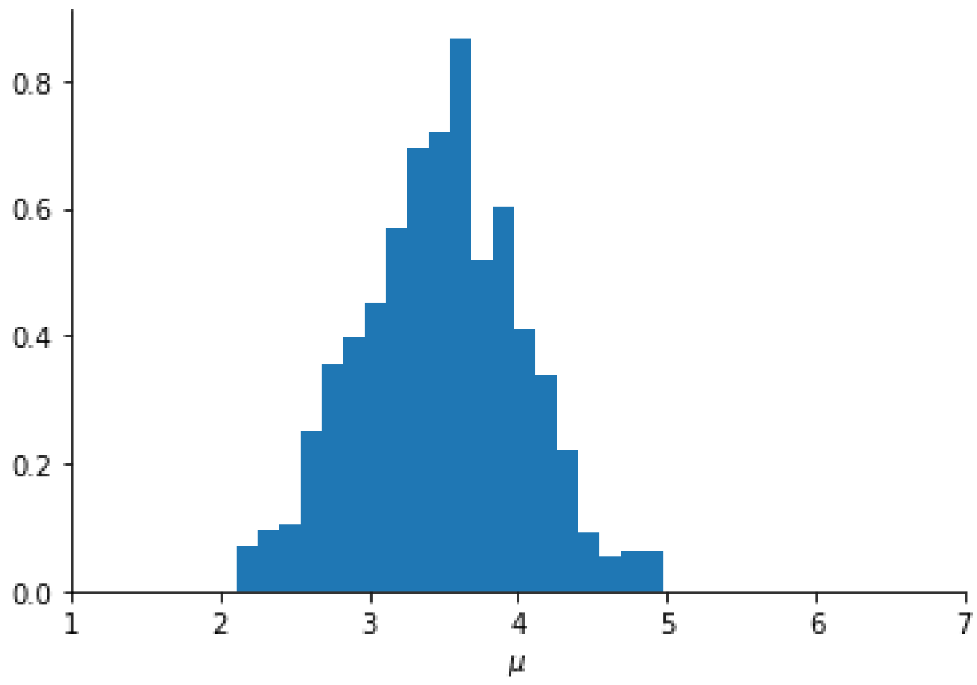
To demonstrate how simple ABC is at its core, let us first implement an own rejection algorithm with quadratic distance, acceptance threshold $\varepsilon = 0.1$, and population size $N = 1000$. Define model, distance, and prior:

[2]:
```python
from scipy.stats import uniform, norm

# observed data
y_obs = 3.5

# noise standard deviation
sigma = 0.5

# model
def model(p):
    return p + sigma * np.random.normal()

# distance
def distance(y, y_obs):
    return np.sum((y - y_obs)**2)

# prior
xmin = 1
xmax = 7
prior = uniform(xmin, xmax-xmin)
```

Implement the ABC algorithm and plot the posterior:

[3]:
```python
# population size
N = 1000

# acceptance threshold
```

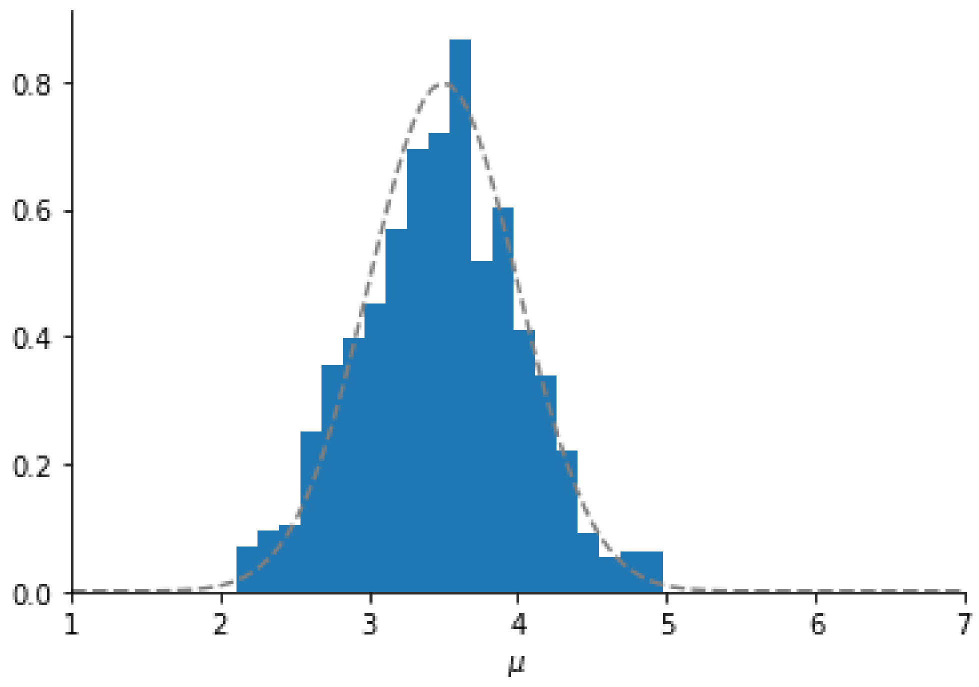
```python
eps = 0.1

# accepted parameters
thetas = []

# ABC loop: sample, simulate, evaluate until enough acceptances
while len(thetas) < N:
    theta = prior.rvs()
    y = model(theta)
    if distance(y, y_obs) < eps:
        thetas.append(theta)

# plot
fig, ax = plt.subplots()
ax.hist(thetas, bins=int(N / 50), density=True)
ax.set_xlim(xmin, xmax)
ax.set_xlabel(r"$\mu$");
```



We can compare this to the true posterior:

[4]:
```python
from scipy.integrate import quad

likelihood = lambda p: norm.pdf(loc=y_obs, scale=sigma, x=p)

posterior_unnormalized = lambda p: likelihood(p) * prior.pdf(p)
```

```
posterior_norm = quad(posterior_unnormalized, xmin, xmax)[0]
posterior = lambda p: posterior_unnormalized(p) / posterior_norm

fig, ax = plt.subplots()
ax.hist(thetas, bins=int(N / 50), density=True)
ax.set_xlim(xmin, xmax)
ax.set_xlabel(r"$\mu$")

xs = np.linspace(xmin, xmax, 100)
true_ys = posterior(xs)
ax.plot(xs, true_ys, color="grey", linestyle="dashed");
```

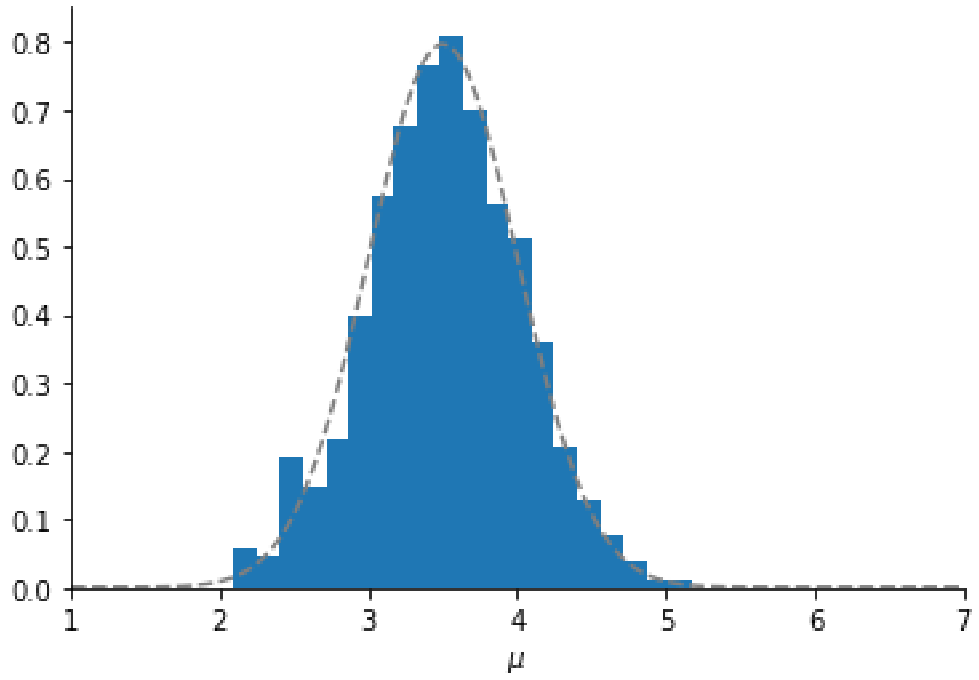


1.2.2 Own ABC-IS

As the prior may be uninformative, in importance sampling we use a different proposal distribution to sample from (and afterwards reweight the particles by prior divided by proposal):

[5]:
```
from scipy.stats import norm

# proposal distribution
proposal = norm(loc=7, scale=2)

# accepted parameters and corresponding weights
thetas = []
weights = []
```

```
# ABC loop: sample, simulate, evaluate until enough acceptances
while len(thetas) < N:
    theta = proposal.rvs()
    y = model(theta)
    if distance(y, y_obs) < eps:
        thetas.append(theta)
        weight = prior.pdf(theta) / proposal.pdf(theta)
        weights.append(weight)

# plot
fig, ax = plt.subplots()
ax.hist(thetas, weights=weights, bins=int(N / 50), density=True)
ax.set_xlim(xmin, xmax)
ax.set_xlabel(r"$\mu$")

xs = np.linspace(xmin, xmax, 100)
ax.plot(xs, posterior(xs), color="grey", linestyle="dashed");
```



What is our effective sample size? It is given as:

$$\text{ESS} = \frac{\left(\sum_i w_i\right)^2}{\sum_i w_i^2}$$

[6]:
```
weights = np.asarray(weights)
ess = weights.sum()**2 / (weights**2).sum()
print(ess)
```

```
814.0092824552288
```

1.2.3 In pyABC

Now, let's use pyABC to perform the same analysis with its implemented ABC-SMC algorithm.

Problem description Specify the model:

[7]:
```
sigma = 0.5

# pyABC accepts arbitrary model functions that return a dictionary of
#  observed values

# more precisely, the following structure is assumed:
#  model(p: Parameter) -> Simulation
#  where Parameter = dict[str, float]
#  and Simulation = dict[str, {float, np.ndarray, pd.DataFrame}]

def model(p):
    return {"y": p["mu"] + sigma * np.random.normal()}
```

The observed data are:

[8]:
```
y_obs = {"y": 3.5}
```

The prior is:

[9]:
```
prior = pyabc.Distribution(mu=pyabc.RV("uniform", xmin, xmax-xmin))
```

Distance function:

[10]:
```
# distances are assumed to follow the structure
#  distance(y: Simulation, y_obs: Simulation) -> float

def distance(y, y_obs):
    return (y["y"] - y_obs["y"])**2
```

ABC analysis The problem being defined, we continue by setting up an `ABCSMC` analysis with $N = 1000$ samples per generation. We have to specify where to log the analysis results. Then, we're all good and can run an analysis that continues until 10 generations have been created, or the acceptance rate falls below 0.01:

[11]:
```
# population size
N = 1000
# ABCSMC instance
```

```
abc = pyabc.ABCSMC(model, prior, distance, population_size=N)

# local sqlite database
db_path = "test.db"
abc.new("sqlite:///" + db_path, y_obs)

history = abc.run(max_nr_populations=10, minimum_epsilon=0.1)
```

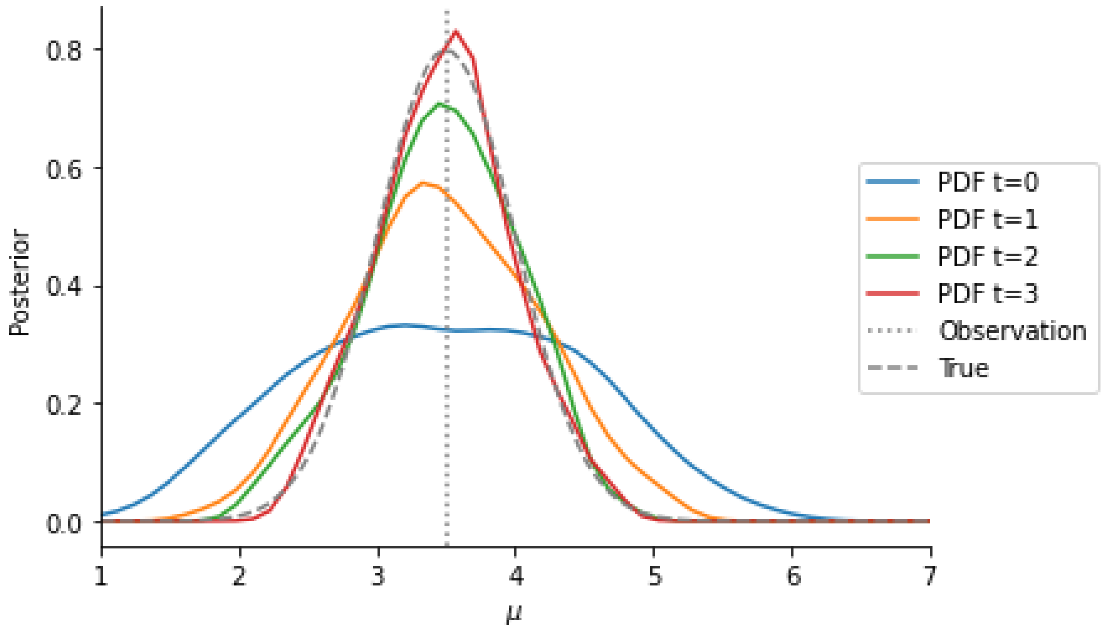
```
ABC.Sampler INFO: Parallelize sampling on 8 processes.
ABC.History INFO: Start <ABCSMC id=1, start_time=2022-08-25 22:54:56>
ABC INFO: Calibration sample t = -1.
ABC INFO: t: 0, eps: 2.14673451e+00.
ABC INFO: Accepted: 1000 / 2085 = 4.7962e-01, ESS: 1.0000e+03.
ABC INFO: t: 1, eps: 5.90703265e-01.
ABC INFO: Accepted: 1000 / 2060 = 4.8544e-01, ESS: 9.7191e+02.
ABC INFO: t: 2, eps: 1.46762388e-01.
ABC INFO: Accepted: 1000 / 3027 = 3.3036e-01, ESS: 8.6851e+02.
ABC INFO: t: 3, eps: 3.99515519e-02.
ABC INFO: Accepted: 1000 / 4892 = 2.0442e-01, ESS: 7.7220e+02.
ABC INFO: Stop: Minimum epsilon.
ABC.History INFO: Done <ABCSMC id=1, duration=0:00:09.837582,
end_time=2022-08-25 22:55:06>
```

1.2.4 Visualization and analysis

Let us visualize the ABC posterior approximation over the generations:

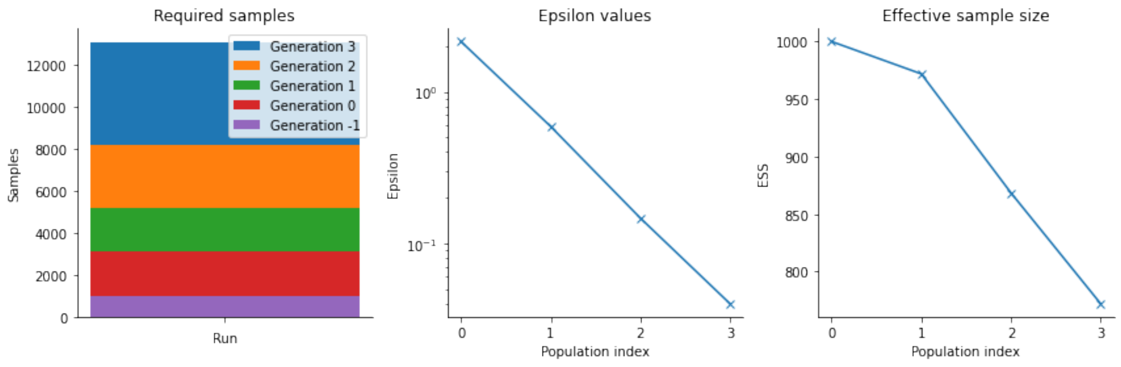
[12]:
```
fig, ax = plt.subplots()
for t in range(history.max_t + 1):
    pyabc.visualization.plot_kde_1d_highlevel(
        history,
        t=t,
        xmin=xmin,
        xmax=xmax,
        x="mu",
        xname=r"$\mu$",
        ax=ax,
        label=f"PDF t={t}",
    )
ax.axvline(x=y_obs["y"], linestyle="dotted", color="grey", label="Observation")

ax.plot(xs, true_ys, color="grey", linestyle="dashed", label="True")

ax.legend(bbox_to_anchor=(0.9, 0.5), loc="center left")
```

[12]:
```
<matplotlib.legend.Legend at 0x7fc48e10b4c0>
```



Next, let us analyze the performance of the sampler by plotting sample numbers, epsilon threshold, and effective sample sizes over the generations:

[13]:
```
fig, arr_ax = plt.subplots(1, 3, figsize=(12, 4))

pyabc.visualization.plot_sample_numbers(history, ax=arr_ax[0])
pyabc.visualization.plot_epsilons(history, ax=arr_ax[1])
pyabc.visualization.plot_effective_sample_sizes(history, ax=arr_ax[2])

fig.tight_layout()
```



That's it, you have run your first pyABC analysis. For more details, illustrations of the various algorithms pyABC implements, and application examples, see https://pyabc.readthedocs.io/en/latest/examples.html.