Self-guided Approximate Linear Programs

Last updated on: December 21, 2023

Topics: approximate dynamic programming, approximate linear programming, model-based reinforcement learning, random Fourier features, inventory management, Options Pricing.

What is this repository?

This repository hosts algorithm implementations and benchmarks discussed in the paper 'Self-Guided Approximate Linear Programs: Randomized Multi-Shot Approximation of Discounted Cost Markov Decision Processes' authored by Parshan Pakiman, Selvaprabu Nadarajah, Negar Soheili, and Qihang Lin, available at Management Science. Specifcally, it includes implementations of following methods:

- Feature-based Approximate Linear Program (FALP)
- Self-guided FALP
- Policy-guided FALP
- Least Squares Monte Carlo (Longstaff and Schwartz 2001)

Furthermore, the repository includes implementations of two algorithms for computing a lower bound on the optimal policy cost. Integrating this lower bound with an upper bound derived from simulating a control policy enables the computation of an optimality gap on the policy's performance. These algorithms are: Information Relaxation and Duality (Brown et al. 2010), and a heuristic based on Constraint Violation Learning (Lin et al. 2020). This repository enables the comparison of the aforementioned methods across two applications: perishable inventory control and Bermudan options pricing. Please note that the code provided here can be used to compute control policies and bounds for other Markov Decision Processes.

How to use this repository?

The following steps are tailored for macOS and Ubuntu users. Windows users may adapt and create similar procedures suitable for their operating system

1. Download the repository on your local system and extract the zip file. For ease of reference, we assume the extracted files are stored in the following path: `/Desktop/Self-Guided-ALPs-Discounted-Cost-master`.
2. Open `Terminal` on your machine and run the following code:

   ```
   cd  /Desktop/Self-Guided-ALPs-Discounted-Cost-master
   ```

3. Please check the version of your Python. For example, run the code below:

   ```
   python3 --version
   ```

4. Please confirm that `Python 3.10` or above is installed on your machine. There are different methods available for installing Python. We leave this step to the user's discretion.

5. Create a virtual Python environment and name it `ALP`. For example, use the following code:

   ```
   python -m venv ALP
   ```

6. Activate the `ALP` environment as follows:

   ```
   source ALP/bin/activate
   ```

7. This code relies on several Python packages. We utilize `pip` for their installation, but users have the flexibility to explore alternative methods. If choosing `pip`, please use the following code for its update:

   ```
   pip install --upgrade pip
   ```

8. Please install the following libraries on the `ALP` environment:

   - `numpy`
   - `pandas`
   - `scipy`
   - `numba`
   - `tqdm`
   - `emcee`
   - `sampyl`
   - `importlib`
   - `sampyl_mcmc`
   - `nengo`
   - `gurobipy`

   For instance, run the following command to install all the aforementioned libraries:

   ```
   python -m pip install numpy pandas scipy numba tqdm emcee sampyl
   importlib sampyl_mcmc nengo gurobipy
   ```

9. This repository depends on Gurobi for solving large-scale linear programs. Please ensure that Gurobi (`gurobipy`) is installed along with its corresponding license. If you are affiliated with academia, Gurobi provides a free academic license, accessible through this page. In some cases, an error might occur when running the `grbgetkey` code to

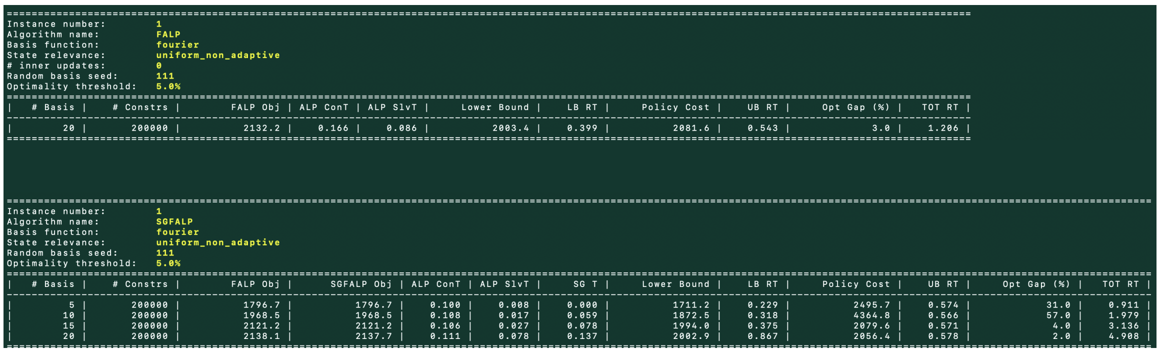
activate your license. Refer to this troubleshooting page for assistance with resolving this issue. If the issue persists, consider installing Gurobi using `conda` instead of `pip`. You can use the following code as an example:

```
conda install -c gurobi gurobi
grbgetkey xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

10. Provided that the `ALP` environment and Gurobi are correctly installed, you can employ the following code to solve test instances of perishable inventory control application. Please run the following code:

    ```
    ./run_PIC.sh
    ```

    The `run_PIC.sh` file solves the first instance of the perishable inventory control problem using the FALP algorithm with 20 random Fourier features, where FALP is fomrulated using a uniform state-relevance distribution (refer to the paper and code for details). You can find the specifications (e.g., cost parameters) of this instance under the path `MDP/PIC/Instances/instance_1.py`. To solve this instance using an alternate algorithm, you can modify the file `run_PIC.sh`. For instance, changing the `algo_name` from `FALP` to `SG-FALP` in this file and rerunning `run_PIC.sh` will display the output for the self-guided FALP algorithm applied to this instance. A screenshot of the output of these algorithms is attached below:



```
==========================================================================================================================
Instance number:          1
Algorithm name:           FALP
Basis function:           fourier
State relevance:          uniform_non_adaptive
# inner updates:          0
Random basis seed:        111
Optimality threshold:     5.0%
==========================================================================================================================
|   # Basis |    # Constrs |         FALP Obj | ALP ConT | ALP SlvT |     Lower Bound |    LB RT |     Policy Cost |    UB RT |      Opt Gap (%) |   TOT RT |
--------------------------------------------------------------------------------------------------------------------------
|        20 |       200000 |           2132.2 |    0.166 |    0.086 |          2003.4 |    0.399 |          2081.6 |    0.543 |              3.0 |    1.206 |
==========================================================================================================================



==================================================================================================================================================
Instance number:          1
Algorithm name:           SGFALP
Basis function:           fourier
State relevance:          uniform_non_adaptive
Random basis seed:        111
Optimality threshold:     5.0%
==================================================================================================================================================
|   # Basis |    # Constrs |         FALP Obj |       SGFALP Obj | ALP ConT | ALP SlvT |     SG T |     Lower Bound |    LB RT |     Policy Cost |    UB RT |      Opt Gap (%) |   TOT RT |
--------------------------------------------------------------------------------------------------------------------------------------------------
|         5 |       200000 |           1796.7 |           1796.7 |    0.100 |    0.008 |    0.000 |          1711.2 |    0.229 |          2495.7 |    0.574 |             31.0 |    0.911 |
|        10 |       200000 |           1968.5 |           1968.5 |    0.108 |    0.017 |    0.059 |          1872.5 |    0.318 |          4364.8 |    0.566 |             57.0 |    1.979 |
|        15 |       200000 |           2121.2 |           2121.2 |    0.106 |    0.027 |    0.078 |          1994.0 |    0.375 |          2079.6 |    0.571 |              4.0 |    3.136 |
|        20 |       200000 |           2138.1 |           2137.7 |    0.111 |    0.078 |    0.137 |          2002.9 |    0.867 |          2056.4 |    0.578 |              2.0 |    4.908 |
==================================================================================================================================================
```

11. To solve the test instance of Bermudan options pricing problem, please run `run_BerOpt.sh` file.

How to adap this repository?

To apply the algorithms from this repository to other problem instances or customize algorithm configurations, users should make adjustments within the `run_PIC.sh` and `run_BerOpt.sh` files. These files enable the modification of algorithm and simulation parameters. For example, a copy of `run_PIC.sh` is provided below:

```
#!/usr/bin/env bash
mdp_name                    ='PIC'
algo_name                   ='FALP'             #FALP, PG-FALP, SG-
FALP
```

```
basis_func_type             ='fourier'          #relu, fourier, lns,
stump
max_num_constr              =200000
max_basis_num               =20
batch_size                  =5
num_cpu_core                =8
state_relevance_inner_itr   =5
for instance_number in 1; do
    for seed in 111; do
        python main_PIC.py $mdp_name $algo_name $basis_func_type
$instance_number $max_num_constr $max_basis_num \
            $batch_size $num_cpu_core $seed
$state_relevance_inner_itr
    done
done
```

Appendix

This code has undergone testing on two systems:

- 2021 MacBook Pro:
  - CPU: M1 Pro
  - Memory: 16 GB
  - OS: Mac OS 14.1.2
- 2022 Mac Studio:
  - CPU: M1 Max
  - Memory: 64 GB
  - OS: Mac OS 14.1.1

Below is the list of Python package versions utilized during the testing phase of this repository:

Package	Version
autograd	1.6.2
emcee	3.1.4
future	0.18.3
gurobipy	11.0.0
importlib	1.0.4
llvmlite	0.41.1
nengo	4.0.0
numba	0.58.1
numpy	1.26.2
pandas	2.1.4

Package	Version
pip	23.3.1
python-dateutil	2.8.2
pytz	2023.3.post1
sampyl-mcmc	0.3
scipy	1.11.4
setuptools	63.2.0
six	1.16.0
tqdm	4.66.1
tzdata	2023.3