




1 checkpoint_schedules: schedules for incremental 2 checkpointing of adjoint simulations

3 **Daiane I. Dolci** ¹, **James R. Maddison**², **David A. Ham** ¹, **Guillaume
4 Pallez** ³, and **Julien Herrmann**⁴

5 **1** Department of Mathematics, Imperial College London, London, SW72AZ, UK. **2** School of
6 Mathematics and Maxwell Institute for Mathematical Sciences, The University of Edinburgh, EH9 3FD **3**
7 Inria, University of Rennes, France. **4** CNRS, IRIT, Université de Toulouse.

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Patrick Diehl](#) 

Reviewers:

- [@matt-graham](#)
- [@KYANJO](#)

Submitted: 28 September 2023

Published: unpublished


License


Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#))

8 Summary

9 *checkpointing_schedules* provides schedules for step based incremental checkpointing of the
10 adjoints to computer models. The schedules contain instructions indicating the sequence of
11 forward and adjoint steps to be executed, and the data storage and retrieval to be performed.
12 These instructions are independent of the model implementation, which enables the model
13 authors to switch between checkpointing algorithms without recoding. Conversely, *checkpoint-*
14 *ing_schedules* provides developers of checkpointing algorithms a direct mechanism to convey
15 their work to model authors. *checkpointing_schedules* has been integrated into **tlm_adjoint**
16 ([James R. Maddison et al., 2019](#)), a Python library designed for the automated derivation
17 of higher-order tangent-linear and adjoint models and work is ongoing to integrate it with
18 **pyadjoint** ([Mitusch et al., 2019](#)). This package can be incorporated into other gradient solvers
19 based on adjoint methods, regardless of the specific approach taken to generate the adjoint
20 model.

21 The use of adjoint calculations to compute the gradient of a quantity of interest resulting
22 from the solution of a system of partial differential equations (PDEs) is widespread and
23 well-established. The resulting gradient may be employed for many purposes, including topology
24 optimisation ([Papadopoulos et al., 2021](#)), inverse problems ([Plessix, 2006](#)), flow control ([Jansen,
25 2011](#)).

26 Solving the adjoint to a non-linear time-dependent PDE requires the forward PDE to be solved
27 first. The adjoint PDE is then solved in a reverse time order, but depends on the forward
28 state. Storing the entire forward state in preparation for the adjoint calculation has a memory
29 footprint linear in the number of time steps. For sufficiently large problems this will exhaust
30 the memory of any computer system 

31 In contrast, checkpointing approaches store only the state required to restart the forward
32 calculation from a limited set of steps. As the adjoint calculation progresses, the forward
33 computation is progressively rerun from the latest available stored state up to the current adjoint
34 step. This enables less forward state to be stored, at the expense of a higher computational
35 cost as forward steps are run more than once. ([Griewank & Walther, 2000](#)) proposed a
36 checkpointing algorithm which is optimal under certain assumptions, including that the number
37 of steps is known in advance, and that all the storage has equal access cost. Subsequent
38 authors have produced checkpointing algorithms that relax these requirements in various ways,
39 such as by accounting for different types of storage (e.g. RAM  and disk) or by not requiring the
40 number of steps to be known in advance, for example ([Aupy et al., 2016](#); [Aupy & Herrmann,
41 2017](#); [Herrmann, 2020](#); [James R. Maddison, 2023](#); [Schanen et al., 2016](#); [Stumm & Walther,
42 2009](#); [Zhang & Constantinescu, 2023](#)).

43 Statement of need

44 This situation is typical across computational mathematics: there exists a diversity of algorithms
45 whose applicability and optimality depends on the nature and parameters of the problem to be
46 solved. From the perspective of users who wish to construct adjoint solvers this creates the
47 need to swap out different checkpointing algorithms in response to changes in the equations,
48 discretisations, and computer systems with which they work. Those users will often lack the
49 expertise or the time to continually reimplement additional algorithms in their framework.
50 Further, such reimplementation work is wasteful and error-prone.

51 *checkpointing_schedules* provides a number of checkpointing algorithms accessible through a
52 common interface and these are interchangeable without recoding. This can be used in
53 conjunction with an adjoint framework such as *tlm_adjoint* or *pyadjoint* and a compatible PDE
54 framework, such as Firedrake (Ham et al., 2023) or FEniCS (Alnaes et al., 2015) to enable
55 users to create adjoint solvers for their choice of PDE, numerical methods, and checkpointing
56 algorithm all without recoding the underlying algorithms.

57 Some of the algorithms supported by *checkpointing_schedules* have been implemented many
58 times, while for others, such as H-Revolve the only previously published implementation was a
59 simple proof of concept in the original paper (Herrmann, 2020). The checkpoint schedule API
60 provided by *checkpoint_schedules* further provides a toolkit for the implementation of further
61 checkpoint schedules, thereby providing a direct route from algorithm developers to users.

62 Software description

63 Currently, *checkpoint_schedules* is able to generate schedules for the following checkpointing
64 schemes: Revolve (Stumm & Walther, 2009); disk-revolve (Aupy et al., 2016); periodic-disk
65 revolve (Aupy & Herrmann, 2017); two-level (Pringle et al., 2016); H-Revolve (Herrmann,
66 2020); and mixed storage checkpointing (James R. Maddison, 2023). It also contains trivial
67 schedules which store the entire forward state. This enables users to support adjoint calculations
68 with or without checkpointing via a single code path.

69 The complete documentation for *checkpoint_schedules* is available on [the Firedrake project website](#).

71 Acknowledgments

72 This work was supported by the Engineering and Physical Sciences Research Council
73 [EP/W029731/1 and EP/W026066/1]. J. R. M. was supported by the Natural Environment
74 Research Council [NE/T001607/1]. G. P. was supported in part by the French National
75 Research Agency (ANR) in the frame of DASH (ANR-17-CE25- 0004).

76 References

- 77 Alnaes, M. S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring,
78 J., Rognes, M. E., & Wells, G. N. (2015). The FEniCS project version 1.5. *Archive of*
79 *Numerical Software*, 3. <https://doi.org/10.11588/ans.2015.100.20553>
- 80 Aupy, G., & Herrmann, J. (2017). Periodicity in optimal hierarchical checkpointing schemes
81 for adjoint computations. *Optimization Methods and Software*, 32(3), 594–624. <https://doi.org/10.1080/10556788.2016.1230612>
- 82
- 83 Aupy, G., Herrmann, J., Hovland, P., & Robert, Y. (2016). Optimal multistage algorithm
84 for adjoint computation. *SIAM Journal on Scientific Computing*, 38(3), C232–C255.
85 <https://doi.org/10.1145/347837.347846>

- 86 Griewank, A., & Walther, A. (2000). Revolve: An implementation of checkpointing for
87 the reverse or adjoint mode of computational differentiation. *ACM Transactions on*
88 *Mathematical Software (TOMS)*, 26(1), 19–45. <https://doi.org/10.1145/347837.347846>
- 89 Ham, D. A., Kelly, P. H. J., Mitchell, L., Cotter, C. J., Kirby, R. C., Sagiya, K., Bouziani,
90 N., Vorderwuelbecke, S., Gregory, T. J., Betteridge, J., Shapero, D. R., Nixon-Hill, R.
91 W., Ward, C. J., Farrell, P. E., Brubeck, P. D., Marsden, I., Gibson, T. H., Homolya,
92 M., Sun, T., ... Markall, G. R. (2023). *Firedrake user manual* (First edition). Imperial
93 College London; University of Oxford; Baylor University; University of Washington. <https://doi.org/10.25561/104839>
- 94
- 95 Herrmann, J. (2020). H-revolve: A framework for adjoint computation on synchronous
96 hierarchical platforms. *ACM Transactions on Mathematical Software (TOMS)*, 46(2), 1–25.
97 <https://doi.org/10.1145/3378672>
- 98 Jansen, J. D. (2011). Adjoint-based optimization of multi-phase flow through porous media –
99 a review. *Computers & Fluids*, 46(1), 40–51. [https://doi.org/10.1016/j.compfluid.2010.](https://doi.org/10.1016/j.compfluid.2010.09.039)
100 [09.039](https://doi.org/10.1016/j.compfluid.2010.09.039)
- 101 Maddison, James R. (2023). On the implementation of checkpointing with high-level algorithmic
102 differentiation. *arXiv Preprint arXiv:2305.09568*. [https://doi.org/10.48550/arXiv.2305.](https://doi.org/10.48550/arXiv.2305.09568)
103 [09568](https://doi.org/10.48550/arXiv.2305.09568)
- 104 Maddison, James R., Goldberg, D. N., & Goddard, B. D. (2019). Automated calculation of
105 higher order partial differential equation constrained derivative information. *SIAM Journal*
106 *on Scientific Computing*, 41(5), C417–C445. <https://doi.org/10.1137/18M1209465>
- 107 Mitusch, S. K., Funke, S. W., & Dokken, J. S. (2019). Dolfin-adjoint 2018.1: Automated
108 adjoints for FEniCS and firedrake. *Journal of Open Source Software*, 4(38), 1292. <https://doi.org/10.21105/joss.01292>
- 109
- 110 Papadopoulos, I. P., Farrell, P. E., & Surowiec, T. M. (2021). Computing multiple solu-
111 tions of topology optimization problems. *SIAM Journal on Scientific Computing*, 43(3),
112 A1555–A1582. <https://doi.org/10.1137/20M1326209>
- 113 Plessix, R.-E. (2006). A review of the adjoint-state method for computing the gradient
114 of a functional with geophysical applications. *Geophys. J. Int*, 167, 495–503. <https://doi.org/10.1111/j.1365-246X.2006.02978.x>
- 115
- 116 Pringle, G., Jones, D. C., Goswami, S., Narayanan, S. H. K., & Goldberg, D. (2016). *Providing*
117 *the ARCHER community with adjoint modelling tools for high-performance oceanographic*
118 *and cryospheric computation*.
- 119 Schanen, M., Marin, O., Zhang, H., & Anitescu, M. (2016). Asynchronous two-level check-
120 pointing scheme for large-scale adjoints in the spectral-element solver Nek5000. *Procedia*
121 *Computer Science*, 80, 1147–1158. <https://doi.org/10.1016/j.procs.2016.05.444>
- 122 Stumm, P., & Walther, A. (2009). Multistage approaches for optimal offline checkpointing.
123 *SIAM Journal on Scientific Computing*, 31(3), 1946–1967. [https://doi.org/10.1137/](https://doi.org/10.1137/080718036)
124 [080718036](https://doi.org/10.1137/080718036)
- 125 Zhang, H., & Constantinescu, E. M. (2023). Optimal checkpointing for adjoint multistage
126 time-stepping schemes. *Journal of Computational Science*, 66, 101913. [https://doi.org/10.](https://doi.org/10.1016/j.jocs.2022.101913)
127 [1016/j.jocs.2022.101913](https://doi.org/10.1016/j.jocs.2022.101913)