SUPPLEMENTARY MATERIAL

Anonymous authors
Paper under double-blind review

CONTENTS

A CODE AVAILABILITY

Our code is publicly available. For reviewing purposes, it is temporarily hosted at https://github.com/pnode-dev/pnode.

B PROOFS

B.1 PROOF OF PROPOSITION 1

Proposition 1. *Assuming $\widetilde{\boldsymbol{\lambda}}_{n+1} = \boldsymbol{\lambda}_{n+1}$, the local discrepancy between the continuous adjoint sensitivity and the discrete adjoint sensitivity for forward Euler with the same step size h is*

$$\|\widetilde{\boldsymbol{\lambda}}_n - \boldsymbol{\lambda}_n\| = h^2 \|\mathcal{H}\left(\boldsymbol{u}_n + \epsilon(\boldsymbol{u}_{n+1} - \boldsymbol{u}_n), \boldsymbol{\theta}, t_n\right) f(\boldsymbol{u}_n, \boldsymbol{\theta}, t_n)\| \|\boldsymbol{\lambda}_{n+1}\|, \tag{1}$$

where $\mathcal{H}$ is the Hessian of f and ϵ is a constant in $(0, 1)$.

Proof. Discretizing the continuous adjoint equation

$$\frac{d\widetilde{\boldsymbol{\lambda}}}{dt} = -\left(\frac{\partial f}{\partial \boldsymbol{u}}\right)^T \widetilde{\boldsymbol{\lambda}} \tag{2}$$

with forward Euler gives

$$\widetilde{\boldsymbol{\lambda}}_n = \widetilde{\boldsymbol{\lambda}}_{n+1} + h\left(\frac{\partial f(\boldsymbol{u}_{n+1}, \boldsymbol{\theta}, t_{n+1})}{\partial \boldsymbol{u}}\right)^T \widetilde{\boldsymbol{\lambda}}_{n+1}. \tag{3}$$

The discrete adjoint of the forward Euler is

$$\boldsymbol{\lambda}_n = \boldsymbol{\lambda}_{n+1} + h\left(\frac{\partial f(\boldsymbol{u}_n, \boldsymbol{\theta}, t_n)}{\partial \boldsymbol{u}}\right)^T \boldsymbol{\lambda}_{n+1}. \tag{4}$$

With (3) and (4) and the assumption $\widetilde{\boldsymbol{\lambda}}_{n+1} = \boldsymbol{\lambda}_{n+1}$, we have

$$\|\widetilde{\boldsymbol{\lambda}}_n - \boldsymbol{\lambda}_n\| = h\| \left(\frac{\partial f(\boldsymbol{u}_{n+1}, \boldsymbol{\theta}, t_{n+1})}{\partial \boldsymbol{u}}\right)^T - \left(\frac{\partial f(\boldsymbol{u}_n, \boldsymbol{\theta}, t_n)}{\partial \boldsymbol{u}}\right)^T \| \|\boldsymbol{\lambda}_{n+1}.\| \tag{5}$$

Using the Taylor's series expansion, we obtain

$$\begin{aligned}\left(\frac{\partial f(\boldsymbol{u}_{n+1}, \boldsymbol{\theta}, t_{n+1})}{\partial \boldsymbol{u}}\right)^T &= \left(\frac{\partial f(\boldsymbol{u}_n, \boldsymbol{\theta}, t_n)}{\partial \boldsymbol{u}}\right)^T + \mathcal{H}\left(\boldsymbol{u}_n + \epsilon(\boldsymbol{u}_{n+1} - \boldsymbol{u}_n)\right)(\boldsymbol{u}_{n+1} - \boldsymbol{u}_n) \\ &= \left(\frac{\partial f(\boldsymbol{u}_n, \boldsymbol{\theta}, t_n)}{\partial \boldsymbol{u}}\right)^T + \mathcal{H}\left(\boldsymbol{u}_n + \epsilon(\boldsymbol{u}_{n+1} - \boldsymbol{u}_n)\right)(hf(\boldsymbol{u}_n)), \end{aligned} \tag{6}$$

where $\mathcal{H}$ is the Hessian of f and ϵ is a constant in $(0, 1)$.

Plugging (6) into (5), we obtain (1) and complete the proof. □

C GRADIENT CALCULATION USING DISCRETE ADJOINT METHODS

C.1 DERIVING THE DISCRETE ADJOINT FOR A TIME-STEPPING ALGORITHM

Let us define the Lagrange multipliers $\boldsymbol{\lambda}_n \in \mathbb{R}^{N_d}, n = 0, \ldots, N$ and define the Lagrangian

$$\mathcal{L}(\boldsymbol{\eta}) = \phi(\boldsymbol{u}_N) - \boldsymbol{\lambda}_0^T(\boldsymbol{u}_0 - \boldsymbol{\eta}) - \sum_{n=0}^{N-1} \boldsymbol{\lambda}_{n+1}^T\left(\boldsymbol{u}_{n+1} - \boldsymbol{\mathcal{N}}(\boldsymbol{u}_n)\right), \tag{7}$$

which depends on the initial state $\boldsymbol{\eta}$ (input data) of an ODE system. Taking the total derivative of Equation (7) with respect to $\boldsymbol{\eta}$ leads to

$$\frac{d\mathcal{L}}{d\boldsymbol{\eta}} = \boldsymbol{\lambda}_0^T - \left(\frac{d\phi}{d\boldsymbol{u}}(\boldsymbol{u}_N) - \boldsymbol{\lambda}_N^T\right)\frac{d\boldsymbol{u}_N}{d\boldsymbol{\eta}} - \sum_{n=0}^{N-1}\left(\boldsymbol{\lambda}_n^T - \boldsymbol{\lambda}_{n+1}^T \frac{d\boldsymbol{\mathcal{N}}}{d\boldsymbol{u}}(\boldsymbol{u}_n)\right)\frac{d\boldsymbol{u}_n}{d\boldsymbol{\eta}}. \tag{8}$$

The discrete adjoint equation is defined as

$$\begin{aligned}\boldsymbol{\lambda}_n &= \left(\frac{d\boldsymbol{\mathcal{N}}}{d\boldsymbol{u}}(\boldsymbol{u}_n)\right)^T \boldsymbol{\lambda}_{n+1}, \quad n = N-1, \ldots, 0, \\ \boldsymbol{\lambda}_N &= \left(\frac{d\phi}{d\boldsymbol{u}}(\boldsymbol{u}_N)\right)^T,\end{aligned} \tag{9}$$

in order to make the last two terms in (8) vanish so that the total derivative can be obtained without computing the forward sensitivities $\frac{d\boldsymbol{u}_n}{d\boldsymbol{\eta}}$.

C.2 Deriving the gradient to parameters and incorporating integrals

We consider a system of ODEs in the general form

$$\boldsymbol{\mathcal{M}}\dot{\boldsymbol{u}} = f(t, \boldsymbol{\theta}, \boldsymbol{u}),\ t \in [t_0, t_F], \tag{10}$$

where $\boldsymbol{\mathcal{M}}$ is a mass matrix. For explicit ODEs, $\boldsymbol{\mathcal{M}}$ is an identity matrix $\boldsymbol{\mathcal{I}}$ of size $N_d \times N_d$.

The loss function with an integral term can be written as

$$\mathcal{L} = \phi(\boldsymbol{u}(t_F)) + \int_{t_0}^{t_F} q(\boldsymbol{u}(t), t)dt. \tag{11}$$

To obtain the gradient of $\mathcal{L}$ to parameters (weights of the neural network), we can extend the ODE system (10) by augmenting the state vector with the parameters and the integrand q, and we obtain a larger system,

$$\underline{\boldsymbol{\mathcal{M}}}\underline{\dot{\boldsymbol{u}}} = \underline{F}(t, \underline{\boldsymbol{\theta}}, \underline{\boldsymbol{u}}),\ t \in [t_0, t_F], \tag{12}$$

where

$$\underline{\boldsymbol{\mathcal{M}}} = \begin{bmatrix} \boldsymbol{\mathcal{M}} & & \\ & \boldsymbol{I}_{N_p \times N_p} & \\ & & 1 \end{bmatrix}, \underline{\boldsymbol{u}} = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\theta} \\ q \end{bmatrix}, \underline{F} = \begin{bmatrix} f \\ \boldsymbol{0}_{N_p \times 1} \\ r \end{bmatrix}.$$

The second equation indicates that the parameters are constant, and the last equation results from the following transformation of the integral:

$$r = \int_{t_0}^{t_F} q(\boldsymbol{u}(t), t)dt. \tag{13}$$

The initial condition for the augmented ODE system is $\underline{\boldsymbol{\eta}}_0 = [\boldsymbol{\eta}\ \boldsymbol{\theta}\ 0]^T$.

For notional brevity, we denote the Jacobian with respect to $\boldsymbol{u}$ and $\boldsymbol{\theta}$ as $\boldsymbol{f_u}$ and $\boldsymbol{f_\theta}$, respectively. Then the extended Jacobian can be written as

$$\underline{F}_{\underline{\boldsymbol{u}}} = \begin{bmatrix} \boldsymbol{f_u} & \boldsymbol{f_\theta} & \boldsymbol{0}_{N_d \times 1} \\ \boldsymbol{0}_{N_p \times N_d} & \boldsymbol{0}_{N_p \times N_p} & \boldsymbol{0}_{N_p \times 1} \\ r_{\boldsymbol{u}} & r_{\boldsymbol{\theta}} & 0 \end{bmatrix}.$$

Now the adjoint variable expands to the combination of three variables, corresponding to the partial derivative of the loss function with respect to the initial system state, the parameters, and the initial value of q, respectively. In the later part of this document, we will use $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ to denote the first and second adjoint variables. The terminal conditions for these two variables are

$$\boldsymbol{\lambda}_N = \left(\frac{\partial\phi}{\partial\boldsymbol{u}}(\boldsymbol{u}_n)\right)^T, \quad \boldsymbol{\mu}_N = \left(\frac{\partial\phi}{\partial\boldsymbol{\theta}}(\boldsymbol{u}_n)\right)^T. \tag{14}$$

The third adjoint variable has a constant value 1 because of the zeros in the last column of $\underline{F}_{\underline{\boldsymbol{u}}}$ and thus does not need to be computed.

C.3 Discrete adjoints of exemplar timestepping methods

This section lists the discrete adjoints for three families of timestepping methods that are widely used. Note that these discrete adjoints are derived and implemented for only selective timestepping methods in `PETSc`, which is also a limitation in this work. Nevertheless, the discrete adjoint methods can be easily extended to other timestepping methods in `PETSc`.

C.3.1 BACKWARD EULER

Forward propagation

$$\boldsymbol{\mathcal{M}}\boldsymbol{u}_{n+1} = \boldsymbol{\mathcal{M}}\boldsymbol{u}_n + h_n\theta\boldsymbol{f}(\boldsymbol{u}_{n+1}) \tag{15}$$

Adjoint propagation

$$\begin{aligned}
\boldsymbol{\mathcal{M}}^T\boldsymbol{\lambda}_s &= \boldsymbol{\lambda}_{n+1} + h_n\,\boldsymbol{f}_{\boldsymbol{u}}^T(\boldsymbol{u}_{n+1})\,\boldsymbol{\lambda}_s + h_n\,r_{\boldsymbol{u}}^T(\boldsymbol{u}_{n+1}),\\
\boldsymbol{\lambda}_n &= \boldsymbol{\lambda}_{n+1} + h_n\,\boldsymbol{f}_{\boldsymbol{u}}^T(\boldsymbol{u}_{n+1})\,\boldsymbol{\lambda}_s + h_n\,r_{\boldsymbol{u}}^T(\boldsymbol{u}_{n+1}),\\
\boldsymbol{\mu}_n &= \boldsymbol{\mu}_{n+1} + h_n\,\boldsymbol{f}_{\boldsymbol{\theta}}^T(\boldsymbol{u}_{n+1})\,\boldsymbol{\lambda}_s + h_n\,r_{\boldsymbol{\theta}}^T(\boldsymbol{u}_{n+1})
\end{aligned}$$

C.3.2 THETA METHODS

Forward propagation

$$\boldsymbol{\mathcal{M}}\boldsymbol{u}_{n+1} = \boldsymbol{\mathcal{M}}\boldsymbol{u}_n + h_n(1-\theta)\boldsymbol{f}(\boldsymbol{u}_n) + h_n\theta\boldsymbol{f}(\boldsymbol{u}_{n+1}) \tag{16}$$

Adjoint propagation

$$\begin{aligned}
\boldsymbol{\mathcal{M}}^T\boldsymbol{\lambda}_s &= \boldsymbol{\lambda}_{n+1} + h_n\theta\,\boldsymbol{f}_{\boldsymbol{u}}^T(\boldsymbol{u}_{n+1})\boldsymbol{\lambda}_s + h_n\theta\,r_{\boldsymbol{u}}^T(t_{n+1},\boldsymbol{u}_{n+1}),\\
\boldsymbol{\lambda}_n &= \boldsymbol{\mathcal{M}}^T\boldsymbol{\lambda}_s + h_n(1-\theta)\boldsymbol{f}_{\boldsymbol{u}}^T(\boldsymbol{u}_n)\boldsymbol{\lambda}_s + h_n(1-\theta)r_{\boldsymbol{u}}^T(t_n,\boldsymbol{u}_n),\\
\boldsymbol{\mu}_n &= \boldsymbol{\mu}_{n+1} + h_n\theta\left(\boldsymbol{f}_{\boldsymbol{p}}^T(\boldsymbol{u}_{n+1})\boldsymbol{\lambda}_s + r_p^T(\boldsymbol{u}_{n+1})\right) + h_n(1-\theta)\left(\boldsymbol{f}_{\boldsymbol{p}}^T(\boldsymbol{u}_n)\boldsymbol{\lambda}_s + r_{\boldsymbol{p}}^T(\boldsymbol{u}_n)\right)
\end{aligned} \tag{17}$$

C.3.3 EXPLICIT RUNGE–KUTTA METHODS

Forward propagation

$$\begin{aligned}
\boldsymbol{U}_i &= \boldsymbol{u}_n + \sum_{j=1}^{i-1} h_n\,a_{ij}\,F(\boldsymbol{U}_j), \quad i = 1,\cdots,s,\\
\boldsymbol{u}_{n+1} &= \boldsymbol{u}_n + \sum_{i=1}^{s} h_n\,b_i\,F(\boldsymbol{U}_i)
\end{aligned} \tag{18}$$

Adjoint propagation

$$\begin{aligned}
\boldsymbol{\lambda}_{s,i} &= h_n\boldsymbol{f}_{\boldsymbol{u}}^T(\boldsymbol{U}_i)\left(b_i\boldsymbol{\lambda}_{n+1} + \sum_{j=i+1}^{s} a_{ji}\,\boldsymbol{\lambda}_{s,j}\right) + h_n\,b_i\,r_{\boldsymbol{u}}^T(\boldsymbol{U}_i)), \quad i = s,\cdots,1,\\
\boldsymbol{\mu}_{s,i} &= h_n\boldsymbol{f}_{\boldsymbol{p}}^T(\boldsymbol{U}_i)\left(b_i\boldsymbol{\lambda}_{n+1} + \sum_{j=i+1}^{s} a_{ji}\,\boldsymbol{\lambda}_{s,j}\right) + h_n\,b_i\,r_{\boldsymbol{p}}^T(\boldsymbol{U}_i), \quad i = s,\cdots,1,\\
\boldsymbol{\lambda}_n &= \boldsymbol{\lambda}_{n+1} + \sum_{j=1}^{s}\boldsymbol{\lambda}_{s,j},\\
\boldsymbol{\mu}_N &= \boldsymbol{\mu}_{n+1} + \sum_{j=1}^{s}\boldsymbol{\mu}_{s,j}
\end{aligned} \tag{19}$$

C.4 CHECKPOINTING

`PETSc` offers transparent and optimal checkpointing strategies on high-performance computing platforms. All of the strategies are accessible through our PNODE framework. In addition to the classic `Revolve` algorithm Griewank & Walther (2000), `PETSc` provides extended algorithms Zhang & Constantinescu (2021) that are tailored for multistage time integration methods. Figure 1 illustrates an optimal schedule for the adjoint calculation given a memory budget for storing 3 checkpoints, with each checkpoint consisting of a solution vector (numbered circles) and the stage values (black dots). This schedule requires a minimal number (6) of extra recomputations in order to reverse 10 time steps. `PETSc` also supports many other checkpointing algorithms, such as online algorithms Stumm & Walther (2010); Wang et al. (2009), multistage algorithms Stumm & Walther (2009), and multilevel algorithms Schanen et al. (2016) for heterogeneous platforms.

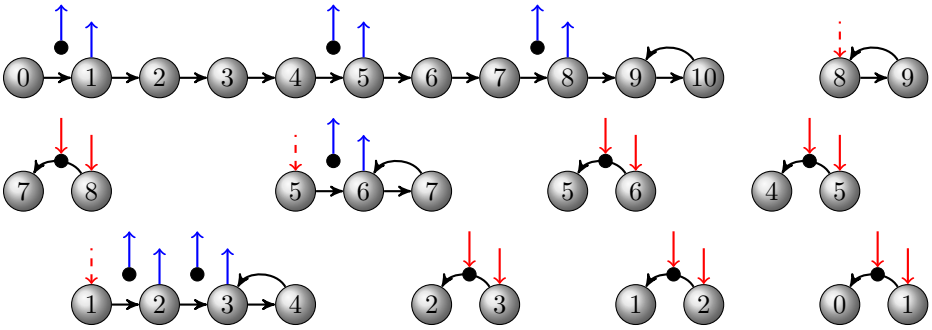


Figure 1: Modified `Revolve` algorithm for multistage time integration methods. Reprinted from Zhang & Constantinescu (2021). A numbered circle stands for the solution at the corresponding time index. A block dot stands for the stage values associated with the time step. The up arrow and down arrow stand for "store" operation and "restore" operation, respectively. When a stack is used for holding the checkpoints, the arrows with solid lines correspond to push and pop operations. The down arrow with dashed line indicates to read the top element on the stack without removing it.

D Details For Numerical Examples

D.1 Image classification

We modify a SqueezeNext network into a neural ODE by replacing all nonactivation blocks with ODE blocks. Our experiment is based on model files from the ANODE Gholaminejad et al. (2019) code repository [1] . All models are trained with the SGD optimizer for 200 epochs, with an initial learning rate 0.1 decayed by a factor of 10 at the 150th epoch.

To further demonstrate the stability and accuracy of PNODE, we train SqueezeNext models using one-step forward Euler and RK4 for 3 independent runs, and we report the mean and standard deviation (STD) across the runs. As shown in Table 1, training with the discrete adjoint reaches lower training loss and higher test accuracy when compared with the neural ODE implementation using the continuous adjoint method (NODE cont).

Training loss				
	NODE disc	ANODE	NODE cont	PNODE
Euler	0.0006 $\pm$ 0.0000	0.0008$\pm$ 0.0002	0.0030 $\pm$ 0.0002	0.0006$\pm$ 0.0001
RK4	0.0005 $\pm$ 0.0000	0.0006 $\pm$ 0.0001	0.0006 $\pm$ 0.0000	0.0005$\pm$ 0.0001
Test accuracy				
	NODE disc	ANODE	NODE cont	PNODE
Euler	0.8830 $\pm$ 0.0085	0.8740$\pm$0.0157	0.5303 $\pm$ 0.0702	0.8780 $\pm$ 0.0036
RK4	0.8803 $\pm$ 0.0032	0.8850 $\pm$ 0.0050	0.8780 $\pm$ 0.0010	0.8803 $\pm$ 0.0068

Table 1: Mean and STD of training loss and test accuracy on classification on CIFAR-10 dataset after 200 epochs of training across 3 runs with different random seeds.

D.2 Continuous normalizing flow

We apply PNODE on FFJORD, which is a free-from continuous generative model, and we follow the architecture and hyperparameter settings chosen by the authors in Grathwohl et al. (2019). The details are given in Table 2.

[1] https://github.com/amirgholami/anode

Dataset	Dimension	Activation	Number of layers in NN	Hidden dim. multiplier	Number of ODE blocks	Batchsize
POWER	6	tanh	3	10	5	10000
BSDS300	63	softplus	3	20	2	10000 (1000*)
MINIBOONE	43	softplus	2	10	1	1000

Table 2: Information of hyperparameters chosen by Grathwohl et al. (2019). *The batch size is reduced to 1000 when using the discrete adjoint, so the memory can fit in the GPU.

D.3 TIME SERIES REGRESSION

We use the same model as in Rackauckas et al. (2020) and Onken & Ruthotto (2020):

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = A\begin{bmatrix} x_1^3 \\ x_2^3 \end{bmatrix}, \quad A = \begin{bmatrix} -0.10 & 2.00 \\ -2.00 & -0.10 \end{bmatrix},$$

with an initial condition $\mathbf{x} = [2, 0]$. The data are generated on 160 equidistant time grid points in $[0, 16]$ by integrating the ODE using an adaptive dopri5 solver. For the training model, we use a single linear layer with 2D input and output applied to the elementwise cubic of x; therefore, the true underlying model can be represented exactly. We train the model on $t \in [0, 1]$ with a step size of 0.1. In each iteration, we use an equal batch size of 20. The loss function is defined as the average of the absolute values of the residuals. The parameters in the model are updated through an RMSprop minimizer with a learning rate of 0.01. The test loss is then evaluated on all the samples.

Here we provide the comparison in terms of loss functions using two second-order schemes—explicit midpoint and Crank–Nicolson (implicit)—using three different random seeds, as shown in Figure 2. The solution using explicit solver blows up during tests for all three runs, while training with implicit solvers always reaches a lower training loss and the solutions always remain finite during testing.

REFERENCES

Amir Gholaminejad, Kurt Keutzer, and George Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 730–736. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/103.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.

Andreas Griewank and Andrea Walther. Algorithm 799: Revolve: An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, March 2000. ISSN 0098-3500. doi: 10.1145/347837.347846.

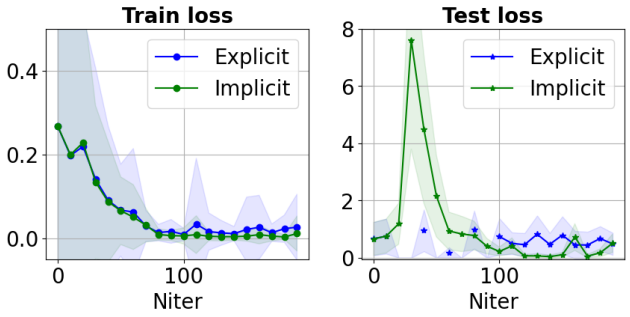
Derek Onken and Lars Ruthotto. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv e-preprints*, 2020.

Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

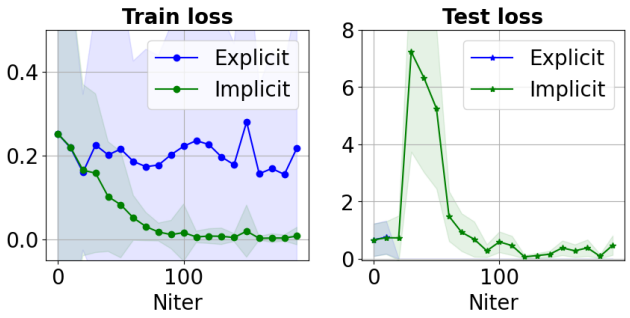
Michel Schanen, Oana Marin, Hong Zhang, and Mihai Anitescu. Asynchronous two-level checkpointing scheme for large-scale adjoints in the spectral-element solver Nek5000. *Procedia Computer Science*, 80: 1147–1158, 2016. ISSN 18770509. doi: 10.1016/j.procs.2016.05.444.

Philipp Stumm and Andrea Walther. MultiStage Approaches for Optimal Offline Checkpointing. *SIAM Journal on Scientific Computing*, 31(3):1946–1967, 2009. ISSN 1064-8275. doi: 10.1137/080718036.

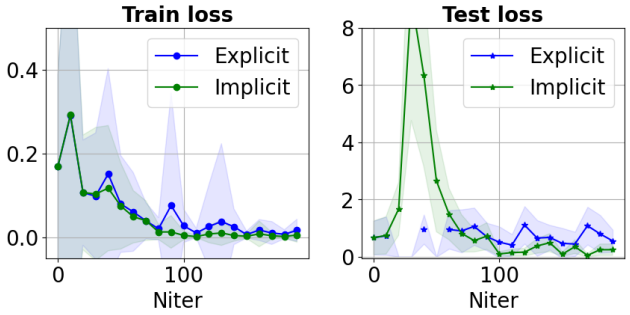
Philipp Stumm and Andrea Walther. New algorithms for optimal online checkpointing. *SIAM Journal on Scientific Computing*, 32(2):836–854, 2010. ISSN 1064-8275. doi: 10.1137/080742439.



(a) Random seed 1



(b) Random seed 2



(c) Random seed 3

Figure 2: Train and test loss with explicit midpoint and Crank–Nicolson (implicit) solvers as functions of number of iterations using different random seeds.

Qiqi Wang, Parviz Moin, and Gianluca Iaccarino. Minimal repetition dynamic checkpointing algorithm for unsteady adjoint calculation. *SIAM Journal on Scientific Computing*, 31(4):2549–2567, 2009. ISSN 1064-8275. doi: 10.1137/080727890.

Hong Zhang and Emil M. Constantinescu. Revolve-based adjoint checkpointing for multistage time integration. In *International Conference on Computational Science*, (online), in main track, 2021.