Definition of the NPS function in python

David Garibay, M.P.P.* Hawre Jalal, MD, Ph.D.†
Fernando Alarid-Escudero, Ph.D.‡§

Code function

This document presents a python implementation of the multivariate categorical sampling function mentioned in the "A Fast Nonparametric Sampling (NPS) Method for Time-to-Event in Individual-Level Simulation Models." manuscript, and used in the R examples provided in the Appendix.

```
def nps_nhpp(a_probs, correction, a_categories=None):
    """
    Generate samples from different probability distributions using a
    multivariate categorical distribution.

    Parameters:
    -----------
    a_probs (array): Array filled with independent and equally-length
    probability distributions.
    correction (str): String defining whether to a random variable between 0
    and 1 to approximate continous time ("uniform") or no modification at all
    after performing the sample ("none").
    a_categories (array): (Optional) Array defining the names of the
    categories to sample.

    Returns:
    -----------
    numpy.array: Array filled with the sampled categories for all the
```

*Health Research Consortium (CISIDAT), Cuernavaca, Morelos, Mexico.
†School of Epidemiology and Public Health, Faculty of Medicine, University of Ottawa, Ottawa, ON, CA.
‡Department of Health Policy, Stanford University School of Medicine, Stanford, CA, USA.
§Center for Health Policy, Freeman Spogli Institute, Stanford University, Stanford, CA, USA.

```
probability distributions.

Example:
-----------
import numpy as np
import scipy.stats as stats

# Number of repetitions
n_samp = 100

# define random numbers
a_unif_samp = np.random.uniform(size = n_rep*2)

# Create an array filled with categories
a_categories = np.arange(1, 101)

# Parameters of normal distribution
## First distribution
norm_mean_1, norm_var_1 = 30, 10
## second distribution
norm_mean_2, norm_var_2 = 60, 10

# Get PDF from distributions
v_disc_PDF_norm_1 = stats.norm.pdf(a_categories, loc = norm_mean_1,
  scale = norm_var_1)
v_disc_PDF_norm_2 = stats.norm.pdf(a_categories, loc = norm_mean_2,
  scale = norm_var_2)

# normalize values
## First version
v_norm_PDF_norm_1 = v_disc_PDF_norm_1/sum(v_disc_PDF_norm_1)
## Second version
v_norm_PDF_norm_2 = v_disc_PDF_norm_2/sum(v_disc_PDF_norm_2)

a_PDF_1_2 = np.array([v_norm_PDF_norm_1, v_norm_PDF_norm_2])

a_choice = np.random.choice(a = [0, 1], size = n_samp, replace = True,
  p = [0.5, 0.5])

a_probs = np.stack(arrays = a_PDF_1_2[a_choice], axis = 0)

nps_nhpp(a_probs= a_probs, correction="none")
```

```
    ----------
    """
    valid_correction = {'none', 'uniform'}

    if correction not in valid_correction:
      print("Warning: correction argument only accepts: 'none' and uniform")

      corresponding_values = None

    if a_categories is None:
      # Get number of categories
      a_categories = np.arange(0, a_probs.shape[1])

    # Check that all PDF's sum up to 1
    if not all(np.isclose(a_probs.sum(axis = 1), 1)):
      a_probs = a_probs/a_probs.sum(axis=1, keepdims=True)

    # Get number of elements to draw
    a_samp_size = a_probs.shape[0]

    # Obtain array filled with random numbers following a uniform distribution
    a_unif_probs = np.vstack(np.random.uniform(size = a_samp_size))

    # Get cumulative probabilities array from `a_probs`
    # Every row is a CDF
    a_cum_probs = np.cumsum(a_probs, axis = 1)

    # Compare uniform probabilities against cumulative probs
    comparison_result = a_cum_probs >= a_unif_probs

    # Getting positions where values are greater than or equal
    positions = np.argmax(comparison_result, axis = 1)

    corresponding_values = a_categories[positions]

    if correction == "uniform":
      corresponding_values = corresponding_values + a_unif_probs

    return corresponding_values
```