The Pencil Code Newsletter

Issue 2025/1

June 9, 2025, 2024, Revision: 1.35

Contents

1 A new steering committee

Since Friday 23 May 2025, the Pencil Code has a new steering committee (PCSC). As on previous occasions, Matthias Rheinhardt set up the adoodle to perform the anonymous election. Clara Dehman from the University of Alicante acted as an independent observer. Here her abbreviated statement from 23 May:

I hereby confirm, in my role as observer, that the election for the PCSC has been successfully concluded. Below is a detailed summary of the participation and the results. A total of 33 individuals were eligible to vote. Of these, 17 submitted their ballots, resulting in a participation rate of 51.5%. One additional participant accessed the voting page but did not cast a vote, and is therefore considered an attentive non-voter (3.03%). The remaining 15 individuals (45.5%) neither accessed the voting page nor submitted a vote and are thus classified as inattentive voters.

The top five candidates in terms of support were:
- *Axel Brandenburg: 15 votes*
- *Philippe Bourdin: 12 votes*
- *Matthias Rheinhardt: 11 votes*
- *Jennifer Schober: 9 votes*
- *Piyali Chatterjee: 8 votes*

With these results, we now have the newly elected members of the PCSC.

Yours sincerely,
Clara Dehman

We wish the new committee good luck with their work. A presentation of most of the candidates was published in issue 2023/3 of our newsletter. The terms of reference of the PCSC are `https://www.nordita.org/~brandenb/pencil-code/PCSC/ToR.pdf`. We thank Nils E. Haugen, who has now been replaced by Jennifer Schober, for his work on the previous committee.

2 Five years newsletter

The first Pencil Code Newsletter was published on 17 July 2020, nearly 5 years ago. We are now in the sixth year and it is time to reflect on its usefulness. The stated goal back then was to update and remind the user community of important results and developments. In Newsletter 2020/2, we said

The Pencil Code *comes with a lot of default settings. Many of the input parameters are set to what was of interest when a particular module was developed. Likewise, many logicals (switches) are set to whatever a*

particular person considered useful at that time and what is imposed by the constraint of backward compatibility. Changing this all of a sudden could make others quite upset. To raise awareness of changes that are considered justified, we use the opportunity to highlight changes of defaults in the Newsletter.

In the present Newsletter, one new default change will be presented. Another development concerns the GPU readiness of the code, which was discussed in issue 2024/2 of the newsletter. A related point was the merge from the development branch `gputestv6` to the main (`master`) branch in March.

In the wake of this major change, but unrelated to it, some potential weaknesses of the handling of the code have been exposed. This led to various discussions about the "Pencil Code philosophy". To have a more detailed account of the different view points, we solicited contributions to this newsletter; see the email to `https://groups.google.com/g/pencil-code-discuss` on May 12. In this newsletter, we begin by reporting on the aforementioned default change which was sent to us by Frederick Gent. We then present contributions that we have received about the Pencil Code philosophy. We also present some views of present and past PCSC members and end with a description of other code changes and enhancements.

3 A proposed default change

by Frederick Gent, Aalto University, Finland
received 24 April 2025

In some cases where there are challenging gradients in the density, particularly associated with highly compressible flows, it can be helpful or indeed essential to include some diffusivity to the continuity equation, although there is no such physical process. In the Pencil Code we have three forms of such diffusivities: a bulk diffusion coefficient `diffrho`, which acts proportional to the Laplacian of density, a shock diffusion coefficient `diffrho_shock` which acts proportional to the product of the flow convergence and the Laplacian of density, and the hyper-diffusion coefficient `diffrho_hyper3`, which acts proportional to the cubic Laplacian of density $(\partial^6/\partial x^6 + \partial^6/\partial y^6 + \partial^6/\partial z^6)$.

These apply an effective mass sink in the system, so to conserve momentum and/or energy, appropriate corrections can be applied to the momentum and/or energy equations, which is enabled by setting the flag `lmassdiff_fix` to `.true.` By default this has been set to `.false.` in the Pencil Code. However, experiments with the shock tube tests (Sod 1978)[1] reported in Gent et al. (2020), `http://doi.org/10.1080/03091929.2019.1634705`, indicate that solutions without the flag set `.true.` yield less accurate solutions. It would therefore be more appropriate to make the default `.true.` This evidently creates some numerical crashes when used with some simulations with high Mach particle flows, so a 2022 change to the default without consultation was recently reverted by request. Now that we have time to consider the matter collectively, I propose that we change the default to `.true.` and explicitly set it in those reference samples to `.false.` which require it so. A warning about the choice should be printed to alert the user about its significance whenever they select mass diffusion. The effect of the switch was not reported in the cited article, but given its subsequent impact on users of the code I propose in the near future to prepare a more comprehensive study of these effects, based on the Sod (1978) solutions.

Because the application of the mass diffusion fix has been found to cause numerical crashes when applied to mass hyper-diffusion, the fix is only effective for Laplacian and shock diffusion. I would also observe that I have found mass hyper-diffusion to be prone to density holes including negative density when solving for linear (non-logarithmic) density. Note, that the Pencil Code is not a conservative code (in its default configuration) and when solving for highly compressible flows, using the linear rather than the logarithmic form of the continuity equation has also been found to be more conservative.

4 Branches

by Wolfgang Dobler, Berlin, Germany
received 20 May 2025

4.1 The problem(s)

The currently dominant development model of the Pencil Code is

- *unstructured:* Commits cannot be grouped predictably, because while you try pushing them, others could push some commits of their own;
- *opaque:* if commits are not grouped, it is hard to see what is going on;

[1] Sod, G. A. (1978). "A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws". *J. Comput. Phys.* **27**, 1–31 (`https://doi.org/10.1016/0021-9991(78)90023-2`).

- *hard to test:*
  - Continuous-integration tests (= CI tests, which we sometimes call Pencil tests or autotests) have hard time constraints and thus cannot be thorough;
- *fragile:* as a consequence of being hard to test;
- *obscuring responsibility:* If I made two commits, and A has pushed 3 commits in between (see *unstructured* above), would I be testing my changes or theirs?

4.2 Advantages of branch-oriented development

- Much deeper testing
  - You can take your time to run all tests before merging your branch back into *main* (= *master*). This allows us to have really thorough tests.
  - Independence from CI. CI is nice, but it is dangerous to rely on it (and it limits the amount of testing)
- Clearer identification of responsibility
- Isolates unrelated changes from each other until they really need to get dealt with
- Friendly competition (survival of the most popular branch)
- You can get new features running and ask interested others to test before you finalize them (instead of: everybody immediately has to live with your new feature, although it is not thoroughly tested yet – you had to push it onto *main* before asking the interested others).
- Reduces stress level, because you can properly work on a feature without interfering with others and later do all the tests and push.

4.3 Note

- We have used branches before (e.g. the GPU branch)

4.4 Feedback and criticism

Here is the gist of some responses I got to an earlier version of the proposal below.

- "Risk of fragmentation"
  - But where is the problem if all branches are pushed to the server?
    - If you need functionality from another branch on your own feature branch, you can (a) merge that branch, or (b) cherry-pick the relevant changes from there.
  - *The important thing is to have all branches pushed to the server.*
    - Currently: If you are not willing (yet) to push, your changes are only local, which is much worse.
- "For short-lived branches which are not shared with anyone else, I would recommend rebasing, as it keeps the history cleaner (this helps later on if one is bisecting to find when a bug was introduced to the master branch)."
  - I am open to that, though I personally find history with (even medium-sized) topic branches cleaner; and `git bisect` works across branches just as well.
- "I fear that if we start using more branching this will result in some branches never being merged back into main. This will eventually lead to several versions of the Pencil Code."
  - There will always be some branches not (yet) merged. If they are relevant, they will eventually get back into *main*, be it that they get merged or cherry-picked. (And if they are not, who cares?) Should there really appear two branches of the code that continue to be popular among users over a long time, we should strive to merge them.
- "...other codes in which multiple groups maintained branches of code which were entirely incompatible with the main branch or each other."
  - Rule 6. below strongly encourages to not let branches diverge too far from *main*. Being able to decide for yourself when exactly you reconcile your changes with *main* is an enormous benefit.

4.5 Proposal

Here is a set of rules that I would like to see implemented:

1. The main person responsible for a commit is its author.

2. The maintainers are important, as often only they know how a sample really works. However, it is not their responsibility to react to commits breaking an auto-test sample.

3. Ideally, any commit that ends up on *main* (the main branch of the code) should not break any of the essential auto-test samples.

   If it turns out that a commit violates this rule, the broken tests need to be fixed swiftly. Either by the commit author committing a fix, or by others reverting the commit in question.

   Any commit that breaks an auto-test may be reverted by anybody without prior notice in order to fix that test.

4. Keep individual commits focused on one logical unit (i.e. keep them small and do not conflate loosely related topics). This is crucial for identifying and reverting problematic commits.

5. For a sequence of commits that belong logically together, it is not very feasible, and also not necessary, to run the test suite for each of them.

   Instead, develop the feature on a branch. When you think the feature is ready, merge *main* into your feature branch and run the tests. If they all run successfully, merge your branch back into *main*.

   Never merge a branch into main *without having verified that the auto-tests work.*

6. Try to keep the lifetime of branches short, even though this will not always be possible. If your branch has to live longer, you should periodically merge *main* into it.

7. *Never quietly adjust reference results in an auto-test sample directory to fit your latest data.*

   If reference data need to be touched up, this must be agreed upon with the maintainer(s) of that test.

8. When making changes that will affect auto tests:

   (a) Discuss this with everybody before you commit the change.

   (b) Isolate auto-tests against the change. E.g. if you are to change the default value of `lmassdiff_fix` from `.false.` to `.true.`, make sure to explicitly set `lmassdiff_fix = .false.` in all auto-test samples that would otherwise be affected by the change.

9. Never rely on the automatic tests run after a commit on the GitHub server, or the bisection that is done. It is the committer's responsibility to run all officially accepted auto-tests.

10. Never rely on automatic emails to reach a maintainer of an auto-test (or in fact anybody else). If you need to discuss an auto-test, contact the maintainer directly and only assume that you reached them once they reply.

5 Comparison with other codes

by Evangelia Ntormousi, Scuola Normale Superiore, Pisa, Italy

received 23 May 2025

The Pencil Code's coding standard is to integrate everyone's contribution into the publicly available version and to test it periodically for incompatibilities and/or errors. This approach has many advantages, the most important one being reproducibility and tractability of the results. Furthermore, younger researchers or new users don't need to reinvent the wheel, since they can immediately see how a certain problem was set up and build from there, if necessary. Very importantly, there are no diverging branches, so all outputs can be processed with the same, universally updated tools. Everyone gets the cutting edge of the development, and has all the elements at their disposal to give credit to other developers for code they reused. Overall, this approach promotes a sense of community and open collaboration, with all the benefits that these qualities entail.

Of course, there is always the danger that a modification from one user or group introduces a bug in all versions. Even in that case though, every user has the chance to spot it and correct it, which is not the case when a code is proprietary. A more probable issue is that, if a code becomes very popular, it could become 'heavy' with many features unless some coding standards are maintained by a steering team of developers.

In my opinion, the Pencil Code method of collaboration is the most transparent out there, which is great for promoting transparent, reproducible results

and avoiding confusion. My experience with other approaches has led to issues with reproducibility and collaborations. The specific examples I have in mind are `RAMSES` and `AREPO`, where the public version is stripped from 'advanced' features, which need to be developed by the user (in this regime `AREPO` is extreme in that the user basically only gets the hydro+gravity solver). This does give a lot of room for innovation and gets different teams to come up with different ideas on an implementation, but without a framework for sharing and exchanging code it eventually leads to many branches of proprietary code that are diverging from each other and the main. This problem inevitably translates into skepticism on a competing team's results, and worse, promotes a culture of withholding code.

6 Additional suggestions

by G. Kishore, Inter-University Centre for Astronomy & Astrophysics, Pune, India
received 23 May 2025

6.1 Contact information for committers

One aspect which I did not see explicitly brought up so far is the issue of contributors not being easily contactable. While granting someone commit access, we should emphasize that the email address used for commits should be one that the contributor checks regularly (in the past, I have found after attempting to contact a contributor through their listed email address that it was an old address that they no longer check regularly).

6.2 Contributor guidelines

More generally, there should be a single set of contributor guidelines listing what is expected from committers, such as:

- the coding style
- the expectation to remain contactable
- the code of conduct
- the recommended development style (if there is one)

This information is currently scattered across various parts of the repository, the webpage, and the Github wiki; having a single document/webpage (which may simply be a set of links to the existing places where each of these aspects is dealt with in detail) would be useful while inducting new contributors. I can help compile such a document if needed.

6.3 Stable tagging

I also think we should bring back the practice of tagging stable versions. Given the observed tendency to 'silence' auto-tests, I don't think this tagging should be done automatically (which I think was the previous practice). My suggestion would be to synchronize the release of stable versions with the PC User Meetings, at which a stable release can be approved after going through a checklist (which would be publicly documented somewhere). Some examples of items which would be in such a checklist:

- create a list of currently working samples (which may just be the set included in `pc_auto-test levels 0--3`) and check that there have been no changes to the reference data of these samples that were not approved by the maintainers (we should have a policy that every commit changing the reference data should explicitly mention which of the maintainers of that sample approved it);
- for all the postprocessing modules (e.g. Python) that have their own test suites, check that the tests still pass (these are currently not included in the periodic autotests);
- discuss proposed changes to default values of input parameters, and include a list of such changed defaults in the release announcement (we may need to come up with a mechanism to propose changes prior to the meeting, which may be Github issues or simply a text file somewhere in the repository);
- if samples which were working in the last stable version are broken and cannot be fixed, this should also be mentioned in the release announcement;
- for all maintainer email addresses listed in the README files of the working samples, check that they are able to receive mails from the auto-test servers.

While going through such a checklist is a non-trivial amount of work, my point of view is that these issues should be dealt with regularly to keep the code healthy; stable version tagging is a compromise that acknowledges that it is not feasible to do this for every single commit.

There are, in a sense, two kinds of branches.

One kind is shared between multiple developers, lives in the central repository, and may be long-lived (e.g. the gputestv6 branch).

The other kind, which is what I think Wolfgang was suggesting (and which is considered a good practice in the software development community), is that for any change (excluding very minor things like updating comments), you create a branch locally on your machine. This would not be pushed to the central repository, and need not be shared with any other developer. Each branch contains a set of logically related changes. For example, you could have one branch where you make changes to an initial condition, and another branch where you are trying to optimize the calculation of power spectra (I have 44 branches right now in my local clone of Pencil). When the changes in a particular branch are 'ready', you then rebase/merge those changes into the master branch. The main advantage of doing things this way is there is no hurry to push (half-baked) changes to the central repository since

A. you do not have to continuously deal with conflicting changes from others (rather, you just have to deal with it once before finally merging the changes; if the branch is short-lived, this should still not be too much of a hassle);

B. you can independently work on multiple things at a time (e.g., if you make a commit partially implementing a new initial condition and you later make another commit fixing a boundary condition, you can choose to push only the latter if both are in separate branches, rather than also being forced to push the half-baked initial condition to the repository).

After adopting this 'micro-branching' development style, it becomes feasible to locally run more extensive tests for each set of changes that you intend to push to the main repository. For example, you can use `bin/pc_isolated-test` or `git worktree` to run tests on a particular branch while you simultaneously work on something else in another local branch.

7 Any commit or merger should be rejected back to the committer if it does not pass the auto-tests

by Chao-Chin Yang, The University of Alabama, Tuscaloosa, USA

received 4 June 2025

The previous suggestion that "any commit that breaks the autotest can be reverted by anyone." still puts pressure on the maintainer. This is the reason a pull request (PR) is better than the current approach: the merger has time to run the auto-tests before merging.

It was pointed out that we already have that in the travis test. The travis could be set up to automatically start auto-tests for each commit coming in, so one could also set up a hook to reject for failed commit.

Further:

(1) If no branching is still favorable, any commit that break any auto-test should be immediately rejected; if branching is the adopted way to go, any merger need to pass all auto-tests before merging into main.

(2) No sample can be modified except by the maintainer. Any people interested in changing a sample must collaborate with the maintainer before any commit can be made. If the maintainer of a sample has clearly left the code or is not accountable for maintaining the sample, we need to find a replacement maintainer. If no replacement can be found, we may want to remove the sample or move it to an obsolete folder.

8 It would be good if Pencil were to adopt PRs and branches

by Daniel Carrera, Iowa State University, Ames, USA

received 4 June 2025

(1) Someone wants to propose a feature for Pencil. They clone the git repository, add the feature, and when it's all tested they go to GitHub and create a pull request. Then someone in charge at Pencil can approve or reject the PR.

(2) Branches give people freedom to make changes that temporarily break the code, as all their work is done in an isolated branch. But when they finish coding the feature, they have to make sure that all tests pass before they can merge.

Point #1 is code review, that's how codes like Athena operate. In the software industry code review is done

every day. However, I understand it would put too much pressure on the developer team, in essence it means that a set of eyes would be necessary to approve any commit. The automated autotest to reject commits that fail could be a compromise.

I would add that another argument for branches and pull requests is to use the capabilities of git and github, which have become the standard in software development. As it is, students that use and develop Pencil are not being trained in the new version control paradigm, but in the old one of svn and cvs. That's the fundamental problem of education: we teach today, with the tools of yesterday, the people of tomorrow. Wearing our educator hat, we should strive to teach a transferable skill, as many of our students will go on to become data scientists or software engineers; while it is not our job to be a training ground for industry, it is a shortcoming if we don't equip them for that possibility. And for what? Just so we don't leave our comfort zone and learn something new ourselves?

9 Views from the PCSC

by Nils E. L. Haugen, SINTEF, Trondheim

The Pencil Code is a versatile and flexible code that is continuously developed by a number of developers working on highly different physics. It is therefore important to facilitate:

a) workflows that allow for efficient and flexible implementation of new code features, while

b) avoiding the code from branching out into multiple incompatible versions

At the same time, this should be achieved while avoiding:

1) new errors being introduced to the code, and

2) that changes to the code make existing runs (not only the official samples) produce different results than previously. (This could for example be caused by the change of default values.)

In order to facilitate a) and b) while avoiding 1) and 2), I proposed the following four improvements to the Pencil Code check-in culture (should be strictly enforced by the entire community):

i. The author of a commit should always run full auto-test (with full debug options, signalling NaNs, etc) before any commit to the main branch (this includes even small commits)

ii. The author should always get approval from the maintainer of a sample before changing a samples reference data file. If the maintainer of the sample does not respond within a reasonable time, the problem should be brought to the PCSC.

iii. If a commit breaks a sample, anybody can simply just revert that commit (while notifying the author). It is of course beneficial if the faulty code could be corrected to make the sample pass the test instead, but this is not a requirement.

iv. Nobody should ever change the value of defaults (even when default values are later found to be unphysical). This is important in order to avoid that older runs suddenly start producing different results.

The above check-in culture is independent of which version control tool that is used (svn or git), and the choice between svn and git is therefore up to the individual user. The important thing is not which tool that is used, but that errors are not introduced to the code, and that a given check-in does not otherwise cause any problems for other users.

by Axel Brandenburg, Nordita, Stockholm

A guiding principle of the Pencil Code was always to provide as much freedom to individuals as possible. The highly modular structure of the code helps in letting different working styles coexist, as long as the different approaches do not interfere with each other. This is particularly true for allowing different check-in practices to coexist. We have never discouraged the use of branches, and the latest GPU test branch is an example.

A difficulty with branches is that they were never short lived, even though there was always the clear intention to merge them with the main branch as soon as possible. When the gputestv6 branch was merged into the main branch on 19 March 2025 at 10:02 in the morning, a total of 365 files were changed, 259 of those in src alone, and 156 of those were f90 files. I have not checked in detail what happened, but for those files that I did check, I could not say that the changes were straightforward. If there was something that broke for somebody it would be difficult to say exactly what the mistake was. On the other hand, of course, we know that everything was carefully tested.

The experience with the gputestv6 branch was an example with a clear beginning and a clear end, but in general, the idea that some code development is ready

and can then be merged into the main branch seems rather alien to me. Usually, I try out an idea and then continue to iterate on it. Eventually, some runs may already have production quality and enter an early draft of a new paper, but the iteration process may well continue beyond this point. Even when the paper is finally submitted, comments from the referees or other colleagues may inspire changes to the code and make additional runs with the modified code necessary, while some of the old runs would still be ok. This development can finally be called "ready" when the paper is published, but that is also possibly the time when one is no longer working with it, so any mistakes that may occur when making this final product ready (and perhaps still more beautiful) will no longer be scrutinized as much as it would when one was still working with it. (This may not apply to everybody, but to me it does.)

Equally plausible is that I stop and won't finish the job, because more exciting things came up. I have also seen others leaving certain developments untouched without there being any paper. Such a development can come in handy for someone needing it for their own work. If such development was hidden or left on a branch, one would never have known about it. One would not have easily known about its existence. Also, why would have anybody have kept it up to a date while many other changes occurred to the rest of the code if it wasn't on the main branch? This is why the common practice of making changes on the main branch has advantages.

Yet another example is the development of a special relativity extension of the hydro module, which was started in 2022, but is still not ready. Nevertheless, the partially ready version has already been used by others for production runs. Keeping things on the branch does not easily facilitate accidental cross-fertilization.

At any point in time, of course, I checked that my changes did not break the autotests and I do my best not to break anybody else's developments whose integrity may not yet be captured by any autotest.

These is no recipe in designing a good autotest. Whenever there is an opportunity, one should verify that there is an autotest that is sensitive to particular aspects of the code, and, for that matter, not sensitive to the numerical representation of zero or to algorithm-dependent representations of sums. One should also not forget that we have in place some tests of spectra, slice files, etc. I can therefore only appeal to everybody to help making autotests as good as possible. This would entail changing existing ones, but one should leave some days between that and any related changes to the code.

It is generally good that everybody runs the autotest also on their own computer. It does take some time, and by the time you remember about your test, you may no longer have the output on your screen. It is therefore good wo write the output to disk using, e.g.,

```
pc_auto-test --log-dir=AXEL_LOG
```

If then still something goes wrong, it would be useful to be sure that this was not because of something you should have been able to see yourself and that it is really because of different settings compared with the autotests on our webpage. Again, such discrepancies should be used as an encouragement to improve the tests.

10 New auto-test infrastructure

With the inauguration of a new server to replace the old `norlx51` server at Nordita, a new scheme of auto-test is introduced. First, the previous hourly and daily tests are now performed only if needed: Without new check-ins, the auto-tests will not be started, which saves precious computing time on the new server. Second, the distribution of tests is now different. Instead of running all tests up to a certain level for ever test run, we now have three tests, where each of them employs disjunct sets of samples, so no repetitions of the same samples. Third, we now have three tests instead of the two old tests: one basic, one normal, and one extended test. Fourth, disjunct auto-test sets are now allowed to run in parallel, which was not possible on the old server.

The new "basic" test is exactly equivalent to the Travis check (test levels 0 and 1), which means it would run every minute, if there are new check-ins. Unlike the original Travis check, this basic test will always run until the end of the test. If there are additional new check-ins while a test is running, the test will first complete and then start again to test the remaining untested check-ins at the next minute.

The new "normal" test contains the remaining set that is equivalent to the old hourly test (level 2). This additional test will always start at 15 minutes after the full hour.

In addition we have the "extended" test to implement the remaining tests necessary to complete the full set of samples that was previously performed only around midnight. This extended set will start now hourly at 55 minutes after the full hour.

With this new and partly parallel test strategy, we are now performing checks similar to the Travis test

every minute — also as a backup infrastructure to the Travis-CI. Additionally, those auto-tests that were previously performed only once per day, are now running each hour.

11 Random tips

11.1 User-specific settings

Sometimes one likes to make certain settings for all runs of one installation of the Pencil Code. For that purpose, a mechanism was introduced at the Pencil Code User Meeting in Graz (2023). A user may create a file `utils/USER/global_run.in`' inside the main `PENCIL_HOME` directory. This file will always be read before the `run.in` file, so that all parameters defined in the global file, are the new default values. After that, the definitions in the local `run.in` file are read and, if present, will supersede the global default run parameters.

To use this functionality, one only needs to set the environment variable `PENCIL_USER` to contain exactly the `USER` name, in which directory to look for the `global_run.in` file, see above. If this environment variable is not present (or the path is wrong), only the local parameters of the `run.in` file will be read.

In case a user wants to give different global run parameter definitions, one can provide multiple file names in the scheme `utils/USER/HOST_global_run.in`. To select one of them to be used, one needs to set another environment variable `PENCIL_HOST` that contains the `HOST` part.

11.2 Safer snapshots

Through setting `lbackup_snap=T` in `run_pars` the code is caused to hold the penultimate instance of `var.dat` as `var.dat.bck`. By this, a failing write of `var.dat` due to, e.g., job canceling or disk failure, would result merely in a loss if `isave` integration timesteps. By default, this feature is off at present, but we may decide to change this in the future.

11.3 Indicator file for snapshot writing

During writing of `var.dat`, `VAR*`, `dvar.dat`, or `crash.dat`, the (empty) file `WRITING` is now present in the working directory.

11.4 Automatic CVS updates

Setting `lupdate_cvs=T` in `run_pars` makes the code executing `cvs ci` after each update of the time series. Default is "`F`". Of course, this is only effective if `pc_cvsci` (or something equivalent) has been executed in this directory before.

11.5 Check-in notifications

Check-in notifications are currently not being sent out because the Google group mailing list does not forward emails from `notifications@github.com`, even though GitHub support checked that emails are delivered there and are reaching Google groups.

12 Next PC User Meeting

The next PC User Meeting will be held in Geneva/Switzerland. Alberto Roper Pol organizes the meeting and has now opened the registration on `https://indico.cern.ch/e/PCUM2025`.

13 Papers since October 2024

As usual, we look here at new papers that make use of the Pencil Code. Since the last newsletter of October, 17 new papers have appeared on the arXiv, plus 27 others, some of which had been just preprints and have now been published in a journal. We list both here, altogether 44. A browsable ADS list of all Pencil Code papers can be found on: `https://ui.adsabs.harvard.edu/public-libraries/iGR7N570Sy6AlhDMQRTe_A`. If something is missing in those entries, you can also include it yourself in: `https://github.com/pencil-code/pencil-code/blob/master/doc/citations/ref.bib`, or otherwise just email brandenb@nordita.org. A compiled version of this file is available as `https://github.com/pencil-code/website/blob/master/doc/citations.pdf`, where we also list a total of now 136 code comparison papers in the last section "Code comparison & reference". Those are not included in our list below, nor among the now total number of 722 research papers that use the Pencil Code.

References

Brandenburg, A., Iarygina, O., Sfakianakis, E.I. and Sharma, R., Magnetogenesis from axion-SU(2) inflation. *J.. Cosmol. Astropart. Phys.*, 2024, **2024**, 057.

Brandenburg, A., Käpylä, P.J., Rogachevskii, I. and Yokoi, N., Helicity Effect on Turbulent Passive and Active Scalar Diffusivities. *Astrophys. J.*, 2025a, **984**, 88.

Brandenburg, A., Larsson, G., Del Sordo, F. and Käpylä, P.J., Magnetorotational instability in a solar mean-field dynamo. *arXiv e-prints*, 2025b, arXiv:2504.16849.

Brandenburg, A. and Ntormousi, E., Magnetic field amplification during a turbulent collapse. *arXiv e-prints*, 2025, arXiv:2505.02885.

Brandenburg, A. and Scannapieco, E., Magnetically Assisted Vorticity Production in Decaying Acoustic Turbulence. *Astrophys. J.*, 2025, **983**, 105.

Brandenburg, A. and Vishniac, E.T., Magnetic helicity fluxes in dynamos from rotating inhomogeneous turbulence. *arXiv e-prints*, 2024, arXiv:2412.17402.

Brandenburg, A. and Vishniac, E.T., Magnetic Helicity Fluxes in Dynamos from Rotating Inhomogeneous Turbulence. *Astrophys. J.*, 2025, **984**, 78.

Brandenburg, A., Yi, L. and Wu, X., Inverse cascade from helical and nonhelical decaying columnar magnetic fields. *arXiv e-prints*, 2025c, arXiv:2501.12200.

Dehman, C. and Brandenburg, A., Reality of inverse cascading in neutron star crusts. *Astron. Astrophys.*, 2025, **694**, A39.

Dwivedi, S., Anandavijayan, C. and Bhat, P., Quasi-two-dimensionality of three-dimensional, magnetically dominated, decaying turbulence. *Open J. Astrophys.*, 2024, **7**, 75.

Elias-López, A., Del Sordo, F. and Viganò, D., Vorticity and magnetic dynamo from subsonic expansion waves: II. Dependence on the magnetic Prandtl number, forcing scale, and cooling time. *Astron. Astrophys.*, 2024, **690**, A77.

Eriksson, L.E.J., Yang, C.C. and Armitage, P.J., Particle fragmentation inside planet-induced spiral waves. *Month. Not. Roy. Astron. Soc.*, 2025, **537**, L26–L32.

Hidalgo, J.P., Käpylä, P.J., Schleicher, D.R.G., Ortiz-Rodríguez, C.A. and Navarrete, F.H., Magnetohydrodynamic simulations of A-type stars: Long-term evolution of core dynamo cycles. *Astron. Astrophys.*, 2024, **691**, A326.

Hidalgo, J.P., Käpylä, P.J., Schleicher, D.R.G., Ortiz-Rodríguez, C.A. and Navarrete, F.H., Shaping core dynamos in A-type stars: The role of dipolar fossil fields. *arXiv e-prints*, 2025, arXiv:2506.01017.

Hosking, D.N., Wasserman, D. and Cowley, S.C., Metastability of stratified magnetohydrostatic equilibria and their relaxation. *J. Plasma Phys.*, 2025, **91**, E35.

Käpylä, P.J., Simulations of entropy rain-driven convection. *arXiv e-prints*, 2025, arXiv:2504.00738.

Kesri, K., Dey, S., Chatterjee, P. and Erdelyi, R., Dependence of Spicule Properties on the Magnetic Field—Results from Magnetohydrodynamics Simulations. *Astrophys. J.*, 2024, **973**, 49.

Kishore, G. and Singh, N.K., Rotational effects on the small-scale dynamo. *arXiv e-prints*, 2025a, arXiv:2502.17178.

Kishore, G. and Singh, N.K., The spectra of solar magnetic energy and helicity. *arXiv e-prints*, 2025b, arXiv:2503.03332.

Lipatnikov, A.N., A priori assessment of a simple approach to evaluating burning rate in large eddy simulations of premixed turbulent combustion. *Phys. Fluids*, 2024, **36**, 115152.

Maity, S.S., Chatterjee, P., Sarkar, R. and Mytheen, I., On the Evolution of Reconnection Flux in Erupting Magnetic Flux Ropes: Insights from Observations and MHD Simulation; in *AGU Fall Meeting Abstracts*, Vol. 2024 of *AGU Fall Meeting Abstracts*, Dec., 2024, pp. SH13B–2928.

Meftah, J., Hydrodynamic simulations of multiple low-mass migrating black holes in AGN disks; in *American Astronomical Society Meeting Abstracts*, Vol. 245 of *American Astronomical Society Meeting Abstracts*, Jan., 2025, p. 402.06.

Mondal, T., Bhat, P., Ebrahimi, F. and Blackman, E.G., Understanding large-scale dynamos in unstratified rotating shear flows. *arXiv e-prints*, 2025, arXiv:2505.03660.

Mtchedlidze, S., Domínguez-Fernández, P., Du, X., Carretti, E., Vazza, F., O'Sullivan, S.P., Brandenburg, A. and Kahniashvili, T., Intergalactic Medium Rotation Measure of Primordial Magnetic Fields. *Astrophys. J.*, 2024, **977**, 128.

Park, K., Effect of turbulent kinetic helicity on diffusive β effect for large scale dynamo. *Phys. Rev. D*, 2025, **111**, 023021.

Qazi, Y., Shukurov, A., Tharakkal, D., Gent, F.A. and Bendre, A.B., Non-linear magnetic buoyancy instability and galactic dynamos. *Month. Not. Roy. Astron. Soc.*, 2025, **540**, 532–544.

Qazi, Y., Shukurov, A., Gent, F.A., Tharakkal, D. and Bendre, A.B., Non-linear magnetic buoyancy instability and galactic dynamos. *arXiv e-prints*, 2024, arXiv:2412.05086.

Rice, K., Baehr, H., Young, A.K., Booth, R., Rowther, S., Meru, F., Hall, C. and Koval, A., Dust density enhancements and the direct formation of planetary cores in gravitationally unstable discs. *Month. Not. Roy. Astron. Soc.*, 2025, **539**, 3421–3435.

Rogachevskii, I., Kleeorin, N. and Brandenburg, A., Theory of the Kinetic Helicity Effect on Turbulent Diffusion of Magnetic and Scalar Fields. *Astrophys. J.*, 2025, **985**, 18.

Roper Pol, A. and Salvino Midiri, A., Relativistic magnetohydrodynamics in the early Universe. *arXiv e-prints*, 2025, arXiv:2501.05732.

Schäfer, U., Johansen, A., Haugbølle, T. and Nordlund, Å., Thousands of planetesimals: Simulating the streaming instability in very large computational domains. *Astron. Astrophys.*, 2024, **691**, A258.

Sharma, P., Vaidya, B., Wadadekar, Y., Bagla, J., Chatterjee, P., Hanasoge, S., Kumar, P., Mukherjee, D., Sajeeth Philip, N. and Singh, N., Computational Astrophysics, Data Science & AI/ML in Astronomy: A Perspective from Indian Community. *arXiv e-prints*, 2025a, arXiv:2501.03876.

Sharma, R., Brandenburg, A., Subramanian, K. and Vikman, A., Lattice simulations of axion-U(1) inflation: gravitational waves, magnetic fields, and black holes. *arXiv e-prints*, 2024, arXiv:2411.04854.

Sharma, R., Brandenburg, A., Subramanian, K. and Vikman, A., Lattice simulations of axion-U(1) inflation: gravitational waves, magnetic fields, and scalar statistics. *J.. Cosmol. Astropart. Phys.*, 2025b, **2025**, 079.

Shchutskyi, N., Schaller, M., Karapiperis, O.A., Stasyszyn, F.A. and Brandenburg, A., Kinematic dynamos and resolution limits for Smoothed Particle Magnetohydrodynamics. *arXiv e-prints*, 2025, arXiv:2505.13305.

Shi, J., Bartelmann, M., Klahr, H. and Dullemond, C.P., Kinetic field theory applied to planetesimal formation I: freely streaming dust particles. *Month. Not. Roy. Astron. Soc.*, 2025, **536**, 1625–1644.

Singh, N.K., Ajay, A. and Rajesh, S.R., Reversals of toroidal magnetic field in local shearing box simulations of accretion disc with a hot corona. *arXiv e-prints*, 2024, arXiv:2410.14497.

Singh, N.K., Ajay, A. and Rajesh, S.R., Reversals of Toroidal Magnetic Field in Local Shearing Box Simulations of Accretion Disk with a Hot Corona. *Astrophys. J.*, 2025, **984**, 113.

Tschernitz, J. and Bourdin, P.A., Granulation and Convectional Driving on Stellar Surfaces. *Astrophys. J. Lett.*, 2025, **979**, L39.

Vachaspati, T. and Brandenburg, A., Spectra of magnetic fields from electroweak symmetry breaking. *arXiv e-prints*, 2024, arXiv:2412.00641.

Vemareddy, P., Simulating the Formation and Eruption of Flux Rope by Magneto-friction Model Driven by Time-dependent Electric Fields. *Astrophys. J.*, 2024, **975**, 251.

Warnecke, J., Korpi-Lagg, M.J., Rheinhardt, M., Viviani, M. and Prabhu, A., Small-scale and large-scale dynamos in global convection simulations of solar-like stars. *Astron. Astrophys.*, 2025, **696**, A93.

Yuvraj, Im, H.G. and Chaudhuri, S., How "mixing" affects propagation and structure of intensely turbulent, lean, hydrogen-air premixed flames. *Combust. Flame*, 2025, **273**, 113903.

Zhou, H. and Lai, D., Understanding the UV/Optical Variability of AGNs through Quasi-Periodic Large-scale Magnetic Dynamos. *arXiv e-prints*, 2024, arXiv:2411.12953.

This PENCIL CODE Newsletter was edited by Axel Brandenburg `<brandenb@nordita.org>`, Nordita, KTH Royal Institute of Technology and Stockholm University, SE-10691 Stockholm, Sweden; and Matthias Rheinhardt `<matthias.rheinhardt@aalto.fi>`, Department of Computer Science, Aalto University, PO Box 15400, FI-00076 Aalto, Finland. See `http://www.nordita.org/~brandenb/pencil-code/newsletter` or `https://github.com/pencil-code/website/tree/master/NewsLetters` for the online version as well as back issues.