PtyRAD is now in internal beta test so it's limited to only the Muller group and invited people

*** Please do NOT share any part of the package to anyone else! ***

PtyRAD: Ptychographic Reconstruction with Automatic Differentiation

This package performs ptychographic reconstruction using an automatic differentiation approach.

Introduction

Ptychographic reconstruction is often solved as an optimization problem using gradient descent methods. Typical ptychographic reconstruction packages (e.g. *PtychoShelves*, *PtyPy*, *py4DSTEM*) use an analytically derived or approximated gradients and apply them for the updates, while *PtyRAD* utilizes automatic differention (AD) to automatically calculate the needed gradients.

The main difference is that automatic differentiation allows simpler implementation for adding and modifying new optimizable variables. For typcial packages that utilize analytical gradients, adding a new optimizable variable (like adding probe position correction or adaptive specimen tilt) requires deriving the corresponding gradient with respect to the objective (loss) function. Manually deriving the gradients for new variables can be tedious and error-prone. On the other hand, automatic differentiation provides instant gradients as long as a differentiable forward model is provided, and flexible control over the optimizable variables including optimizing the amplitude and phase of the object individually.

Additionally, automatic differentiation is the backbone of backpropagation, which is the enabling training algorithm for all modern deep learning networks. *PtyRAD* uses *PyTorch*'s `autograd` module for its AD architecture, which allows us to benefit from the active PyTorch community for potentially extended capabilities and improved performance over the time.

Getting Started

Major dependencies

- Python 3.10 or above
- PyTorch 2.1 or above
- While *PtyRAD* can run on CPU, GPU is strongly suggested for high-speed ptychographic reconstructions.
  - *PtyRAD* supports both NVIDIA GPUs with CUDA and Apple Silicon (MPS)
- *PtyRAD* was tested on Windows, MacOS, and Linux

Recommended Tool (Optional)

If you're not comfortable using the terminal, we suggest installing *Visual Studio Code*. *VS Code* makes it easier to manage Python environments, open terminals in the correct folder, and run Jupyter notebooks — all in one interface. After installing *VS Code*, you can:

- Open the `ptyrad_review/` folder in *VS Code*
- Seamlessly switch between different created Python environments
- Easily launch the notebook or run scripts inside the same window

Step-by-Step Guide

> 💡 **Note:** All the commands in this `README.md` are single-line commands — copy and paste them as-is, even if they wrap in your PDF viewer.

Step 1. Install a Python environment management software

We recommend *Miniforge*, a lightweight and open-source alternative to *Anaconda*. It provides the conda ecosystem, while solving environments much faster and taking up much less disk space.

Once *Miniforge* is installed:

1. Open the "*Miniforge Prompt*" on Windows, or a new terminal window on MacOS/Linux.

   > 💡 **Tip:** On Windows, use the Start menu to search for "*Miniforge Prompt*".
   >
   > If you prefer to use *cmd* or *PowerShell* on your Windows machine, you can optionally run the following command in your *Miniforge Prompt*:
   >
   > ```
   > # This command will allow other terminals like *cmd* or *PowerShell* to directly access `conda`
   > conda init
   > ```
   >
   > This command only need to be executed once.

   > 💡 **Tip:** For macOS and Linux, `conda` should be automatically available after installing *Miniforge* when you open a new terminal. If not, try restarting your terminal, or manually activate it with:
   >
   > ```
   > # Adjust the path if you installed Miniforge elsewhere
   > source ~/miniforge3/bin/activate
   > ```

2. Navigate to the `ptyrad_review/` folder that contains this `README.md` as well as the main `ptyrad/` code.

   You can do this using the `cd` command, which means **changing directory**:

   ```
   cd path/to/ptyrad_review
   ```

   Replace `path/to/ptyrad_review` with the actual path to this folder. This step is required so that all the example scripts and paths work correctly.

3. You can check that you're in the right folder by running:

   ```
   ls  # or `dir` on Windows
   ```

   You should see files like `README.md`, `run_ptyrad.py`, and a folder named `ptyrad/`.

You're now ready to set up a dedicated Python environment for *PtyRAD*. Creating separate environments for each project is strongly recommended — it helps avoid version conflicts and keeps your setup clean and reproducible.

Step 2. Create the Python environment required for *PtyRAD*

Based on your system and whether you have GPU support, choose the appropriate installation command below and run it in your terminal.

- **2a. For Windows and Linux system with CUDA-supported GPUs**
  - Option 1 (Recommended): Create the environment using the provided YAML file

    ```
    # Ensure you are inside the `ptyrad_review/` directory so the environment YAML file is visible
    conda env create -f environment_ptyrad_review.yml
    ```

  - Option 2: Manually specifying the packages for more control

    ```
    conda create -n ptyrad_review python pytorch pytorch-cuda accelerate torchvision optuna matplotlib scikit-learn scipy h5py tifffile jupyter -c nvidia -c pytorch -c conda-forge
    ```

- **2b. For MacOS users, or Windows/Linux without a GPU**

  ```
  conda create -n ptyrad_review python pytorch accelerate torchvision optuna matplotlib scikit-learn scipy h5py tifffile jupyter -c pytorch -c conda-forge
  ```

> 💡 **Note:** It’s completely fine to use the GPU install options even on systems without a GPU. Packages like `pytorch-cuda` will simply remain inactive and unused. If your *PyTorch* installation isn’t built with CUDA support, or if *PtyRAD* can't detect a compatible GPU, it will automatically fall back to running on the CPU.

The created Python environment will be named `(ptyrad_review)` and stored under `miniforge3/envs/ptyrad_review/`. Throughout this guide, we use `(ptyrad_review)` to refer to the Python environment, and `ptyrad_review/` to refer to the repository folder. The parentheses `(...)` help keep them distinct.

Step 3. Install the *PtyRAD* package

Once the Python environment `(ptyrad_review)` is created:

1. Activate the `(ptyrad_review)` environment:

   In your terminal, run the following command:

   ```
   conda activate ptyrad_review
   ```

   Your prompt should now begin with `(ptyrad_review)` to indicate the environment has been activated.

2. Navigate to the `ptyrad_review/` folder if you haven't done so.

   You can confirm you're in the right place by listing the contents and you will need `pyproject.toml` for the installation of *PtyRAD*.

   ```
    ls  # or `dir` on Windows
   ```

3. Install *PtyRAD* as an editable Python package inside the `(ptyrad_review)` environment

   ```
   # Note that there is a space and period after "-e"
    pip install -e .
   ```

   > 💡 **Note:** This editable install registers *PtyRAD* as a Python package, so you can import it from anywhere — without worrying about your working directory or `sys.path`. While directly importing the `ptyrad/` folder is possible, using `pip install -e .` is much more robust and user-friendly.
   >
   > This command also creates a `ptyrad.egg-info/` folder for storing metadata. You can safely ignore or delete this folder — it won't affect the installed package.

Step 4. Try the demo script / notebook

Once your `(ptyrad_review)` environment is activated, and you've navigated to the `ptyrad_review/` folder and installed *PtyRAD*, you're ready to run a quick demo using one of two interfaces:

- Interactive interface (Recommended):

  Use `run_ptyrad_quick_example.ipynb` to quickly reconstruct the demo dataset in a *Jupyter notebook*

- Command-line interface:

  Use `run_ptyrad.py` to run *PtyRAD* directly from your *Miniforge Prompt* terminal:

```
python ./run_ptyrad.py --params_path "params/ptyrad_examples_tBL_WSe2.yml" -
-gpuid 0
```

We've included two example datasets (PSO, tBL_WSe2) in the `data/` folder and three preconfigured parameter files in `params/` to get you started. The full datasets and all parameter files used in the manuscript will be released on Zenodo.

A Note on this Repo Structure

This repository is intentionally structured for **demonstration purposes** to provide an **out-of-the-box experience** — including example data under `data/` and configuration files under `params/` for quick testing.

In general, we do not recommend mixing input data or local configuration files within the codebase, especially in production or collaborative settings. Once *PtyRAD* is released on `PyPI` and `conda-forge`, it will be installed like any other Python package, and users will be expected to organize their own workspace, and provide their own data and configuration files externally.

Beyond the Demo: Using *PtyRAD* with Your Own Data

PtyRAD is designed to be **fully driven and specified by configuration files** — no code editing required. Each reconstruction is specified using a YAML params file, which includes the location of your data, preprocessing options, experimental parameters, and all other relevant reconstruction parameters (e.g., optimizer algorithm, constraints, loss, output files, etc).

To reconstruct your own dataset:

1. Prepare your data and place it in any folder of your choosing (e.g., a `data/` directory in your workspace).

2. Create or edit a YAML params file with the appropriate paths and settings for your data. You can keep this file anywhere — as long as your script or notebook knows where to find it.

3. Run *PtyRAD* using the same notebook or script provided in this repo, but pointing it to your customized params file. Note that the `output/` folder will be automatically generated under your current working directory.

Author

Chia-Hao Lee (cl2696@cornell.edu)

Developed at the Muller Group, Cornell University.

Acknowledgments

This package gets inspiration from lots of community efforts, and specifically from the following packages. Some of the functions in *PtyRAD* are directly translated or modified from these packages as noted in their comments to give explicit acknowledgment.

- PtychoShelves
- fold_slice
- py4dstem
- adorym
- SciComPty