Example 3 using python

Time to event from parametric hazards

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

Code function

This document presents the python code corresponding to the third example presented in the "A Fast Nonparametric Sampling (NPS) Method for Time-to-Event in Individual-Level Simulation Models." manuscript.

```
# 01 Initial Setup ------------------------------------------------------

# Import required modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
import pandas as pd


# Define `nps_nhpp` function
def nps_nhpp(a_probs, correction, a_categories=None):

    valid_correction = {'none', 'uniform'}

    if correction not in valid_correction:
      print("Warning: incrrect inputs, allowed values: 'none' and 'uniform'")
      corresponding_values = None
```

*School of Epidemiology and Public Health, Faculty of Medicine, University of Ottawa, Ottawa, ON, CA.
†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.

```
    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 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
```

```
# 02 Load base data ---------------------------------------------------

df_mort_data_raw = pd.read_csv(
  filepath_or_buffer = "../data/all_cause_mortality.csv")
```

```
# 03 Data wrangling ---------------------------------------------------

# Keep data only for year 2015
df_mort_data_1 = df_mort_data_raw.query("Year == 2015").copy()
```

```
# Reset index (row ennumeration) of new data set
df_mort_data_1.reset_index(drop = True, inplace = True)

df_mort_data_1.sort_values(by = ["Sex", "Year", "Age"], inplace = True)

df_grouped = df_mort_data_1.groupby('Sex')

# H(t) - Cumulative hazard
df_mort_data_1['H_t'] = df_grouped['Rate'].cumsum()
# S(t) - Cumulative survival
df_mort_data_1['S_t'] = df_mort_data_1['H_t'].apply(lambda x: np.exp(-x))
# F(t) - Cumulative probability: 1 - S(t)
df_mort_data_1['F_t'] = 1 - df_mort_data_1['S_t']
# f(t) - Instantaneous probability
df_mort_data_1['p_t'] = (df_mort_data_1.groupby('Sex')['F_t'].
  diff().fillna(df_mort_data_1['F_t']))

#* Check sum of probabilities.
df_mort_data_1.groupby(["Sex"])["p_t"].sum()

#* It we incomplete, so we will create another row to fill the probabilities
df_mort_append_0 = df_mort_data_1.groupby(["Sex"]).tail(1)

df_mort_append_0.loc[:, "Age"] = 101
df_mort_append_0.loc[:, "p_t"] = 1 - df_mort_append_0["F_t"]
df_mort_append_0.loc[:, "F_t"] = df_mort_append_0[["F_t", "p_t"]].sum(axis=1)
df_mort_append_0.loc[:, "S_t"] = 1 - df_mort_append_0["F_t"]
df_mort_append_0.loc[:, "H_t"] = np.nan
df_mort_append_0.loc[:, ["H_t", "Rate"]]  = np.nan


# Concatenate the original and the extra dataframes
df_mort_data_2 = pd.concat([df_mort_data_1, df_mort_append_0])
df_mort_data_2.sort_values(by = ["Sex", "Year", "Age"], inplace = True)
df_mort_data_2.reset_index(drop = True, inplace = True)

# Now the sum of probs by Sex == 1
df_mort_data_2.groupby(["Sex"])["p_t"].sum()

#* Conver into wide format
#* - Year will be discarded while turning data into wide format
df_mort_data_2_wide = df_mort_data_2.pivot(
```

```
  columns = "Age",
  index   = ["Year", "Sex"],
  values  = "p_t")

df_mort_data_2_wide.reset_index(inplace = True)

df_mort_data_2_wide.rename_axis(None, axis = 1, inplace = True)
```

```
# 04 Sample times to events from heterogeneous cohorts ------------------

# Set seed for reproducibility
np.random.seed(seed = 1234)

# We will use the "Male" and "Female" sex categories
# to sample 100,000 individuals
a_sex = ["Male", "Female"]

# Sample size
n_samples = int(1e5)

# Allowed ages (0 to 102)
a_age_values = np.arange(0, 102)

# Sex proportions (50% males and 50% females)
p_sex = [0.5, 0.5]

# Instantiate base dataset
df_test_sex = pd.DataFrame(data = {"Year": np.repeat(2015, n_samples)})

# Draw sex of the individuals
df_test_sex["Sex"] = np.random.choice(
  a       = a_sex,
  size    = n_samples,
  replace = True,
  p       = p_sex)

# Append probability distribution based on Year and Sex
df_test_sex_probs = pd.merge(
  df_test_sex,
  df_mort_data_2_wide,
  how = "left",
  on  = ["Year", "Sex"])
```

```
# Obtain probability arrays
a_pob_probs = df_test_sex_probs.loc[:, 0:101].to_numpy()

#* Sample age of death for every individual distribution using the
#* `nps_nhpp` function
np.random.seed(seed = 234090) # Set seed for reproducibility
df_test_sex["age_death"] = nps_nhpp(
  a_probs    = a_pob_probs,
  correction = "none")

# Remove seed
np.random.seed(seed = None)
```