CUBmods

Release 0.0.1

Massimo Pierini

Sep 27, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Preface

This package has been implemented by Massimo Pierini as a Bachelor's thesis (Pierini, 2024).

It is the first implementation of CUB class models in Python and is mainly based upon the work of Domenico Piccolo and the `CUB` package in R (Iannario *et al.*, 2022), mainteined by Rosaria Simone.

1.2 Background

The class of CUB (Combination of Uniform and Binomial) models, proposed by Professor Domenico Piccolo in 2003 (Piccolo, 2003) within the cotext of rating and preference data analysis, hypothesizes that the ordinal responses provided by the raters are not the simple result of a reasoned choice, but rather the complex combination of a multitude of factors, both internal and external.

Simplifying, two main components can be distinguished: *feeling* and *uncertainty*.

The primary component of feeling is due to sufficient awareness and understanding of the topic based on knowledge and experience. The secondary component of uncertainty is instead generated by an *intrinsic fuzziness*, due to various circumstances: limited knowledge, lack of interest, timing of the survey, method of administration, boredom, and so on.

The simplest way to consider these two aspects is a distribution resulting from a mixture of a shifted Binomial component for the first and Uniform Discrete for the second which takes the form of the CUB family models, subsequently extended to consider further factors such as the overdispersion of Binomial component, the effect of shelter choice, and so on.

The most updated paper by Piccolo and Simone, 2019 will be used as a reference for terminology, theory and inferential issues.

1.3 Motivation

Currently the class of CUB models has been implemented in statistical and econometric programming languages such as R (Iannario *et al.*, 2022), Stata (Cerulli *et al.*, 2021), Gretl (Simone *et al.*, 2019) and GAUSS (Piccolo, 2006). However, given the recent increase in the development of the Python programming language also in the statistical field (Pittard and Li, 2020), their implementation in this environment could be useful to the scientific community.

1.4 Notes

To simplify the notation, the complete matrix of the covariates will be occasionally indicated by $\boldsymbol{T}$ and the column vector of model's parameters by $\boldsymbol{\theta}$.

Generally speaking, for models with covariates three different probability functions are available:

1. **`.pmfi()` (probability distribution matrix)**

$$\Pr(R_i = r|\boldsymbol{\theta};\boldsymbol{T}_i), \begin{cases} i = 1,\ldots,n \\ r = 1,\ldots,m \end{cases}$$

   which is a matrix $n \times m$ of the probability distribution for each i-th subject given the estimated parameters and the covariates. This is an auxiliary function for `.draw()`. Notice that each row sums to 1, i.e.

$$\sum_{r=1}^{m} \Pr(R_i = r|\boldsymbol{\theta};\boldsymbol{T}_i) = 1,\ \forall i$$

2. **`.pmf()` (average probability distribution)**

$$\frac{1}{n}\sum_{i=1}^{n} \Pr(R_i = r|\boldsymbol{\theta};\boldsymbol{T}_i),\ r = 1,\ldots,m$$

   which is a row vector $1 \times m$ of the average probability given the estimated parameters and the covariates. This is an auxiliary function of `.plot_ordinal()`. Notice that it sums to 1 because

$$\begin{aligned}\sum_{r=1}^{m} \frac{1}{n}\sum_{i=1}^{n} \Pr(R_i = r|\boldsymbol{\theta};\boldsymbol{T}_i) &= \frac{1}{n}\sum_{i=1}^{n}\sum_{r=1}^{m} \Pr(R_i = r|\boldsymbol{\theta};\boldsymbol{T}_i) \\ &= \frac{1}{n}\sum_{i=1}^{n} 1 = \frac{1}{n}n = 1\end{aligned}$$

3. **`.prob()` (observed sample probability)**

$$\Pr(R_i = r_i|\boldsymbol{\theta};\boldsymbol{T}_i),\ i = 1,\ldots,n$$

   which is a column vector $n \times 1$ of the probabilities for each i-th subject of the observed response r_i given the estimated parameters and the covariates. This has not been implemented for all models and can be an auxiliary function of `.loglik()`. Notice that usually it doesn't sum to 1.

CHAPTER

TWO

MANUAL

The package `cubmods` can be used to build models within the CUB class given an observed sample (and, eventually, the covariance matrix) in order to estimate the parameters. Also, for each family, random samples can be drawn from a specified model.

Currently, six families have been defined and implemented:

- CUB (Combination of Uniform and Binomial)
- CUBSH (CUB + a SHelter choice)
- CUSH (Combination of Uniform and a SHelter choice)
- CUSH2 (Combination of Uniform and 2 SHelter choices)
- CUBE (Combination of Uniform and BEta-binomial)
- IHG (Inverse HyperGeometric)

For each family, a model can be defined with or without covariates for one or more parameters.

Details about each family and examples are provided in the following chapters.

Even if each family has got its own *Maximum Likelihood Estimation* function `mle()` that could be called directly, for example `cub.mle()`, the function `gem.estimate()` provides a simplified and generalised procedure for MLE.

Similarly, even if each family has got its own *Random Sample Drawing* function `draw()` that could be called directly, for example `cub.draw()`, the function `gem.draw()` provides a simplified and generalised procedure to draw a random sample.

In this manual `gem` functions will be used for the examples.

Notice that, the Dissimilarity index is computed for models with covariates also: it should be interpreted as the fraction of the sample to be changed to achive a perfect fit to the estimated average probability distribution (see Introduction Notes).

The last chapter, shows the basic usage for the tool `multicub`.

2.1 GeM usage

GeM (Generalized Mixture) is the main module of `cubmods` package, which provides simplified and generalized functions to both estimate a model from an observed sample and draw a random sample from a specified model.

The function `gem.estimate()` is the main function for the estimation and validation of a model from an observed sample, calling for the corresponding `.mle()` function of the specified family, with or without covariates.

The function `gem.draw()` is the main function for drawing a random sample from a specified model, calling for the corresponding `.draw()` function of the corresponding family, with or without covariates.

Reference guide

2.1.1 The *formula* syntax

Both functions need a `formula` that is a **string** specifying the name of the ordinal variable (before the tilde ~ symbol) and of the covariates of the components (after the tilde symbol ~). Covariates for each component are separated by the pipeline symbol `|`. The *zero* symbol `0` indicates no covariates for a certain component. The *one* symbol `1` indicates that we want to estimate the parameter of the constant term only. If more covariates explain a single component, the symbol + concatenates the names. Qualitative variables names, must be placed between brackets `()` leaded by a `C`, for example `C(varname)`.

Warning

No columns in the DataFrame should be named `constant`, `1` or `0`. In the column names, only letters, numbers, and underscores _ are allowed. Spaces **SHOULD NOT BE** used in the column names, but replaced with _.

For example, let's suppose we have a DataFrame where `response` is the ordinal variable, `age` and `sex` are respectively a quantitative and a qualitative variable to explain the *feeling* component only, in a `cub` family model. The formula will be `formula = "response ~ 0 | age + C(sex)"`.

Note

Python will automatically order qualitative variables in alphanumeric order. So, for instance, a variable `sex` with two categories `"M"` and `"F"` will be ordered as `["F", "M"]` thus the dummy variabile will be equal to `0` where `sex=="F"` and equal to 1 where otherwise `sex=="M"`. Consequently, the estimated parameters will be the `constant` for `sex=="F"` and `C.sex_M` for `sex=="M"`. If you want a different order for the categorical variables, you must specify it in *DataFrame*, for instance with the `pandas` funtion `Categorical`. In the example:

Listing 1: Script

```
df["sex"] = pd.Categorical(
    df["sex"],
    categories=["M", "F"],
    ordered=True
)
```

Notice that spaces are allowed between symbols and variable names in the formula but they aren't needed: a formula `"ord ~ X | Y1 + Y2 | Z"` is the same as `"ord~X|Y1+Y2|Z"`.

Warning

The number of fields separated by the pipeline `|` in a formula **MUST BE** equal to the number of parameters specifying the model family. Therefore: two for `cub` and `cush2`, three for `cube` and `cub` with shelter effect, one for `cush` and `ihg`.

2.1.2 Arguments of `estimate` and `draw`

Within the function `estimate` the number of ordinal categories `m` is internally retrieved if not specified (taking the maximum observed category) but it is advisable to pass it as an argument to the call if some category has zero frequency. Within the function `draw` instead, the number of ordinal categories `m` must always be specified.

A `pandas` DataFrame must always be passed to the function `estimate`, with the *kwarg* (keyword argument) `df`. It should contain, at least, a column of the observed sample and the columns of the covariates (if any). If no `df` is passed to the function `draw` for a model without covariates instead, an empty DataFrame will be created.

The number `n` of ordinal responses to be drawn should always be specified in the function `draw` for models without covariates. For model with covariates instead, `n` is not effective because the number of drawn ordinal responses will be equal to the passed DataFrame rows.

A `seed` could be specified for the function `draw` to ensure reproducibility. Notice that, for models with covariates, `seed` cannot be `0` (in case, it will be automatically set to `1`).

If no `model` is declared, the function takes `"cub"` as default. Currently implemented models are: `"cub"` (default), `"cush"`, `"cube"`, and `"ihg"`. CUB models with shelter effect are automatically implemented using `model="cub"` and specifying a shelter choice with the *kwarg* `sh`. CUSH2 models are automatically implemented using `model="cush"` and passing a list of two categories to the *kwarg* `sh` instead of an integer, for instance `sh=[2, 7]`.

To the `draw` method, the parameters' values (with the *kwargs* of the corresponding family) must be passed: for example, `pi` and `xi` for CUB models without covariates, `beta` and `gamma` for CUB models with covariates for both feeling and uncertainty, etc. See the `.draw()` function reference of the corresponding family module for details.

If `model="cub"` (or nothing), then a CUB mixture model is fitted to the data to explain uncertainty, feeling (`ordinal~Y|W`) and possible shelter effect by further passing the extra argument `sh` for the corresponding category. Subjects' covariates can be included by specifying covariates matrices in the formula as `ordinal~Y|W|X`, to explain uncertainty (Y), feeling (W) or shelter (X). Notice that covariates for the shelter effect can be included only if specified for both feeling and uncertainty too (GeCUB models) because, as in the R package `CUB`, only the models without covariates and with covariates for all components have been implemented. Nevertheless, the symbol `1` could be used to specify a different combination of components with covariates. For example, if we want to specify a CUB model with covariate `cov` for uncertainty only, we could pass the formula `ordinal ~ cov | 1 | 1`: in this case, for feeling and shelter effect, the constant terms only (γ_0 and ω_0) will be estimated and the values of the estimated ξ and δ could be computed as $\hat{\xi} = \text{expit}(\hat{\gamma}_0)$ and $\hat{\delta} = \text{expit}(\hat{\omega}_0)$, where $\text{expit}(x) = 1/(1+\exp(-x))$. See *this example* for the GeCUB model.

If `family="cube"`, then a CUBE mixture model (Combination of Uniform and Beta-Binomial) is fitted to the data to explain uncertainty, feeling and overdispersion. Subjects' covariates can be also included to explain the feeling component or all the three components by specifying covariates matrices in the Formula as `ordinal~Y|W|Z` to explain uncertainty (Y), feeling (W) or overdispersion (Z). For different combinations of components with covariates, the symbol 1 can be used. Notice that $\hat{\phi} = e^{\hat{\alpha}_0}$.

If `family="ihg"`, then an IHG model is fitted to the data. IHG models (Inverse HyperGeometric) are a peculiar case of CUBE models, for $\phi = 1 - \xi$ (Iannario, 2012). The parameter θ gives the probability of observing the first category and is therefore a direct measure of preference, attraction, pleasantness toward the investigated item. This is the reason why θ is customarily referred to as the preference parameter of the IHG model. Covariates for the preference parameter θ have to be specified in matrix form in the Formula as `ordinal~V`.

If `family="cush"`, then a CUSH model is fitted to the data (Combination of Uniform and SHelter effect). If a category corresponding to the inflation should be passed via argument `sh` a CUSH model is called and covariates for the shelter parameter δ are specified in matrix form Formula as `ordinal~X`. If two category corresponding to the inflation should be passed via argument `sh` (as a *list* or *array*) a CUSH2 model is called and covariates for the shelters' parameters (δ_1, δ_2) are specified in matrix form Formula as `ordinal~X1|X2`. Notice that, to specify covariates for a single shelter choice in a CUSH2 model, the formula should be `ordinal~X1|0` and not `ordinal~0|X2`.

Extra arguments include the maximum number of iterations `maxiter` for the optimization algorithm, the required error tolerance `tol`, and a dictionary of parameters of a known model `ass_pars` (assumed parameters) to be compared with

the estimates: these could be the parameters used to draw the sample, theoretical parameters, or howsoever specified parameters we want to (graphically) compare with the estimates.

2.1.3 Methods of `estimate` and `draw`

For both functions, the methods `.summary()` and `.plot()` are always available calling the main functions to print a summary and plot the results, respectively. For `.plot()` arguments and options, see here the `CUBsample` Class (for object returned by `draw`) and the extended `CUBres` Classes of the corresponding family (for objects returned by `estimate`), defined in each family module.

Calling `.as_dataframe()` will return a DataFrame of parameters' names and values for objects of the Class `CUBsample` returned by `draw`. For objects of the extended Base Class `CUBres` returned by `estimate` instead, will return a DataFrame with parameters' component, name, estimated value, standard error, Wald test statistics and p-value.

Calling the method `.save(fname)` the object can be saved on a file called `fname.cub.sample` (for `draw`) or `fname.cub.fit` (for `estimate`). Saved objects can then be loaded using the function `general.load_object(fname)`. See *this example*.

2.1.4 Attributes of `estimate` and `draw`

For both objects returned by `estimate` and `draw`, the attributes `.formula` and `.df` are always available. The function `draw` will return the original DataFrame (if provided) with an extra column of the drawn ordinal response called as specified in the formula.

Many other attributes can be called from objects of the Base Class `CUBres` returned by `estimate`, such as the computed loglikelihood, the AIC and BIC, ectcetera. For details, see here the Base Class `CUBres` reference guide.

2.2 CUB family

Basic family of the class CUB. See the references for details: Piccolo, 2003; D'Elia and Piccolo, 2005; Piccolo, 2006; Iannario and Piccolo, 2010; Iannario and Piccolo, 2009; Iannario *et al.*, 2014; Iannario *et al.*, 2022; Piccolo and Simone, 2019.

2.2.1 Without covariates

Reference guide

A model of the CUB family for responses with m ordinal categories, without covariates is specified as

$$\Pr(R = r|\boldsymbol{\theta}) = \pi \binom{m-1}{r-1} (1-\xi)^{r-1} \xi^{m-r} + \frac{1-\pi}{m},\ r = 1, 2, \ldots, m$$

where π and ξ are the parameters for respectively the *uncertainty* and the *feeling* components.

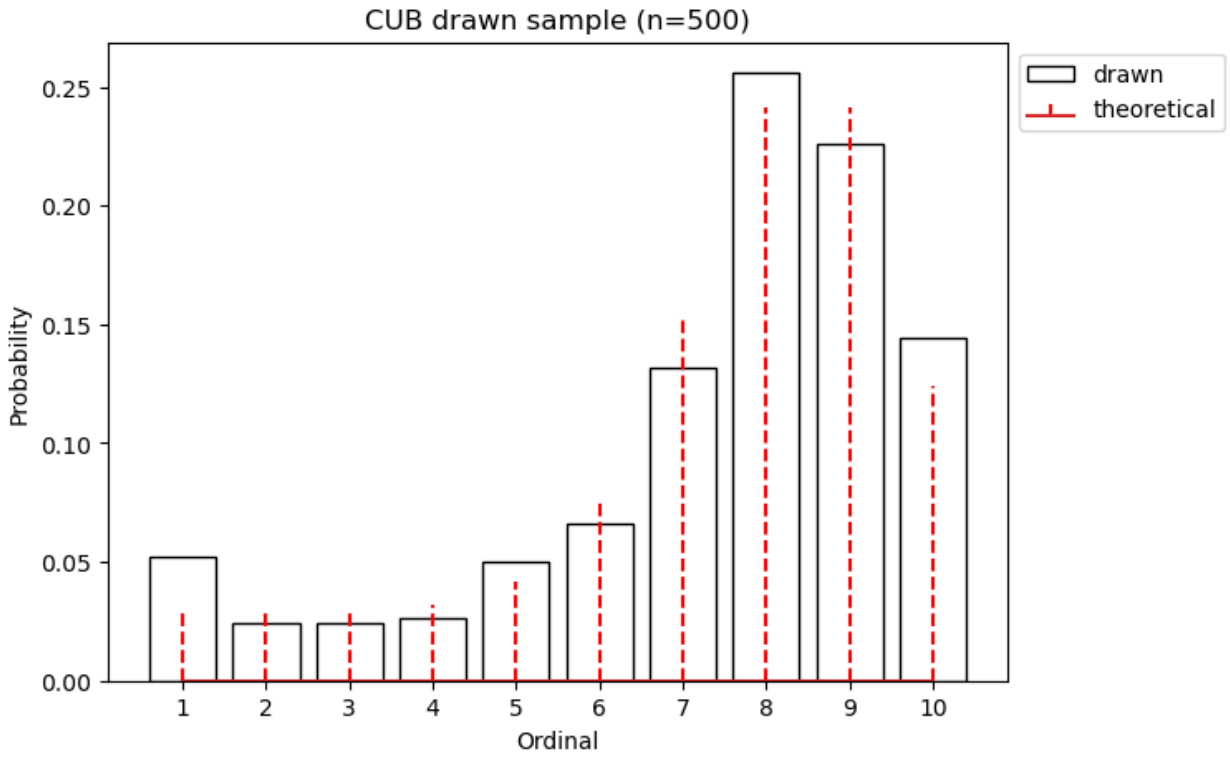
Note that $(1-\pi)$ is the weight of the Uncertainty component and $(1-\xi)$ is the Feeling component for common *positive wording*.

In the following example, a sample will be drawn from a CUB model of $n = 500$ observations of an ordinal variable with $m = 10$ ordinal categories and parameters $(\pi = .7, \xi = .2)$. A `seed=1` will be set to ensure reproducibility.

Listing 2: Script

```python
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw

# draw a sample
drawn = draw(
    formula="ord ~ 0 | 0",
    m=10, pi=.7, xi=.2,
    n=500, seed=1)
# print the summary of the drawn sample
print(drawn.summary())
# show the plot of the drawn sample
drawn.plot()
plt.show()
```

```
========================================================================
=====>>> CUB model <<<===== Drawn random sample
========================================================================
m=10  Sample size=500  seed=1
formula: ord~0|0
------------------------------------------------------------------------
  component parameter  value
Uncertainty        pi    0.7
    Feeling        xi    0.2
========================================================================
Sample metrics
Mean     = 7.368000
Variance = 5.687952
Std.Dev. = 2.384943
------------------------------------------------------------------------
Dissimilarity = 0.0650938
========================================================================
```



Notice that, since the default value of the *kwarg* `model` is `"cub"` we do not need to specify it.

Calling `drawn.as_dataframe()` will return a DataFrame with the parameters

```
     component parameter  value
0  Uncertainty        pi    0.7
1      Feeling        xi    0.2
```

Using the previously drawn sample, in the next example the parameters $(\hat{\pi}, \hat{\xi})$ will be estimated.

Note that in the function `gem.estimate`:

- `df` needs to be a `pandas` DataFrame; the attribute `drawn.df` will return a DataFrame with `ord` as column name of the drawn ordinal response (as previuosly speficied in the formula)
- `formula` needs the ordinal variable name (`ord` in this case) and the covariates for each component (none in this case, so `"0|0"`)
- if `m` is not provided, the maximum observed ordinal value will be assumed and a warning will be raised
- with `ass_pars` dictionary, the parameters of a known model (if any) can be specified; in this case, we'll specify the known parameters used to draw the sample

Listing 3: Script

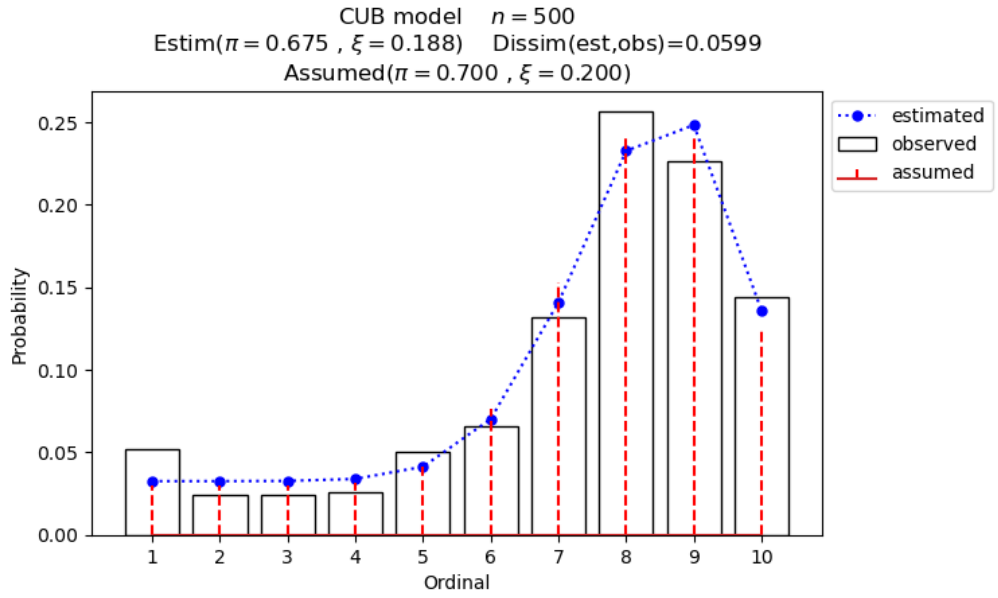
```
# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    formula="ord~0|0",
    ass_pars={
        "pi": drawn.pars[0],
        "xi": drawn.pars[1]
```

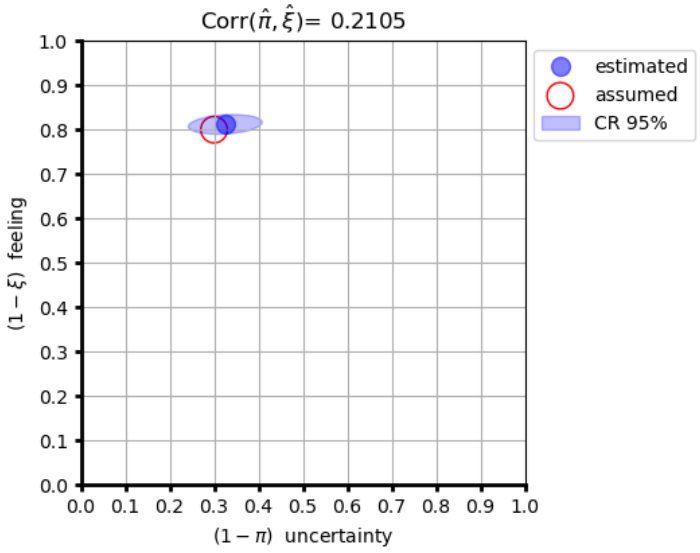
(continues on next page)

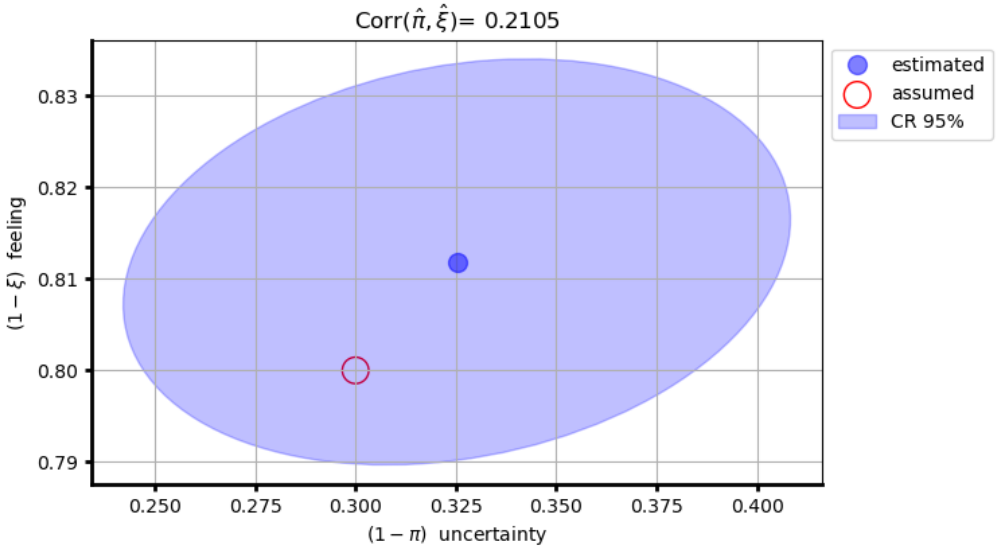
(continued from previous page)

```
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

```
=========================================================================
=====>>> CUB model <<<===== ML-estimates
=========================================================================
m=10  Size=500  Iterations=13  Maxiter=500  Tol=1E-04
-------------------------------------------------------------------------
Uncertainty
    Estimates  StdErr    Wald  p-value
pi      0.675  0.0340  19.872   0.0000
-------------------------------------------------------------------------
Feeling
    Estimates  StdErr    Wald  p-value
xi      0.188  0.0090  20.808   0.0000
-------------------------------------------------------------------------
Correlation   = 0.2105
=========================================================================
Dissimilarity = 0.0599
Loglik(sat)   = -994.063
Loglik(MOD)   = -1000.111
Loglik(uni)   = -1151.293
Mean-loglik   = -2.000
Deviance      = 12.096
-------------------------------------------------------------------------
AIC = 2004.22
BIC = 2012.65
=========================================================================
Elapsed time=0.00202 seconds =====>>> Thu Sep 26 18:00:53 2024
=========================================================================
```


CUB model $n = 500$
Estim($\pi = 0.675$, $\xi = 0.188$) Dissim(est,obs)=0.0599
Assumed($\pi = 0.700$, $\xi = 0.200$)
estimated
observed
assumed
Probability
Ordinal


Corr($\hat{\pi}, \hat{\xi}$)= 0.2105
estimated
assumed
CR 95%
$(1-\xi)$ feeling
$(1-\pi)$ uncertainty


Corr($\hat{\pi}, \hat{\xi}$)= 0.2105
estimated
assumed
CR 95%
$(1-\xi)$ feeling
$(1-\pi)$ uncertainty

Calling `fit.as_dataframe()` will return a DataFrame with parameters' estimated values and standard errors

```
     component parameter  estimate    stderr       wald        pvalue
0  Uncertainty        pi   0.67476  0.033954  19.872485  7.042905e-88
1      Feeling        xi   0.18817  0.009043  20.807551  3.697579e-96
```

As an example, we can now save the `fit` object to file. By default, it will be saved as a `pickle` file.

Listing 4: Script

```
fit.save(fname="cub_mle_results")
```

The previous code, will save a file `cub_mle_results.cub.fit`.

We can then load the saved file with the code

Listing 5: Script

```
from cubmods.general import load_object

myfit = load_object("cub_mle_results.cub.fit")
```

and we can apply to `myfit` the same methods and attributes of the original `fit` object.

2.2.2 With covariates

Reference guide (0|W)

Reference guide (Y|0)

Reference guide (Y|W)

$$\Pr(R_i = r|\boldsymbol{\theta}, \boldsymbol{y}_i, \boldsymbol{w}_i) = \pi_i \binom{m-1}{r-1} (1-\xi_i)^{r-1} \xi_i^{m-r} + \frac{1-\pi_i}{m},\ r = 1, 2, \ldots, m$$

$$\begin{cases} \pi_i = \dfrac{1}{1+\exp\{-\boldsymbol{y}_i\boldsymbol{\beta}\}} \\ \xi_i = \dfrac{1}{1+\exp\{-\boldsymbol{w}_i\boldsymbol{\gamma}\}} \end{cases} \equiv \begin{cases} \text{logit}(1-\pi_i) = -\boldsymbol{y}_i\boldsymbol{\beta} \\ \text{logit}(1-\xi_i) = -\boldsymbol{w}_i\boldsymbol{\gamma} \end{cases}$$

All three combinations of covariates has been implemented for CUB family in both Python and R: for *uncertainty* only, for *feeling* only, and for *both*.

Here we'll show an example with covariates for *feeling* only.

First of all, we'll draw a random sample with two covariates for the *feeling* component: `W1` and `W2`. Note that, having two covariates, we'll need three γ parameters, to consider the constant term too.

Listing 6: Script

```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

(continues on next page)

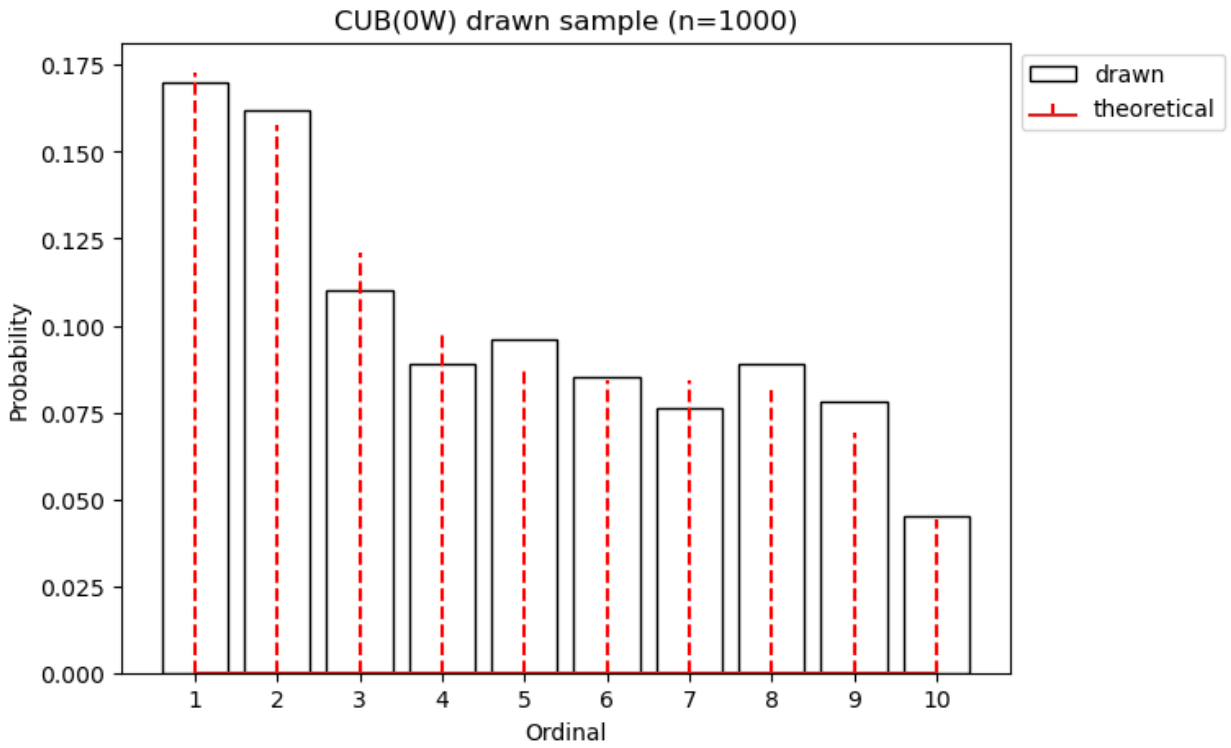
(continued from previous page)

```
from cubmods.gem import draw, estimate

# Draw a random sample
n = 1000
np.random.seed(1)
W1 = np.random.randint(1, 10, n)
np.random.seed(42)
W2 = np.random.random(n)
df = pd.DataFrame({
    "W1": W1, "W2": W2
})
drawn = draw(
    formula="response ~ 0 | W1 + W2",
    df=df,
    m=10, n=n,
    pi=0.8,
    gamma=[2.3, 0.2, -5],
)
# print the summary
print(drawn.summary())
```

```
========================================================================
=====>>> CUB(0W) model <<<===== Drawn random sample
========================================================================
m=10  Sample size=1000  seed=None
formula: res~0|W1+W2
------------------------------------------------------------------------
  component parameter  value
Uncertainty        pi    0.8
    Feeling  constant    2.3
    Feeling        W1    0.2
    Feeling        W2   -5.0
========================================================================
Sample metrics
Mean     = 4.566000
Variance = 8.089734
Std.Dev. = 2.844246
------------------------------------------------------------------------
Dissimilarity = 0.0307673
========================================================================
```

Listing 7: Script

```
# plot the drawn sample
drawn.plot()
plt.show()
```



Listing 8: Script

```
# print the parameters' values
print(drawn.as_dataframe())
```

```
     component parameter  value
0  Uncertainty        pi    0.8
1      Feeling  constant    2.3
2      Feeling        W1    0.2
3      Feeling        W2   -5.0
```

Listing 9: Script

```
# print the updated DataFrame
print(drawn.df)
```

```
     W1        W2  res
0     6  0.374540    2
1     9  0.950714    7
2     6  0.731994    8
3     1  0.598658    8
```

(continues on next page)

(continued from previous page)

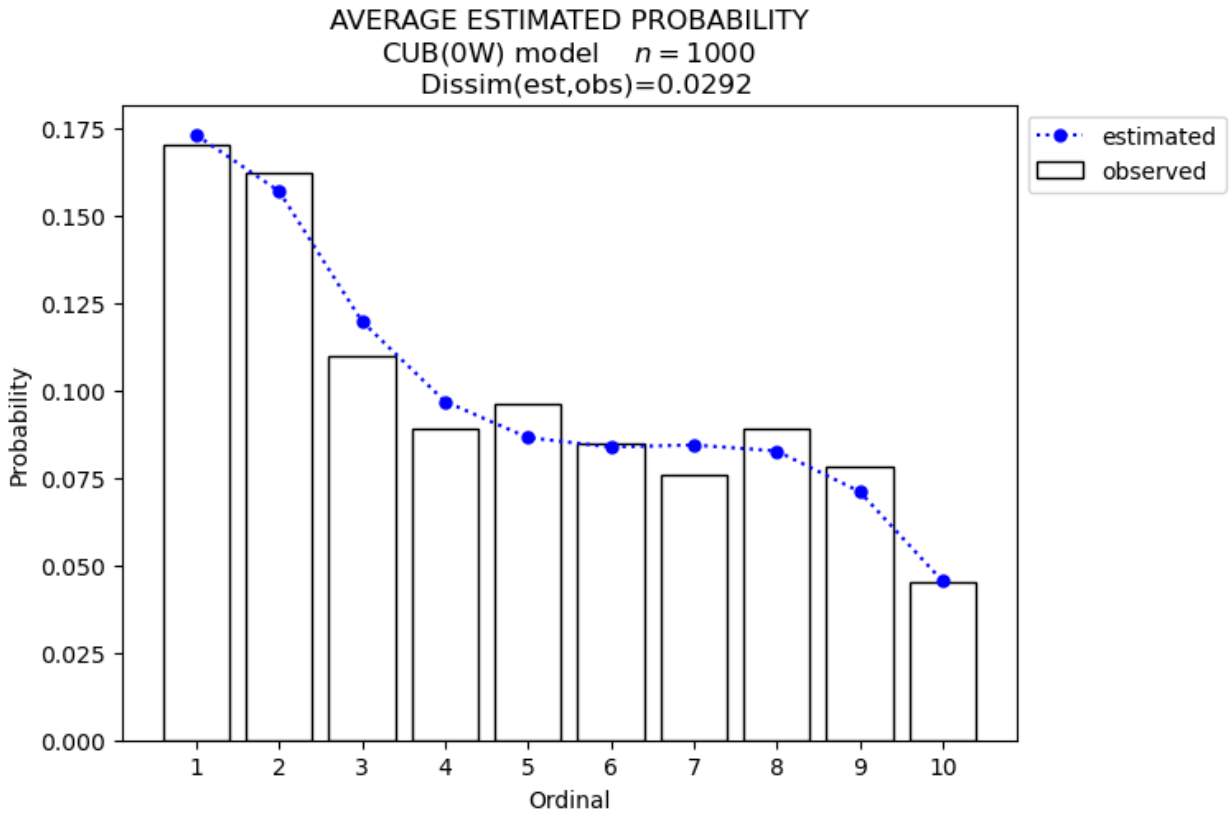
```
4     1  0.156019    4
..   ..       ...  ...
995   3  0.091582    2
996   9  0.917314    9
997   4  0.136819    1
998   7  0.950237    3
999   8  0.446006    2

[1000 rows x 3 columns]
```

Finally, we'll call `estimate` to estimate the parameters given the observed (actually, drawn) sample.

Listing 10: Script

```
# MLE estimation
fit = estimate(
    formula="response ~ 0 | W1+W2",
    df=drawn.df,
)
# Print MLE summary
print(fit.summary())
# plot the results
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUB(0W) model <<<===== ML-estimates
================================================================================
m=10  Size=1000  Iterations=18  Maxiter=500  Tol=1E-04
--------------------------------------------------------------------------------
Uncertainty
          Estimates  StdErr     Wald  p-value
pi            0.800  0.0198   40.499   0.0000
--------------------------------------------------------------------------------
Feeling
          Estimates  StdErr     Wald  p-value
constant      2.353  0.1001   23.514   0.0000
W1            0.194  0.0138   14.034   0.0000
W2           -5.076  0.1454  -34.909   0.0000
================================================================================
Dissimilarity = 0.0292
Loglik(MOD)   = -1807.052
Loglik(uni)   = -2302.585
Mean-loglik   = -1.807
--------------------------------------------------------------------------------
AIC = 3622.10
BIC = 3641.74
================================================================================
Elapsed time=0.09656 seconds =====>>> Thu Aug 15 18:31:21 2024
================================================================================
```



2.3 CUBSH family

Basic family of the class CUB with shelter effect.

See the references for details: Corduas *et al.*, 2009; Iannario, 2012; Piccolo and Simone, 2019.

2.3.1 Without covariates

Reference guide

A model of the CUB family with shelter effect for responses with m ordinal categories, without covariates is specified as

$$\Pr(R = r|\boldsymbol{\theta}) = \delta D_r^{(c)} + (1-\delta)\left(\pi b_r(\xi) + \frac{1-\pi}{m}\right),\ r = 1, 2, \ldots, m$$

where π and ξ are the parameters for respectively the *uncertainty* and the *feeling* components, and δ is the weight of the shelter effect.

Other parametrizations have been proposed, such as

$$\Pr(R = r|\boldsymbol{\theta}) = \lambda b_r(\xi) + (1-\lambda)\left[\eta/m + (1-\eta)D_r^{(c)}\right],\ r = 1, 2, \ldots, m$$

where

$$\begin{cases} \lambda = \pi(1-\delta) \\ \eta = \dfrac{(1-\pi)(1-\delta)}{1-\pi(1-\delta)} \end{cases}$$

See Piccolo and Simone, 2019 (pp 412-413) for the parameters' interpretation.

Another parametrization, particularly useful for inferential issues is

$$\Pr(R = r|\boldsymbol{\theta}) = \pi_1 b_r \xi + \pi_2/m + (1 - \pi_1 - \pi_2) D_r^{(c)}$$

where

$$\begin{cases} \pi_1 = (1-\delta)\pi \\ \pi_2 = (1-\delta)(1-\pi) \end{cases}$$

See the references for further details.

In the next example, we'll draw an ordinal response and then estimate the parameters given the sample.

Listing 11: Script

```
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate

# draw a sample
drawn = draw(
    formula="ord ~ 0 | 0 | 0",
    m=7, sh=1,
    pi=.8, xi=.4, delta=.15,
    n=1500, seed=42)

print(drawn.as_dataframe())
```

```
     component parameter  value
0      Uniform       pi1   0.68
1     Binomial       pi2   0.17
2      Feeling        xi   0.40
3  Uncertainty       *pi   0.80
4      Shelter    *delta   0.15
```

Notice that:

- since `"cub"` is default value of the *kwarg* `model`, we do not need to specify it
- we'll pass to `estimate` *kwarg* values taken from the object `drawn`

The method `.plot()` (of the `fit` object) shows, in the parameters space, the trivariate confidence ellipsoid too, which has not been implemented yet in the `CUB` package in R. The plot includes the marginal bivariate confidence ellipses too. Notice that, as proven in Pierini, 2024 pp 28-30, the confidence level of the marginal ellipses is greater than the ellipsoid's confidence level. Indeed, the radius r of a standardized sphere at confidence level $(1-\alpha_3)$ is equal to $r = \sqrt{F^{-1}_{\chi^2_{(3)}}(1-\alpha_3)}$, thus the confidence level of the bivariate marginal ellipses (which is a section of a trivariate cylinder) is $(1-\alpha_2) = F_{\chi^2_{(2)}(r^2)}$.

Listing 12: Script

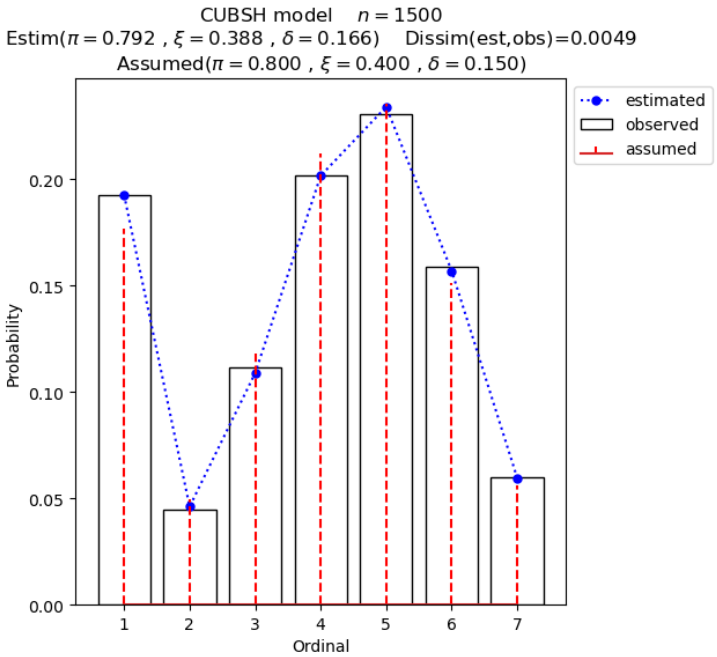
```
# inferential method on drawn sample
fit = estimate(
    df=drawn.df, sh=drawn.sh,
    formula=drawn.formula,
    ass_pars={
```

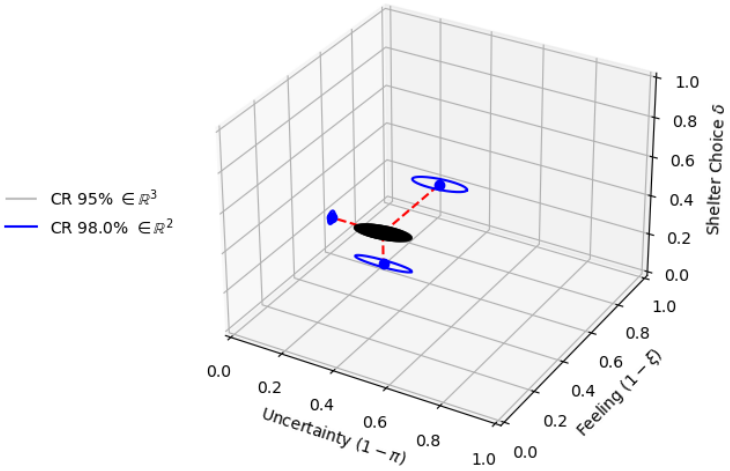
(continues on next page)

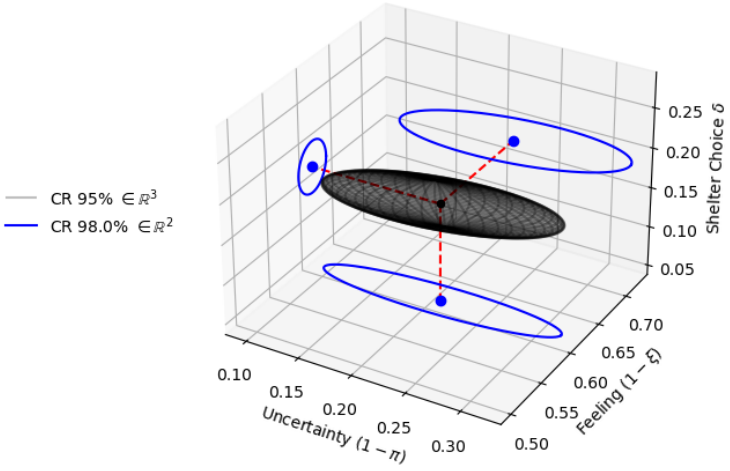
(continued from previous page)

```
        "pi1": drawn.pars[0],
        "pi2": drawn.pars[1],
        "xi": drawn.pars[2],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
==========================================================================
=====>>> CUBSH model <<<===== ML-estimates
==========================================================================
m=7  Shelter=1  Size=1500  Iterations=59  Maxiter=500  Tol=1E-04
--------------------------------------------------------------------------
Alternative parametrization
       Estimates  StdErr    Wald  p-value
pi1        0.661  0.0307  21.508   0.0000
pi2        0.174  0.0344   5.041   0.0000
xi         0.388  0.0077  50.592   0.0000
--------------------------------------------------------------------------
Uncertainty
       Estimates  StdErr    Wald  p-value
pi         0.792  0.0400  19.813   0.0000
--------------------------------------------------------------------------
Feeling
       Estimates  StdErr    Wald  p-value
xi         0.388  0.0077  50.592   0.0000
--------------------------------------------------------------------------
Shelter effect
       Estimates  StdErr    Wald  p-value
delta      0.166  0.0116  14.327   0.0000
==========================================================================
Dissimilarity = 0.0049
Loglik(sat)   = -2734.302
Loglik(MOD)   = -2734.433
Loglik(uni)   = -2918.865
Mean-loglik   = -1.823
Deviance      = 0.263
--------------------------------------------------------------------------
AIC = 5474.87
BIC = 5490.81
==========================================================================
```


CUBSH model $n = 1500$
Estim($\pi = 0.792$, $\xi = 0.388$, $\delta = 0.166$) Dissim(est,obs)=0.0049
Assumed($\pi = 0.800$, $\xi = 0.400$, $\delta = 0.150$)
estimated
observed
assumed
Probability
Ordinal


CR 95% $\in \mathbb{R}^3$
CR 98.0% $\in \mathbb{R}^2$
Uncertainty $(1-\pi)$
Feeling $(1-\xi)$
Shelter Choice δ


CR 95% $\in \mathbb{R}^3$
CR 98.0% $\in \mathbb{R}^2$
Uncertainty $(1-\pi)$
Feeling $(1-\xi)$
Shelter Choice δ

2.3.2 With covariates

Reference guide

$$\Pr(R_i = r|\boldsymbol{\theta}, \boldsymbol{y}_i, \boldsymbol{w}_i, \boldsymbol{x}_i) = \delta_i D_r^{(c)} + (1-\delta_i)\left(\pi_i b_r(\xi_i) + \frac{1-\pi_i}{m}\right),\ r = 1, 2, \ldots, m$$

$$\begin{cases} \pi_i = \dfrac{1}{1+\exp\{-\boldsymbol{y}_i\boldsymbol{\beta}\}} \\ \xi_i = \dfrac{1}{1+\exp\{-\boldsymbol{w}_i\boldsymbol{\gamma}\}} \\ \delta_i = \dfrac{1}{1+\exp\{-\boldsymbol{x}_i\boldsymbol{\omega}\}} \end{cases} \equiv \begin{cases} \text{logit}(1-\pi_i) = -\boldsymbol{y}_i\boldsymbol{\beta} \\ \text{logit}(1-\xi_i) = -\boldsymbol{w}_i\boldsymbol{\gamma} \\ \text{logit}(\delta_i) = \boldsymbol{x}_i\boldsymbol{\omega} \end{cases}$$

Only the model with covariates for all components has been currently defined and implemented, as in the R package `CUB`.

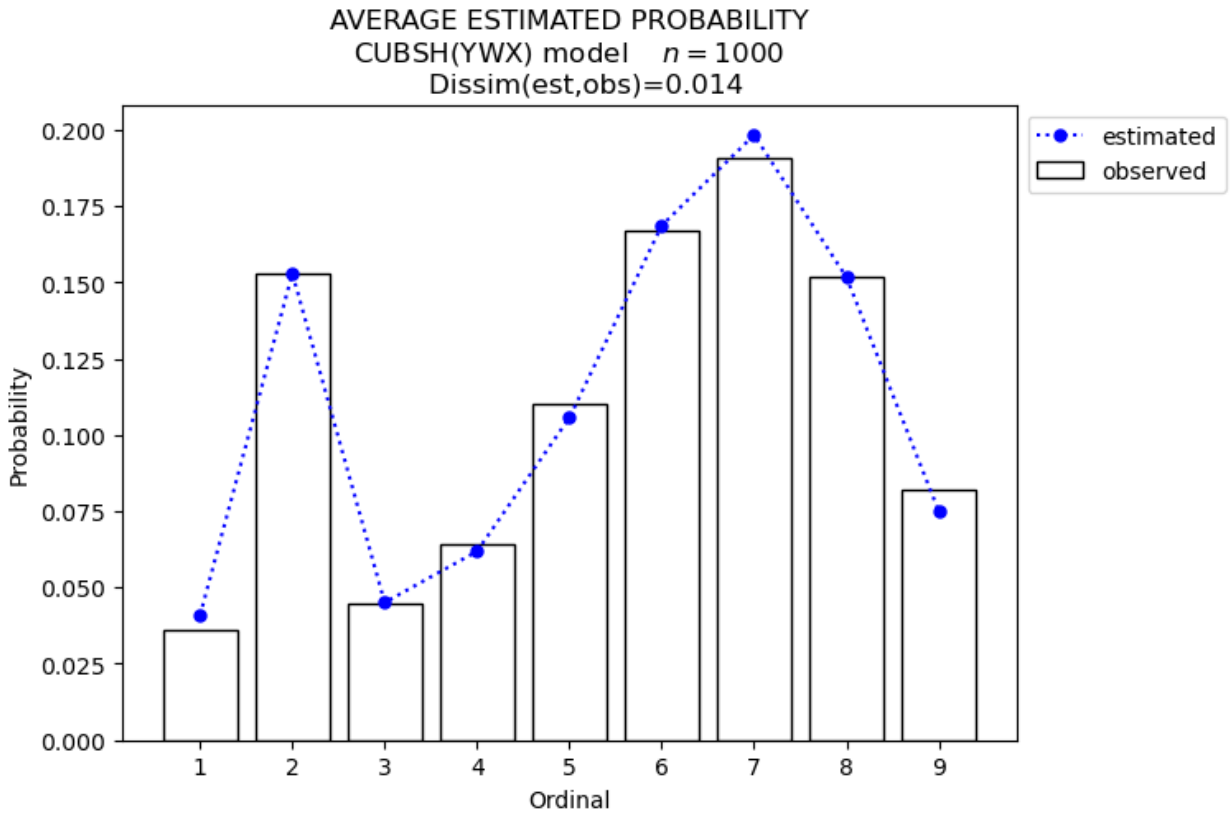
Nevertheless, thanks to the symbol 1 provided by the *formula*, we can specify a different combination of covariates.

For example, we'll specifiy a model CUB with shelter effect, with covariates for uncertainty only. We'll use the function `logit` to have better 'control' of the parameters values, because $\gamma_0 = \text{logit}(\xi)$ and similarly for π and δ.

Listing 13: Script

```python
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.general import expit, logit
from cubmods.gem import draw, estimate

# Draw a random sample
n = 1000
np.random.seed(1)
W1 = np.random.randint(1, 10, n)
df = pd.DataFrame({
    "W1": W1,
})
drawn = draw(
    formula="fee ~ W1 | 1 | 1",
    df=df,
    m=9, sh=2,
    beta=[logit(.8), -.2],
    gamma=[logit(.3)],
    omega=[logit(.12)],
)

# MLE estimation
fit = estimate(
    formula="fee ~ W1 | 1 | 1",
    df=drawn.df, sh=2,
)
# Print MLE summary
print(fit.summary())
# plot the results
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUBSH(YWX) model <<<===== ML-estimates
================================================================================
m=9  Shelter=2  Size=1000  Iterations=25  Maxiter=500  Tol=1E-04
--------------------------------------------------------------------------------
Uncertainty
          Estimates  StdErr     Wald  p-value
constant      0.992  0.3314    2.994   0.0028
W1           -0.127  0.0569   -2.228   0.0259
--------------------------------------------------------------------------------
Feeling
          Estimates  StdErr     Wald  p-value
constant     -0.902  0.0381  -23.662   0.0000
--------------------------------------------------------------------------------
Shelter effect
          Estimates  StdErr     Wald  p-value
constant     -2.074  0.1260  -16.462   0.0000
================================================================================
Dissimilarity = 0.0139
Loglik(MOD)   = -2069.978
Loglik(uni)   = -2197.225
Mean-loglik   = -2.070
--------------------------------------------------------------------------------
AIC = 4147.96
BIC = 4167.59
================================================================================
Elapsed time=1.43850 seconds =====>>> Thu Aug 15 19:39:49 2024
================================================================================
```



To get the estimated values of $\hat{\xi}$ and $\hat{\delta}$ we can use the function `expit` because $\hat{\xi} = \text{expit}(\hat{\gamma}_0)$ and similarly for $\hat{\delta}$. Then, we can use the delta-method to compute the standard errors of both $\hat{\xi}$ and $\hat{\delta}$, for instance $\widehat{es}(\xi) = \text{expit}[\hat{\gamma}_0 + \widehat{es}(\gamma_0)] - \hat{\xi}$.

Listing 14: Script

```
est_xi = expit(fit.estimates[2])
est_de = expit(fit.estimates[3])
est_xi_se = expit(fit.estimates[2]+fit.stderrs[2]) - est_xi
est_de_se = expit(fit.estimates[3]+fit.stderrs[3]) - est_de
print(
    "      estimates  stderr\n"
    f"xi       {est_xi:.4f}  {est_xi_se:.4f}"
    "\n"
    f"delta    {est_de:.4f}  {est_de_se:.4f}"
)
```

```
     estimates  stderr
xi      0.2886  0.0079
delta   0.1116  0.0131
```

which, in fact, match the values used to draw the sample.

2.4 CUSH family

Basic family of the class CUSH with a single shelter effect.

See the references for details: Capecchi and Piccolo, 2017; Piccolo and Simone, 2019.

2.4.1 Without covariates

Reference guide

$$\Pr(R = r|\boldsymbol{\theta}) = \delta D_r^{(c)} + (1 - \delta)/m,\ r = 1, 2, \ldots, m$$

In the example, we'll draw a sample from a CUSH model without covariates and then estimate the parameter given the observed sample.

Notice that, since the `model` is not the default `"cub"`, we need to specify it.

Listing 15: Script

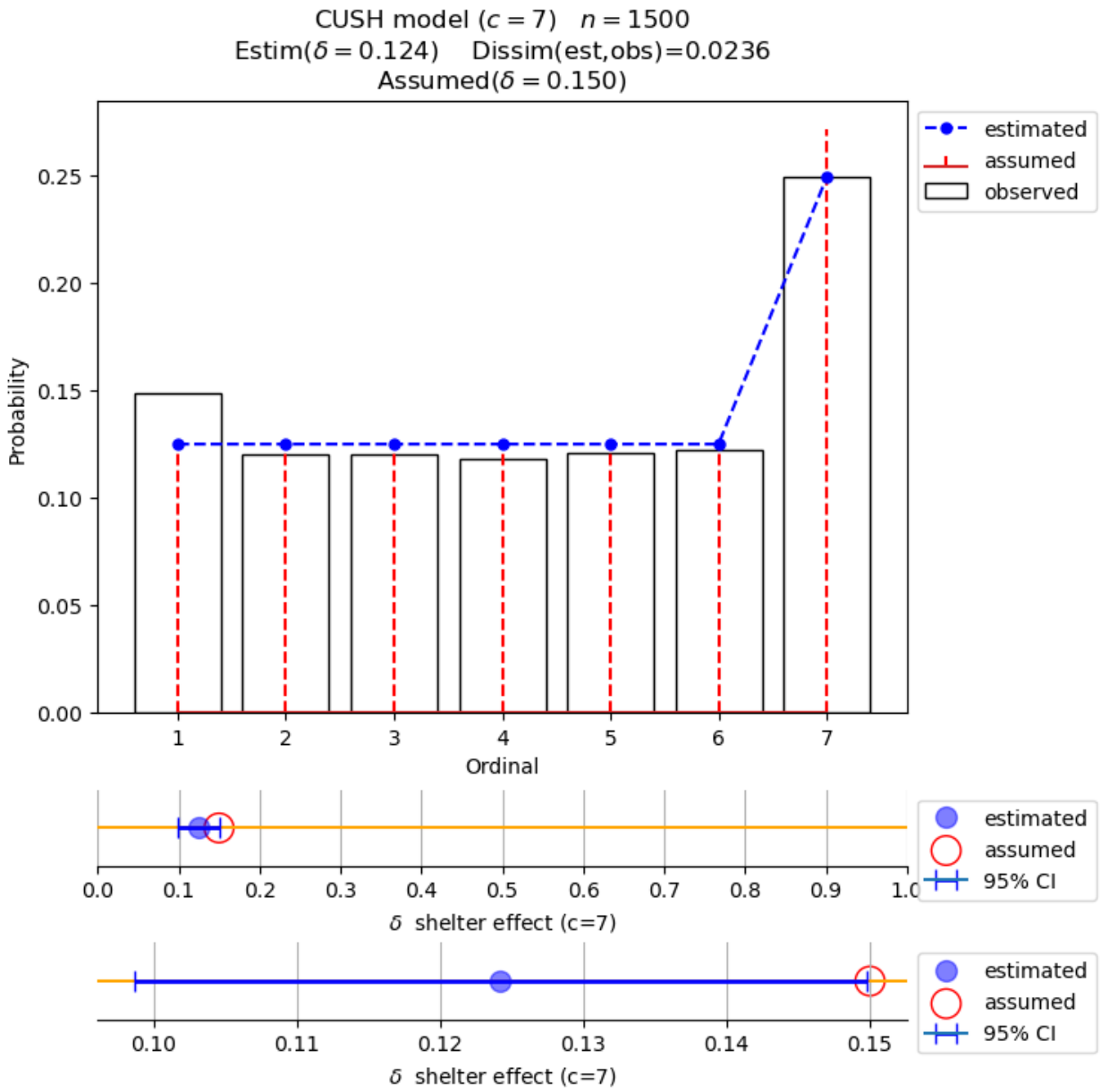
```
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate

# draw a sample
drawn = draw(
    formula="ord ~ 0",
    model="cush",
    sh=7,
    m=7, delta=.15,
    n=1500, seed=76)

# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    model="cush",
    formula="ord~0",
    sh=7,
    ass_pars={
        "delta": drawn.pars[0],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUSH model <<<===== ML-estimates
================================================================================
m=7  Shelter=7  Size=1500
--------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
Shelter effect
       Estimates  StdErr   Wald  p-value
delta      0.124  0.0130  9.532   0.0000
========================================================================
Dissimilarity = 0.0236
Loglik(sat)   = -2856.039
Loglik(MOD)   = -2859.923
Loglik(uni)   = -2918.865
Mean-loglik   = -1.907
Deviance      = 7.768
------------------------------------------------------------------------
AIC = 5721.85
BIC = 5727.16
========================================================================
Elapsed time=0.00113 seconds =====>>> Fri Aug 16 10:44:07 2024
========================================================================
```



2.4.2 With covariates

Reference guide

$$\Pr(R_i = r|\boldsymbol{\theta}, \boldsymbol{x}_i) = \delta_i D_r^{(c)} + (1 - \delta_i)/m,\ r = 1, 2, \ldots, m$$

$$\delta_i = \frac{1}{1 + \exp\{-\boldsymbol{x}_i\boldsymbol{\omega}\}} \quad \equiv \quad \text{logit}(\delta_i) = \boldsymbol{x}_i\boldsymbol{\omega}$$

In the example, we'll draw a sample from a CUSH model with covariates and then estimate the parameter given the observed sample.

Notice that, since the `model` is not the default `"cub"`, we need to specify it.

Listing 16: Script

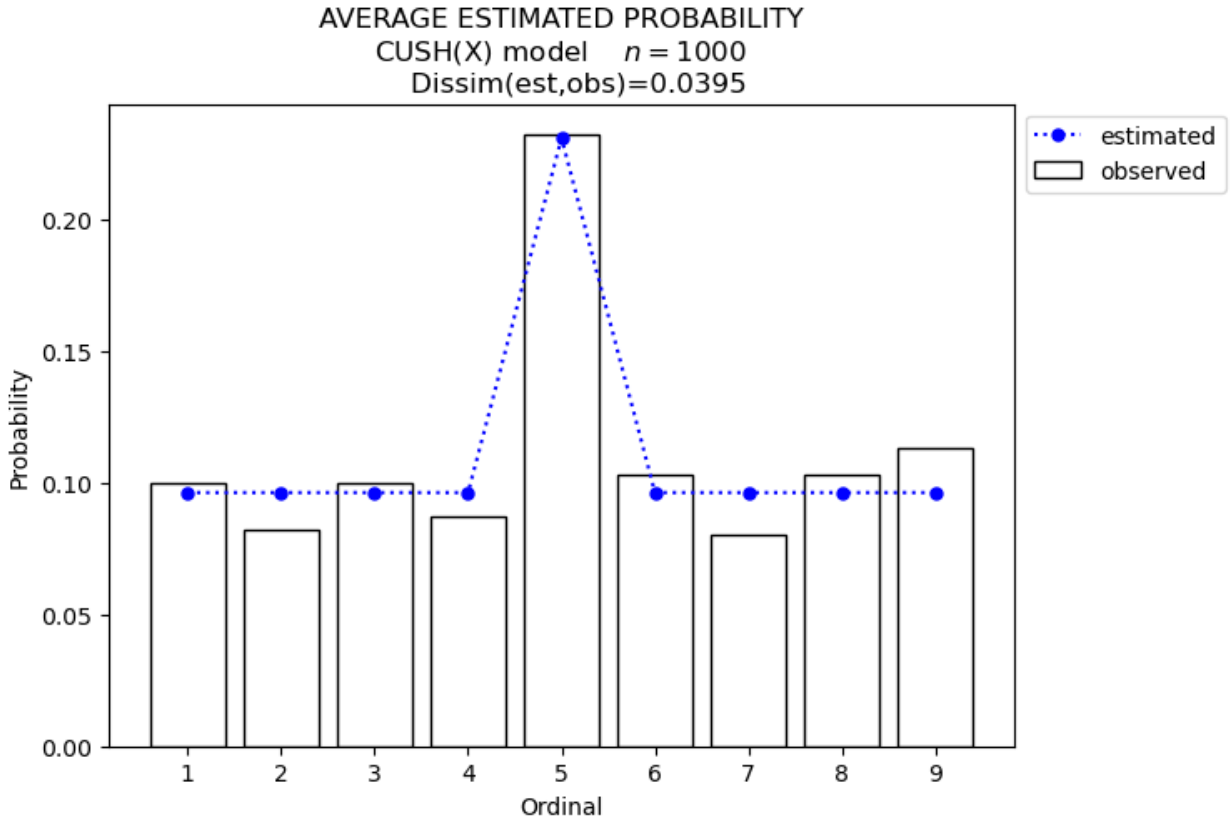
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.general import logit
from cubmods.gem import draw, estimate

# Draw a random sample
n = 1000
np.random.seed(1)
X = np.random.randint(1, 10, n)
df = pd.DataFrame({
    "X": X,
})
drawn = draw(
    formula="fee ~ X",
    model="cush",
    df=df,
    m=9, sh=5,
    omega=[logit(.05), .2],
)

# MLE estimation
fit = estimate(
    formula="fee ~ X",
    model="cush",
    df=drawn.df, sh=5,
)
# Print MLE summary
print(fit.summary())
# plot the results
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUSH(X) model <<<===== ML-estimates
================================================================================
m=9  Shelter=5  Size=1000
--------------------------------------------------------------------------------
Shelter effect
          Estimates  StdErr    Wald  p-value
constant     -3.131  0.4361  -7.180   0.0000
X             0.229  0.0629   3.642   0.0003
================================================================================
Dissimilarity = 0.0395
Loglik(MOD)   = -2130.030
Loglik(uni)   = -2197.225
Mean-loglik   = -2.130
--------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
AIC = 4264.06
BIC = 4273.87
================================================================================
Elapsed time=0.01704 seconds =====>>> Fri Aug 16 10:54:11 2024
================================================================================
```



2.5 CUSH2 family

Family of the class CUSH with two shelter effects (CUSH2). See the references for details.

This family has been introduced by Pierini, 2024 (pp 16-20) and first implemented in this Python package. See Piccolo and Simone, 2019 as a reference for the CUB class models.

These models are particularly useful whenever the shelter choices are not *polarized*, i.e. they're not at the extremes of the ordinal variable support. In these cases, finite mixtures of the Beta Discretized distribution can be used (Simone, 2022).

2.5.1 Without covariates

Reference guide

$$\Pr(R = r|\boldsymbol{\theta}) = \delta_1 D_r^{(c_1)} + \delta_2 D_r^{(c_2)} + (1 - \delta_1 - \delta_2)/m,\ r = 1, 2, \ldots, m$$

In the example, we'll draw a sample from a CUSH2 model without covariates and then estimate the parameter given the observed sample.

Notice that, since the `model` is not the default `"cub"`, we need to specify it. Passing a list of two shelter categories with the *kwarg* `sh`, a CUSH2 model will be called.

Listing 17: Script

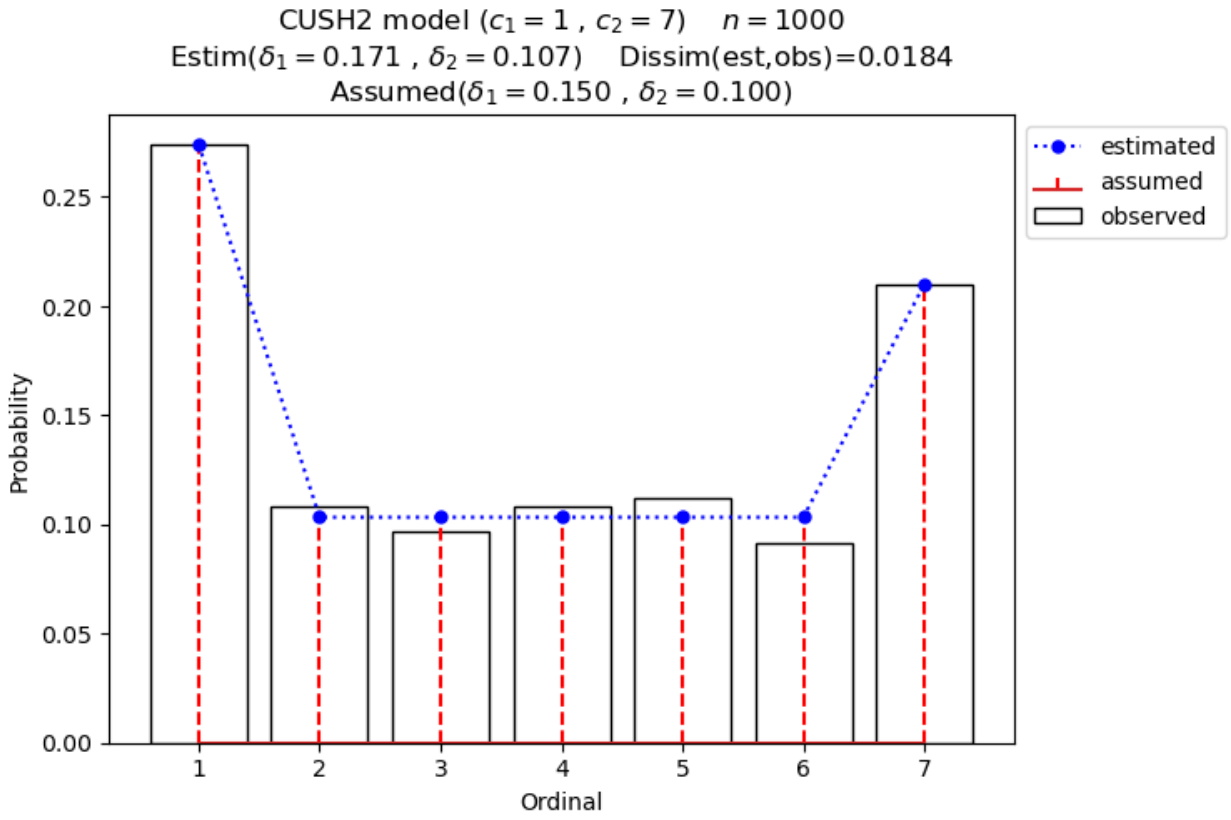
```
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate

# draw a sample
drawn = draw(
    formula="ord ~ 0 | 0",
    model="cush",
    sh=[1,7],
    m=7,
    delta1=.15, delta2=.1,
    n=1000, seed=42)

# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    model="cush",
    formula="ord~0|0",
    sh=[1,7],
    ass_pars={
        "delta1": drawn.pars[0],
        "delta2": drawn.pars[1],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

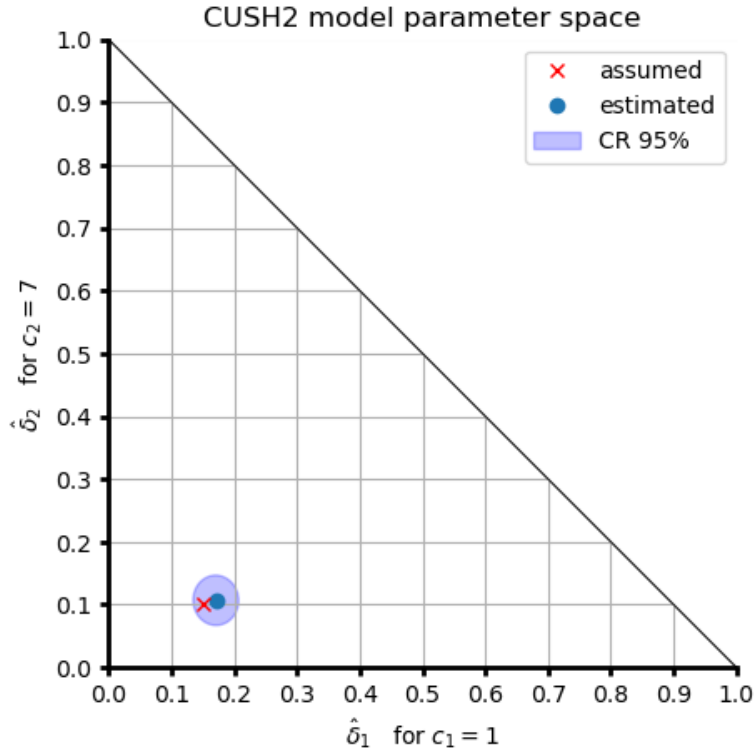
```
warnings.warn("No m given, max(ordinal) has been taken")
=========================================================================
=====>>> CUSH2 model <<<===== ML-estimates
=========================================================================
m=7  Shelter=[1 7]  Size=1000
-------------------------------------------------------------------------
Shelter effects
        Estimates  StdErr    Wald  p-value
delta1      0.171  0.0148  11.535   0.0000
delta2      0.107  0.0163   6.555   0.0000
```

(continues on next page)

(continued from previous page)

```
========================================================================
Dissimilarity = 0.0184
Loglik(sat)   = -1852.818
Loglik(MOD)   = -1854.344
Loglik(uni)   = -1945.910
Mean-loglik   = -1.854
Deviance      = 3.053
------------------------------------------------------------------------
AIC = 3712.69
BIC = 3722.50
========================================================================
Elapsed time=0.00228 seconds =====>>> Fri Aug 16 11:15:21 2024
========================================================================
```


CUSH2 model ($c_1 = 1$, $c_2 = 7$) $n = 1000$
Estim($\delta_1 = 0.171$, $\delta_2 = 0.107$) Dissim(est,obs)=0.0184
Assumed($\delta_1 = 0.150$, $\delta_2 = 0.100$)
estimated
assumed
observed
Probability
0.00
0.05
0.10
0.15
0.20
0.25
1
2
3
4
5
6
7
Ordinal


CUSH2 model parameter space
assumed
estimated
CR 95%
$\hat{\delta}_2$ for $c_2 = 7$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
$\hat{\delta}_1$ for $c_1 = 1$

2.5.2 With covariates

Reference guide (X1|0)

Reference guide (X1|X2)

$$\Pr(R_i = r|\boldsymbol{\theta}, \boldsymbol{x}_{1i}, \boldsymbol{x}_{2i}) = \delta_{1i} D_r^{(c_1)} + \delta_{2i} D_r^{(c_2)} + (1 - \delta_{1i} - \delta_{2i})/m,\ r = 1, 2, \ldots, m$$

$$\begin{cases} \delta_{1i} = \dfrac{1}{1 + \exp\{-\boldsymbol{x}_{1i}\boldsymbol{\omega}_1\}} \\ \delta_{2i} = \dfrac{1}{1 + \exp\{-\boldsymbol{x}_{2i}\boldsymbol{\omega}_2\}} \end{cases} \equiv \begin{cases} \text{logit}(\delta_{1i}) = \boldsymbol{x}_{1i}\boldsymbol{\omega}_1 \\ \text{logit}(\delta_{2i}) = \boldsymbol{x}_{2i}\boldsymbol{\omega}_2 \end{cases}$$

Two CUSH2 models with covariates have been defined and implemented: for the first shelter choice only and for both.

In this example we'll draw a sample from a CUSH2 model with covariates for the first shelter choice only and will then estimate the parameters with a CUSH2 model with covariates for both shelter choices but using the symbol 1 in the formula for the second shelter choice to estimate the constant parameter only. This is usually not needed, but we do it here to confirm that $\text{expit}(\hat{\omega}_{20}) = \hat{\delta}_2$.

Notice that, since the `model` is not the default `"cub"`, we need to specify it.

Listing 18: Script

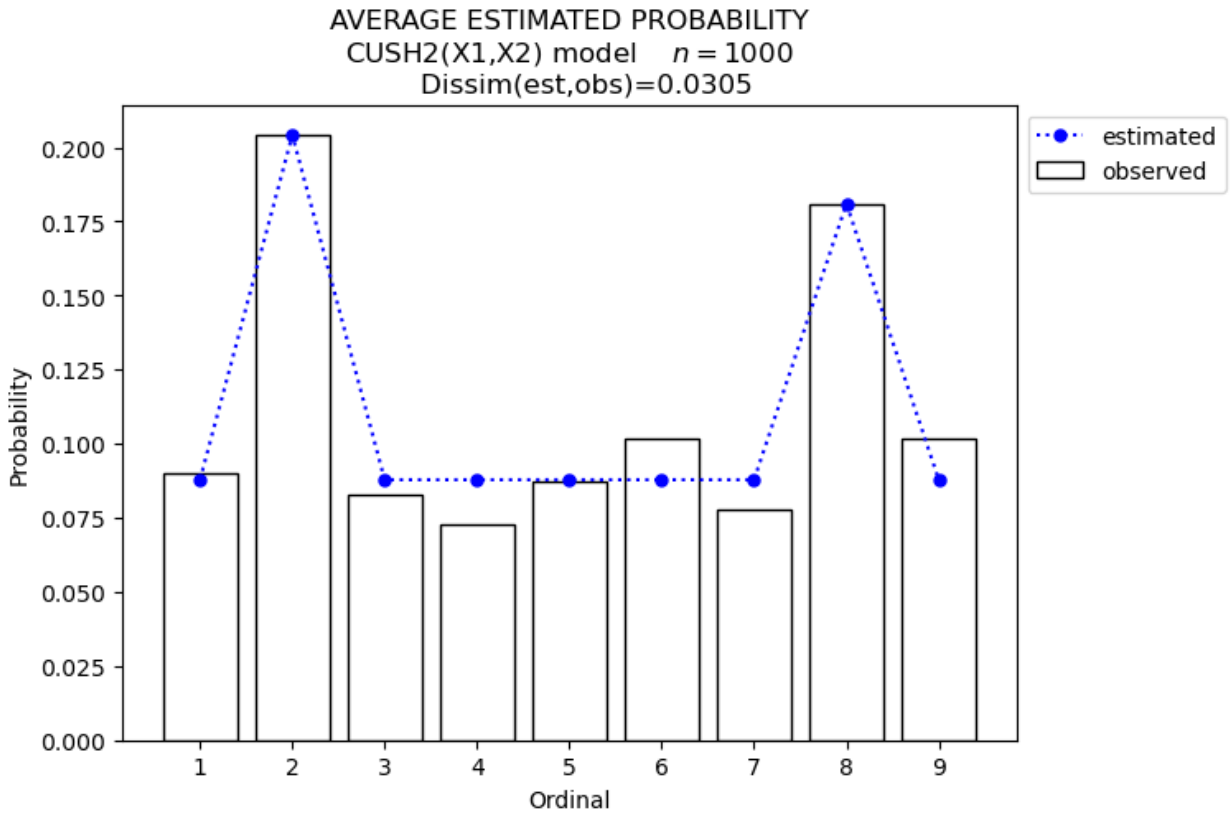
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.general import logit, expit
from cubmods.gem import draw, estimate

# Draw a random sample
n = 1000
np.random.seed(1)
X = np.random.randint(1, 10, n)
df = pd.DataFrame({
    "X": X,
})
drawn = draw(
    formula="fee ~ X | 0",
    model="cush",
    df=df,
    m=9, sh=[2, 8],
    omega1=[logit(.05), .2],
    delta2=.1
)

# MLE estimation
fit = estimate(
    formula="fee ~ X | 1",
    model="cush",
    df=drawn.df, sh=[2, 8],
)
# Print MLE summary
print(fit.summary())
# plot the results
```

(continues on next page)

(continued from previous page)

```
fit.plot()
plt.show()

est_de2 = expit(fit.estimates[2])
est_de2_es = expit(fit.estimates[2]+fit.stderrs[2]) - est_de2
print(
    "     estimates  stderr\n"
    f"delta2  {est_de2:.4f}  {est_de2_es:.4f}"
)
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUSH2(X1,X2) model <<<===== ML-estimates
================================================================================
m=9  Shelter=[2 8]  Size=1000
--------------------------------------------------------------------------------
Shelter effect 1
          Estimates  StdErr     Wald  p-value
constant     -3.170  0.4216   -7.519   0.0000
X             0.207  0.0613    3.379   0.0007
--------------------------------------------------------------------------------
Shelter effect 2
          Estimates  StdErr     Wald  p-value
constant     -2.276  0.1609  -14.149   0.0000
================================================================================
Dissimilarity = 0.0305
Loglik(MOD)   = -2122.463
Loglik(uni)   = -2197.225
Mean-loglik   = -2.122
--------------------------------------------------------------------------------
AIC = 4250.93
BIC = 4265.65
================================================================================
Elapsed time=0.06553 seconds =====>>> Fri Aug 16 11:29:11 2024
================================================================================
```




```
     estimates  stderr
delta2  0.0931  0.0145
```

2.6 CUBE family

Family of the class CUBE (Combination of Uniform and BEtaBinomial). CUB models are nested into CUBE models: in fact, a CUB model is equal to a CUBE model with the overdispersion parameter $\phi = 0$. Notiche that $0 \geq \phi \geq 0.2$ is the usual range of the overdispersion parameter.

See the references for details: Iannario, 2014; Piccolo, 2015; Piccolo and Simone, 2019.

2.6.1 Without covariates

Reference guide

$$\Pr(R = r|\boldsymbol{\theta}) = \pi\beta e(\xi, \phi) + \frac{1-\pi}{m},, \; r = 1, 2, \ldots, m$$

In this example, we'll draw a sample from a CUBE model and then will estimate the parameters given the observed sample.

Notice that, since the `model` is not the default `"cub"`, we need to specify it.

The *.plot()* method of the object *fit* will show trivariate and bivariate confidence regions too, as in CUBSH models. See *here* for the values of confidence levels.

Listing 19: Script

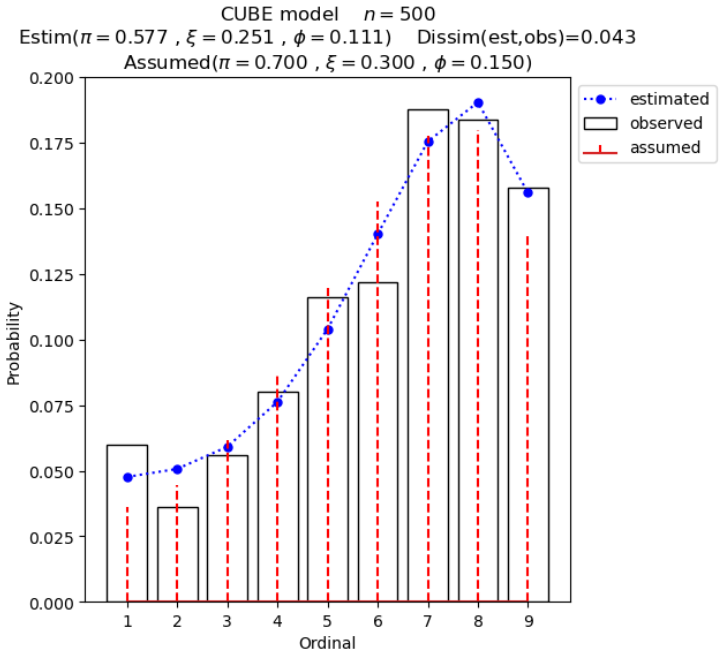
```python
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate

# draw a sample
drawn = draw(
    formula="ord ~ 0 | 0 | 0",
    model="cube",
    m=9, pi=.7, xi=.3, phi=.15,
    n=500, seed=1)

# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    formula="ord~0|0|0",
    model="cube",
    ass_pars={
        "pi": drawn.pars[0],
        "xi": drawn.pars[1],
        "phi": drawn.pars[2],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

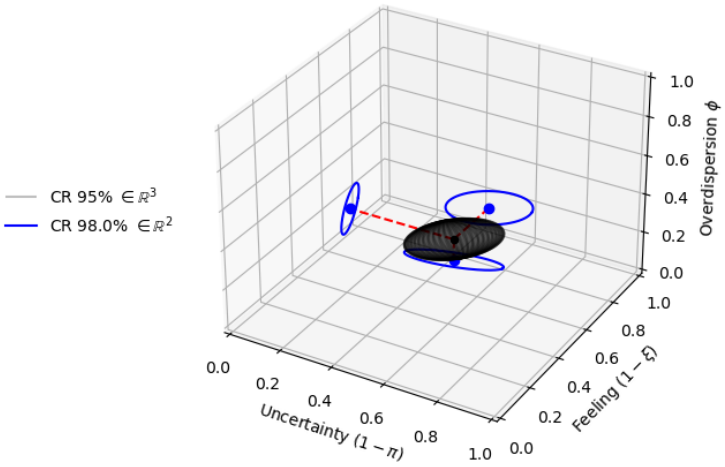
```
warnings.warn("No m given, max(ordinal) has been taken")
==========================================================================
=====>>> CUBE model <<<===== ML-estimates
==========================================================================
m=9  Size=500  Iterations=62  Maxiter=1000  Tol=1E-06
--------------------------------------------------------------------------
Uncertainty
     Estimates  StdErr    Wald  p-value
pi       0.577  0.0633   9.108   0.0000
--------------------------------------------------------------------------
Feeling
     Estimates  StdErr    Wald  p-value
xi       0.251  0.0217  11.560   0.0000
--------------------------------------------------------------------------
Overdispersion
     Estimates  StdErr    Wald  p-value
phi      0.111  0.0402   2.754   0.0059
==========================================================================
Dissimilarity = 0.0426
Loglik(sat)   = -1037.855
Loglik(MOD)   = -1041.100
Loglik(uni)   = -1098.612
```

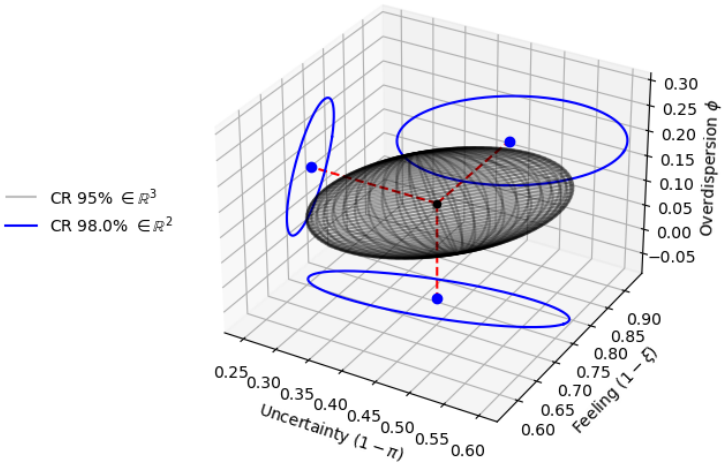
(continues on next page)

(continued from previous page)

```
Mean-loglik   = -2.082
Deviance      = 6.491
------------------------------------------------------------------------
AIC = 2088.20
BIC = 2100.84
========================================================================
Elapsed time=0.07919 seconds =====>>> Fri Aug 16 12:18:49 2024
========================================================================
```


CUBE model $n = 500$
Estim($\pi = 0.577$, $\xi = 0.251$, $\phi = 0.111$) Dissim(est,obs)=0.043
Assumed($\pi = 0.700$, $\xi = 0.300$, $\phi = 0.150$)
estimated
observed
assumed
Probability
Ordinal


CR 95% $\in \mathbb{R}^3$
CR 98.0% $\in \mathbb{R}^2$
Overdispersion ϕ
Uncertainty $(1 - \pi)$
Feeling $(1 - \xi)$


CR 95% $\in \mathbb{R}^3$
CR 98.0% $\in \mathbb{R}^2$
Overdispersion ϕ
Uncertainty $(1 - \pi)$
Feeling $(1 - \xi)$

2.6.2 With covariates

Reference guide (0|W|0)

Reference guide (Y|W|Z)

$$\Pr(R_i = r|\boldsymbol{\theta}; \boldsymbol{y}_i, \boldsymbol{w}_i; \boldsymbol{z}_i) = \pi_i \beta e(\xi_i, \phi_i) + \frac{1-\pi_i}{m}, , \; r = 1, 2, \ldots, m$$

$$\begin{cases} \pi_i = \dfrac{1}{1+\exp\{-\boldsymbol{y}_i\boldsymbol{\beta}\}} \\ \xi_i = \dfrac{1}{1+\exp\{-\boldsymbol{w}_i\boldsymbol{\gamma}\}} \\ \phi_i = \exp\{\boldsymbol{z}_i\boldsymbol{\alpha}\} \end{cases} \equiv \begin{cases} \text{logit}(1-\pi_i) = -\boldsymbol{y}_i\boldsymbol{\beta} \\ \text{logit}(1-\xi_i) = -\boldsymbol{w}_i\boldsymbol{\gamma} \\ \log\phi_i = \boldsymbol{z}_i\boldsymbol{\alpha} \end{cases}$$

Currently, as in the R package CUB, two CUBE models with covariates have been defined and implemented: for the *feeling* only and for all components. Nevertheless, the symbol 1 can always be used in the formula for different combinations of covariates.

In this example, we'll draw a sample with covariates for *feeling* only and then will estimate the parameters given the observed sample.

Listing 20: Script

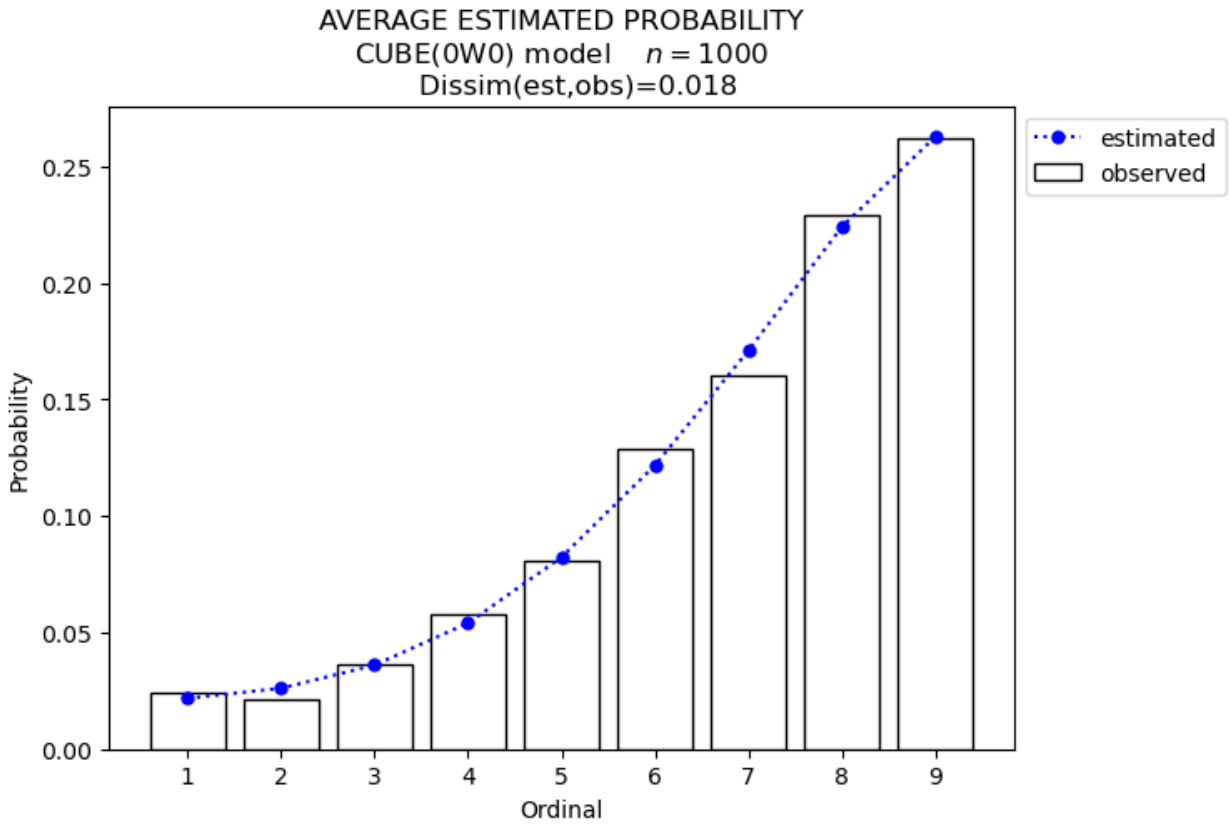
```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.general import expit, logit
from cubmods.gem import draw, estimate

# Draw a random sample
n = 1000
np.random.seed(76)
W = np.random.randint(1, 10, n)
df = pd.DataFrame({
    "W": W,
})
drawn = draw(
    formula="fee ~ 0 | W | 0",
    model="cube",
    df=df,
    m=9,
    pi=.8,
    gamma=[logit(.3), -.1],
    phi=.12,
)

# MLE estimation
fit = estimate(
    formula="fee ~ 0 | W | 0",
    model="cube",
    df=drawn.df,
)
# Print MLE summary
print(fit.summary())
```

(continues on next page)

(continued from previous page)

```
# plot the results
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
==============================================================================
=====>>> CUBE(0W0) model <<<===== ML-estimates
==============================================================================
m=9  Size=1000
------------------------------------------------------------------------------
Uncertainty
          Estimates  StdErr    Wald  p-value
pi            0.815  0.0343  23.733   0.0000
------------------------------------------------------------------------------
Feeling
          Estimates  StdErr    Wald  p-value
constant     -0.770  0.1012  -7.612   0.0000
W            -0.116  0.0191  -6.052   0.0000
------------------------------------------------------------------------------
Overdisperson
          Estimates  StdErr    Wald  p-value
phi           0.150  0.0260   5.779   0.0000
==============================================================================
Dissimilarity = 0.0183
Loglik(MOD)   = -1886.654
Loglik(uni)   = -2197.225
Mean-loglik   = -1.887
------------------------------------------------------------------------------
AIC = 3781.31
BIC = 3800.94
==============================================================================
Elapsed time=2.30903 seconds =====>>> Fri Aug 16 12:31:10 2024
==============================================================================
```



Notice that the same results can be achieved using a CUBE model with covariates for all components and passing the symbol 1 to the *uncertainty* and *overdispersion* components.

Listing 21: Script

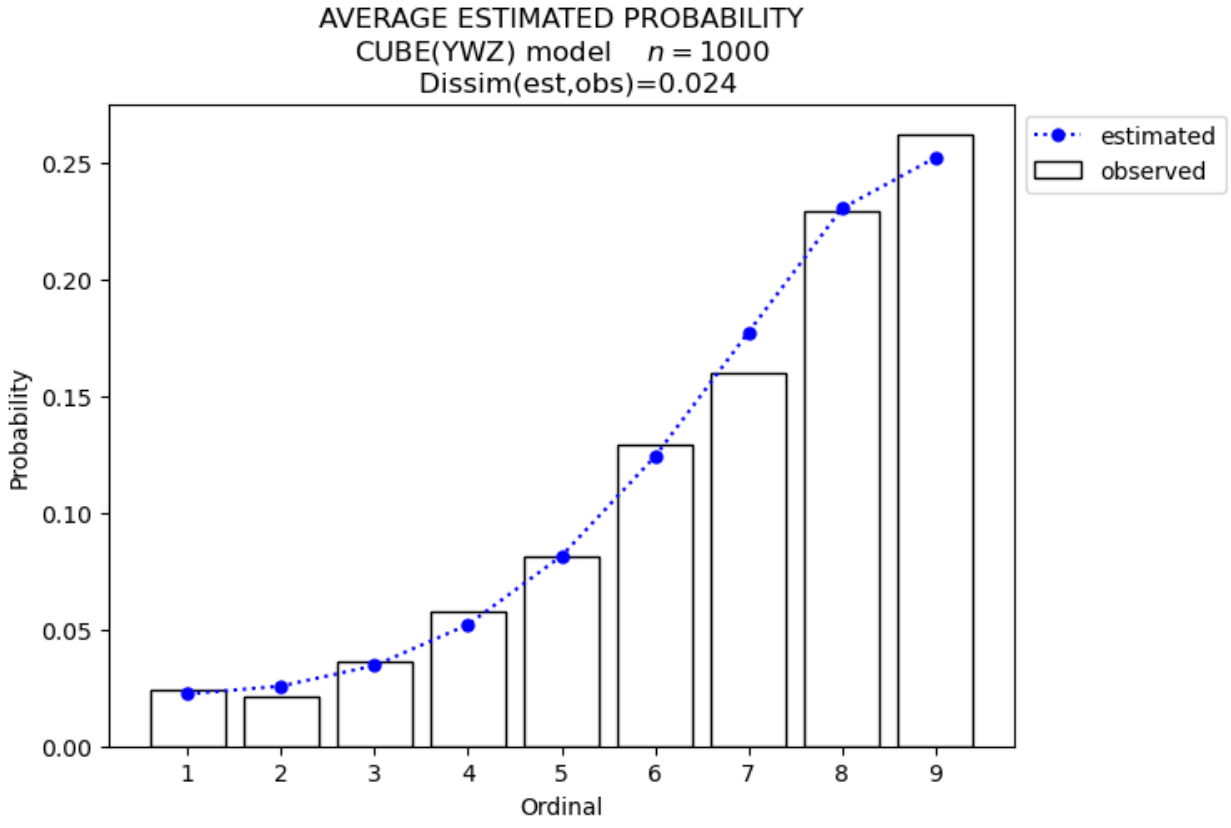
```
# MLE estimation
fit = estimate(
    formula="fee ~ 1 | W | 1",
    model="cube",
    df=drawn.df,
)
# Print MLE summary
print(fit.summary())
# plot the results
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> CUBE(YWZ) model <<<===== ML-estimates
================================================================================
m=9  Size=1000  Iterations=29  Maxiter=1000  Tol=1E-02
--------------------------------------------------------------------------------
Uncertainty
          Estimates  StdErr     Wald  p-value
constant      1.423  0.2183    6.518   0.0000
```

(continues on next page)

(continued from previous page)

```
------------------------------------------------------------------------
Feeling
          Estimates  StdErr     Wald  p-value
constant     -0.778  0.1018   -7.639   0.0000
W            -0.117  0.0193   -6.074   0.0000
------------------------------------------------------------------------
Overdispersion
          Estimates  StdErr     Wald  p-value
constant     -1.930  0.1756  -10.989   0.0000
========================================================================
Dissimilarity = 0.0239
Loglik(MOD)   = -1886.690
Loglik(uni)   = -2197.225
Mean-loglik   = -1.887
------------------------------------------------------------------------
AIC = 3781.38
BIC = 3801.01
========================================================================
Elapsed time=50.02969 seconds =====>>> Fri Aug 16 12:33:36 2024
========================================================================
```



In fact:

Listing 22: Script

```
est_pi = expit(fit.estimates[0])
est_ph = np.exp(fit.estimates[3])
est_pi_se = expit(fit.estimates[0]+fit.stderrs[0]) - est_pi
est_ph_se = np.exp(fit.estimates[3]+fit.stderrs[3]) - est_ph
print(
    "     estimates  stderr\n"
    f"pi       {est_pi:.4f}  {est_pi_se:.4f}"
    "\n"
    f"phi      {est_ph:.4f}  {est_ph_se:.4f}"
)
```

```
     estimates  stderr
pi      0.8058  0.0319
phi     0.1451  0.0279
```

2.7 IHG family

Family of the class IHG (Inverse HyperGeometric).

See the references for details: D'Elia, 2003; D'Elia *et al.*, 2005; Piccolo and Simone, 2019.

2.7.1 Without covariates

Reference guide

$$\begin{cases} \Pr(R=1|\theta)=\theta \\ \Pr(R=r+1|\theta)=\Pr(R=r|\theta)(1-\theta)\dfrac{m-r}{m-1-r(1-\theta)}, \ r=1,2,\ldots,m-1 \end{cases}$$

which is equivalent to

$$\Pr(R=r|\theta)=\frac{\binom{m+B-r-1}{m-r}}{\binom{m+B-1}{m-1}}, \ r=1,2,\ldots,m$$
$$\text{with } B=(m-1)\theta/(1-\theta)$$

In this example, we'll draw a sample from an IHG model and the estimate the parameter from the observed sample.

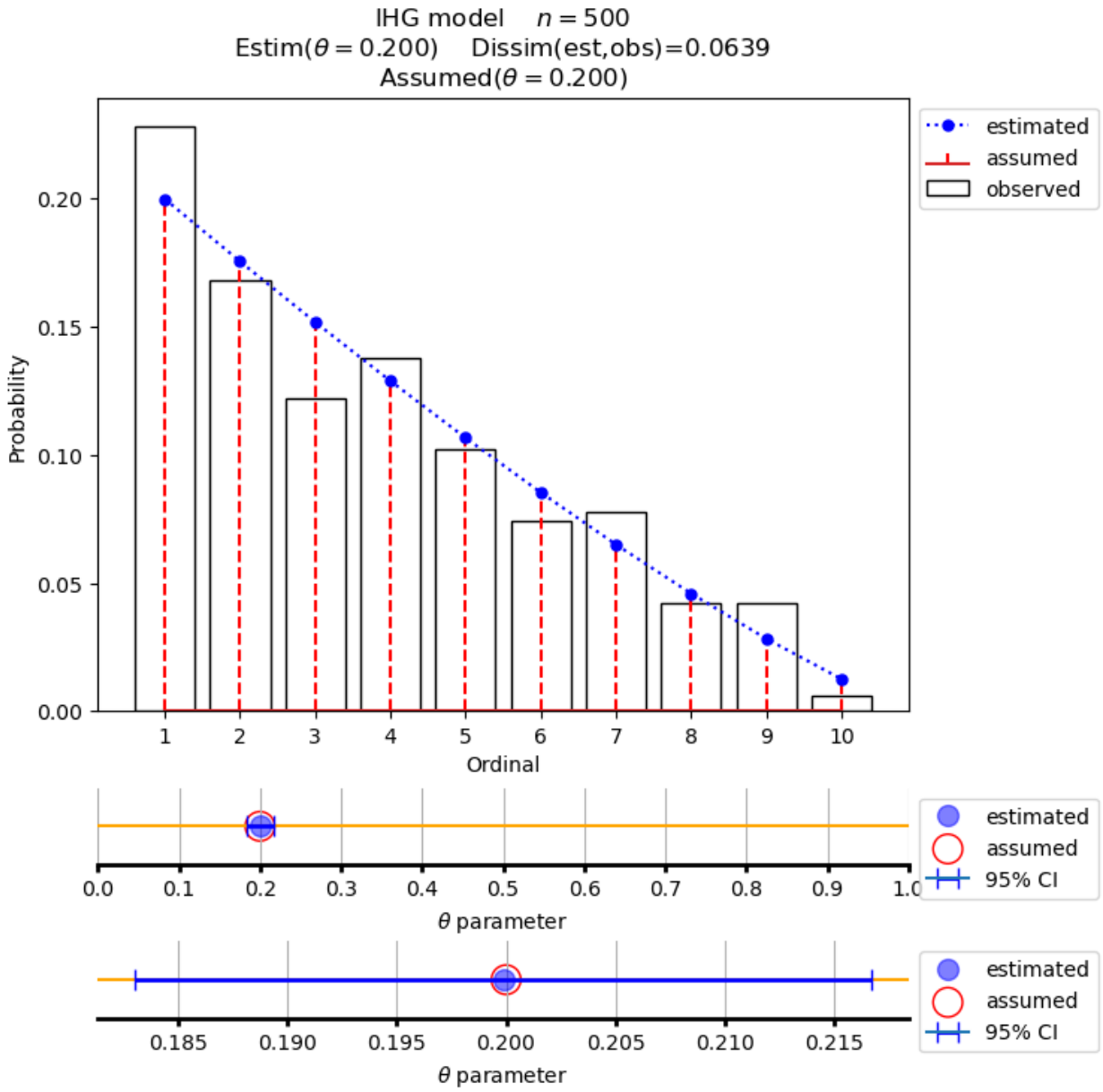
```
# import libraries
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate

# draw a sample
drawn = draw(
    formula="ord ~ 0",
    model="ihg",
    m=10, theta=.2,
    n=500, seed=42)
```

(continues on next page)

(continued from previous page)

```
# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    formula="ord ~ 0",
    model="ihg",
    ass_pars={
        "theta": drawn.pars[0],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> IHG model <<<===== ML-estimates
================================================================================
m=10  Size=500
--------------------------------------------------------------------------------
Theta
       Estimates  StdErr    Wald  p-value
theta      0.200  0.0086  23.292   0.0000
================================================================================
Dissimilarity = 0.0639
Loglik(sat)   = -1044.100
Loglik(MOD)   = -1050.513
Loglik(uni)   = -1151.293
Mean-loglik   = -2.101
Deviance      = 12.824
--------------------------------------------------------------------------------
AIC = 2103.03
BIC = 2107.24
================================================================================
Elapsed time=0.00464 seconds =====>>> Fri Aug 16 12:47:55 2024
================================================================================
```



2.7.2 With covariates

Reference guide

$$\begin{cases} \Pr(R_i = 1|\boldsymbol{\theta}; \boldsymbol{v}_i) = \theta_i \\ \Pr(R_i = r+1|\boldsymbol{\theta}; \boldsymbol{v}_i) = \Pr(R_i = r|\boldsymbol{\theta}; \boldsymbol{v}_i)(1-\theta_i)\dfrac{m-r}{m-1-r(1-\theta_i)}, \ r = 1, \ldots, m-1 \end{cases}$$

$$\theta_i = \frac{1}{1+\exp\{-\boldsymbol{v}_i\boldsymbol{\nu}\}} \quad \equiv \quad \text{logit}(\theta_i) = \boldsymbol{v}_i\boldsymbol{\nu}$$

In this example we'll draw a sample from an IHG with two covariates and then will estimate the parameters given the observed sample. Notice that IHG models without covariates are unimodals but, however, IHG models with covariates can be bimodal, as the one in the following example.

Listing 23: Script

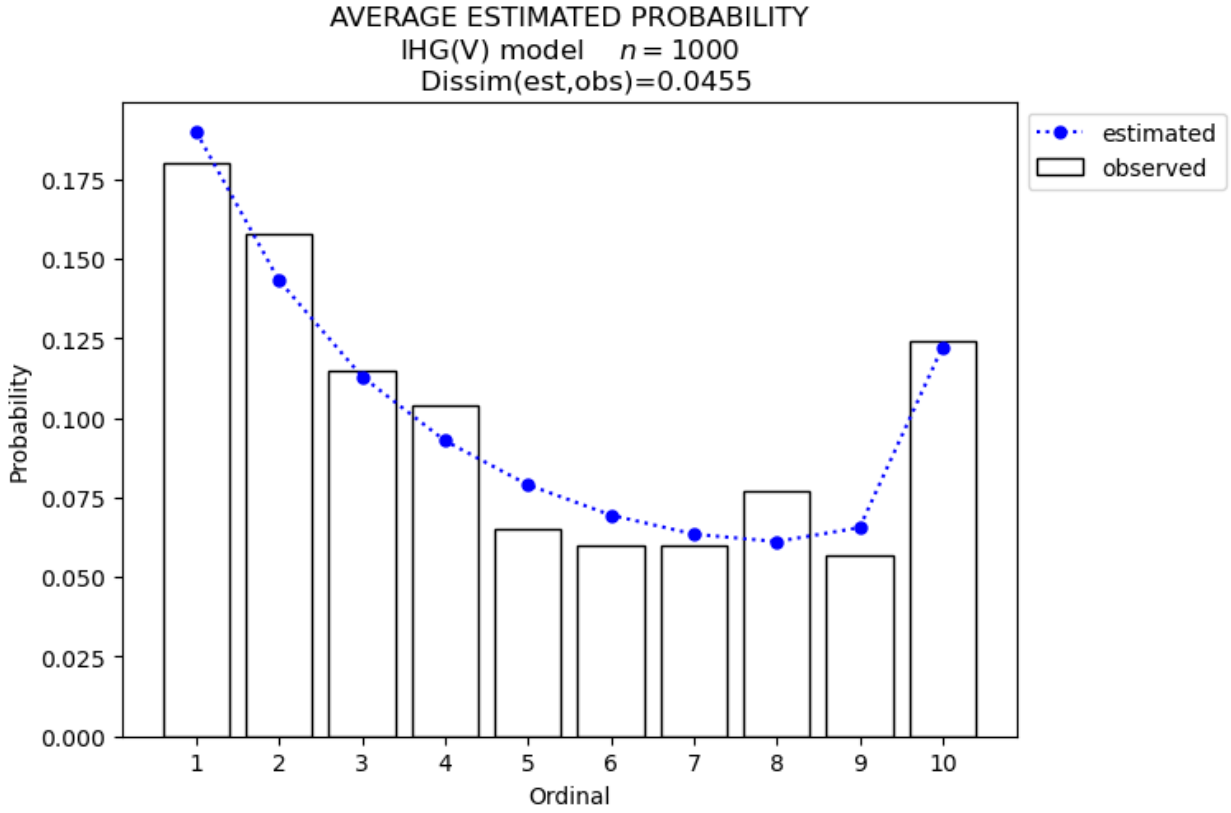
```python
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.gem import draw, estimate
from cubmods.general import logit

# Draw a random sample
n = 1000
np.random.seed(1)
V1 = np.random.random(n)
np.random.seed(42)
V2 = np.random.random(n)
df = pd.DataFrame({
    "V1": V1, "V2": V2
})

# draw a sample
drawn = draw(
    df=df,
    formula="ord ~ V1 + V2",
    model="ihg",
    m=10,
    nu=[logit(.1), -2, 3],
    seed=42)

# inferential method on drawn sample
fit = estimate(
    df=drawn.df,
    formula=drawn.formula,
    model="ihg",
    ass_pars={
        "theta": drawn.pars[0],
    }
)
# print the summary of MLE
print(fit.summary())
# show the plot of MLE
fit.plot()
plt.show()
```

```
warnings.warn("No m given, max(ordinal) has been taken")
================================================================================
=====>>> IHG(V) model <<<===== ML-estimates
================================================================================
m=10  Size=1000
--------------------------------------------------------------------------------
Theta
          Estimates  StdErr     Wald  p-value
constant     -2.368  0.0998  -23.741   0.0000
```

(continues on next page)

(continued from previous page)

```
V1           -1.973  0.1438  -13.721   0.0000
V2            3.230  0.1451   22.261   0.0000
=============================================================================
Dissimilarity = 0.0455
Loglik(MOD)   = -1958.475
Loglik(uni)   = -2302.585
Mean-loglik   = -1.958
-----------------------------------------------------------------------------
AIC = 3922.95
BIC = 3937.67
=============================================================================
Elapsed time=1.10664 seconds =====>>> Fri Aug 16 12:53:12 2024
=============================================================================
```



2.8 MULTICUB

See the Piccolo and Simone, 2019 as a reference.

Reference guide

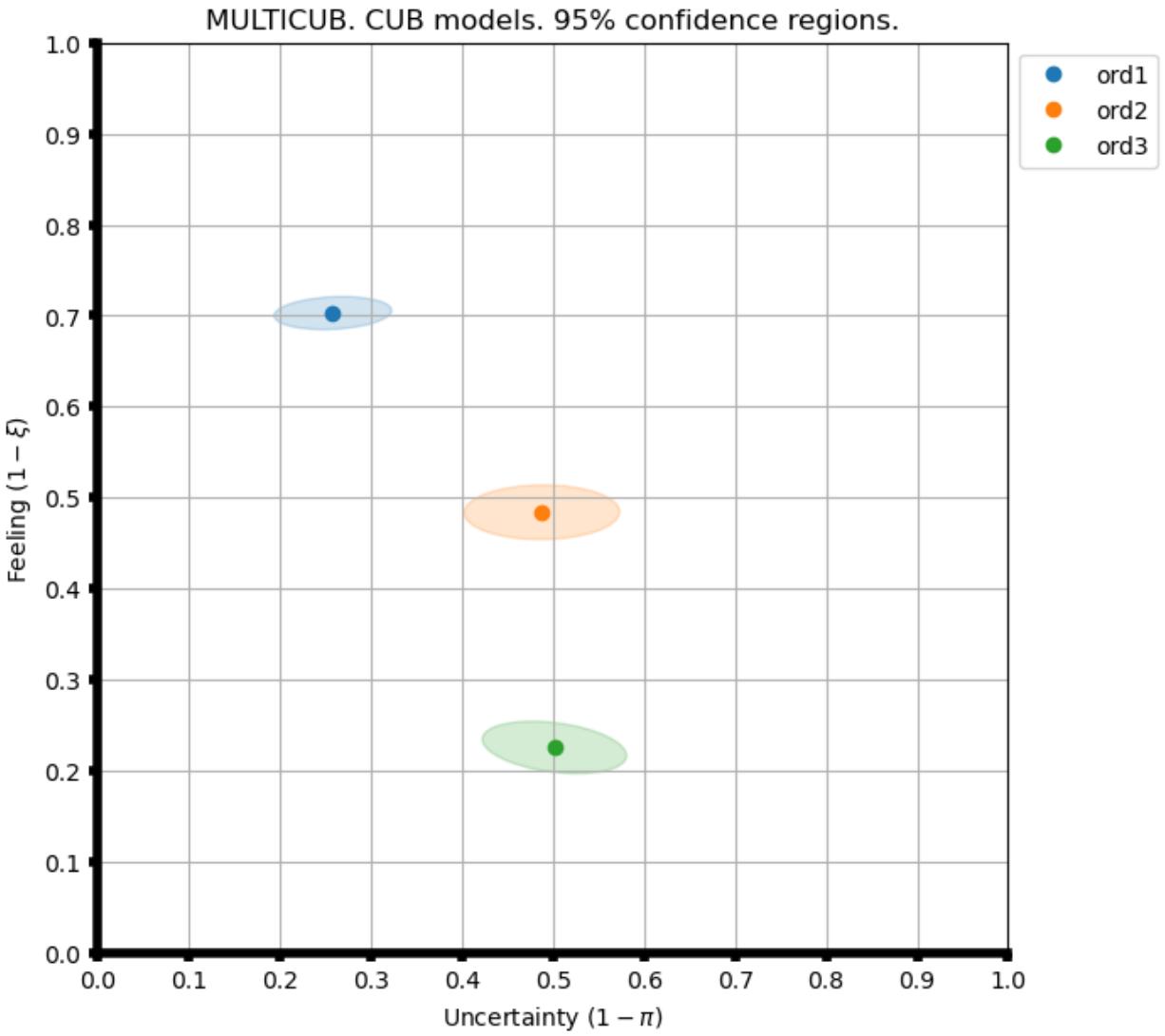
With the **multicub** tool, parameters estimated from multiple observed samples can be shown in a single plot.

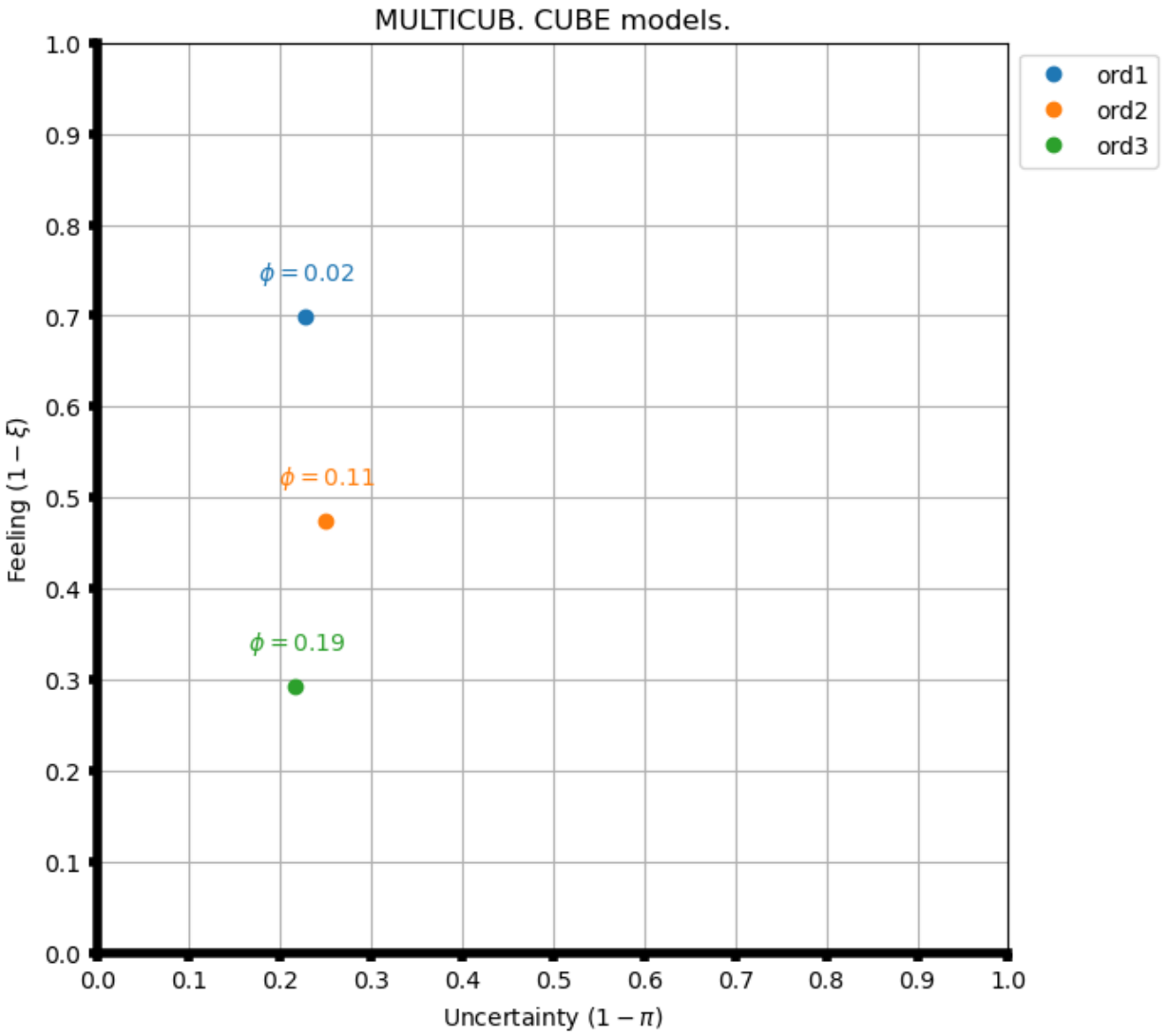
In this example, we'll draw three samples from CUBE models and *manually* add a shelter category. Then we'll use the **multicub** tool for CUB models, CUBE models and CUBSH models (that aren't yet implemented in the R package `CUB`).

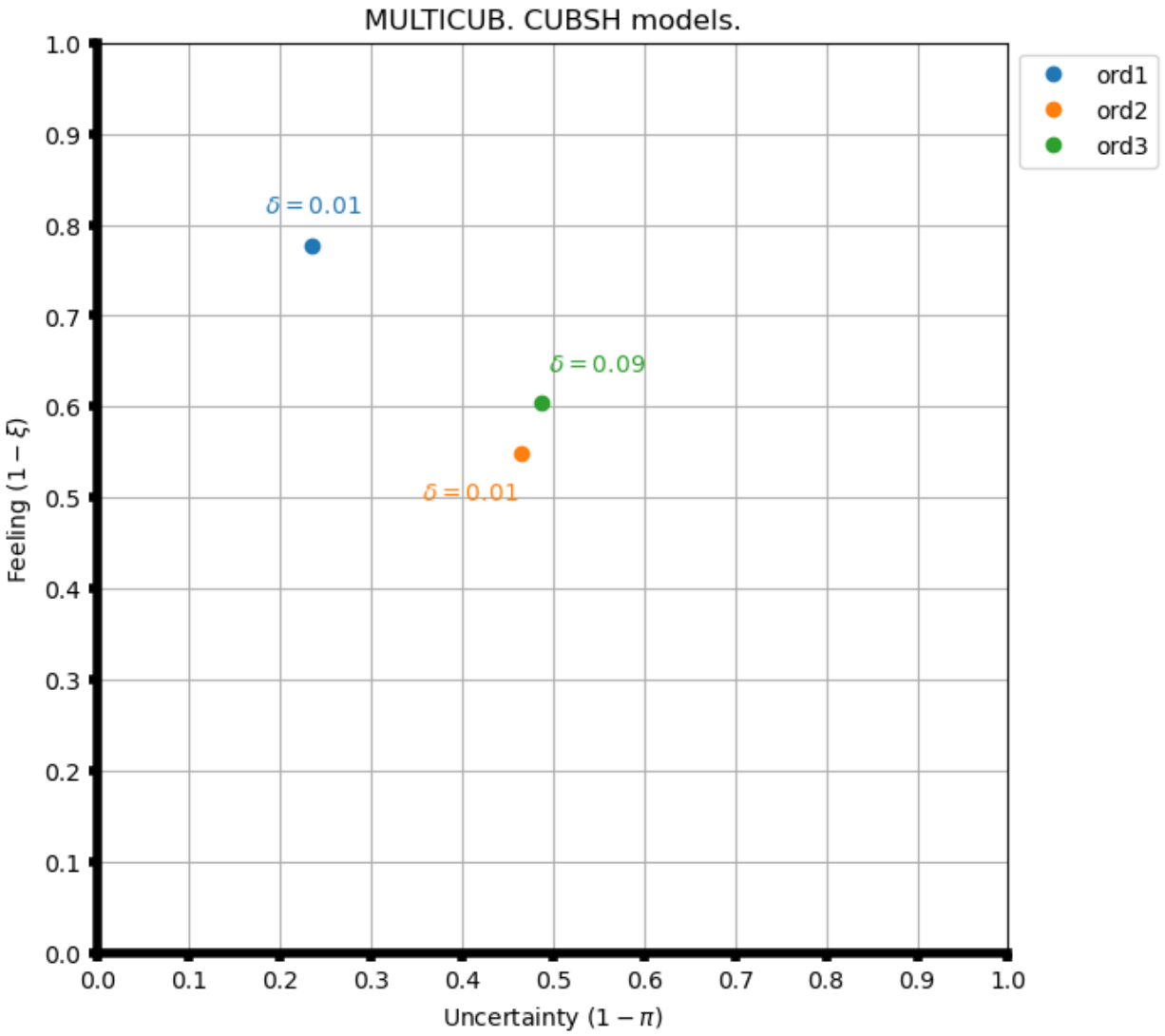
The **multicub** tool in cubmods package can also show confidence ellipses for CUB models.

Listing 24: Script

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from cubmods.gem import draw
from cubmods.multicub import multi

# draw random samples
df = pd.DataFrame()
for i, (pi, xi, phi) in enumerate(
    zip([.9, .8, .7], [.3, .5, .7], [.05, .1, .15])
    ):
    drawn = draw(
        formula="ord ~ 0 | 0 | 0",
        m = 9, model="cube", n=1000,
        pi=pi, xi=xi, phi=phi,
        seed=1976
    )
    # add a shelter category at c=1
    df[f"ord{i+1}"] = np.concatenate((
        drawn.rv, np.repeat(1, 25)
    ))

# MULTI-CUB
multi(
    ords=df, ms=9, model="cub"
)
plt.show()
# MULTI-CUBE
multi(
    ords=df, ms=9, model="cube"
)
plt.show()
# MULTI-CUBSH
multi(
    ords=df, ms=9, model="cub", shs=1,
    pos=[1, 6, 2]
)
plt.show()
```


MULTICUB. CUB models. 95% confidence regions.
ord1
ord2
ord3
Feeling $(1-\xi)$
Uncertainty $(1-\pi)$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


MULTICUB. CUBE models.
ord1
ord2
ord3
$\phi = 0.02$
$\phi = 0.11$
$\phi = 0.19$
Feeling $(1 - \xi)$
Uncertainty $(1 - \pi)$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0


MULTICUB. CUBSH models.
ord1
ord2
ord3
$\delta = 0.01$
$\delta = 0.09$
$\delta = 0.01$
Feeling $(1 - \xi)$
Uncertainty $(1 - \pi)$
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

CHAPTER

THREE

CUBMODS

3.1 cubmods package

3.1.1 Submodules

3.1.2 cubmods.cub module

3.1.3 cubmods.cub_0w module

3.1.4 cubmods.cub_y0 module

3.1.5 cubmods.cub_yw module

3.1.6 cubmods.cube module

3.1.7 cubmods.cube_0w0 module

3.1.8 cubmods.cube_ywz module

3.1.9 cubmods.cubsh module

3.1.10 cubmods.cubsh_ywx module

3.1.11 cubmods.cush module

3.1.12 cubmods.cush2 module

3.1.13 cubmods.cush2_x0 module

3.1.14 cubmods.cush2_xx module

3.1.15 cubmods.cush_x module

3.1.16 cubmods.gem module

3.1.17 cubmods.general module

3.1.18 cubmods.ihg module

3.1.19 cubmods.ihg_v module

3.1.20 cubmods.multicub module

CHAPTER

FOUR

REFERENCES

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[CP17] Stefania Capecchi and Domenico Piccolo. Dealing with heterogeneity in ordinal responses. *Quality & Quantity*, 51:2375–2393, 2017.

[CSDI+21] Giovanni Cerulli, R Simone, Francesca Di Iorio, D Piccolo, CF Baum, and others. The Stata module for CUB models for rating data analysis. In *London Stata Conference 2021*, number 16. Stata Users Group, 2021.

[CIP09] Marcella Corduas, Maria Iannario, and Domenico Piccolo. A class of statistical models for evaluating services and performances. In *Statistical methods for the evaluation of educational services and quality of products*, pages 99–117. Springer, 2009.

[DEliaP+05] A D'Elia, Domenico Piccolo, and others. The moment estimator for the IHG distribution. In *S. Co. Modelli complessi e metodi computazionali intensivi per la stima e la previsione*, pages 245–250. CLEUP, 2005.

[DEliaP05] Angela D'Elia and Domenico Piccolo. A mixture model for preferences data analysis. *Computational Statistics & Data Analysis*, 49(3):917–934, 2005.

[DElia03] Angela D'Elia. Modelling ranks using the inverse hypergeometric distribution. *Statistical modelling*, 3(1):65–78, 2003.

[Ian12a] Maria Iannario. Cube models for interpreting ordered categorical data with overdispersion. *Quaderni di statistica*, 14:137–140, 2012.

[Ian12b] Maria Iannario. Modelling shelter choices in a class of mixture models for ordinal responses. *Statistical Methods & Applications*, 21:1–22, 2012.

[Ian14] Maria Iannario. Modelling uncertainty and overdispersion in ordinal data. *Communications in Statistics-Theory and Methods*, 43(4):771–786, 2014.

[IP09] Maria Iannario and Domenico Piccolo. A program in R for CUB models inference. *Available via Internet, URL http://www. dipstat. unina. it/CUBmodels/, Version*, 2009.

[IP10] Maria Iannario and Domenico Piccolo. A new statistical model for the analysis of customer satisfaction. *Quality Technology & Quantitative Management*, 7(2):149–168, 2010.

[IP+14] Maria Iannario, Domenico Piccolo, and others. Inference for CUB models: a program in R. *Statistica & Applicazioni*, 12(2):177–204, 2014.

[IPSMaintainer22] Maria Iannario, Domenico Piccolo, and Rosaria Simone (Maintainer). Package 'CUB'. *CRAN*, 2022.

[Pic03] Domenico Piccolo. On the moments of a mixture of uniform and shifted binomial random variables. *Quaderni di Statistica*, 5(1):85–104, 2003.

[Pic06] Domenico Piccolo. Observed information matrix for MUB models. *Quaderni di Statistica*, 8(1):33–78, 2006.

[Pic15] Domenico Piccolo. Inferential issues on cube models with covariates. *Communications in Statistics-Theory and Methods*, 44(23):5023–5036, 2015.

[PS19] Domenico Piccolo and Rosaria Simone. The class of CUB models: statistical foundations, inferential issues and empirical evidence (with discussion and rejoinder). *Statistical Methods & Applications*, 28:389–493, 2019.

[Pie24] Massimo Pierini. Modelli della classe CUB in python. *Universitas Mercatorum, Rome, IT*, pages 1–172, June 2024. (Bachelor's thesis L-41).

[PL20] W Stephen Pittard and Shuzhao Li. The essential toolbox of data science: python, R, git, and docker. *Computational Methods and Data Analysis for Metabolomics*, pages 265–311, 2020.

[Sim22] Rosaria Simone. On finite mixtures of Discretized Beta model for ordered responses. *TEST*, 31(3):828–855, 2022.

[SDIL19] Rosaria Simone, Francesca Di Iorio, and Riccardo Lucchetti. CUB for gretl. *GNU Regression, Econometrics and Time Series Library*, pages 147, 2019.

PYTHON MODULE INDEX