GETTSIM

Release 0.2.1

Nov 20, 2019

Table of Contents

GETTSIM aims at providing a depiction of the German Taxes and Transfers System that is usable across a wide range of research applications, ranging from highly complex dynamic programming models to extremely detailed microsimulation studies.

GETTSIM is implemented in Python, thereby achieving both user friendliness and flexibility. All features are extensively tested.

The current version is usable, but probably only by the initiated. Please get in touch (ideally by joining our Zulip Chat at https://gettsim.zulipchat.com) if you want to use it already, else keep an eye out for what there is to come!

CHAPTER 1

Getting Started

1.1 Installation

The recommended way to install GETTSIM is via conda, the standard package manager for scientific Python libraries. If conda is not installed on your machine, please follow the installation instructions of its user guide.

With conda available on your path, installing GETTSIM is as simple as typing

```
$ conda install -c gettsim gettsim
```

in a command shell.

To validate the installation, start a Python shell and type

```python
import gettsim

gettsim.test()
```

1.2 Usage

Say you have a dataset for the year 2019 that you store in a pandas DataFrame using the variable name `data`. Your dataset needs to adhere to a particular format; all the columns specified in *Required columns in input data* below need to be present.

To calculate the taxes and transfers for the households / tax units / individuals in the dataset, you can use the following Python code:

```python
from gettsim.tax_transfer import calculate_tax_and_transfers


calculate_tax_and_transfers(dataset=data, year_of_policy=2019)
```

The function `calculate_tax_and_transfers` will update your dataset `data` with the columns specified below in *Columns returned by the simulator*. Note that the dataset needs to be in long format, i.e. each person no matter what age has his/her own row.

1.3 Required columns in input data

Variable	Explanation
hid	Household identifier
tu_id	Tax Unit identifier
pid	Personal identifier
head_tu	Whether individual is head of tax unit
head	Whether individual is head of household
adult_num	Number of adults in household
child0_18_num	Number of children between 0 and 18 in household
hh_wealth	Wealth of household
m_wage	Monthly wage of each individual
age	Age of Individual
selfemployed	Whether individual is self-employed
east	Whether location in former east or west germany
haskids	Whether individual has kids
m_self	Monthly wage of selfemployment of each individual
m_pensions	Monthly pension payments of each individual
pkv	Whether individual is (only) privately health insured
m_wage_l1	Average monthly earnings, previous year
months_ue	Months in unemployment, current year
months_ue_l1	Months in unemployment, previous year
months_ue_l2	Months in unemployment, two years before
w_hours	Weekly working hours of individual
child_num_tu	Number of children in tax unit
adult_num_tu	Number of adults in tax unit
byear	Year of birth
exper	Labor market experience, in years
EP	Earning points for pension claim
child	Dummy: Either below 18yrs, or below 25 and in education
pensioner	Dummy: Pensioner employment status
m_childcare	Monthly childcare expenses
m_imputedrent	Monthly value of owner-occupied housing
m_kapinc	Monthly capital income
m_vermiet	Monthly rental income
miete	Monthly rent expenses (without heating)
heizkost	Monthly heating expenses
renteneintritt	Statutory retirement age (might be in the future)
handcap_degree	Handicap degree (between 0 and 100)
wohnfl	Size of dwelling in square meters
zveranl	Dummy: Married couple filing jointly for income tax
ineducation	Dummy: Employment status "in education"
alleinerz	Dummy: Single parent
eigentum	Dummy: owner-occupied housing
cnstyr	Construction year of dwelling (1: <1965,2:1966-2000,3:>2000)

Continued o

Table 1 – continued from previous page

Variable	Explanation
m_transfers	Sum of monthly public/private transfers not simulated. E.g. transfers from parents, alimonies, maternity leave pay

1.4 Columns returned by the simulator

Note that if one of these columns exists, it will be overwritten.

Variable	Explanation	Type
svbeit	Monthly amount employee soc. sec. contributions	Float
rvbeit	Monthly amount employee old-age pensions contrib.	Float
avbeit	Monthly amount employee unempl. insurance contrib.	Float
gkvbeit	Monthly amount employee health insurance contrib.	Float
m_alg1	Monthly amount of unemployment assistance	Float
pensions_sim	Monthly amount of old-age pensions	Float
gross_e1	Inc. from self-employment subject to tax, individual	Float
gross_e5	Inc. from Capital subject to tax, individual	Float
gross_e6	Inc. from Rents subject to tax, individual	Float
gross_e7	Inc. from Pensions subject to tax, individual	Float
gross_e1_tu	Inc. from Self-Employment subject to tax, couple sum	Float
gross_e4_tu	Inc. from Earnings subject to tax, couple sum	Float
gross_e5_tu	Inc. from Capital subject to tax, couple sum	Float
gross_e6_tu	Inc. from Rents subject to tax, couple sum	Float
gross_e7_tu	Inc. from Pensions subject to tax, couple sum	Float
abgst_tu	Monthly capital cncome tax due, couple sum	Float
abgst	Monthly capital cncome tax due, individual	Float
soli	Monthly solidarity surcharge due, individual	Float
soli_tu	Monthly solidarity surcharge due, couple sum	Float
kindergeld	Monthly child Benefit, individual	Float
kindergeld_tu	Monthly child Benefit, household sum	Float
incometax	Monthly income Tax Due, individual	Float
incometax_tu	Monthly income Tax Due, couple sum	Float
uhv	Alimony advance payment, individual	Float
regelbedarf	Household socio-economic *need*, incl. housing cost	Float
regelsatz	Household socio-economic *need*, lump-sum	Float
alg2_kdu	Housing cost covered by social assistance	Float
uhv_hh	Alimony advance payment, household sum	Float
kiz	Monthly additional child benefit, household sum	Float
wohngeld	Monthly housing benefit, household sum	Float
m_alg2	Monthly social assistance, household sum	Float
dpi_ind	Monthly disposable income, individual	Float
dpi	Monthly disposable income, household sum	Float
gross	Monthly market income	Float

CHAPTER 2

How To Contribute

Contributions are always welcome. Everything ranging from small extensions of the documentation to implementing new features is appreciated. Of course, the bigger the change the more it is necessary to reach out to us in advance for an discussion. You can start an discussion by posting an issue which can be a bug report or a feature request or something else.

To get acquainted with the code base, you can also check out the documentation or our issue tracker for some immediate and clearly defined tasks.

To contribute to the project, adhere to the following process.

1. The process starts differently for regular contributors and newcomers. As a contributor you might have been granted privileges to push to the GETTSIM repository. Thus, you can clone the repository directly using

   ```
   $ git clone https://github.com/iza-institute-of-labor-economics/gettsim
   ```

   As a newcomer or infrequent contributor, you must first create a fork of GETTSIM which is a copy of the repository into your account where you have unlimited access. Go to the Github page of GETTSIM and click on the fork button in the upper right corner. Then, clone your fork onto your disk with

   ```
   $ git clone https://github.com/<user>/gettsim
   ```

2. In the next step, go into the GETTSIM folder and set up the Python environment with

   ```
   $ conda env create
   ```

   Then, activate the environment and install the current GETTSIM version in the repository in development mode. Also, install pre-commits which are checks before a commit is accepted.

   ```
   $ conda activate gettsim
   $ conda develop .
   $ pre-commit install
   ```

3. We always develop new features in new branches. Thus, create a new branch by picking an appropriate name, e.g., `kindergeld-freibetrag` or `ubi`. Make sure to branch off from master and not any other branch.

```
$ git checkout -b <branch-name>
```

4. Now, develop the new feature on this branch. Before you commit the changes, make sure they pass our test suite which can be started with the following command.

   ```
   $ pytest
   ```

   Sometimes you want to push changes even if the tests fail because you need feedback. Then, skip this step.

5. In the next step, try to commit the changes to the branch with an appropriate commit message.

   ```
   $ git commit -am "Changed ... ."
   ```

   A commit starts the pre-commits which are additional checks, mostly formatting and style checks. If an error occurs, the commit is rejected and you need to review the log in your terminal to fix the issues. If an reported error is unclear to you, try to use Google for more help. After fixing all issues, you need to commit the changes again.

5. If your commit passes, push your changes to the repository. Then, go to either the official GETTSIM or your fork's Github page. A banner will be displayed asking you whether you would like to create a PR. Follow the link and the instructions of the PR template. Fill out the PR form to inform everyone else on what you are trying to accomplish and how you did it.

   The PR also starts a complete run of the test suite on a continuous integration server. The status of the tests is shown in the PR. You can follow the links to Azure Pipelines to get more details on why the tests failed. Reiterate on your changes until the tests pass on the remote machine.

6. Ask one of the main contributors to review your changes. Include their remarks in your changes.

7. The final PR will be merged by one of the main contributors.

2.1 FAQ

Question: I want to re-run the Azure Pipelines test suite because some random error occurred, e.g., a HTTP timeout error.

Answer: Starting from the Github page of the PR, select the tab called "Checks". In the upper right corner you find a button to re-run all checks. In a column on the left-hand-side you can re-run tests for individual platforms.

CHAPTER 3

How To Maintain

This document is dedicated to maintainers of GETTSIM.

3.1 Versioning

GETTSIM adheres in large parts to semantic versioning. Thus, for a given version number `major.minor.patch`

- `major` is incremented when you make incompatible API changes.
- `minor` is incremented when you add functionality which is backwards compatible.
- `patch` is incremented when you make backwards compatible bug fixes.

3.2 Branching Model

The branching model for GETTSIM is very simple.

1. New major and minor releases of GETTSIM are developed on the master branch.
2. For older major and minor releases there exist branches for maintenance called, for example, `0.1` or `1.3`. These branches are used to develop new patch versions.

   Once a minor version will not be supported anymore, the maintenance branch should be deleted.

3.3 How To Release

To release a new major or minor version of GETTSIM, do the following.

1. To start the release process for any new version, e.g., `0.2`, first create a new milestone on Github. Set the name to the version number (format is `v[major].[minor]`, in this example: `v0.2`) to collect issues and PRs.

A consensus among developers determines the scope of the new release. Note that setting up the milestone and determining the scope of the release will typically happen quite some time before the next steps.

2. Once all PRs in a milestone are closed:

   a. Update *Changes* with all necessary information regarding the new release.

   b. Use `bumpversion [major|minor|patch]` to increment all version strings. For example, to bump the version from `0.1.x` to `` ` ``0.2.0, type

   ```
   $ bumpversion minor
   ```

   c. Merge it to the master branch and create a maintenance branch `[major].[minor]`, i.e., `0.2` in this example.

3. The following step assigns a version and documents the release on Github. Go to the page for releases and draft a new release. The tag and title become `vx.y.z`. Make sure to target the master or maintenance branch. A long description is not necessary as the most important information is documented under *Changes*. Release the new version by clicking "Publish release".

4. On your local machine, pull the latest changes to the repository, check out the new release tag and run

```
$ python release.py
```

   which uploads the new release to the repository on Anaconda.org.

3.4 How To Maintain Previously Released Versions

Most changes to previously released versions come in the form of backports. Backporting is the process of re-applying a change to future versions of GETTSIM to older versions.

As backports can introduce new regressions, the scope is limited to critical bug fixes and documentation changes. Performance enhancements and new features are not backported.

3.4.1 Procedure

In the following we will consider an example where GETTSIM's stable version is `0.2.0`. Version `0.3.0` is currently developed on the master branch. There is a maintenance branch `0.2` to receive patches for the `0.2.x` line of releases. And a critical bug was found, which should be fixed in both `0.3.0` and in `0.2.1`.

1. Create a PR containing the bug fix which targets the master branch.
2. Add a note to the release notes for version 0.2.1.
3. Squash merge the PR into master and note down the commit sha.
4. Create a new PR against branch `0.2`. Call the branch for the PR `backport-pr[No.]-to-0.2.1` where `[No.]` is the PR number.
5. Use `git cherrypick -x <commit-sha>` with the aforementioned commit sha to apply the fix to the branch. Solve any merge conflicts, etc..
6. Add the PR to the milestone for version `0.2.1` so that all changes for a new release can be collected.
7. The release process for a patch version works as above in *How To Release* to release `0.2.1`; just that it is released off the maintenance branch, not off master.

3.5 FAQ

Question: I want to re-run the Azure Pipelines test suite because a merge to the master branch failed due to some random error, e.g., a HTTP timeout error.

Answer: Go to https://dev.azure.com/iza-institute-of-labor-economics/gettsim/_build. Make sure you are signed in. First click on the build which merged the PR to master. On the ensuing page, click on the button with the three vertical dots. Choose "Edit pipeline". Do not edit the configuration, but select "Run" in the upper right corner.

CHAPTER 4

Changes

This is a record of all past `gettsim` releases and what went into them in reverse chronological order. We follow semantic versioning and all releases are available on Anaconda.org.

4.1 0.2.1 - 2019-11-20

- Fix error with real SOEP data and "Wohngeld" for households with more than 12 household members (@Eric-Sommer, @MaxBlesch)
- Better description of required input and output columns (@MaxBlesch, @Eric-Sommer)
- Fix dependencies for conda package (@tobiasraabe)
- Fill changelog and include in docs (@tobiasraabe, @hmgaudecker)
- Add maintenance section to website (@tobiasraabe)

4.2 0.2.0 - 2019-11-06

This will be the initial release of `gettsim`.

- Set up as a conda-installable package (@tobiasraabe)
- Migration of the parameter database from xls to yaml (@mjbloemer, @MaxBlesch)
- Migration of test parameters from xls to csv (@MaxBlesch, @tobiasraabe)
- Get the main entry point to work, change interface (@MaxBlesch, janosg, @Eric-Sommer, @hmgaudecker, @tobiasraabe)
- Tax and transfer module uses apply instead of loops (@MaxBlesch, @hmgaudecker)
- Correct tax treatment of child care costs (@Eric-Sommer)
- Improve calculation of housing allowance (@Eric-Sommer)

4.3 0.1 and prior work - 2019-09-30

Most code written by @Eric-Sommer based on IZAΨMOD, a policy microsimulation model developed at IZA.

CHAPTER 5

GETTSIM Enhancement Protocols

Setting up GETTSIM and making major changes is done via the process of GETTSIM Enhancement Protocols, short GEPs. They serve the purpose of summarising discussions that may happen in chats, issues, pull requests, in person, or by any other means. They describe the basis for GETTSIM's architecture and set a process for adopting major changes (or not).

These GEPs are currently in place:

5.1 GEP 0 - Purpose and Process

Author Hans-Martin von Gaudecker

Status Provisional

Type Process

Created 2019-10-22

5.1.1 What is a GEP?

GEP stands for GETTSIM Enhancement Proposal. A GEP is a design document providing information to the GETTSIM community, or describing a new feature for GETTSIM or its processes or environment. The GEP should provide a concise technical specification of the feature and a rationale for the feature.

We intend GEPs to be the primary mechanisms for proposing major new features, for collecting community input on an issue, and for documenting the design decisions that have gone into GETTSIM. The GEP author is responsible for building consensus within the community and documenting dissenting opinions.

Because the GEPs are maintained as text files in a versioned repository, their revision history is the historical record of the feature proposal[1].

[1] This historical record is available by the normal git commands for retrieving older revisions, and can also be browsed on GitHub.

Types

There are three kinds of GEPs:

1. A **Standards Track** GEP describes a new feature or implementation for GETTSIM.
2. An **Informational** GEP describes a GETTSIM design issue, or provides general guidelines or information to the Python community, but does not propose a new feature. Informational GEPs do not necessarily represent a GETTSIM community consensus or recommendation, so users and implementers are free to ignore Informational GEPs or follow their advice.
3. A **Process** GEP describes a process surrounding GETTSIM, or proposes a change to (or an event in) a process. Process GEPs are like Standards Track GEPs but apply to areas other than the GETTSIM language itself. They may propose an implementation, but not to GETTSIM's codebase; they require community consensus. Examples include procedures, guidelines, changes to the decision-making process, and changes to the tools or environment used in GETTSIM development. Any meta-GEP is also considered a Process GEP.

5.1.2 GEP Workflow

The GEP process begins with a new idea for GETTSIM. It is highly recommended that a single GEP contain a single key proposal or new idea. Small enhancements or patches often don't need a GEP and can be injected into the GETTSIM development workflow with a pull request to the GETTSIM repository. The more focused the GEP, the more successful it tends to be. If in doubt, split your GEP into several well-focused ones.

Each GEP must have a champion—someone who writes the GEP using the style and format described below, shepherds the discussions in the appropriate forums, and attempts to build community consensus around the idea. The GEP champion (a.k.a. Author) should first attempt to ascertain whether the idea is suitable for a GEP. A message in #general/geps on Zulip is the best way to go about doing this.

The proposal should be submitted as a draft GEP via a GitHub pull request to the `doc/geps` directory with the name `gep-<n>.rst` where `<n>` is an appropriately assigned two-digit number (e.g., it is `gep-00.rst` for this document). The draft must use the *GEP X — Template and Instructions* file.

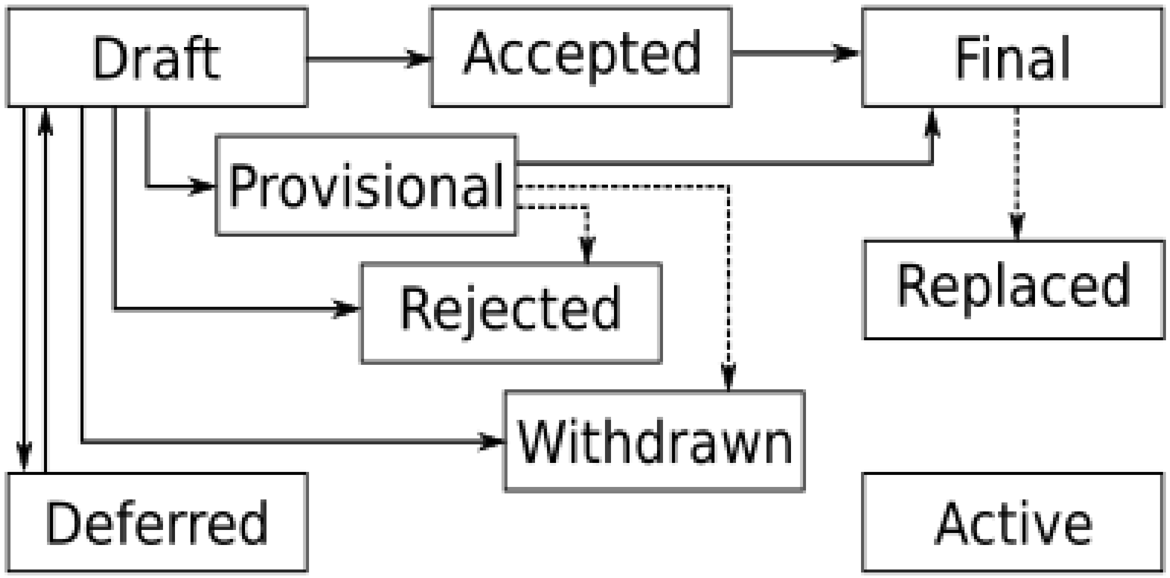
Once the PR is in place, the GEP should be announced on the in #general/geps on Zulip for discussion. Discussion about implementation details will take place on the pull request, but once editorial issues are solved, the PR should be merged, even if with draft status. The #general/geps topic will contain the GEP upto the section titled "Backward compatibility", so as to make it digestible to a wide audience. The #general/geps topic discussion is intended to target end-users, and thus, discussion on the proposed usage and possible impact should take place in #general/geps.

At the earliest convenience, the PR should be merged (regardless of whether it is accepted during discussion). Additional PRs may be made by the Author to update or expand the GEP, or by maintainers to set its status, discussion URL, etc.

Standards Track GEPs consist of two parts, a design document and a reference implementation. It is generally recommended that at least a prototype implementation be co-developed with the GEP, as ideas that sound good in principle sometimes turn out to be impractical when subjected to the test of implementation. Often it makes sense for the prototype implementation to be made available as PR to the GETTSIM repository (making sure to appropriately mark the PR as a WIP).

Review and Resolution

GEPs are discussed in #general/geps. The possible paths of the status of GEPs are as follows:



All GEPs should be created with the `Draft` status.

Eventually, after discussion, there may be a consensus that the GEP should be accepted — see the next section for details. At this point the status becomes `Accepted`.

Once a GEP has been `Accepted`, the reference implementation must be completed. When the reference implementation is complete and incorporated into the main source code repository, the status will be changed to `Final`.

To allow gathering of additional design and interface feedback before committing to long term stability for a language feature or standard library API, a GEP may also be marked as "Provisional". This is short for "Provisionally Accepted", and indicates that the proposal has been accepted for inclusion in the reference implementation, but additional user feedback is needed before the full design can be considered "Final". Unlike regular accepted GEPs, provisionally accepted GEPs may still be Rejected or Withdrawn even after the related changes have been included in a Python release.

Wherever possible, it is considered preferable to reduce the scope of a proposal to avoid the need to rely on the "Provisional" status (e.g. by deferring some features to later GEPs), as this status can lead to version compatibility challenges in the wider GETTSIM ecosystem.

A GEP can also be assigned status `Deferred`. The GEP author or a core developer can assign the GEP this status when no progress is being made on the GEP.

A GEP can also be `Rejected`. Perhaps after all is said and done it was not a good idea. It is still important to have a record of this fact. The `Withdrawn` status is similar — it means that the GEP author themselves has decided that the GEP is actually a bad idea, or has accepted that a competing proposal is a better alternative.

When a GEP is `Accepted`, `Rejected`, or `Withdrawn`, the GEP should be updated accordingly. In addition to updating the status field, at the very least the `Resolution` header should be added with a link to the relevant thread in the Zulip archives.

GEPs can also be `Superseded` by a different GEP, rendering the original obsolete. The `Replaced-By` and `Replaces` headers should be added to the original and new GEPs respectively.

Process GEPs may also have a status of `Active` if they are never meant to be completed, e.g. GEP 0 (this GEP).

How a GEP becomes Accepted

A GEP is `Accepted` by consensus of all interested contributors. We need a concrete way to tell whether consensus has been reached. When you think a GEP is ready to accept, send a message with a first line like:

> Proposal to accept GEP #<number>: <title>

In the body of your message, you should:

- link to the latest version of the GEP,
- briefly describe any major points of contention and how they were resolved,
- include a sentence like: "If there are no substantive objections within 7 days from this message, then the GEP will be accepted; see GEP 0 for more details."

After you send the message, you should make sure to link to the message thread from the `Discussion` section of the GEP, so that people can find it later.

Generally the GEP author will be the one to send this message, but anyone can do it – the important thing is to make sure that everyone knows when a GEP is on the verge of acceptance, and give them a final chance to respond. If there's some special reason to extend this final comment period beyond 7 days, then that's fine, just say so in the message. You shouldn't do less than 7 days, because sometimes people are traveling or similar and need some time to respond.

In general, the goal is to make sure that the community has consensus, not provide a rigid policy for people to try to game. When in doubt, err on the side of asking for more feedback and looking for opportunities to compromise.

If the final comment period passes without any substantive objections, then the GEP can officially be marked `Accepted`. You should send a follow-up message notifying the community (celebratory emoji optional but encouraged), and then update the GEP by setting its `:Status:` to `Accepted`, and its `:Resolution:` header to a link to your follow-up message.

If there *are* substantive objections, then the GEP remains in `Draft` state, discussion continues as normal, and it can be proposed for acceptance again later once the objections are resolved.

Maintenance

In general, Standards track GEPs are no longer modified after they have reached the Final state as the code and project documentation are considered the ultimate reference for the implemented feature. However, finalized Standards track GEPs may be updated as needed.

Process GEPs may be updated over time to reflect changes to development practices and other details. The precise process followed in these cases will depend on the nature and purpose of the GEP being updated.

5.1.3 Format and Template

GEPs are UTF-8 encoded text files using the reStructuredText format. Please see the *GEP X — Template and Instructions* file and the reStructuredTextPrimer for more information. We use Sphinx to convert GEPs to HTML for viewing on the web[2].

Header Preamble

Each GEP must begin with a header preamble. The headers must appear in the following order. Headers marked with `*` are optional. All other headers are required:

[2] The URL for viewing GEPs on the web is https://gettsim.readthedocs.io/en/latest/geps.

```
  :Author: <list of authors' real names and optionally, email addresses>
  :Status: <Draft | Active | Accepted | Deferred | Rejected | Withdrawn | Final |
           Superseded>
  :Type: <Standards Track | Process>
  :Created: <date created on, in dd-mmm-yyyy format>
* :Requires: <gep numbers>
* :GETTSIM-Version: <version number>
* :Replaces: <gep number>
* :Replaced-By: <gep number>
* :Resolution: <url>
```

The Author header lists the names, and optionally the email addresses of all the authors of the GEP. The format of the Author header value must be

> Random J. User <address@dom.ain>

if the email address is included, and just

> Random J. User

if the address is not given. If there are multiple authors, each should be on a separate line.

5.1.4 Discussion

- Reference to any discussions on PRs etc.

5.1.5 References and Footnotes

5.1.6 Acknowledgements

This document has been slightly adapted from NumPy's *NEP 0 <https://numpy.org/neps/nep-0000>*.

5.1.7 Copyright

This document has been placed in the public domain.

`Download the template here.`

5.2 GEP X — Template and Instructions

Author <list of authors' real names and optionally, email addresses>

Status <Draft | Active | Accepted | Deferred | Rejected | Withdrawn | Final | Superseded>

Type <Standards Track | Process>

Created <date created on, in yyyy-mm-dd format>

Resolution <url> (required for Accepted | Rejected | Withdrawn)

5.2.1 Abstract

The abstract should be a short description of what the GEP will achieve.

Note that the — in the title is an elongated dash, not -.

5.2.2 Motivation and Scope

This section describes the need for the proposed change. It should describe the existing problem, who it affects, what it is trying to solve, and why. This section should explicitly address the scope of and key requirements for the proposed change.

5.2.3 Usage and Impact

This section describes how users of GETTSIM will use features described in this GEP. It should be comprised mainly of code examples that wouldn't be possible without acceptance and implementation of this GEP, as well as the impact the proposed changes would have on the ecosystem. This section should be written from the perspective of the users of GETTSIM, and the benefits it will provide them; and as such, it should include implementation details only if necessary to explain the functionality.

5.2.4 Backward compatibility

This section describes the ways in which the GEP breaks backward compatibility.

The #general/geps topic will contain the GEP up to and including this section. This is to avoid losing users who are not interested in implementation details and instead focus the discussion on usage and impact of the intended features.

5.2.5 Detailed description

This section should provide a detailed description of the proposed change. It should include examples of how the new functionality would be used, intended use-cases and pseudo-code illustrating its use.

5.2.6 Related Work

This section should list relevant and/or similar technologies, possibly in other libraries. It does not need to be comprehensive, just list the major examples of prior and relevant art.

5.2.7 Implementation

This section lists the major steps required to implement the GEP. Where possible, it should be noted where one step is dependent on another, and which steps may be optionally omitted. Where it makes sense, each step should include a link to related pull requests as the implementation progresses.

Any pull requests or development branches containing work on this GEP should be linked to from here. (A GEP does not need to be implemented in a single pull request if it makes sense to implement it in discrete phases).

5.2.8 Backward compatibility

This section describes the ways in which the GEP breaks backward compatibility.

5.2.9 Alternatives

If there were any alternative solutions to solving the same problem, they should be discussed here, along with a justification for the chosen approach.

5.2.10 Discussion

This section may just be a bullet list including links to any discussions regarding the GEP:

- Links to relevant GitHub issues, pull requests.
- Discussion on XXX

5.2.11 References and Footnotes

5.2.12 Copyright

This document has been placed in the public domain.[1]

[1] Each GEP must either be explicitly labeled as placed in the public domain (see this GEP as an example) or licensed under the Open Publication License.

CHAPTER 6

Code of Conduct (CoC)

The GETTSIM community is made up of members from around the globe with a diverse set of skills, personalities, and experiences. It is through these differences that our community experiences great successes and continued growth. When you're working with members of the community, this CoC will help steer your interactions and keep GETTSIM a positive, successful, and growing community.

6.1 Our Community

Members of the GETTSIM community are open, considerate, and respectful. Behaviours that reinforce these values contribute to a positive environment, and include:

- *Being open.* Members of the community are open to collaboration, whether it's on feature requests, patches, problems, or otherwise.
- *Focusing on what is best for the community.* We're respectful of the processes set forth in the community, and we work within them.
- *Acknowledging time and effort.* We're respectful of the volunteer efforts that permeate the GETTSIM community. We're thoughtful when addressing the efforts of others, keeping in mind that often times the labor was completed simply for the good of the community.
- *Being respectful of differing viewpoints and experiences.* We're receptive to constructive comments and criticism, as the experiences and skill sets of other members contribute to the whole of our efforts.
- *Showing empathy towards other community members.* We're attentive in our communications, whether in person or online, and we're tactful when approaching differing views.
- *Being considerate.* Members of the community are considerate of their peers – other GETTSIM users.
- *Being respectful.* We're respectful of others, their positions, their skills, their commitments, and their efforts.
- *Gracefully accepting constructive criticism.* When we disagree, we are courteous in raising our issues.
- *Using welcoming and inclusive language.* We're accepting of all who wish to take part in our activities, fostering an environment where anyone can participate and everyone can make a difference.

6.2 Our Standards

Every member of our community has the right to have their identity respected. The GETTSIM community is dedicated to providing a positive experience for everyone, regardless of age, gender identity and expression, sexual orientation, disability, physical appearance, body size, ethnicity, nationality, race, or religion (or lack thereof), education, or socio-economic status.

6.2.1 Inappropriate Behavior

Examples of unacceptable behavior by participants include:

- Harassment of any participants in any form.
- Deliberate intimidation, stalking, or following.
- Logging or taking screenshots of online activity for harassment purposes.
- Publishing others' private information, such as a physical or electronic address, without explicit permission.
- Violent threats or language directed against another person.
- Incitement of violence or harassment towards any individual, including encouraging a person to commit suicide or to engage in self-harm.
- Creating additional online accounts in order to harass another person or circumvent a ban.
- Sexual language and imagery in online communities or in any conference venue, including talks.
- Insults, put downs, or jokes that are based upon stereotypes, that are exclusionary, or that hold others up for ridicule.
- Excessive swearing.
- Unwelcome sexual attention or advances.
- Unwelcome physical contact, including simulated physical contact (eg, textual descriptions like "hug" or "back-rub") without consent or after a request to stop.
- Pattern of inappropriate social contact, such as requesting/assuming inappropriate levels of intimacy with others.
- Sustained disruption of online community discussions, in-person presentations, or other in-person events.
- Continued one-on-one communication after requests to cease.
- Other conduct that is inappropriate for a professional audience including people of many different backgrounds.

Community members asked to stop any inappropriate behavior are expected to comply immediately.

6.3 Consequences

If a participant engages in behavior that violates this CoC, the GETTSIM community CoC team may take any action they deem appropriate, including editing/removing inappropriate content, warning the offender or expulsion from the community. The CoC team is made up of GETTSIM contributors and users and will react promptly to reports. The current team includes

- Boryana Ilieva
- Holger Stichnoth
- Maximilian Blesch

All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The GETTSIM CoC team is obligated to maintain confidentiality with regard to the reporter of an incident. In cases of conflicts of interest, CoC team members have to remove themselves from decision process without the need to further specify their conflict.

6.4 A Last Note

Thank you for helping make this a welcoming, friendly community for everyone.

6.5 Acknowledgment

This CoC is derived if not copied from the CoC of the Python Software Foundation (PSF) and includes parts from the CoC of pandas.

CHAPTER 7

Initiated by

CHAPTER 8

Indices and tables

- genindex
- modindex
- search