Reproducibility Framework Tool

Deliverable 9.5

Authors: Stratos Gerakakis, Maria Ieronymaki, Maksym Brilenkov

Date: December 1st, 2020

Work Package: WP9 - Reproducibility in Science

DocId: pkh112-13-1.0

Revision History

Version	Authors	Date	Changes
1.0	Stratos Gerakakis Maria Ieronymaki Maksym Brilenkov	December 1st, 2020	Initial Version

Contents

Applicable and Reference Documents

[AD01] pkh112-06-1.0 Deliverable 9.3: Reproducibility in Science Report

1 Introduction

This deliverable describes not only the tool itself, but also the work that was produced in the Reproducibility Workpackage 9 in the later stages of the BeyondPlanck project.

2 Reproducible Science

We started our research on Reproducibility in Science by examining the latest developments on the topic. We identified several tools and services that are available online and aim to provide solutions towards reproducible science. The different tools that were identified cover a variety of provided functionalities that constitute the current state of the art in reproducibility in science. We present here our evaluation of these tools and workflows and we briefly present them, together with our impressions and conclusions after trying them out.

2.1 Reproducibility Theory

The reuse and reproduction of scientific experiments as they are currently described in publications is hard. Most often it requires additional information, data, tooling or support beyond what is usually provided in the text of a traditional publication. There are a growing number of individuals, groups, and initiatives - all trying to improve the state of scholarly publication. These range from domain specific to general, and from the practical and immediately actionable, to the more visionary and experimental solutions. The subject of Reproducibility and the ability of scientists to exactly reproduce and confirm a given result, is central to Science in general. At the theoretical level, Research Object have produced plenty of useful theoretical information on their website, regarding the subject of reproducibility, but except for a list of suggested initiatives and resources, the wandering scientist in search for a concrete reproducibility workflow will still be left without explicit direction.

2.2 Workflow Tools

One group of reproducibility tools offered to scientists fall in the category of Workflow definition tools. These help scientists define and execute a specific set of tasks, implemented by executing local (or sometimes remote) code, scripts, and other subworkflows. Each component only being responsible for a small fragment of functionality, therefore many components working together in a pipeline order to obtain the ultimate goal of the workflow, performing a useful task. Two such popular tools are Taverna and the Kepler Project Both are open source tools and provide tools for the definition of workflows. By using the specific applications provided by these tools, an execution workflow is defined. The use of the same tool chain must also be used by whoever wants to reproduce the specific workflows. This adds an additional burden to the original creators (and everyone else that wants to follow up with reproducibility) to learn and be intimately familiar with the workings of yet another specific application that most of the times scientists do not want to invest time in doing so, preferring to spend their time doing their original research.

Another set of tools require the scientist to use online services. These services offer specific tool sets that require all work to be done online through the use of their web applications. Some of the major players in this area that we evaluated were Open Science Framework, Codeocean, Zenodo. One of the major advantages of these services is that, by definition, all work is performed online which makes it easier for dissemination. At the same time, depending on the plans offered by each provider, authors can easily run out of hosting space or online computational time, requiring them to update their accounts. While this makes financial sense, from the side of the hosting companies, we consider this to be a big disadvantage for authors, just for the sake of reproducibility purposes. A solution like this might make sense for small projects, but for larger and heavier collaborations the cost can be very prohibitive.

2.3 Traditional online development platforms

All three of the major services in this section Bitbucket Github and Gitlab (among many other similar online development platforms) offer similar tools that mainly cater for the hosting and online development of source code.

Gitlab and Bitbucket, in addition to publicly available repositories, are also offering private repositories. This option might make them a better candidate for users that want to start their project as a private repository, but later on, closer to publication time, switch to a fully public repository. Being implemented by well established organizations, they offer a full suite of online development tools, with bug/issues management, wiki pages, and file hosting capabilities. All of them offer API access to the files hosted there and between the three of them, they have captured the majority of online code development. We are very familiar with the list of tools offered by these companies, and we plan to integrate with their services in our Reproducibility in Science efforts.

2.4 Analysis and Usability

From the list of the existing Reproducibility services that we chose to evaluate, we came to the following conclusions. Most of the online services offer a very streamlined and user friendly interface that specializes in the features that each service has decided are the most important to their users. In our evaluation, we did not find any service that provides a complete, unobtrusive, workflow that will explain and fit the needs for a complete reproducible workflow. Most of the services are free to use, although some of them are business endeavors that offer a free trial or a free tier. Unfortunately for a more serious and heavy usage the users would require to pay fees in order to continue using the service, which is something that we do not believe our audience is willing to do, simply to have their work in a reproducible format. Furthermore, the online services we evaluated did not offer a clear way to fully automate the complete process of reproducing the authors work. Some offered

online storage space only, while others offered computational resources so the produced code could execute online. But none offered an easy way to reproduce the full cycle. Some services offered a very important functionality, of offering free DOI (Digital Object Identifiers) numbers for the final published datasets, but they were only acting as a file storage and not as a computational environment. This feature would have been helpful as a final hosting repository for the published results, so that they would be uniquely identified, cited and addressed.

Overall, from what we evaluated, we saw plenty of functionality offered in the existing services, with interesting features spread between the various services, but we were not happy enough to use or suggest a single one of the online providers as being able to offer a complete and user friendly reproducible environment. On the other hand, it would be really beneficial if one would utilize some of the functionalities offered from the existing services (especially the free to use ones) and combine the best of each one.

2.5 Reproducibility Survey

At the beginning of the project an anonymous online questionnaire containing more than 40 questions about demographics, occupational capacity and reproducibility related queries was circulated in order to collect feedback from the project participants regarding the state of reproducibility in their day to day experiences.

The questionnaire contained questions asking about their level of working experience, familiarity with source control systems, backup and restore procedures used to safekeep their work, their coding experience, operating systems in use, familiarity with virtual machines, familiarity with existing reproducibility frameworks, their participation and involvement in reproducing other papers or being asked for help reproducing their own work.

We received 39 filled in responses over a period of a couple of months. A quick overview of the demographics revealed that 80% of the responders were male and 18% female. Age of the participants was roughly split in the 25-30, 31-35, 35-40 and 41- 45 age groups. One third of the responders were doing science for 4-8 years, another third for 9-15 and the final third for more than 15 years. Almost half of them were research fellows and close to 20% were professors.

56% of the responders were actively using a source code repository to store their work. The most popular source code control software used was git followed by svn. On the matter of safekeeping of their work, 62% responded that they keep their own personal backups on external hard disks, 46% said they used free online services to store their backups, while another 41% said that they trust the backups already offered by their institution.

On the part of the questionnaire regarding reproducibility, more than 80% responded that

they have recreated work described in another paper. 62% did so because the methods used there were interesting enough and they wanted to know more about them, 60% because they wanted to extend the work presented there and produce further results, while 33% did so because the results presented there were suspicious and wanted to verify the validity of them. These reproducibility attempts were performed to 1-5 papers (for 70% of the responders), 6-15 papers (for 20%) and for more than 15 papers for the remaining 10% of the participants. For each reproduced paper, 73% claimed that they would do anything in their power to recreate another paper if it was important to their work, whereas 15% claimed that they would spend up to 2-3 days trying to recreate it. 10% claimed that they were willing to spend up to 1 week trying to get successful results. 97% would first ask for assistance from colleagues in their attempts to recreate other papers, while 60% would directly contact the original author/s asking for clarifications.
From the participants that actually contacted the original authors, 68% mentioned that the author was helpful in assisting, 21% claimed that the author's reply was ambiguous and not clear enough, while 5% said that they never heard back from the original author. 51% of the participants claimed that they were contacted for help in recreating their own work. 43% of them once only, and 24% of them more than 10 separate times. 20% of those contacted refused to reply. From the ones that replied, 47% spent only a few hours assisting, while another 43% had communications that lasted several weeks until the issue was resolved.

From the participants that failed to respond to the queries for help, 60% mentioned that the requests was very trivial, 40% mentioned that they were lacking free time at the time of the request, while other responders mentioned that the requests were not clear, or insulting in nature or that the work was partly covered by copyright or of confidential nature. 37% of the responders said that they were willing to spend up to a couple of extra days to make sure that their work is reproducible, 24% that they were willing to spend up to a few hours only, while 26% allocate up to one week.

The majority of the responders (78%) said that they would like to have a reproducibility workflow embedded in their work, and only 10% claimed otherwise. Smaller percentages said that it depends on the nature or the complexity of the paper. Of the people that wanted to make their work reproducible, 54% attempted to provide a reproducibility workflow of their own. The remaining 46% attempted to find online an already established reproducibility workflow. 56% of those that looked online were not able to find a clear and easy to follow reproducibility workflow. From the 44% of actively searched online for solutions, 46% ended up just being a little more descriptive in the explanations provided in their paper, 25% ended up asking other colleagues what they use in order to provide reproducibility in their papers, while 21% of them gave up while of being able to find anything actionable.

From the reproducibility survey we concluded that Reproducibility is a desirable trait in a scientific paper, although the process to do so is not clearly defined. It seems that authors realize the importance of reproducibility but given the difficulty finding easy ways to do so and the limited amount of time they devote on doing so, often leads them in dropping that

feature from their papers.

3 Reproducibility in BeyondPlanck

Based on the overview of the existing Reproducibility services and tools, as we evaluated them, and on the reproducibility survey results we concluded that the existing services are not able on their own to offer a convenient and unobtrusive way for scientists to make their work easily reproducible. Our proposal is to introduce a command line tool that will automate the process of recreating the computational pipeline implemented with BeyondPlanckThe tool should provide an easy way for users to reproduce the BeyondPlanck pipeline and easily recreate it locally on their own infrastructure. The reproducibility tool is utilizing Docker containers and is capable of recreating locally the execution environment of the BeyondPlanck pipeline without making any changes to the currently logged in user's environment.

We understand that it will be impossible to satisfy the tastes and/or needs of all scientists and all scientific needs and purposes, but we strongly believe that by introducing a tool that automates the majority of the required processes in an easy and extensible way will make reproducibility a lot more approachable. Furthermore it does not tie the process to a specific online service and no special accounts or subscriptions are required for implementing it. Also, we understand that the provided reproducibility workflow might not be applicable in certain edge case scenarios, due to their complexity. Special cases with either exceptional executable phases (requiring really big amounts of execution resources, more applicable to a computing grid) or in their usage of input or output files (for example, big datasets that are really prohibitive in downloading and storing locally) might not be candidates for this specific reproducibility workflow. It is not impossible to devise such an extension to the proposed Reproducibility workflow, capable of handling these complicated edge cases, but this would be something that could be expanded on, in a further iteration of the tool.

Due to the inherent complexity of supporting any and all kinds of possible scientific workflows, we purposefully chose to cater to a specific subset of scientific workflows. We chose to provide a tool that provides an automated way of reproducing the scientific work produced in a paper, while the scope of its capabilities have been decided so that the development of the tool would fit in the allocated time of this project.

The criteria by which a project would benefit from this proposed workflow are:

- Have an exact definition of initial input files. The author should be able to define what input files are required and where these can be found.
- Have a process that programmatically manipulates these input files (along with any other files that are generated for the purposes of this paper).
- Does not have excessive computational requirements for the execution of the above programmatic code, preferably able to be executed in a modern laptop or even a high end desktop machine.

- Similarly it should have modest file system requirements, able to be fit, once again, on a laptop or high end desktop machine.
- Produces some results (files, diagrams, tables etc) that is the output of the computational phase. Although these might appear as very limiting criteria, we believe that they will be able to cover a large number of scientific publications. In addition, to the best of our ability we believe that a fully automated reproducibility workflow such as this, has never been attempted before.

3.1 Docker environment for user-friendly data access and code exploration

In the course of the BeyondPlanck project, we implement a command line tool that is available for all major Operating Systems (Linux, macOS and Windows). The tool is fully documented in a companion website that acts as an online guide, a frequently asked questions resource, a way for users to provide feedback, and as a hub where all development of it takes place.

The command line tool offers three fully automated major functionalities:

- Allows the automated retrieval of required input data files from various online sources
- Allows the execution of required computational tasks that operate in the input data files
- Recreates the exact BeyondPlanck results, or allows execution of similar products by altering configuration files.

The tool allows the original author to consistently recreate their own work, but it can also be used by any other users that would like to create derivative works. Although developed in the context of the BeyondPlanck project, it is a general purpose tool that could be utilized in similar workflows for other projects. The tool is working from a text based configuration file that defines all requirements necessary to make the task of reproducing the majority of scientific workflows as automated as possible. During execution, it offers clear, concise and user friendly error messages for the cases where the automation process cannot continue unassisted. In the majority of the cases, a solution or additional help should be already available in the companion website. For the missing cases, the companion website can be used as a discussion forum to request further assistance from fellow users of the system.

3.2 Source code structure and compilation with CMake

Any project, no matter how big or small, consists of both planning and execution. This usually includes clever time management, careful resource allocation, and proper results' presentation. Exactly for these reasons the questions of code design, compilation, installation and distribution is one of the most crucial parts of any successful project. Things

such as compilers, linkers, and package managers undoubtedly build up the task complexity of software development. While every platform has some tool to address this issue (e.g. Visual Studio on Windows, XCode on MacOS, or Autotools on Linux), we cannot really utilize them properly if our aim is to have cross platform support. And even if the target consisted of only Linux based Operating System (OS), which appears to be a preference for High Performance Computing (HPC) centers, we cannot simply neglect the abundance of different flavours within the Linux (e.g. Debian- or RPM-based) itself.

In addition, the complicacy of the CMB analysis task may become quite daunting if one starts to think of how to install Commander code dependencies on: – Beehives/OWLs - RedHat based, small cluster maintained by Institute of Theoretical Astrophysics of University of Oslo [footnote to docs], Norway; – Saga, Stallo and Fram - CentOS based, the Norwegian HPC cluster [footnote to docs], Norway. – Cori - Cray XC40 supercomputer maintained by National Energy Research Scientific Computing Center (NERSC), USA; – and many many more... That is even before installing the code itself!

That is why, in BeyondPlanck we were focused on automating the processes as far as possible in order to:

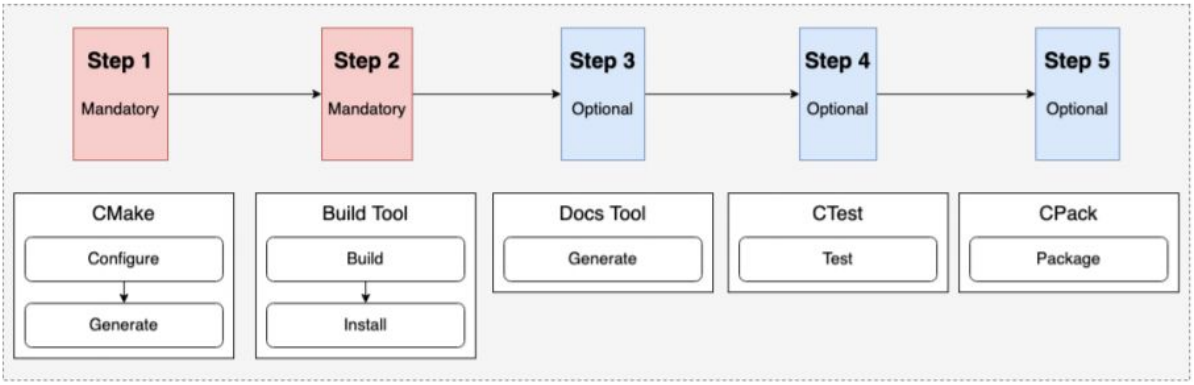
1. save time on technical details and implementation in favor of actual science
2. cover as many platforms as possible, in terms of code distribution/installation; and
3. take care of reproducibility, which is an essential part of any good science.

And here is where CMake - the set of tools, which covers the entire process starting from build/compilation up to package distribution while supporting a range of compilers and platforms - comes into play. CMake is a part of a lot of cosmology related projects (TOAST and HDF5 to name a few) and one of the industry standards in terms of code distribution. It comes with an extensive reference documentation, accessible from the official website. In the section we describe the CMake-based generation of Commander binaries, which are later can be utilized to analyze an entire set of CMB datasets.

3.3 Code Organization and CMake workflow

The code organization is tightly connected to CMake workflow. The first step, which is a combination of two, uses CMake script(s), CMakeLists.txt, starting from the top one (located inside the root directory of the project) and generates a set of configuration files later used to compile the project. Here we should make an important remark that CMake is not a build tool per se, but a build file generator, - it generates files used by the user's favorite build tool (Makefiles, Ninja files etc.) to compile the project afterwards.
Needless to say that the amount of those files can be quite significant; thus, we adopted the out-of-source build approach, recommended by CMake creators. The idea is simple and consists of storing the source code (written by developers) and build files (produced by CMake) in separate locations. Usually, the folder for the latter is called build and it is created inside the root directory.

In this way, the user would simply invoke an appropriate CMake command to configure the project from that folder and then use his/her favorite tool's command to compile it. CMake also supports a unified form of the command, which would result in the same outcome.



3.4 Commander Superbuild

Usually, each sub directory would contain its own CMakeLists.txt with all relevant information on how to compile the code(s). However, we decided to treat the Fortran and CMake source codes separately, with the main chunk of CMake code concentrated inside the cmake folder.

Not only does this approach makes the directory structure cleaner, easier to maintain (and, if needed, expand) the existing code base, but it also complements the so-called CMake Superbuild pattern which allows us to build Commander and all its dependencies in one go. This process is completely automated and requires only a handful of commands each of which supports a variety of variables to assure the fine grained control over the installation [footnote to documentation]. In this way the dependency is treated as a separate entity, while CMake manages the correct sequence of compilation / installation and passes the details from one project to another. In brief, the process can be summarized as follows.

During the configuration step, CMake will scan the host system for required libraries and, if those were found, will save their locations together with user configuration for later use. If some (or all) of the libraries were not found, CMake will also remember this, so then during the build phase, it would download, configure, compile and install the missing ones and link them all to Commander code.

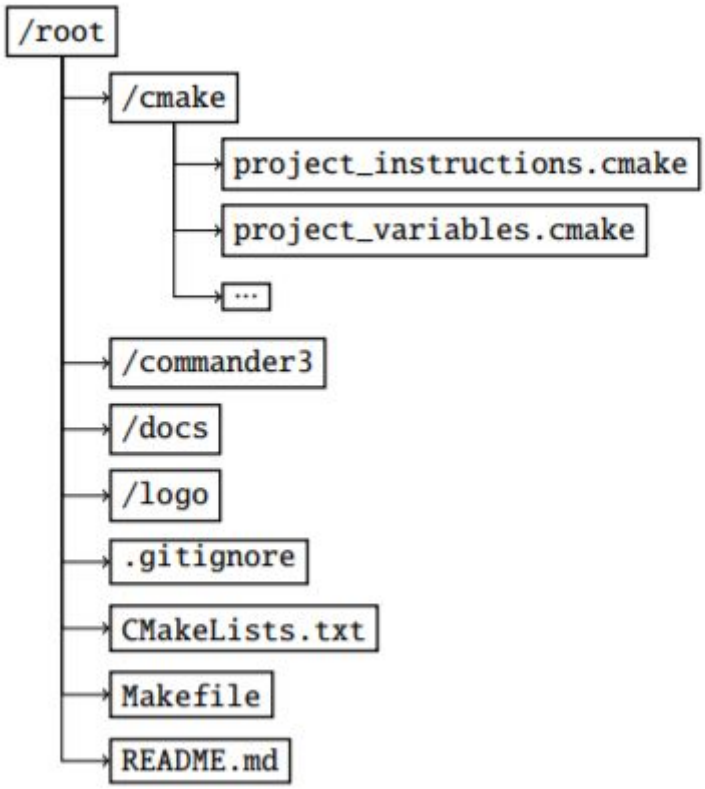
The CMake module, which allows such behavior is called, not surprisingly, ExternalProject and its main purpose is exactly to allow the download and build of the projects, which are not a direct part of the main project. In this way, the Commander dependency is treated as a completely independent entity. This isolation allows the build to be performed the same way on different platforms, with completely different build settings (e.g. compiler flags) and/or

with completely different build system (e.g. Autotools).

Under the hood, it defines the set of so-called targets each representing the particular step for the build process of an external project. These steps are then collected under the unified name (in our case, sub project name), used later in the code. CMake also remembers information about each performed step, which, if performed successfully, will not be repeated again. This allows us to compile Commander dependencies only once for several different types of build. The default steps are as follows:

1. Download. The project is downloaded via external link as .zip or .tar.gz archive, or from the git repository. We use the MD5 hashes used whenever possible to ensure that the downloaded files are correct;
2. Update/Patch. This step applies patch to the downloaded archive or, in case of git, brought up to date. As we download the release versions of the packages, we skip this step;
3. Configure. This step can use CMake and other build tools alike. In our case, most of the libraries use autoconfig scripts and Makefile to compile;
4. Build. In this stage, we use the default built tool as the rest of the project.;
5. Install. The sub project is installed to a local directory specified by the user during the CMake configuration stage;
6. Test. Run the tests provided by the sub project developer given that the test stage is enabled by user [Maksym: not implemented yet!];

However, not every project (e.g. HEALPix) has an install command. In this case, we would simply copy the compiled binaries and files into the specified directory. Once the sub project is installed, the build will move to the next one and so on until the end of installation. We enabled the writing logs option provided by CMake, which will save the logs of the installation inside the appropriate folder specified by the user (the default value is logs located inside build). There is also a possibility to generate out-of-source documentation. With this information in mind, the entire Commander folder structure can be summarized as follows:



where:

- root - root folder of the project;
- cmake - directory which contains all CMake related files;
- commander3 - directory which contains all Commander related files;
- docs - directory which contains instructions to generate outof-source documentation.
- logo - directory which contains Commander logos;
- .gitignore - git version control file;
- CMakeLists.txt - top level CMake file. The project processing starts from here.;
- Makefile - old Makefile used to compile Commander3. Although can be used, it is not recommended to do so;
- README.md - file which describes the project (on GitHub).

3.5 Installation regimes

All libraries are produced in Release format with all optimization flags enabled, whereas Commander can be installed in 4 different regimes, namely:

- Release. Builds Commander with all optimization flag enabled, tuned for specific compiler and platform. By the time of writing this document, the following Fortran compilers were supported: Intel, GNU [Maksym: I am trying to add NVIDIA];
- Debug. Builds Commander executable without optimization, but with debug symbols;
- RelWithDebInfo (Release With Debug Information). Somehow the compromise

between the two above. It builds Commander binary with less aggressive optimizations and with debug symbols;
- MinSizeRel (Minimal Size Release). Builds Commander executable with optimizations that do not increase object code size.

As the targeted platforms are HPCs with a lot of actual disk space, this feature is not used; thus, not thoroughly tested. As we already mentioned, CMake remembers all the steps it undertook and these are saved inside CMakeCache.txt file in the root of the build folder.

If one wants to recompile Commander in different regimes, one can simply delete this file and rerun CMake configuration again. Commander Superbuild is using CMake predefined (looks for cURL, HDF5 libraries), our custom (FFTW3 and HEALPix libraries) and third party (CFITSIO library) find modules. These modules search for common paths inside the host system, or for package information provided by package managers (e.g. homebrew on MacOS) to identify the installed libraries. In this way, if the system somehow knows about previous installation (it was either specified in PATH or any other dedicated environment variable), CMake will find the libraries and so will not recompile them again.

4 Dissemination

4.1 Commander and BeyondPlanck reference guide

The documentation for the BeyondPlanck pipeline and Commander are publicly available on the documentation page of the official BeyondPlanck website. Due to the dynamic nature of the BeyondPlanck project, we expect the documentation to continually evolve with the addition of new experiments and other features to BeyondPlanck. Active participation by the community is, therefore, essential to maintain and build on the documentation.

The documentation consists of three main sections. The first section considers Commander and gives a detailed description on the following: 1) how to install Commander based on the specific use-case; 2) a detailed description of all the parameters used in Commander; and, 3) a description of the data and the various input file formats used by Commander. The second section describes the BeyondPlanck pipeline with focus on the precalculation and execution stages. The third section considers reproducibility as implemented by BeyondPlanck with a step by step guide on how the BeyondPlanck results can be reproduced.

4.2 Websites

The online "presence" of the BeyondPlanck project was broken down to multiple independent websites that all worked seamlessly together to address different needs of the project. Each website has its own source code repository and is developed in isolation from the other websites. They all work seamlessly under the umbrella of the BeyondPlanck domain name of `beyondplanck.science`

The main website is located at https://beyondplanck.science.

Online documentation for Commander 3, along with parameter files are available under https://docs.beyondplanck.science

The BeyondPlanck Online conference information is hosted under https://conferences.beyondplanck.science

Finally an online discussion forum is hosted under https://forums.beyondplanck.science/

4.3 Online Conference Support

The online Conference was presented as a Zoom teleconference meeting. The average number of participants over the five days of presentations and tutorials was at about 150 participants.

Presentation slides and videos for all the sessions are available at our conferences site.

4.4 Discussion Forum

BeyondPlanck comes with a discussion web forum. The discussion forum is the first point of contact for help regarding any technical questions users might have related to the code, but also for more general scientific questions regarding the project. The goal is that the discussion forum will act as a database of Frequently Asked Questions once it has accumulated some questions.