Peppy: An AI-Assisted Workflow for Tight Convergence Analysis of Optimization Algorithms

Jaewook J. Suh [*1] **TaeHo Yoon** [*2] **Edward D. H. Nguyen** [3] **Bicheng Ying** [4] **Shiqian Ma** [1]

Abstract

This paper presents Peppy, an AI-assisted workflow for discovering tight, analytic convergence proofs for optimization algorithms. While existing automated tools can generate numerical convergence bounds for first-order methods, translating these numerical certificates into analytic Lyapunov proofs independent of iteration number remains a complex manual task. Peppy bridges this gap by combining structured, verifiable blocks with AI agents. Case studies on first-order methods demonstrate its ability to support a rigorous, practical, and reproducible paradigm for AI-assisted theorem synthesis in optimization.

1. Introduction

The velocity of AI's advancement, especially in large language models (LLMs), has opened many new avenues in how the discipline of research should be conducted (Brown et al., 2020; Bommasani et al., 2021; Wang et al., 2024). Approaches vary dramatically in terms of scope and granularity. For example, significant efforts have been allocated to developing completely autonomous "AI scientists" which aim to automate the entire scientific process including literature review, novel idea formulation, and experimental verification (Lu et al., 2026; Schmidgall et al., 2025). "AI4Math" is another similarly ambitious direction which uses LLMs and formal methods to enable AI agents to autonomously prove theorems (Yang et al., 2025; Wu et al., 2022; Yang et al., 2023; Trinh et al., 2024). These directions, while promising, can be difficult to adopt in everyday research practice, as they may require substantial computational resources, specialized infrastructure, or access to proprietary models (Yang et al., 2023). Furthermore, because the scope of these AI systems is so broad, they may lack key domain knowledge and consistency in their outputs (Lu et al., 2026; Jansen et al., 2025).

This paper presents a complementary, more focused direction of leveraging AI in optimization theory. We propose an AI-facilitated workflow Peppy for searching convergence proofs based on the Performance Estimation Problem (PEP) (Drori & Teboulle, 2014; Taylor et al., 2017b) framework. PEP is often viewed as a computer-assisted proof framework, as it provides numerical evidence and guidance for constructing convergence analyses of first-order optimization methods. It has been proved useful both for finding tight convergence rates and designing novel optimization algorithms (Kim & Fessler, 2016; Taylor et al., 2018; Taylor & Bach, 2019; Ryu et al., 2020; Kim, 2021; Dragomir et al., 2022; Taylor & Drori, 2023; Das Gupta et al., 2024; Upadhyaya et al., 2024).

Research process utilizing PEP typically combines generation of numerical certificates with analytic proof construction. Existing packages support the computational side by helping users formulate PEPs, call numerical solvers, and extract certificates (Taylor et al., 2017c; Goujaud et al., 2024; Upadhyaya et al., 2025). However, converting these numerical certificates into analytic proofs has remained a largely manual art that requires significant effort from human experts.

Peppy combines guided AI assistance with the `PEPFlow` library to simulate steps that were traditionally performed by humans, where `PEPFlow` (Suh et al., 2025) provides the backend for implementing and solving PEPs while exposing the information needed for analytic extensions. As a result, for an input optimization algorithm, Peppy can output a rigorous *theorem* stating its convergence rate with its proof, with varying degrees of human guidance.

[*]Equal contribution [1]Department of Computational Applied Mathematics and Operations Research, Rice University [2]Department of Applied Mathematics and Statistics, Johns Hopkins University [3]Department of Electrical and Computer Engineering, Rice University [4]Google Inc.. Correspondence to: Jaewook J. Suh <jaewookjacksuh@gmail.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026.
Copyright 2026 by the author(s).

Our workflow has several advantages over existing approaches in "AI4Math" which typically combine LLMs with formal verification. We list them as follows:

- Peppy provides an effectively tailored approach toward AI for optimization, powered by domain-specific machinery, PEP. At the same time, all PEP-related components are black-boxed via `PEPFlow` implementation that AI agent can work with.[1] Hence, the user only needs to understand how Peppy operates at a high level, as outlined in Section 2.
- While the final output of Peppy is a verifiable convergence proof providing a concrete worst-case rate, it does not require any costly Lean-style formalization. This is because the PEP formulation itself is naturally verification-friendly: it allows us to rapidly check the numerical feasibility of a candidate certificate, and the final analytic proof can be verified through symbolic matrix identities. This verification step is embedded into Peppy.
- The proof output by Peppy is optimized, i.e., its certified bound cannot be improved within the chosen proof system. This is because PEP, by its formulation, detects numerical certificates corresponding to a convergence proof that is optimal among all proofs constructible from the specified set of inequalities.

2. Peppy: Design principle with example

The goal of Peppy is to produce a convergence theorem with its proof based on Lyapunov analysis. Given a user input of the form described in Section 2.1, a successful run produces a theorem and its proof as in Section 2.5, with varying degrees of human intervention. Below, we outline the theoretical design principle of each step of the workflow via a representative example of gradient descent (GD) for smooth convex minimization.

2.1. PEP implementation of input algorithm

In the first stage of Peppy, the user feeds in the problem class, algorithm's update rule, performance metric and initial condition into the coding agent. Input format is flexible, as agents can handle various inputs; we provide a standardized example format in Section 3.1. Once the agent understands all this information, it writes a `PEPFlow` code for setting up the PEP construction, which can be numerically solved in the next stage.[2]

[1] The code is available at: https://github.com/pepflow-lib/PEPFlow/tree/peppy-workshop-v1/examples_peppy

[2] The characterization of the function class by inequality (1) is established in (Taylor et al., 2017a).

Setup: f is L-smooth convex function, characterized by

$$I(x,y) := f(y) - f(x) + \langle \nabla f(y), x-y\rangle + \frac{1}{2L}\|\nabla f(x) - \nabla f(y)\|^2 \le 0, \quad \forall x, y. \tag{1}$$

Performance metric: $f(x_N) - f(x_\star)$
Initial condition: $\|x_0 - x_\star\| \le R$
Algorithm definition:

$$x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k).$$

2.2. Solving PEP and determining certificates

Given the PEP implementation, the second stage actually builds and solves the PEP problem to obtain numerical certificates, which provide guidance toward the analytical convergence proof. The PEP certificates obtained here can be understood as the numerical version of the following generic identity:

$$\texttt{(Performance metric)} - \tau_N \times \texttt{(Initial condition)} = \texttt{(Weighted sum of } I(\cdot,\cdot)\texttt{'s)} - \texttt{(Sum of squares)}.$$

Specifically for GD, we can write

$$f(x_N) - f(x_\star) - \frac{L}{4N+2}\|x_0 - x_\star\|^2 = \sum_{i=1}^{N} \lambda_{i-1,i} I(x_{i-1}, x_i) + \sum_{i=0}^{N} \lambda_{\star,i} I(x_\star, x_i) - \sum_{i=0}^{N} d_i \|s_i\|^2 \tag{2}$$

If the above identity is analytically determined, with forms of PEP certificates $\lambda_{i,i+1}, \lambda_{\star,i}, d_i \ge 0$ and s_i clarified, then from (1) we conclude that $f(x_N) - f(x_\star) \le \frac{L}{4N+2}\|x_0 - x_\star\|^2$. However, the PEP certificates are only numerical and searching their analytical form traditionally required human endeavor (Taylor et al., 2017a; Kim & Fessler, 2016). In Peppy, this is often successfully carried out by the coding agent. On the other hand, even if (2) is established analytically, it holds only for a fixed horizon N specified in the beginning and does not guarantee an analytical proof that holds for arbitrary N. To build an extendable proof, we employ the Lyapunov-style proof approach outlined in the subsequent sections.

2.3. Defining Lyapunov function

In this stage, we decompose the PEP proof (2) into partial sums V_k for $k = 1, \dots, N$, as:

$$V_k := \sum_{i=1}^{k} \lambda_{i-1,i} I(x_{i-1}, x_i) + \sum_{i=0}^{k} \lambda_{\star,i} I(x_\star, x_i) - \sum_{i=0}^{k} d_i \|s_i\|^2$$

By definition, due to (1), V_k is a nonincreasing *Lyapunov function*. This construction and the subsequent

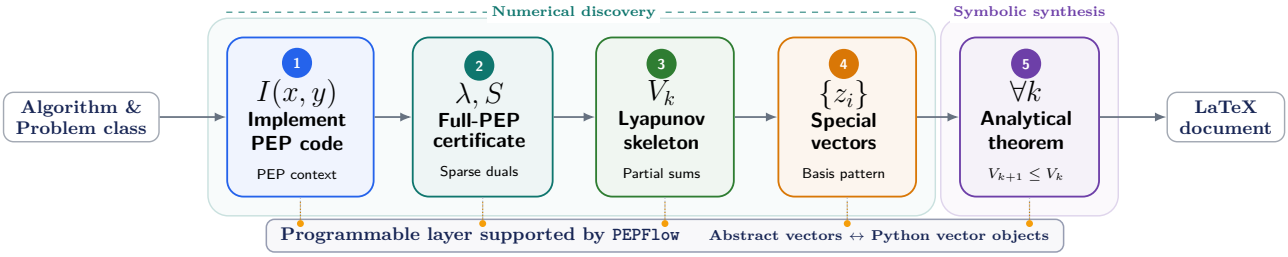

Figure 1. A schematic description of Peppy's 5-stage workflow.

steps of Peppy follows Yoon et al. (2026), which provided a systematic procedure for utilizing the Lyapunov function construction of the above form and deriving a concise and interpretable analytic proof from it. Notably, Goujaud et al. (2023) presented a similar idea of using partial sum of nonpositive terms as a Lyapunov function, while Yoon et al. (2026) can be viewed as an extension and completion of the theory surrounding it.

2.4. Finding a summation-free expression for V_k

In this stage, we systematically simplify V_k into a summation-free form. This is done via nontrivial linear algebraic tricks[3] that we have implemented under the hood. These core functions are packaged into the new `lyapunov_utils.py` component, built upon the `PEPFlow` library. Utilizing them, we can extract the basis vectors that can express V_k as a simple (low-rank) quadratic form. This is the most novel component introduced by Peppy, which enables the conversion of (2) into an analytical, N-independent proof. Applying this step to GD, in particular, we obtain:

$$V_k = \frac{k+1}{2N-k}\big(f(x_k) - f(x_\star)\big) - \frac{L}{4N+2}\|x_0 - x_\star\|^2 - \frac{k+1}{2L(2N-k)}\|\nabla f(x_k)\|^2 + \frac{L(2N-2k-1)}{2(2N-k)^2}\|x_{k+1} - x_\star\|^2$$

Note that this stage, on its own, used to be a research-level task requiring human experts, as done in prior work (Park et al., 2023; d'Aspremont et al., 2021; Lee et al., 2021).

2.5. Analytic proof by Lyapunov analysis

In this final step, we verify $V_{k+1} \leq V_k$ through symbolic calculation. In a successful use case, we let the Peppy to additionally produce LaTeX code for the corresponding proof and the resulting theorem statement, as shown below. Examples of proofs generated by Peppy are provided in Section 4.

[3]We provide a brief explanation in Section A.

Theorem. Let f be convex and L-smooth, and let $x_\star$ minimize f. If $\|x_0 - x_\star\|^2 \leq R$, then the gradient descent iterates

$$x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k)$$

satisfy the following stronger Lyapunov statement. Set $V_0 = 0$ and, for $1 \leq k \leq N-1$,

$$V_k = \frac{k+1}{2N-k}\big(f(x_k) - f(x_\star)\big) - \frac{L}{4N+2}\|x_0 - x_\star\|^2 - \frac{k+1}{2L(2N-k)}\|\nabla f(x_k)\|^2 + \frac{L(2N-2k-1)}{2(2N-k)^2}\|x_{k+1} - x_\star\|^2,$$

and $V_N = f(x_N) - f(x_\star) - \frac{L}{4N+2}\|x_0 - x_\star\|^2$. Then $V_0 \geq V_1 \geq \cdots \geq V_N$. Consequently, for every $N \geq 1$,

$$f(x_N) - f(x_\star) \leq \frac{LR}{4N+2}.$$

3. Peppy: Agentic implementation

Peppy is implemented as a sequence of reusable commands, each written in a Markdown file. Here, a *command* is a prompt specification for executing each modular block of theoretical workflow described in Section 2.

The user feeds Peppy with a prompt including the command name and the required input arguments. Below, we briefly outline the function of each command at a high level, deferring detailed explanation to Section B. See Section 3.1 for a concrete use case example.

(1) **`/pep-implement`:** Turns the algorithm description into an executable `PEPFlow` setup module.

(2) **`/pep-full-proof`:** Solves PEP with a fixed N and extracts proof certificates, e.g., λ, d, and s in (2), and guesses their closed-form expressions.

(3) **`/lyap-define`:** Constructs a candidate Lyapunov function V_k as partial sums of weighted inequalities and square terms from the PEP proof.

(4) **`/lyap-vectors`:** Identifies the vectors that are used for expressing V_k, and thereby simplifies it into a summation-free expression.

(5) **`/lyap-closed-form`:** Finalizes the analytic coefficients in V_k, symbolically verifies the Lyapunov analysis, and drafts a theorem with proof outline.

The role of AI assistance in Peppy is two-fold:

- AI generates `PEPFlow`-based codes for structured tasks such as solving PEPs, constructing Lyapunov groupings, and searching for basis vectors.
- AI assists in automating tasks that require mathematical insight, including selecting sparsity patterns, guessing closed-form rates and symbolic formulas, and analytically identifying V_k.

  Here, to prevent proceeding with hallucinated outputs, Peppy performs specific numerical or symbolic verification steps on AI agent's outputs.

At the end of each block, the outputs are saved into a `.json` state file, and a human-readable summary is written into a Jupyter notebook, which enables both Peppy and the user to keep track of intermediate results. The user may inspect the progress and provide additional guidance to Peppy when needed.

3.1. How to use: Prompt example

While the input prompts can be flexible, we provide a standardized example format that we found successful, for the case of GD for smooth convex minimization. We start by running the first command, `/pep-implement`, together with the required arguments.

```
/pep-implement

Function: f is convex and L-smooth
Parameters: L, R
Initial condition: ||x_0 - x_star|| <= R, where
     grad f(x_star) = 0
Performance metric: f(x_N) - f(x_star)
Algorithm: Gradient descent with fixed step
    size 1/L

For k >= 0:
x_{k+1} = x_k - (1/L) * grad f(x_k)

Conjectured rate: unknown
```

Alternatively, one can choose to input a screenshot of the algorithm's information, e.g., the box in Section 2.1. This also worked in our experimental tests.

In successful cases, `/pep-implement` produces an output that can be used as an input to the next block, `/pep-full-proof`. This can be triggered by prompting "command name + algorithm name." For example,

```
/pep-full-proof gd
```

will let Peppy proceed to the next stage. A simpler alternative, which we used in our experiments, is:

```
proceed with the next step
```

Then repeat this until the final block is reached.

4. Experiments

We let Peppy analyze PGM, OGM (Kim & Fessler, 2016), BPPM, FEG (Lee & Kim, 2021), and Dual-FEG (Yoon et al., 2024). In our experiments, we used OpenAI Codex (GPT-5, medium effort) as AI assistant. We briefly present the result for PGM, and defer the details and results for other algorithms to Section C.

4.1. Proximal gradient descent method (PGM)

Let f be convex and L-smooth, let g be closed proper convex, and let $x_\star$ minimize $h = f + g$. Consider proximal gradient descent method:

$$x_{k+1} = \mathrm{prox}_{g/L}\left(x_k - L^{-1}\nabla f(x_k)\right).$$

We now apply our workflow to obtain a new Lyapunov analysis. We use similar input prompts as in Section 3.1; see Section C.1 for the full prompting details. For this example, we provided Peppy an explicit decomposition of sum-of-squares terms. For all the other examples in Section C, no human intervention was involved.

The full run took approximately 30 minutes across the five blocks:

Block 1	Block 2	Block 3	Block 4	Block 5
6m 43s	5m 42s	3m 15s	6m 17s	9m 45s

We obtained a Lyapunov function: for $k = 1, \ldots, N-1$,

$$\begin{aligned} V_k = &\frac{k+1}{2N-k}\left(f(x_k) - f(x_\star)\right) + \frac{k}{2N-k}\left(g(x_k) - g(x_\star)\right) \\ &- \frac{L}{4N}\|x_0 - x_\star\|^2 + \frac{L(N-k)}{(2N-k)^2}\|x_k - x_\star\|^2 \\ &- \frac{1}{2N-k}\langle x_k - x_\star, \nabla f(x_k)\rangle \\ &+ \frac{1}{2L(2N-k)}\|\nabla f(x_k) - \nabla f(x_\star)\|^2. \end{aligned}$$

We provide the full output in Section C.1. The output provides a new Lyapunov analysis that recovers the convergence rate $h(x_N) - h(x_\star) \le \frac{L}{4N}\|x_0 - x_\star\|^2$ from (Teboulle & Vaisbourd, 2023).

References

Auslender, A. and Teboulle, M. Interior Gradient and Proximal Methods for Convex and Conic Optimization. *SIAM Journal on Optimization*, 16(3):697–725, 2006. ISSN 1052-6234.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Neural Information Processing Systems*, 2020.

Das Gupta, S., Van Parys, B. P. G., and Ryu, E. K. Branch-and-bound performance estimation programming: A unified methodology for constructing optimal optimization methods. *Mathematical Programming*, 204:567–639, 2024.

d'Aspremont, A., Scieur, D., and Taylor, A. Acceleration methods. *Foundations and Trends® in Optimization*, 5(1-2):1–245, 2021.

Dragomir, R.-A., Taylor, A. B., d'Aspremont, A., and Bolte, J. Optimal complexity and certification of bregman first-order methods. *Mathematical Programming*, 194(1):41–83, 2022.

Drori, Y. and Teboulle, M. Performance of first-order methods for smooth convex minimization: A novel approach. *Mathematical Programming*, 145(1-2):451–482, 2014.

Goujaud, B., Dieuleveut, A., and Taylor, A. On fundamental proof structures in first-order optimization. In *Conference on Decision and Control*, pp. 3023–3030, 2023.

Goujaud, B., Moucer, C., Glineur, F., Hendrickx, J., Taylor, A., and Dieuleveut, A. PEPit: Computer-assisted worst-case analyses of first-order optimization methods in Python. *Mathematical Programming Computation*, 16:337–367, 2024.

Jansen, P., Tafjord, O., Radensky, M., Siangliulue, P., Hope, T., Dalvi Mishra, B., Majumder, B. P., Weld, D. S., and Clark, P. CodeScientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics*, 2025.

Kim, D. Accelerated proximal point method for maximally monotone operators. *Mathematical Programming*, 190(1–2):57–87, 2021.

Kim, D. and Fessler, J. A. Optimized first-order methods for smooth convex minimization. *Mathematical Programming*, 159(1-2):81–107, 2016.

Lee, J., Park, C., and Ryu, E. K. A geometric structure of acceleration and its role in making gradients small fast. In *Neural Information Processing Systems*, 2021.

Lee, S. and Kim, D. Fast extra gradient methods for smooth structured nonconvex-nonconcave minimax problems. *Neural Information Processing Systems*, 2021.

Lu, C., Lu, C., Lange, R. T., Yamada, Y., Hu, S., Foerster, J., Ha, D., and Clune, J. Towards end-to-end automation of AI research. *Nature*, 651:914–919, 2026.

Park, C., Park, J., and Ryu, E. K. Factor-$\sqrt{2}$ acceleration of accelerated gradient methods. *Applied Mathematics & Optimization*, 88(3):77, 2023.

Ryu, E. K., Taylor, A. B., Bergeling, C., and Giselsson, P. Operator splitting performance estimation: Tight contraction factors and optimal parameter selection. *SIAM Journal on Optimization*, 30(3):2251–2271, 2020.

Schmidgall, S., Su, Y., Wang, Z., Sun, X., Wu, J., Yu, X., Liu, J., Moor, M., Liu, Z., and Barsoum, E. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

Suh, J. J., Ying, B., Jiang, X., and Nguyen, E. D. H. PEPFlow: A python library for the workflow of performance estimation of optimization algorithms. In *NeurIPS Workshop on GPU-accelerated and Scalable Optimization*, 2025.

Taylor, A. and Bach, F. Stochastic first-order methods: Non-asymptotic and computer-aided analyses via potential functions. *Conference on Learning Theory*, 2019.

Taylor, A. and Drori, Y. An optimal gradient method for smooth strongly convex minimization. *Mathematical Programming*, 199(1-2):557–594, 2023.

Taylor, A., Hendrickx, J., and Glineur, F. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161:307–345, 2017a.

Taylor, A., Van Scoy, B., and Lessard, L. Lyapunov functions for first-order methods: Tight automated convergence guarantees. *International Conference on Machine Learning*, 2018.

Taylor, A. B., Hendrickx, J. M., and Glineur, F. Smooth strongly convex interpolation and exact worst-case performance of first-order methods. *Mathematical Programming*, 161(1-2):307–345, 2017b.

Taylor, A. B., Hendrickx, J. M., and Glineur, F. Performance estimation toolbox (PESTO): Automated worst-case analysis of first-order optimization methods. In *Conference on Decision and Control*, 2017c.

Teboulle, M. and Vaisbourd, Y. An elementary approach to tight worst case complexity analysis of gradient based methods. *Mathematical Programming*, 201(1):63–96, 2023.

Trinh, T. H., Wu, Y., Le, Q. V., He, H., and Luong, T. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. doi: 10.1038/s41586-023-06747-5.

Upadhyaya, M., Banert, S., Taylor, A. B., and Giselsson, P. Automated tight Lyapunov analysis for first-order methods. *Mathematical Programming*, 2024.

Upadhyaya, M., Taylor, A. B., Banert, S., and Giselsson, P. AutoLyap: A Python package for computer-assisted Lyapunov analyses for first-order methods. 2025.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen, J. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*, 2022.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. LeanDojo: Theorem proving with retrieval-augmented language models. In *Neural Information Processing Systems*, 2023.

Yang, K., Poesia, G., He, J., Li, W., Lauter, K. E., Chaudhuri, S., and Song, D. Position: Formal mathematical reasoning—a new frontier in ai. In *International Conference on Machine Learning*, 2025.

Yoon, T., Kim, J., Suh, J. J., and Ryu, E. K. Optimal Acceleration for Minimax and Fixed-Point Problems is Not Unique. *International Conference on Machine Learning*, 2024.

Yoon, T., Suh, J. J., Nguyen, E. D. H., Ying, B., and Ma, S. Toward a systematic understanding and interactive search of lyapunov-style proofs in optimization. *arXiv:2606.26077*, 2026.

A. Omitted details of Section 2.4

Here, we briefly explain Step 4, which is implemented under the hood in Peppy, using GD as a representative example. For a more detailed explanation, we refer the readers to (Yoon et al., 2026).

The PEP proof for gradient descent (GD) $x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k)$ for convex and L-smooth f is equivalent to establishing the *Lyapunov analysis* $V_{k+1} \le V_k$ for each $k = 0, \dots, N-1$, where we define

$$V_k = \sum_{i=1}^{k} \lambda_{i-1,i} I(x_{i-1}, x_i) + \sum_{i=0}^{k} \lambda_{\star,i} I(x_\star, x_i) - \sum_{i=0}^{k} d_i \|s_i\|^2 \tag{3}$$

where

$$\lambda_{i-1,i} = \frac{i}{2N+1-i}, \quad i = 1, \dots, N, \qquad \lambda_{\star,i} = \begin{cases} \lambda_{0,1} & i = 0 \\ \lambda_{i,i+1} - \lambda_{i-1,i} & i = 1, \dots, N-1 \\ 1 - \lambda_{i-1,i} & i = N, \end{cases}$$

$$\begin{aligned} d_i &= \frac{L}{2}\frac{4Ni + 2N - 2i^2 + 1}{(2N-i)^2}, \quad i = 0, \dots, N-1 \\ s_i &= \frac{1}{2N+1-i}(x_i - x_\star) - \frac{1}{L}\nabla f(x_i) \\ I(x_{i-1}, x_i) &= \left(f(x_i) - f(x_{i-1}) + \langle \nabla f(x_i), x_{i-1} - x_i \rangle + \frac{1}{2L}\|\nabla f(x_{i-1}) - \nabla f(x_i)\|^2 \right) \le 0 \\ I(x_\star, x_i) &= \left(f(x_i) - f(x_\star) + \langle \nabla f(x_i), x_\star - x_i \rangle + \frac{1}{2L}\|\nabla f(x_i)\|^2 \right) \le 0. \end{aligned}$$

Chaining the inequalities, we obtain $V_{N-1} \le V_1 \le 0$, and this would imply $f(x_N) - f(x_\star) \le \frac{LR}{4N+2}$. Stage 4 checks whether each V_k can be simplified into a summation-free formula written in terms of small number of vectors, not increasing with k. Direct computation, either numerical or symbolic, quickly reveals that the function value terms in (3) simplifies to

$$\lambda_{k,k+1}(f(x_k) - f(x_\star)) = \frac{k+1}{2N-k}(f(x_k) - f(x_\star)).$$

However, systematically checking that the sum of all inner product and squared norm terms can be simplified is a more involved process. Below, we outline how this is performed in our code `gd_recover_example_lyap.ipynb`.

Define the sum of only the inner product terms, excluding the function values, within (3) as

$$\begin{aligned} \widetilde{V}_k :&= \sum_{i=1}^{k} \lambda_{i-1,i} \left(\langle \nabla f(x_i), x_{i-1} - x_i \rangle + \frac{1}{2L}\|\nabla f(x_{i-1}) - \nabla f(x_i)\|^2 \right) \\ &+ \sum_{i=0}^{k} \lambda_{\star,i} \left(\langle \nabla f(x_i), x_\star - x_i \rangle + \frac{1}{2L}\|\nabla f(x_i)\|^2 \right) \\ &- \sum_{i=0}^{k} d_i \left\| \frac{1}{2N+1-i}(x_i - x_\star) - \frac{1}{L}\nabla f(x_i) \right\|^2. \end{aligned} \tag{4}$$

The Gram formulation used in PEP allows us to rewrite this quantity in terms of tangible matrices that we can handle with linear algebraic tricks. Without loss of generality, assume $x_\star = \mathbf{0}$. We let

$$\mathbf{P} = \begin{bmatrix} x_0 & \nabla f(x_0) & \cdots & \nabla f(x_N) \end{bmatrix} \in \mathbb{R}^{d \times (N+2)},$$

and define $\mathbf{x}_0 = \mathbf{e}_1$ and $\mathbf{g}_i = \mathbf{e}_{i+2}$ for $i = 0, \dots, N$ so that $\mathbf{P}\mathbf{x}_0 = x_0$ and $\mathbf{P}\mathbf{g}_i = \nabla f(x_i)$. Define $\mathbf{x}_1, \dots, \mathbf{x}_N$ by

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \frac{1}{L}\mathbf{g}_i = \mathbf{x}_0 - \frac{1}{L}\sum_{j=0}^{i} \mathbf{g}_j$$

for $i = 0, \dots, N-1$ and let $\mathbf{x}_\star = \mathbf{0}$, so that $\mathbf{P}\mathbf{x}_i = x_i$ for all $i \in \{0, \dots, N, \star\}$. Now consider the Gram matrix $\mathbf{G} = \mathbf{P}^\mathsf{T}\mathbf{P} \in \mathbb{S}_+^{N+2}$. Then we can express the inner product terms as, for example, for $i = 1, \dots, k$,

$$\langle \nabla f(x_i), x_{i-1} - x_i \rangle = \langle \mathbf{P}\mathbf{g}_i, \mathbf{P}(\mathbf{x}_{i-1} - \mathbf{x}_i) \rangle = \mathbf{g}_i^\mathsf{T}\mathbf{P}^\mathsf{T}\mathbf{P}(\mathbf{x}_{i-1} - \mathbf{x}_i) = \mathbf{g}_i^\mathsf{T}\mathbf{G}(\mathbf{x}_{i-1} - \mathbf{x}_i) = \operatorname{Tr}\left(\mathbf{G}\mathbf{g}_i(\mathbf{x}_{i-1} - \mathbf{x}_i)^\mathsf{T}\right).$$

The last expression can be symmetrized: $\operatorname{Tr}\left(\mathbf{G}\mathbf{g}_i(\mathbf{x}_{i-1} - \mathbf{x}_i)^\mathsf{T}\right) = \operatorname{Tr}\left(\mathbf{G} \cdot \frac{1}{2}\left(\mathbf{g}_i(\mathbf{x}_{i-1} - \mathbf{x}_i)^\mathsf{T} + (\mathbf{x}_{i-1} - \mathbf{x}_i)\mathbf{g}_i^\mathsf{T}\right)\right)$. Applying the similar trick to all terms in (4), we can express $\widetilde{V}_k = \operatorname{Tr}(\mathbf{G}\mathbf{V}_k)$ with

$$\begin{aligned}\mathbf{V}_k :&= \sum_{i=1}^{k} \lambda_{i-1,i}\left(\mathbf{B}_{i-1,i} + \frac{1}{2L}\mathbf{C}_{i-1,i}\right) + \sum_{i=0}^{k} \lambda_{\star,i}\left(\mathbf{B}_{\star,i} + \frac{1}{2L}\mathbf{C}_{\star,i}\right) \\ &- \sum_{i=0}^{k} d_i\left(\frac{1}{(2N+1-i)^2}\mathbf{A}_{\star,i} + \frac{2}{L(2N+1-i)}\mathbf{B}_{\star,i} + \frac{1}{L^2}\mathbf{C}_{\star,i}\right)\end{aligned}$$

where $\mathbf{A}_{i,j}, \mathbf{B}_{i,j}, \mathbf{C}_{i,j} \in \mathbb{S}^{N+2}$ are the symmetric matrices

$$\begin{aligned}\mathbf{A}_{i,j} &= (\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^\mathsf{T} \\ \mathbf{B}_{i,j} &= \frac{1}{2}\left(\mathbf{g}_j(\mathbf{x}_i - \mathbf{x}_j)^\mathsf{T} + (\mathbf{x}_i - \mathbf{x}_j)\mathbf{g}_j^\mathsf{T}\right) \\ \mathbf{C}_{i,j} &= (\mathbf{g}_i - \mathbf{g}_j)(\mathbf{g}_i - \mathbf{g}_j)^\mathsf{T}.\end{aligned}$$

Now, if the matrices $\mathbf{V}_k$ have constant low rank r across k, then this indicates that each $\widetilde{V}_k$ can be expressed in terms of r vectors that are linear combinations of $x_0, \nabla f(x_0), \dots, \nabla f(x_N)$. The process of identifying the low-rank structure of $\mathbf{V}_k$ is based on the following linear algebraic argument. If $\operatorname{rank}(\mathbf{V}_k) = r$ and $\mathcal{B}_k = \{\mathbf{v}_{k,1}, \dots, \mathbf{v}_{k,r}\}$ is a basis of the column space $\operatorname{col}(\mathbf{V}_k)$, then there exist $\mathbf{C}_k \in \mathbb{R}^{r \times r}$ such that

$$\mathbf{V}_k = \begin{bmatrix}\mathbf{v}_{k,1} & \cdots \mathbf{v}_{k,r}\end{bmatrix} \mathbf{C}_k \begin{bmatrix}\mathbf{v}_{k,1} & \cdots & \mathbf{v}_{k,r}\end{bmatrix}^\mathsf{T}. \tag{5}$$

Then $\widetilde{V}_k$ can be expressed as inner products of $\mathbf{P}\mathbf{v}_{k,1}, \dots, \mathbf{P}\mathbf{v}_{k,r}$, and the associated coefficients are the entries of $\mathbf{C}_k$. The process of finding $\mathcal{B}_k$ and $\mathbf{C}_k$ is as follows.

1. Compute $r = \operatorname{rank}(\mathbf{V}_k)$. For GD, $r = 3$.

2. Build the set of candidate vectors as

   $$\mathcal{T} = \{\mathbf{x}_0, \dots, \mathbf{x}_N, \mathbf{x}_\star, \mathbf{g}_0, \dots, \mathbf{g}_N\}, \quad \mathcal{S} = \mathcal{T} \cup \{\mathbf{u} - \mathbf{v} : \mathbf{u}, \mathbf{v} \in \mathcal{T}\}.$$

   Then, use the `vectors_in_column_space` function to determine the candidate vectors that belong to $\operatorname{col}(\mathbf{V}_k)$.

3. Use the `find_basis_with_sparsest_coefficients` function to search over linearly independent subsets of $\widetilde{\mathcal{B}}_k$ with r elements and select the one that yields the sparsest $\mathbf{C}_k$ in the decomposition (5). This will be our $\mathcal{B}_k$. In case of GD, we find $\mathcal{B}_k = \{\mathbf{x}_0 - \mathbf{x}_\star, \mathbf{x}_{k+1} - \mathbf{x}_\star, \mathbf{g}_k\}$.

4. Identify the closed form of the Lyapunov coefficients, i.e., the entries of $\mathbf{C}_k$. This yields the analytic expression of $\widetilde{V}_k$, and thus completes the simplification of V_k. For GD, we find

   $$\begin{aligned}V_k &= \frac{k+1}{2N-k}\big(f(x_k) - f(x_\star)\big) - \frac{L}{4N+2}\|x_0 - x_\star\|^2 \\ &- \frac{k+1}{2L(2N-k)}\|\nabla f(x_k)\|^2 + \frac{L(2N-2k-1)}{2(2N-k)^2}\|x_{k+1} - x_\star\|^2\end{aligned}$$

B. Additional details for Section 3

This appendix documents the commands used in Peppy to turn an algorithm description into an analytic Lyapunov analysis. The five commands are executed sequentially:

`/pep-implement` → `/pep-full-proof` → `/lyap-define` → `/lyap-vectors` → `/lyap-closed-form`.

Each command is written in a Markdown file in the directory `.claude/commands`. The first command, `/pep-implement`, starts by parsing the user-given information, where the required information is specified as follows:

Step 1 — Parse

From `$ARGUMENTS`, identify:

Field	Description
`ALGO_NAME`	`snake_case` identifier, e.g. `heavy_ball`
`PROBLEM_TYPE`	`smooth_convex` / `smooth_strongly_convex` / `monotone_operator` / `composite`
`OBJ_TAG`	`"f"` (functions) or `"A"` (operators)
`PARAMS`	All symbolic parameters (e.g. `L`, `alpha`, `beta`, `R`)
`PERFORMANCE_METRIC`	e.g. `f(x_N) - f(x_star)` or `||A(x_N)||²`
`INITIAL_CONDITION`	e.g. `||x_0 - x_star||²<= R²`
`CONJECTURED_RATE`	Known bound or `"unknown"`

After `/pep-implement` finishes its task based on the parsed information, it creates the output file

`examples/{ALGO_NAME}/state/{ALGO_NAME}_b1.json`,

and initializes a Jupyter notebook

`examples/{ALGO_NAME}/{ALGO_NAME}_example_lyap.ipynb`,

which displays the process in a user-friendly format. Each subsequent command typically reads the state generated by the previous block, performs a specific discovery or verification task, writes a new state file, and updates the same Lyapunov notebook. The following table briefly summarizes the tasks performed by each command and their main outputs.

Command	Tasks	Main output
`/pep-implement`	• Parse the algorithm information. • Implement the algorithm. • Solve PEP problem for various $N = 1, \ldots, 7$ and identify convergence rate • Initialize the Lyapunov notebook.	• Setup module. • Numerical sweep data. • Block 1 state `{ALGO_NAME}_b1.json`.
`/pep-full-proof`	• Find sparsity patterns of the dual variables. • Find a decomposition of the PSD matrix corresponding to the sum-of-squares term in (2). • Try to find closed-form expressions for the dual variables.	• Closed-form λ. • Gram-dual data. • Proof-valid flag. • Block 2 state `{ALGO_NAME}_b2.json`.
`/lyap-define`	• Build Lyapunov partial sums V_k from the full-PEP certificate. • Compute the rank of each Gram matrix corresponding to $V_0, \ldots, V_N$ and construct the rank profile.	• Partial-sum construction of V_k, stored in the list `lyap`. • Rank profile `[r_0, r_1, ..., r_N]`. • Block 3 state `{ALGO_NAME}_b3.json`.
`/lyap-vectors`	• Find special vectors that yield a summation-free expression of V_k. • Find basis templates: k-indexed patterns of the special vectors. • Extract coefficient matrices and infer closed-form coefficient patterns.	• Basis templates. • Coefficient matrices and formula. • Block 4 state `{ALGO_NAME}_b4.json`.
`/lyap-closed-form`	• Assemble the final closed-form Lyapunov proof. • Verify the result symbolically.	• Final notebook. • Block 5 state `{ALGO_NAME}_b5.json` • LaTeX theorem/proof.

Some tasks are straightforward for a coding agent to perform by following the guideline, while other nontrivial tasks may not succeed automatically. For such tasks, Peppy presents the result it has obtained and asks the user for guidance. In this way, users can interact with Peppy during the workflow. The following is an example of a user-interaction guideline provided in `/pep-full-proof`.

Nontrivial checkpoint: If no small sparsity pattern can be found that preserves the rate (even expanded hypothesis fails), display the dense λ fully and ask the user which entries look structurally important.

The final result is a Jupyter notebook file that contains a LaTeX theorem statement and proof outline, together with the detailed results of the whole workflow. The Jupyter notebook files are available at:

https://github.com/pepflow-lib/PEPFlow/tree/peppy-workshop-v1/examples_peppy

Examples of the theorem statements and proofs are provided in Section C.

C. Omitted details of Section 4

In this section, we provide the details of the experiments. In all experiments, we used OpenAI Codex as the AI assistant, with the GPT-5 model family and the reasoning setting set to medium. We note that the contents of the subsections titled *"Output theorem statement and proof"* were generated fully automatically by Peppy.

C.1. Proximal gradient method.

Let f be convex and L-smooth, let g be closed proper convex, and let $x_\star$ minimize $h = f + g$. Consider proximal gradient descent method:

$$x_{k+1} = \mathrm{prox}_{g/L}\left(x_k - L^{-1}\nabla f(x_k)\right). \tag{PGM}$$

The convergence rate

$$h(x_N) - h(x_\star) \le \frac{L}{4N}\|x_0 - x_\star\|^2$$

was studied in (Teboulle & Vaisbourd, 2023), although they did not consider a Lyapunov function. In this section, we provide a Lyapunov analysis obtained through Peppy that recovers the same convergence rate.

We point out that this example involves human intervention. The decomposition of the sum-of-squares terms (which plays the same role as $\sum_{i=0}^{N-1} d_i \|s_i\|^2$ in (2)) is manually provided in `examples_peppy/_references/pgm/pgm_example.ipynb`. Peppy crucially uses this information to successfully proceed through the blocks `lyap-define` and `lyap-vectors`.

C.1.1. Input prompt

```
/pep-implement

Problem type: composite_convex_minimization.
Function: f is convex and L-smooth.
Function: g is closed, proper, and convex.
Objective: h = f + g.
Parameters: L, R.
Initial condition: ||x_0 - x_star||^2 <= R^2, where x_star minimizes h.
Performance metric: h(x_N) - h(x_star).

Algorithm: Proximal gradient descent with fixed step size 1/L.

For k >= 0:
x_{k+1} = prox_{g/L}(x_k - (1/L) * grad f(x_k))

Conjectured rate: unknown.
```

Then, repeat:

```
proceed with the next step
```

We obtained the output in the next subsection as a result.

C.1.2. Output theorem statement and proof

Theorem.
Assume f is convex and L-smooth, g is closed, proper, and convex, and $x_\star$ minimizes

$$h(x) = f(x) + g(x).$$

Let x_0 satisfy $\|x_0 - x_\star\|^2 \le R^2$, and let proximal gradient descent generate

$$x_{k+1} = \mathrm{prox}_{g/L}\left(x_k - \frac{1}{L}\nabla f(x_k)\right), \qquad k = 0, \dots, N-1.$$

Then

$$h(x_N) - h(x_\star) \le \frac{LR^2}{4N}.$$

For $1 \le k < N$, define

$$\begin{aligned} V_k = {} & \frac{k+1}{2N-k}\big(f(x_k) - f(x_\star)\big) + \frac{k}{2N-k}\big(g(x_k) - g(x_\star)\big) - \frac{L}{4N}\|x_0 - x_\star\|^2 \\ & + \frac{L(N-k)}{(2N-k)^2}\|x_k - x_\star\|^2 - \frac{1}{2N-k}\langle x_k - x_\star, \nabla f(x_k)\rangle + \frac{1}{2L(2N-k)}\|\nabla f(x_k) - \nabla f(x_\star)\|^2. \end{aligned}$$

The terminal expression is

$$V_N = h(x_N) - h(x_\star) - \frac{L}{4N}\|x_0 - x_\star\|^2.$$

Proof outline.

For f, use the smooth-convex interpolation residual

$$I_f(u,v) = f(v) - f(u) + \langle \nabla f(v), u - v\rangle + \frac{1}{2L}\|\nabla f(u) - \nabla f(v)\|^2 \le 0.$$

For g, use the convex interpolation residual

$$I_g(u,v) = g(v) - g(u) + \langle s_v, u - v\rangle \le 0, \qquad s_v \in \partial g(v).$$

The proximal update gives $s_{x_i} = -\nabla f(x_{i-1})$ for $i \ge 1$. Define the nonnegative square certificate $S_i = S_i^{(1)} + S_i^{(2)} + S_i^{(3)}$, where

$$S_i^{(1)} = \frac{N}{(2N+1-i)(2N-i)L}\left\|\frac{i}{2N}\nabla f(x_i) + \frac{2N-i}{2N}\nabla f(x_{i-1}) - \nabla f(x_\star)\right\|^2,$$

$$S_i^{(2)} = \left(\frac{i-1}{2N-i}\frac{2N+2}{2N+1} + \frac{1}{(2N-i)^2}\frac{2N}{2N+1}\right)\frac{1}{2L}\left\|\nabla f(x_{i-1}) + s_{x_i} - \frac{L}{2N+1-i}(x_{i-1} - x_\star)\right\|^2,$$

and

$$S_i^{(3)} = \frac{i}{2N+1-i}\frac{2N}{2N+1}\frac{1}{2L}\left\|\frac{2N+1}{2N}\big(\nabla f(x_i) + s_{x_i}\big) - \frac{1}{2N}\big(\nabla f(x_{i-1}) + s_{x_i}\big) - \frac{L}{2N-i}(x_i - x_\star)\right\|^2.$$

The base identity is

$$\begin{aligned} V_1 = {} & \frac{1}{2N}I_f(x_\star, x_0) + \frac{1}{2N}I_f(x_0, x_1) + \left(\frac{2}{2N-1} - \frac{1}{2N}\right)I_f(x_\star, x_1) \\ & + \frac{1}{2N-1}I_g(x_\star, x_1) - S_1. \end{aligned}$$

For $1 \le k < N-1$, the step identity is

$$\begin{aligned} V_{k+1} - V_k = {} & \frac{k+1}{2N-k}I_f(x_k, x_{k+1}) + \left(\frac{k+2}{2N-k-1} - \frac{k+1}{2N-k}\right)I_f(x_\star, x_{k+1}) \\ & + \frac{k}{2N-k}I_g(x_k, x_{k+1}) + \frac{2N}{(2N-k)(2N-k-1)}I_g(x_\star, x_{k+1}) - S_{k+1}. \end{aligned}$$

All interpolation residuals are nonpositive and all S_i are nonnegative, so $V_1 \le 0$ and $V_{k+1} \le V_k$. Therefore $V_N \le 0$, and the boundary identity gives

$$h(x_N) - h(x_\star) - \frac{L}{4N}\|x_0 - x_\star\|^2 = V_N \le 0.$$

Using $\|x_0 - x_\star\|^2 \le R^2$ proves the stated guarantee.

Here, $s_{x_i} = -\nabla f(x_{i-1})$ is a minor typo. It should be $s_{x_i} = L(x_{i-1} - x_i) - \nabla f(x_{i-1})$.

C.2. OGM

Let f be L-smooth convex function. Fix $N > 0$. Optimized gradient method (OGM, (Kim & Fessler, 2016)) is defined as

$$\begin{aligned} y_{k+1} &= x_k - \frac{1}{L}\nabla f(x_k) \\ z_{k+1} &= z_k - \frac{2\theta_k}{L}\nabla f(x_k) \\ x_{k+1} &= \left(1 - \frac{1}{\theta_{k+1}}\right)y_{k+1} + \frac{1}{\theta_{k+1}}z_{k+1}, \end{aligned} \tag{OGM}$$

where θ_k is defined as

$$\theta_k = \begin{cases} \theta_{-1} = 0, & k = -1 \\ \frac{1+\sqrt{4\theta_{k-1}^2+1}}{2}, & k = 0, \dots, N-1, \\ \theta_N = \frac{1+\sqrt{8\theta_{N-1}^2+1}}{2}, & k = N. \end{cases}$$

OGM achieves the convergence rate

$$f(x_N) - f(x_\star) \leq \frac{L}{2\theta_N^2}\|x_0 - x_\star\|^2$$

and its Lyapunov analysis was later given in (d'Aspremont et al., 2021; Park et al., 2023). In this section, we recover this Lyapunov analysis using Peppy.

C.2.1. Input prompt

```
/pep-implement
Function: f is convex and L-smooth
Parameters: L, R
Initial condition: ||x_0 - x_star|| <= R, where
grad f(x_star) = 0
Performance metric: f(x_N) - f(x_star)
Algorithm: Optimized gradient method with fixed step size 1/L

Define theta_{-1} = 0. For a fixed horizon N, define
theta_N = (1/2) * (1 + sqrt(8 * theta_{N-1}^2 + 1)),
and for 0 <= k < N,
theta_k = (1/2) * (1 + sqrt(4 * theta_{k-1}^2 + 1)).

Initialize:
z_0 = x_0

For k = 0, ..., N-1:
y_k = x_k - (1/L) * grad f(x_k)
z_{k+1} = z_k - (2/L) * theta_k * grad f(x_k)
x_{k+1} = (1 - 1/theta_{k+1}) * y_k + (1/theta_{k+1}) * z_{k+1}

Conjectured rate: unknown
```

Then, repeat:

```
proceed with the next step
```

C.2.2. Output theorem statement and proof

Theorem.
Let f be convex and L-smooth, and let $x_\star$ satisfy $\nabla f(x_\star) = 0$. For the fixed-horizon OGM recurrence in this notebook, assume $\|x_0 - x_\star\| \leq R$. Let $\theta_{-1} = 0$, let $\theta_k = \frac{1}{2}(1 + \sqrt{4\theta_{k-1}^2 + 1})$ for $0 \leq k < N$, and let $\theta_N = \frac{1}{2}(1 + \sqrt{8\theta_{N-1}^2 + 1})$. For $1 \leq k < N$, define the Lyapunov certificate

$$V_k = \frac{2\theta_k^2}{\theta_N^2}\big(f(x_k) - f(x_\star)\big) - \frac{L}{2\theta_N^2}\|x_0 - x_\star\|^2 - \frac{\theta_k^2}{L\theta_N^2}\|\nabla f(x_k)\|^2 + \frac{L}{2\theta_N^2}\|z_{k+1} - x_\star\|^2.$$

Then the final iterate satisfies

$$f(x_N) - f(x_\star) \leq \frac{LR^2}{2\theta_N^2}.$$

Proof outline. For a smooth convex interpolation residual, write

$$I_f(u, v) = f(v) - f(u) + \langle \nabla f(v), u - v\rangle + \frac{1}{2L}\|\nabla f(u) - \nabla f(v)\|^2 \leq 0.$$

The base identity is

$$V_1 = \frac{2}{\theta_N^2} I_f(x_\star, x_0) + \frac{2}{\theta_N^2} I_f(x_0, x_1) + \frac{2\theta_1}{\theta_N^2} I_f(x_\star, x_1).$$

For $1 \le k < N-1$, using $\theta_{k+1}^2 - \theta_{k+1} = \theta_k^2$,

$$V_{k+1} - V_k = \frac{2\theta_k^2}{\theta_N^2} I_f(x_k, x_{k+1}) + \frac{2\theta_{k+1}}{\theta_N^2} I_f(x_\star, x_{k+1}) \le 0.$$

Thus V_k is nonincreasing under the chosen sign convention. The terminal certificate uses

$$S = \frac{L}{2\theta_N^2} \left\| x_0 - x_\star - \sum_{i=0}^{N-1} \frac{2\theta_i}{L} \nabla f(x_i) - \frac{\theta_N}{L} \nabla f(x_N) \right\|^2 \ge 0$$

and the boundary identity

$$\frac{L}{2\theta_N^2} \|x_0 - x_\star\|^2 - \big(f(x_N) - f(x_\star)\big) + V_N - S = 0.$$

Since $V_N \le V_0 = 0$ and $S \ge 0$, the claimed bound follows.

C.3. BPPM

Let f be closed, proper, convex, and differentiable relative to the Bregman kernel h. Let h be closed, proper, convex, and differentiable, and define

$$D_h(x, y) = h(x) - h(y) - \langle \nabla h(y), x - y \rangle.$$

For a fixed $\alpha > 0$, consider Bregman proximal point method (BPPM)

$$x_{k+1} = \mathrm{prox}_{\alpha f}^h(x_k). \tag{BPPM}$$

BPPM achieves the convergence rate (Auslender & Teboulle, 2006)

$$f(x_N) - f(x_\star) \le \frac{1}{\alpha N} D_h(x_\star, x_0).$$

In this section, we recover this result through a Lyapunov analysis obtained through `Peppy`.

C.3.1. Input prompt

```
/pep-implement

Problem type: composite_convex_minimization.
Function: f is closed, proper, convex, and differentiable relative to the Bregman kernel h.
Function: h is closed, proper, convex, and differentiable; h defines the Bregman distance.
Objective: f.
Parameters: alpha, R.
Initial condition: D_h(x_star, x_0) <= R, where x_star minimizes f.
Performance metric: f(x_N) - f(x_star).
Algorithm: Bregman proximal point method with fixed step size alpha.
For k >= 0:
x_{k+1} = prox^h_{alpha * f}(x_k)
In PEPFlow:
x_{k+1} = f.bregman_prox(x_k, alpha, h)
Conjectured rate: R / (alpha * N).
```

Then, repeat:

```
proceed with the next step
```

C.3.2. Output theorem statement and proof

In the proof below, $I_g(u, v) = g(u) - g(v) - \langle \nabla g(v), u - v \rangle \le 0$ contains a minor typo. It should be $I_g(u, v) = g(v) - g(u) + \langle \nabla g(v), u - v \rangle \le 0$.

Theorem.
Assume f is closed, proper, convex, and differentiable relative to the Bregman kernel h, and assume h is closed, proper, convex, and differentiable. Let

$$D_h(u,v) = h(u) - h(v) - \langle \nabla h(v), u - v\rangle.$$

Run the Bregman proximal point method with fixed $\alpha > 0$,

$$x_{k+1} = \operatorname{prox}^h_{\alpha f}(x_k), \qquad k \geq 0,$$

equivalently $\alpha \nabla f(x_{k+1}) + \nabla h(x_{k+1}) - \nabla h(x_k) = 0$. If $x_\star \in \operatorname{argmin} f$ and $D_h(x_\star, x_0) \leq R$, then for every $N \geq 1$,

$$f(x_N) - f(x_\star) \leq \frac{R}{\alpha N}.$$

The Lyapunov certificate recovered below is

$$V_k = \frac{k}{N}\left(f(x_k) - f(x_\star)\right) + \frac{D_h(x_\star, x_k) - D_h(x_\star, x_0)}{\alpha N}, \qquad 1 \leq k < N,$$

with terminal form

$$V_N = f(x_N) - f(x_\star) - \frac{D_h(x_\star, x_0)}{\alpha N}.$$

Proof outline.
For any convex function g, write the interpolation residual as

$$I_g(u,v) = g(u) - g(v) - \langle \nabla g(v), u - v\rangle \leq 0.$$

The BPPM optimality condition is encoded by PEPFlow as

$$\nabla h(x_{k+1}) = \nabla h(x_k) - \alpha \nabla f(x_{k+1}).$$

The first step satisfies

$$V_1 = \frac{1}{N} I_f(x_\star, x_1) + \frac{1}{\alpha N} I_h(x_1, x_0) \leq 0.$$

For $1 \leq k < N-1$,

$$\begin{aligned} V_{k+1} - V_k = {} & \frac{1}{N} I_f(x_\star, x_{k+1}) + \frac{k}{N} I_f(x_k, x_{k+1}) \\ & + \frac{k+1}{\alpha N} I_h(x_{k+1}, x_k) + \frac{k}{\alpha N} I_h(x_k, x_{k+1}) \leq 0. \end{aligned}$$

The final step has the same form with $k = N-1$ and the extra boundary interpolation residual:

$$\begin{aligned} V_N - V_{N-1} = {} & \frac{1}{N} I_f(x_\star, x_N) + \frac{N-1}{N} I_f(x_{N-1}, x_N) + \frac{1}{\alpha} I_h(x_N, x_{N-1}) \\ & + \frac{N-1}{\alpha N} I_h(x_{N-1}, x_N) + \frac{1}{\alpha N} I_h(x_\star, x_N) \leq 0. \end{aligned}$$

Thus $V_N \leq V_{N-1} \leq \cdots \leq V_1 \leq 0$. Finally,

$$f(x_N) - f(x_\star) - \frac{R}{\alpha N} = V_N + \frac{D_h(x_\star, x_0) - R}{\alpha N} \leq 0,$$

because $D_h(x_\star, x_0) - R \leq 0$.

C.4. FEG

Assume that $\mathbb{A}$ is monotone and L-Lipschitz where $L > 0$, and let $x_\star$ satisfy $\mathbb{A}(x_\star) = 0$. Let the Fast Extragradient (FEG) method be

$$\begin{aligned} x_{k+1/2} &= x_k + \frac{1}{k+1}(x_0 - x_k) - \frac{k}{k+1}\frac{1}{L}\mathbb{A}(x_k) \\ x_{k+1} &= x_k + \frac{1}{k+1}(x_0 - x_k) - \frac{1}{L}\mathbb{A}(x_{k+1/2}). \end{aligned} \tag{FEG}$$

Then for every $N \geq 1$,

$$\|\mathbb{A}(x_N)\|^2 \leq \frac{4L^2}{N^2}\|x_0 - x_\star\|^2.$$

Peppy provided essentially the same Lyapunov function as in (Lee & Kim, 2021), but in a slightly different expression.

C.4.1. Input prompt

```
/pep-implement


Problem type: monotone_operator.
Operator: A is monotone and L-Lipschitz.
Parameters: L, R.
Initial condition: ||x_0 - x_star||^2 <= R^2, where A(x_star)=0.
Performance metric: ||A(x_N)||^2.

Algorithm:
Fast Extragradient (FEG) method for monotone L-Lipschitz operators.
For k = 0, tag the initial point x_0 also as x_{0.5}; do not create a separate intermediate iterate.

For k >= 0:
x_{k+1/2} = x_k + 1/(k+1) * (x_0 - x_k) - k/(k+1) * (1/L) * A(x_k)
x_{k+1} = x_k + 1/(k+1) * (x_0 - x_k) - (1/L) * A(x_{k+1/2})

Conjectured rate: unknown.
```

C.4.2. Output theorem statement and proof

Theorem.
Assume A is monotone and L-Lipschitz, $A(x_\star) = 0$, and $\|x_0 - x_\star\|^2 \leq R^2$. Let the FEG iterates be

$$x_{k+1/2} = x_k + \frac{1}{k+1}(x_0 - x_k) - \frac{k}{k+1}\frac{1}{L}A(x_k),$$

$$x_{k+1} = x_k + \frac{1}{k+1}(x_0 - x_k) - \frac{1}{L}A(x_{k+1/2}),$$

with $x_{1/2} = x_0$. For $1 \leq k \leq N$, define

$$V_k = \frac{2k^2}{N^2}\|A(x_k)\|^2 - \frac{4kL}{N^2}\langle A(x_k), x_0 - x_k\rangle.$$

Then for every integer $N \geq 1$,

$$\|A(x_N)\|^2 \leq \frac{4L^2}{N^2}\|x_0 - x_\star\|^2.$$

Proof outline.
Use the nonpositive residual convention

$$I^{\text{mon}}(u, v) = -\langle u - v, A(u) - A(v)\rangle \leq 0,$$

$$I^{\text{lip}}(u, v) = \|A(u) - A(v)\|^2 - L^2\|u - v\|^2 \leq 0.$$

The base identity is

$$V_1 - \frac{2}{N^2}I^{\text{lip}}(x_0, x_1) = 0.$$

For $1 \leq k \leq N - 1$, the one-step identity is

$$V_{k+1} - V_k - \frac{4Lk(k+1)}{N^2}I^{\text{mon}}(x_k, x_{k+1}) - \frac{2(k+1)^2}{N^2}I^{\text{lip}}(x_{k+1/2}, x_{k+1}) = 0.$$

Finally, with

$$S = \left\| A(x_N) - \frac{2L}{N}(x_0 - x_\star) \right\|^2 \geq 0,$$

the boundary identity is

$$\frac{4L^2}{N^2}\|x_0 - x_\star\|^2 - \|A(x_N)\|^2 + V_N + \frac{4L}{N} I^{\text{mon}}(x_N, x_\star) - S = 0.$$

Because $V_{k+1} - V_k$ is a positive-weighted sum of nonpositive residuals $I \leq 0$, the sequence V_k is nonincreasing and $V_N \leq 0$. The terminal residual is also nonpositive and $S \geq 0$, so the boundary identity implies the claimed residual bound.

C.5. Dual-FEG

Let $\mathbb{A}$ be monotone and L-Lipschitz with $\mathbb{A}(x_\star) = 0$. Fix $N > 0$. Dual-FEG (Yoon et al., 2024) with stepsize $\alpha = 1/L$ is defined as

$$\begin{aligned} x_{k+1/2} &= x_k - \alpha z_k - \alpha \mathbb{A}(x_k), \\ x_{k+1} &= x_{k+1/2} - \frac{N-k-1}{N-k}\alpha \left(\mathbb{A}(x_{k+1/2}) - \mathbb{A}(x_k) \right), \\ z_{k+1} &= \frac{N-k-1}{N-k} z_k - \frac{1}{N-k}\mathbb{A}(x_{k+1/2}), \end{aligned} \tag{Dual-FEG}$$

with $z_0 = 0$. Then, for every $N \geq 1$,

$$\|\mathbb{A}(x_N)\|^2 \leq \frac{4L^2}{N^2}\|x_0 - x_\star\|^2.$$

We show that Peppy recovers the Lyapunov analysis of (Yoon et al., 2024).

C.5.1. Input prompt

```
/pep-implement

Problem type: monotone_operator_variational_inequality.
Operator: A is monotone and L-Lipschitz continuous.
Parameters: L, N, R.
Step size: alpha = 1/L.
Initial condition: ||x_0 - x_star||^2 <= R^2, where x_star satisfies A(x_star) = 0.
Performance metric: unknown.
Algorithm: Dual FEG
step size alpha = 1/L.
For k = 0, ..., N - 1:

x_{k+1/2} = x_k - alpha z_k - alpha A(x_k)

x_{k+1} = x_{k+1/2} - ((N - k - 1) / (N - k)) * alpha * (A(x_{k+1/2}) - A(x_k))

z_{k+1} = ((N - k - 1) / (N - k)) * z_k - (1 / (N - k)) * A(x_{k+1/2})

Conjectured rate: unknown.
```

C.5.2. Output theorem statement and proof

Theorem.
Assume A is monotone and L-Lipschitz continuous, $A(x_\star) = 0$, and $\|x_0 - x_\star\|^2 \leq R^2$. Let Dual FEG use $\alpha = 1/L$ and $z_0 = 0$:

$$x_{k+1/2} = x_k - \alpha z_k - \alpha A(x_k),$$

$$x_{k+1} = x_{k+1/2} - \frac{N-k-1}{N-k}\alpha\left(A(x_{k+1/2}) - A(x_k)\right),$$

$$z_{k+1} = \frac{N-k-1}{N-k}z_k - \frac{1}{N-k}A(x_{k+1/2}).$$

For $N \geq 2$, define $V_0 = 0$. For $1 \leq k \leq N-1$,

$$V_k = 2\|A(x_N)\|^2 - 2\|z_k + A(x_N)\|^2 - \frac{4L}{N}\langle A(x_N), x_0 - x_N\rangle + \frac{4L}{N-k}\langle z_k + A(x_N), x_k - x_N\rangle.$$

The terminal term is

$$V_N = V_{N-1} + 2I^{\mathrm{lip}}(x_{N-1}, x_N).$$

Then

$$\|A(x_N)\|^2 \leq \frac{4L^2}{N^2}\|x_0 - x_\star\|^2 \leq \frac{4L^2R^2}{N^2}.$$

Proof outline.
Use the nonpositive residuals

$$I^{\mathrm{lip}}(u,v) = \|A(u) - A(v)\|^2 - L^2\|u - v\|^2 \leq 0, \qquad I^{\mathrm{mon}}(u,v) = -\langle A(u) - A(v), u - v\rangle \leq 0.$$

The Lyapunov function is the closed-form V_k stated above. Its increments satisfy the exact identities

$$V_1 - V_0 = \frac{2}{N^2}I^{\mathrm{lip}}(x_0, x_{1/2}) + \frac{4L}{(N-1)N}I^{\mathrm{mon}}(x_{1/2}, x_N),$$

and, for $1 \leq k \leq N-2$,

$$V_{k+1} - V_k = \frac{2}{(N-k)^2}I^{\mathrm{lip}}(x_k, x_{k+1/2}) + \frac{4L}{(N-k-1)(N-k)}I^{\mathrm{mon}}(x_{k+1/2}, x_N).$$

The final increment is

$$V_N - V_{N-1} = 2I^{\mathrm{lip}}(x_{N-1}, x_N).$$

All coefficients are nonnegative and all residuals are nonpositive, so $V_N \leq V_0 = 0$. The boundary identity is

$$\|A(x_N)\|^2 - \frac{4L^2}{N^2}\|x_0 - x_\star\|^2 - V_N - \frac{4L}{N}I^{\mathrm{mon}}(x_N, x_\star) + \left\|A(x_N) - \frac{2L}{N}(x_0 - x_\star)\right\|^2 = 0.$$

Since $-V_N \geq 0$, $-I^{\mathrm{mon}}(x_N, x_\star) \geq 0$, and the square is nonnegative, the claimed residual bound follows.