1 Contributing to HARK

This section provides an overview of how users can contribute new code to HARK, what sort of contributions are most desired, and style conventions for the toolKit. Before attempting to extend HARK, we strongly recommend that you familiarize yourself with its features and models by looking at the source code.

1.1 What Does HARK Want?

The easiest and most straightforward way to contribute to HARK is by writing new general purpose tools for inclusion in one of the top-level modules. This might be a commonly known data analysis technique (e.g. a kernel density estimator) for `HARKutilities`, an optimization method (local or global) for `HARKestimation`, an interpolation method for `HARKinterpolation`, etc. If you've found a technique useful in your own research and believe it could be of use to others on entirely different projects, then it probably belongs in HARK. Likewise, if you read through a HARK module expecting to find a certain tool or function (because of similar functions that are present, or merely because *it's very useful*) but don't, this is a good sign that HARK has a hole that you can fill.[1]

Secondly, the development team wants to expand HARK to include more canonical models. Most of the models available in the toolKit at the time of its public beta release concern agents with (geometrically discounted) CRRA utility over a unitary consumption good, who receive permanent and transitory shocks to income and have access to a single risk free asset. This is an important class of models, but that description has an awful lot of qualifiers. In short, there are many more common features of dynamic household models that are not yet in HARK, but would be very valuable for the toolKit. New model contributions should "play nicely" with existing models and tools, following HARK style guidelines (see sections 5.4 and 5.5) to the greatest extent possible. For ideas of potential model extensions for HARK, see section **??**.[2]

Finally, contributions to HARK are not restricted to entirely new functionality. While we have done our best to provide flexible frameworks and useful tools, we have almost certainly overlooked an unexpected case that would not be (properly) handled by the toolKit as is. Moreover, there might even be flaws in our solution or simulation methods. If you find a mistake or error, we strongly encourage you to contribute a fix for the problem, or at least to flag it as an Issue on GitHub so that someone else might. The "mistake" you find might not even be in the code itself, but a deficiency in the documentation: a misnamed (or

[1] Case in point: As of this moment, `HARKutilities` has no function for making a discrete approximation to the normal distribution, even though there are several well known methods for doing so and it is an obviously useful function.

[2] Although that list is limited to extensions and variations on consumption-saving models, HARK has no such restrictions. If there is an entirely different sort of dynamic model that fits in the HARK framework, it can be contributed.

incorrectly typed) parameter or return variable in the docstring, an incorrect (or missing) comment, even a grammatical or spelling error. No matter how niggling, we want your fix.

For those who want to contribute to the HARK project without programming a new model or digging through the code for errors and omissions, consider reading through the Issues page on GitHub to see what other users have flagged for development. You might find several pieces of "low hanging fruit" for you to quickly fix.

1.2 Git and GitHub

Git is a distributed version control system that has gained increasing popularity in the decade since its creation; it is designed to enable collaboration and "non-linear workflow" among a (potentially disperse) programming team. As HARK is to be developed by volunteer researchers all around the world, git is an excellent fit for the toolKit. Originally designed by Linus Torvalds as a command line tool, others have since extended git with further functionality and a user interface more friendly to those more familiar with GUIs. Foremost among these tools is GitHub.com, where the HARK project is hosted. This section provides a brief overview of how to use GitHub to interact with (and potentially contribute to) HARK.

To prepare to contribute to HARK, follow these simple steps:

1. **Make a GitHub account:** Go to github.com; the homepage should prompt you to choose a username and password. Signing up is quick, easy, and free.[3]

2. **Install GitHub Desktop**: Go to desktop.github.com and download the Desktop application for your operating system, then install it. After opening Desktop for the first time, enter your GitHub login to connect it to your account.

3. **Fork the HARK repository:** Go to the HARK repository page, click on the Fork button in the upper right, and choose to fork HARK to your own GitHub account. This creates a new repository identical to the one hosted by Econ-Ark at the moment you fork it, but hosted on your account instead.

4. **Add the repository to Desktop:** Open GitHub Desktop, click on the + icon in the upper left, and select clone. Find the repository named `yourusername/HARK`, click the Clone button at the bottom of the dialogue window, and choose a directory on your local machine.

You now control a "fork" of the main HARK repository, hosted by GitHub, and have a clone of this fork on your local machine. You are free to edit your fork of HARK in any way you'd like; other GitHub users can view your fork (and fork it themselves), but they cannot make changes to your fork unless you give them permission by adding them as a contributor. Changes made to the local copy of your fork do not get automatically sent to

[3]There are non-free ways to sign up for GitHub, but you don't need to worry about them.

the remote copy on GitHub for public viewing. To make changes to your fork on GitHub, follow these steps:

1. **Make local changes:** Edit your local copy of the repository in any way you'd like: add or delete lines from existing files; add, delete, or rename entire files or directories.

2. **Commit local changes:** Open the repository in GitHub Desktop and click on the Changes tab. In the lower left, provide a short description of the changes made in the Summary field, and (optionally) longer comments about these changes in the Description field. Click the check-mark Commit button.

3. **Push the commit:** Click on the Sync button near the upper right to "push" your local commits to the remote server; this will also pull down remote commits from other contributors to your fork that were made since the last time you synced.

The left hand panel of the Changes tab has a summary of the files that have been changed, added, or removed since your last commit; the right hand panel shows the specific changes made to each changed file. After step 2, a record of your changes has been stored locally by the git system, but is not yet reflected in the public remote copy on GitHub.[4] The "timeline" summary of the repository at the top of the Desktop window shows the history of commits for the repository: each dot represents a commit that was pushed to the remote server, and each open ring represents a local commit that has not been pushed to the remote server; the large broken ring on the far right represents changes that have not been locally committed.[5] The History tab lists all previous commits, including the timestamp, contributor, and a record of changes made.

Suppose you have mucked about in your fork for a while and have made changes to HARK that you think should be included in the main repository hosted by Econ-Ark. To make your dream a reality, follow these steps:

1. **Isolate the changes:** If you mucked around for quite a while, making various changes here and there, and then later made some contribution that you want to include in the main repository, you should *isolate the desired changes in a new branch of your fork*. Make a new branch by clicking the button toward the upper left of GitHub Desktop, revert that branch to the commit just before you forked from the main repository,[6] then edit this branch to include *only the changes you want to push to the main repository.*

2. **Issue a pull request:** Go to HARK's GitHub page and click the "New pull request" button. Click the text that says "compare across forks", and select `econ-ark/HARK` as the base fork, `master` as the base, your fork as the head fork, and the branch to

[4] You can make several local commits without syncing with the remote server. This is useful for working without an internet connection. Say, on an intercontinental flight.

[5] Clicking on a dot or ring will display the changes made in that commit (or uncommitted changes).

[6] Or since your last update from the main repository hosted by Econ-Ark.

be merged as the "compare". In the boxes below, give a name to your pull request and provide sufficient comments about your contribution so that a development team member can understand what you've done. Click the "Create pull request" button.

3. **Be patient:** Someone from the development team will look at your contribution and either merge it into the main repository or return it with comments. See section 1.3 for procedures followed by the HARK team when evaluating outside contributions.

1.3 Submission Approval Process

Before submitting to HARK, the development team would like you to make sure that your intended contribution is listed as an Issue on the GitHub page. If you are going to fix or address an Issue that is already posted, reply to that post with a note indicating that you are about to work on it and an estimate of when you will make a pull request for it. If you want to do something that no one has mentioned yet as an Issue, make a new Issue on GitHub (briefly) describing what you intend to do.[7] This will both help prevent two people from working on the same thing (and one of them potentially wasting time on it), and lets the development team know what's coming. Moreover, if you propose a contribution that is (a) already in HARK, (b) not in the spirit of HARK, or (c) a fix for something that's not actually broken, this gives the team a chance to let you know not to spend effort on it.

After creating a pull request to submit your contribution, it will be evaluated by one or more members of the HARK team. Criteria that the HARK team will consider when choosing whether to accept a submission include:

1. **Does the contribution break previous code?** If you change or expand an existing function, you should make sure that this does not cause some models or applications to work incorrectly. In the near future, HARK will have a core set of tests to run to check for full compatibility.

2. **Is the code well documented?** Code submissions should have properly formatted docstrings for each new function, class, method, and module (see section 1.5) and enough comments for a reader to follow the algorithm.

3. **Is the contribution relevant?** The development team intends for HARK to be diverse, but not everything goes in HARK. If your submission concerns a representative agent model or a method for estimating a reduced form model, then it probably doesn't belong in HARK.

4. **Is the code demonstrated with an example?** If your solve a new model in HARK, you should include a set of example parameters for other users to easily

[7] If your contribution is a tiny adjustment or to fix/add some documentation or comments, you shouldn't make an Issue.

test. If you've contributed a new function, consider providing a snippet of code demonstrating its use.[8]

5. **Is the code correct?** Your code should actually do what it claims to do.

6. **Does the code use HARK conventions?** To the best of your ability, contributions to HARK should follow the style guidelines in section 1.4. We don't demand perfection, as these are guidelines rather than rules.

7. **Is the code well integrated in HARK?** In the beta release of HARK, we have tried to present a "taxonomy" of consumption-saving models that build up from each other. When possible, new models in HARK should try to continue this pattern. A submission that introduces a model in a new module that could have easily been made as an extension of an existing module might be rejected. However, we acknowledge that there is value in providing a solver for a new model, and that a future contributor might make her mark by integrating it into the existing hierarchy.

8. **Is this already in HARK?** If your contribution is already in HARK, then it will almost certainly be rejected. Of course, if you submit a version that dominates the existing implementation (or even one that works better in a well described, non-trivial set of circumstances), it belongs in HARK.

The HARK team intends to release more specific guidelines in the near future, so that contributors are better able to judge their submissions before issuing a pull request. We are in the process of recruiting "czars" to oversee particular aspects of HARK, providing their knowledge and expertise.[9] While czars might directly evaluate submissions as a HARK team member, their more important role is to help establish these guidelines for their domain, more specifically nailing down what we mean by "correct" or "well integrated".

1.4 Naming Conventions

Object naming conventions in HARK are fairly different than existing standards, and differ somewhat between tool modules vs model or application modules. The following conventions apply throughout HARK:

- Functions and methods are always in "camel case": no underscores, first letter is lower case, first letter of each subsequent word is capitalized. E.g. `approxLognormal`.

- Function and method names should accurately and concisely describe what the function does; the first word in the name *must be a verb.*[10]

[8] In the future, HARK will make use of in-line testing.

[9] If you have knowledge and/or expertise and would like to contribute as a czar, please contact a HARK team member so that we can judge your credentials even though we have way less expertise.

[10] HARK standards have specified several "keyword verbs" that have specific meaning.

- Variable and class names *should not* have a verb as their first word.
- Class names should use no underscores and capitalize the first letter of each word; moreover, a class name *must include a noun*. E.g. `ConsPerfForesightSolver`.

When naming variables in model modules, the HARK team strongly discourages using single letter names, like `c` for consumption. Instead, we encourage contributors to use longer, more descriptive variable names using additional words and common abbreviations to specify the variable more precisely. In `/Documentation/NARK.pdf`, we list standard single letter variable "bases" and an array of prefixes and suffixes to adjust them. Economic variables in model modules should (usually) not use underscores, instead using camel case to the greatest extent possible.[11] The development team prefers this standard so that users can translate between Python code and LaTeX script with minimal work.

Conventions for naming variables in HARK's tool modules are significantly closer to more commonly used standards. Variable names should be in all lower case, with underscores between words, e.g. `data_to_match`. The functions and classes in these modules are more general and almost surely do not have any inherent "economic content"; they are numerical or algorithmic objects, not variables that might appear in an equation in an article for a (non-computational) economics journal. Variable names in application modules (e.g. the files that execute the `cstwMPC` estimations) are a mix of the conventions for tool and model files, as appropriate for each variable. That is, variables that are directly related to "economic variables" in model modules should follow those conventions, while objects created solely for data manipulation or reporting should use the style of tool modules.

1.5 Documentation Conventions

The HARK team wants the toolKit to be as accessible to users as possible; our greatest fear[12] is that a new user will open up our code, get hopelessly confused trying to read it, and then never look at HARK again. To prevent this tragic outcome, we have tried hard to provide comprehensive, accurate documentation and comments within the code describing our methods.[13] Moreover, HARK uses the Sphinx utility to generate a website with online documentation for all of our tool and model modules, so that users can learn about what's available in HARK without digging through the source code. When making contributions to HARK, the development team asks users to format their inline documentation to work with Sphinx by following a few simple rules.

- The top of every module should begin with a "docstring" providing a clear description of the contents of the module. The first sentence should concisely summarize the file,

[11] For "non-economic" variables that are only used temporarily, underscores are permissible.

[12] Other than spiders, of course.

[13] To the extent that documentation and comments are missing, we encourage users to rectify this in their contributions.

as it may appear in an index or summary of all modules without the remaining sentences. A docstring at the top of a module should be formatted as:[14]

```
"""
Specifies an economic model and provides methods for solving it.
More specific description of the key features of the model and variations
of it in this module.  Maybe some comments about the solution method
or limitations of the model.  Your bank account routing number.
"""
```

- The line directly below the declaration of a function, method or class should begin a docstring describing that object. As with modules, the first sentence should concisely summarize the function or class, as it might be included in an index or summary. For functions and methods, the docstring should be formatted as:

```
def functionName(input1,input2):
    """
    Concise description of the function.  More details about what
    the function does, options or modes available, and maybe mathematical
    methods used.  Credit to a source if you poached their algorithm.

    Parameters
    ----------
    input1:  type
        Description of what input1 represents.
    input2:  type
        Description of what input2 represents.

    Returns
    -------
    output_name:  type
        Description of the output(s) of the function.  Might have
        multiple entries.  If no output, this is just "None".
    """
```

[14]The triple quote to begin and end a docstring can use either the apostrophe ' or quotations marks ".

- Provide ample comments within a function or method so that a relatively intelligent reader can follow along with your algorithm. Short comments can follow at the end of a line, longer comments (or descriptions of the step or task about to be performed) should precede a block of code on the line(s) above it.

Finally, if you write a new model module, the HARK team asks that you also provide a short mathematical writeup of the model as a PDF. This document does not need to go into great detail about the solution method for the model or the functions and classes included in the module, merely specify the economic model and provide a summary of how it is solved. See `/Documentation/ConsumptionSavingModels.pdf` for an example of this.