pybuck

Release 0.1

Jan 15, 2022

Contents:

CHAPTER 1

Overview

This package supports dimensional analysis in Python.

TODO

CHAPTER 2

Installation

Change directories to the repo location and run `python setup.py install`.

CHAPTER 3

Application Guide

This is a short description of applications of `pybuck`. See the demo for an executable version of this tour.

3.1 Setting up an analysis

The first stage of performing dimensional analysis is to describe the physical dimensions of the problem. Using `col_matrix`, we can succinctly define a dimension matrix. As a running example, we consider the inputs for the *Reynolds pipe flow problem* [1]. There are five input quantities, described in the table below.

Expressing this information with `pybuck`, we specify each column of the matrix as a Python `dict` of non-zero entries.

```
from pybuck import *

df_dim = col_matrix(
    rho = dict(M=1, L=-3),
    U   = dict(L=1, T=-1),
    D   = dict(L=1),
    mu  = dict(M=1, L=-1, T=-1),
    eps = dict(L=1)
)
df_dim
```

```
  rowname  rho  U  D  mu  eps
0       T    0 -1  0  -1    0
1       M    1  0  0   1    0
2       L   -3  1  1  -1    1
```

The *dimension matrix* is now assigned to `df_dim`—each entry is an exponent, associated with an input and physical dimension. For instance the `rho` column has the entry `-3` in the `L` row, indicating that `rho` has a factor of L^{-3} in its physical dimensions.

Note that we did not need to assign the zeros in the matrix, and both variable and dimension names are provided by keyword argument (not by string). Finally, note that the ordering of row labels in `rowname` is automatically handled by `col_matrix()`; for example:

```
col_matrix(
    U = dict(L=+1, T=-1),
    V = dict(T=-1, L=+1)
)
```

```
  rowname   U   V
0       T  -1  -1
1       L   1   1
```

3.2 Buckingham Pi

The central result of dimensional analysis is the buckingham pi theorem. This result provides a means for a priori dimension reduction; a lossless reduction in the number of inputs for a physical system. Using the dimension matrix, we can compute a basis for the set of dimensionless numbers.

```
df_pi = pi_basis(df_dim)
df_pi
```

```
  rowname       pi0       pi1
0     rho -0.521959  0.115207
1       U -0.521959  0.115207
2       D -0.413385 -0.632884
3      mu  0.521959 -0.115207
4     eps -0.108575  0.748091
```

This output indicates that, despite there being five inputs, only two dimensionless numbers are necessary to fully describe the system.

3.3 Re-expression

The dimensionless numbers above are a basis for the pi subspace—the set of all valid dimensionless numbers for the problem at hand. However, they are fairly difficult to physically interpret. *Re-expressing* the dimensionless numbers in a user-selected basis can help us with interpretation.

First, we define a "standard" dimensionless basis.

```
df_standard = col_matrix(
    Re = dict(rho=1, U=1, D=1, mu=-1), # Reynolds number
    R  = dict(eps=1, D=-1)             # Relative roughness
)
df_standard
```

```
  rowname  Re  R
0     rho   1  0
1       U   1  0
2     eps   0  1
3       D   1 -1
4      mu  -1  0
```

Re is the Reynolds number, which represents the ratio of inertial to viscous forces. R is the relative roughness, which represents the ratio of roughness to bulk lengthscales. We can re-express df_pi in terms of these standard numbers to make them more physically interpretable.

```
df_pi_prime = express(df_pi, df_standard)
df_pi_prime
```

```
  rowname       pi0       pi1
0      Re -0.521959  0.115207
1       R -0.108575  0.748091
```

Based on the weights above, we can see that pi0 is mostly weighted towards Re, while pi1 is mostly weighted towards R. However, both are mixtures of the two standard dimensionless numbers.

3.4 Empirical Dimension Reduction

Next we demonstrate combining *empirical dimension reduction* with dimensional analysis. This allows one to equip data-driven methods with physical interpretation. First, we generate some data for the Reynolds pipe flow problem. This follows the setup described in Reference 2.

```
import statsmodels.formula.api as smf
import numpy as np
import pandas as pd
from model_pipe import fcn

## Simulate collecting data
np.random.seed(101)
n_data = 500

Q_names = ["rho", "U", "D", "mu", "eps"]
Q_lo = np.array([1.0, 1.0e+0, 1.3, 1.0e-5, 0.5e-1])
Q_hi = np.array([1.4, 1.0e+1, 1.7, 1.5e-5, 2.0e-1])
Q_all = np.random.random((n_data, len(Q_lo))) * (Q_hi - Q_lo) + Q_lo

F_all = np.zeros(n_data)
for i in range(n_data):
    res = fcn(Q_all[i])
    F_all[i] = res

df_data = pd.DataFrame(
    data=Q_all,
    index=range(n_data),
    columns=Q_names
)
df_data["f"] = F_all
```

To perform empirical dimension reduction, we will carry out ordinary least squares to regress the output `f` on the inputs `rho, U, D, mu, eps`. However, if we **log transform** our inputs, any *linear* dimension reduction can be interpreted as a product of the inputs [2]. This will allow us to combine dimension reduction with dimensional analysis. To illustrate:

```
df_log = df_data.copy()
df_log[Q_names] = np.log(df_log[Q_names])
df_log

lm = smf.ols(
    "f ~ rho + U + D + mu + eps",
    data=df_log
).fit()
```

We extract the regression coefficients with the following recipe.

```
df_dr = pd.DataFrame({
    "rowname": lm.params.index[1:],
    "pi": lm.params.values[1:]
})
df_dr
```

```
  rowname        pi
0     rho  0.000658
1       U -0.000247
2       D -0.049711
3      mu -0.000014
4     eps  0.045981
```

We now check the physical units of the proposed direction.

```
inner(df_dim, df_dr)
```

```
  rowname        pi
0       M  0.000644
1       L -0.005936
2       T  0.000261
```

This is very nearly dimensionless. We can re-express this number in terms of our standard basis.

```
express(df_dr, df_standard)
```

```
  rowname        pi
0      Re -0.000412
1       R  0.047640
```

Re-expression reveals that the empirical dimension reduction has recovered the relative roughness `R`, which fully describes the output variation in the setting considered.

3.5 Lurking Variables

Finally, we slightly modify the problem above to demonstrate *lurking variable detection*.

Suppose that during data collection we did not know that `eps` is a physical input. In this case, we would not know to vary it in our experiments, and it might remain fixed to an unknown value. To model this, we fix `eps=0.1` in data generation.

```
## Generate frozen-eps data
Fp_all = np.zeros(n_data)
Qp_all = Q_all
Qp_all[:, 4] = [0.1] * n_data

for i in range(n_data):
    Fp_all[i] = fcn(Qp_all[i])

df_frz = pd.DataFrame(
    data=Qp_all,
    index=range(n_data),
    columns=Q_names
)
df_frz["f"] = Fp_all
```

We repeat computing a linear dimension reduction on the frozen data.

```
df_frz_log = df_data.copy()
df_frz_log[Q_names] = np.log(df_frz_log[Q_names])
df_frz_log

lm_frz = smf.ols(
    "f ~ rho + U + D + mu",
    data=df_frz_log
).fit()

df_frz_dr = pd.DataFrame({
    "rowname": lm_frz.params.index[1:],
    "pi": lm_frz.params.values[1:]
})
```

Let’s inspect the physical dimensions of `df_frz_dr`.

```
inner(df_dim, df_frz_dr)
```

```
  rowname        pi
0       M -0.005219
1       L -0.049977
2       T  0.005100
```

This direction is not dimensionless! This indicates that a lurking variable is present, and it has units of `L`. This procedure has correctly identified the presence of our lurking variable `eps` which has dimensions $[\epsilon] = L$.

3.6 References

[1] O. Reynolds, “An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels” (1883) *Royal Society*

[2] Z. del Rosario, M. Lee, and G. Iaccarino, “Lurking Variable Detection via Dimensional Analysis” (2019) *SIAM/ASA Journal on Uncertainty Quantification*

CHAPTER 4

Dimensional Analysis Theory

TODO

CHAPTER 5

References

CHAPTER 6

Indices and tables

- genindex
- modindex
- search