Response to reviewers

We would like to warmly thank both reviewers for their careful revision and valuable feedback, which helped to improve the quality of our submission. Based on their comments we have significantly re-structured and enhanced our contribution. A detailed point-by-point response and explanation of the modifications made is provided below.

Reviewer #1 (@tiburoch)

- **Comment #1:**

Dear **@Habiba-Eldababy** ,

Your code is geared towards Continuum Damage Mechanics, yet there is no information in the documentation nor the paper about the specific features of your code.

Specifically, which damage models are implemented (local/non-local, specific formulation)? What is the behavior of the bulk? Can you solve 3D problems? What are the 2D conditions (plane strain/plane stress)? What type of boundary conditions can you use? Is your code solving static or dynamic problems? All those questions (and maybe more) should be addressed in the documentation and the description of your software.

In addition to specifying the features of your code, you should also compare them to the capabilities of other accessible software. What is the main difference of your code compared to existing FEM libraries that include damage mechanics in parallel? For example, I can think of akantu that includes parallel phase-field models (https://joss.theoj.org/papers/10.21105/joss.05253), oofem (https://oofem.org/doku.php?id=en:oofem) that includes both parallelism and many damage models (I am not sure if they work together though...), FEniCS (https://fenicsproject.org/) etc...

This will help clarify what your software does and the statement of needs of the JOSS paper.

Response:

We wish to thank the reviewer for this comment, and we have addressed all the issues raised in the revised submission. First, a complete list of the code's features and functionality capabilities has been added as Section I in the "ParallelCDM_Documentation". The main features are listed below:

- Both local and non-local damage models are implemented
- Quasi-brittle materials are analyzed, and two damage formulations are currently implemented (Mazars' and Geers' damage laws)
- The code is currently written for 2D (plane-stress or plane-strain) static problems
- Dirichlet boundary conditions are imposed
- Both structured and unstructured quadrilateral meshes can be analyzed
- Two numerical solvers are available (Newton-Raphson and the Unified Arc-Length)

Regarding the comparison to existing platforms, our software differs in two main key ways. First, our code is developed in MATLAB, which is an easily accessible language widely used by early-stage researchers. This is in contrast with more advanced platforms, like FEniCSx or Acantu, and renders our code particularly suitable as an entry point to continuum damage mechanics for students or less-experienced coders. Second, our software features the implementation of a newly proposed [3] arc-length solver (UAL). The strength of UAL compared to other solvers was already demonstrated in [3], providing the user with a tool capable of efficiently capturing really challenging damage mechanics responses with snap-back features. As UAL is still under in-house development and not available in any other platform, it clearly differentiates our contribution from other existing approaches. To address the above, a relevant discussion has been added in the 3rd paragraph of the "Statement of Need" section in the main document.

- **Comment #2:**

Dear **@Habiba-Eldababy**,

I believe there is a lack of documentation for your code, which I think is partly related to #1 . It is not clear what is implemented and how a new user can interact with the code apart from changing the parameters in the input files, which are not documented either. To make your code accessible to new users, you should include documentation regarding:

- The minimum structure of a working example, and a description of the corresponding functions. Right now, I would be unable to write a script by myself. How do I read material parameters? How do I apply boundary conditions? How do I call the solver? Etc... As a suggestion, this might benefit from factorizing the code a bit, as there are many variables exposed on the main file, which makes it difficult to read.
- The mesh. What are the supported formats? Maybe you can also suggest a software that generates such mesh format. What types of elements are supported? Can I use an unstructured mesh? Etc...
- The input parameters. Every parameter should be documented. In the case of purely numerical (non-physical) parameters for the solver, can you specify if/how they might impact the validity of the results?
- The content of the outputs.

Documenting all those points will not only allow users to get started with your code, but will also allow them to potentially modify or expand it to their needs.

Response:

Based on the reviewer's comment, we have significantly upgraded the documentation of the submission. To this end, we have created a "ParallelCDM_Documentation" file which addresses all the points raised. Overall, this document includes information about the software features and software prerequisites, detailed explanation of the code and files structure, detailed steps on how to run the code, explanation of the hyperparameters, and more.

With regards to each specific point raised in the comment above:

- W.r.t. the first bullet point: Each model is fully defined within a mesh file and a parameter file (both ".txt"), and several sample working examples exist in the folder "Test problems". The steps to run the code are presented in the Documentation file.
    - Regarding the suggestion for factorization: we have increased the code efficiency by cleaning up several files, and moving all user-defined variables in the parameter ".txt" file. To run the code, the user now only needs to edit the file path, model name, and parallel pool in the main script. This is indicated by relevant comments in the documentation.
- W.r.t. the second bullet point: The mesh-related files include definitions of nodal coordinates, element connectivity, and imposed Dirichlet boundary conditions. The parameter file includes the parameters of the problem. Though not restricting, we suggest the creation of the mesh files with the GMSH software. Our code supports both structured and unstructured meshes. The elements should be quadrilateral elements with 4 Gauss (integration) points.
- W.r.t. the third bullet point: The model hyperparameters are fully documented in section "VII. Details of parameter files". A detailed investigation of the impact of each parameter is considered out-of-scope for this submission, as it would significantly exceed the acceptable length of the article. Instead, in this submission we mainly explore the impact of parameters which are pertinent to the parallelization (threads, processes, etc.)
- W.r.t. the fourth bullet point: The code outputs a data file stored in the "**Saved_Files**" folder at each increment as well as contour plots for damage, local strain, and nonlocal strain at the last load increment stored in the "**Images**" folder.

This information is explained in the documentation so users are able to interact with the code and potentially modify it to their needs.

- **Comment #3**

Dear **@Habiba-Eldababy** ,

You need to implement tests for your software, either by comparing a simulation to an analytical solution (if available) or with a benchmark test.

Response:
We thank the reviewer for this comment which increases the reliability of our code. We have tested our code with the direct tension test problem with L=50mm configuration reported by Peerlings et al. and reported the result in the paper in the section "Testing and Results". The code and the solution in Peerlings et al. show the results are in excellent agreement, an observation which increases the robustness and trustworthiness of our code.

- **Comment #4**

@Habiba-Eldababy , please indicate in the README that your code requires the package Parallel Computing Toolbox from Matlab.

Response:
We have included a section in the Documentation file to indicate the MATLAB Parallel Computing Toolbox as a prerequisite for our code.

- **Comment #5**

Dear **@Habiba-Eldababy** ,

Can you please indicate in the README or the documentation how third parties can:

- Contribute to the software
- Report issues or problems with the software
- Seek support

Response:
We have mentioned areas of potential contributions in the Documentation file, in section "IX. Contributions, Support, and Reporting Issues". These include extending the code to other element types, other nonlinear solvers, other damage/phase-field models, or even extending it to 3D or multi-physics problems. To this end, we have indicated that users can open an issue on Github or use alternate form of communication if they prefer (emails of 2 of the listed authors).

- **Comment #6**

Dear **@Habiba-Eldababy** ,

The Methodology and Software Implementation paragraph of your paper focuses on the parallelization of your code. Most of the section explains how different parallelization routines work in Matlab, but the main message is summarized in the last part of this paragraph:

"In the solver mode of the code, the function will assemble global matrices, so we implement 'spmd' since ordered execution and data sharing between workers is needed. In the plotting properties portion, we use 'parfor' as the loop body is independent, and iterations can be executed in any order."

I think this paragraph can be significantly shortened and more focused towards your code.

Response:

Following the reviewer's comment, this section has now been shortened in order to increase clarity. The additional space was used to discuss the test comparison between our code and the result in Peerlings et al.

Reviewer #2 (@vijaysm)

- **Comment #1:**

I second many of the review points that **@tiburoch** has already raised. The repository is missing several recommended and required practices.

1. community contribution guidelines,
2. documentation related to additional dependencies (parallel computing toolbox) and pointers to tools for generating new meshes for test cases,
3. detailed documentation describing the test suite and explaining functionality using tutorials/examples
4. description of drivers FEM_Main_Script_2D.m and FEM_Main_Script_2D_HPC.m, their usage, and along with the differences between the two versions
5. documentation detailing the control of process vs thread-based parallelized runs and methods to change the number of tasks (pointers to MATLAB documentation may suffice along with instructions to run the driver)

I also have a general comment about the paper.

- In Figure 1., what happens when you update the parameters? There is perhaps an arrow missing to restart the iterations?
- In Figure 2., the solver workflow, I only see the assembly of the stiffness matrix being done in parallel. Does the solver use any parallelization? This is not indicated in the flowchart.
- For performance results in Figures 5 and 6, it may be useful to consider your "medium" mesh as the coarse one, and use another level of refinement for the fine one to showcase better results on the workstation and the cluster.
- Is the parallel efficiency of MATLAB thread-level parallelism limited? The scaling plots plateau immediately beyond 2-3 threads on the workstation and 8 threads on a cluster. While there is a reduction in total runtime compared to serial execution, does the author think that leveraging MATLAB's thread-level parallelism provides a speedup comparable to domain-decomposition-based approaches? You can discuss these implications as part of your conclusions.

Please address these suggestions and let me know when the repository/paper are ready to be reviewed again

We would like to thank the reviewer for the attentive review and thoughtful questions regarding the parallelization. We have addressed all comments raised, and below we provide our response:

- W.r.t. point 1: the repository has now been updated to include the required and recommended practices mentioned. Community contribution is most welcome and we have indicated how this can be done in the README and Documentation file.

- W.r.t. point 2: Information on the software prerequisites (MATLAB and the Parallel Computing Toolbox) and mesh-generation software (such as GMSH) is included in the pertinent sections of the Documentation file.
- W.r.t. point 3: A test for our code is available and detailed in the paper, along with working examples ready for use in the folder "Test problems". Additionally, with regards to the documentation aspect of the question, we note that each code function includes an opening comment that describes its role, in order to facilitate user interaction and future modifications.
- W.r.t. point 4: Following the re-structuring of our code, we note that now there is only one "FEM_Main_Script_2D.m" file. The user can use this file alone to run the code on any device (local or HPC), with parallel computing capability, by simply modifying the number of cores/threads to be used in the main script.
- W.r.t. point 5: In the revised version of the code we describe the parallel pool and number options in section "VIII. Tips for testing and improved performance" of the Documentation file. In addition, the user will encounter the option to modify the pool in the user input area of the "FEM_Main_Script_2D.m" file.

We also appreciate the reviewer's thoughtful comments on the paper and address them below one-by-one.

- There was indeed a missing - this has been updated to indicate that the iterations restart. The flowcharts have also been edited to increase readability.
- The solver itself does not use any parallelization – hence we do not make a statement about that in the workflow.
- On the front of using a finer mesh, we underline that we are currently limited by memory constraints that arise when running models with more elements than those reported. We believe our mesh discretization scheme, which leads up to 10,201 elements, is sufficient to demonstrate the trend for progressive speedup (at least 3 times faster).
- The plateauing behavior observed beyond 2–3 threads on the workstation and 8–16 threads on the HPC cluster is primarily due to the overhead costs inherent to thread-based parallelism in MATLAB. These are related to the worker setup, memory synchronization, and data sharing overhead, which begin to outweigh the benefits as thread count increases without a proportionally larger problem size. These findings are illustrated in Fig. 7 and Fig. S2 of our paper. With regards to its comparison to domain decomposition methods, we acknowledge that such methods tend to show stronger trends of scalability for large-scale problems solved on distributed memory systems. Libraries like PETSc, deal.II, and OpenSees are designed to work with MPI-based parallelism and are well-suited to large-scale simulations. However, domain decomposition comes with considerable implementation complexity and typically requires users to have expertise in low-level parallel programming and mesh partitioning. In contrast, our code adopts MATLAB's thread-based shared-memory model, which offers a practical and user-friendly approach to accelerate small- to medium-sized CDM problems. Despite the inherent limitations of thread-level parallelism, we demonstrate that our implementation provides 3–10× speedup over serial execution, particularly when the mesh size is sufficient and memory is not a limiting factor.