Efficient Linear System Solvers for Least-Squares Temporal Difference (LSTD) Methods

Mrudani Pimpalkhare (2023101133), Bhavya Ahuja (2023111035), Vishal Rao (2023101091)

International Institute of International Technology, Hyderabad

Abstract :

Least-Squares Temporal Difference (LSTD) methods are widely used in reinforcement learning (RL) for policy evaluation by estimating the value function of a given policy. Unlike standard Temporal Difference (TD) methods, which update value estimates incrementally, LSTD solves for the value function by solving a linear system of equations. The computational cost of solving this system increases significantly with the size of the state space, necessitating the development of efficient solvers for practical applications. The goal of this project is to investigate the structure of the linear system in LSTD and explore efficient numerical techniques for solving it. We focus on special properties of the system, such as sparsity and low rank, and evaluate methods such as deflation, domain decomposition, and multigrid solvers to improve efficiency. We also explore how the structure of this system varies with different RL environments, such as CartPole, MountainCar, GridWorld, etc.

Index Terms: LSTD, Reinforcement Learning, Linear System.

1 Introduction

To evaluate a fixed policy π in reinforcement learning, the goal is to estimate the value function $V^\pi(s)$, which represents the expected discounted return starting from state s and following policy π thereafter. This value function satisfies the Bellman equation and is often difficult to solve exactly in environments with large or continuous state spaces. The Least Squares Temporal Difference (LSTD) method provides a sample-efficient way to estimate $V^\pi(s)$ by approximating it using a linear combination of features. This results in a system of linear equations that can be solved to obtain the best-fit weights for the value function under the given policy.

$$V^\pi(s) = \mathbb{E}[r(s,a) + \gamma V^\pi(s') \mid s, a], \tag{1}$$

where s is the state, a is the action, s' is the next state, and γ is the discount factor.

In LSTD(λ), we approximate the value function using a linear feature representation:

$$V^\pi(s) \approx \phi(s)^T w, \tag{2}$$

where $\phi(s)$ is the feature vector for state s and w is the weight vector.

The core computation in LSTD involves solving the following linear system:

$$Aw = b, \tag{3}$$

where:

$$A = \sum_{t=0}^{T} \phi(s_t)\left(\phi(s_t) - \gamma\phi(s_{t+1})\right)^T, \qquad b = \sum_{t=0}^{T} \phi(s_t) r_t. \tag{4}$$

This system is high-dimensional and can be challenging to solve efficiently when the feature dimension is large.

1.1 Task

To evaluate the performance and scalability of various solvers for the LSTD linear system, we propose the following action plan:

- **Iterative Solvers**:
  - Apply **Conjugate Gradient (CG)** for symmetric positive definite matrices. Assess performance with and without preconditioning (e.g., ILU).
  - Use **GMRES** for cases where the matrix A may be non-symmetric (less common in LSTD, but worth considering for extensions).
- **Deflation-Based Solvers**:
  - Implement **Deflated Conjugate Gradient** to project out low-eigenvalue directions that cause ill-conditioning.
  - Identify and eliminate nearly redundant features to improve numerical stability and effective rank of matrix A.
- **Multilevel Methods (PyAMG)**:
  - Evaluate `PyAMG` (Algebraic Multigrid) to solve large sparse systems using multilevel hierarchy.
  - Investigate how well PyAMG builds coarse-to-fine hierarchies depending on the sparsity and structure of A.
- **Baseline Solvers for Comparison**:
  - Include **LU** and **QR decomposition** as direct methods to establish baseline accuracy and runtime, especially for small to medium-sized matrices.
- **Evaluation across Multiple Environments**:
  - Test all methods on a range of reinforcement learning environments:
    * `MountainCar-v0` — continuous, smooth transitions
    * `CartPole-v1` — continuous, unstable transitions
    * `FrozenLake-v1` — discrete, grid-like structure
    * `Acrobot-v1` — chaotic, highly nonlinear transitions
  - For each environment, analyze the structure of matrix A:

  - **Rank** — affected by feature richness and state coverage
  - **Sparsity** — depends on basis functions (tabular vs polynomial)
  - **Conditioning** — sensitive to redundant features or small eigenvalues
- Determine which solver performs best for each environment and justify based on the observed properties of A.

1.2 What this report contains:

In the following sections, we will be discussing each environment in detail and the results obtained from it. Further, we will analyse these environments, and generate a flow-chart of some sort, to help predict the best solvers for the envrionments that have not been tested.

2 Testing on different environments

2.1 MountainCar-v0

The MountainCar environment is a classic reinforcement learning task involving a car situated between two hills. The agent's goal is to drive up the right hill, but it does not have enough power to do so directly and must build momentum by going back and forth. The state space is continuous, defined by the car's position and velocity. The reward is sparse (−1 per step until the goal is reached), and transitions are smooth and deterministic.

Solver Analysis PyAMG was seen to have the best accuracy over the other solvers, followed by LU and QR. The polynomial feature expansion for a continuous environment can create a large, structured system. Multigrid methods thrive on systems that have some notion of continuity or grid-like adjacency (even if it's in feature space). PyAMG can exploit this structure by smoothing out long-range errors quickly at coarse levels.

Matrix Properties Smoother transitions in the environment, like in MountainCar, lead to better conditioning. This is because the eigenvalues are more uniformly spread out. More episodes, lead to more coverage, which may lead to a full rank matrix, or maybe slightly lower than full This helps PyAMG due to gradual variation in values across neighboring states

2.2 FrozenLake-v1 (GridWorld)

FrozenLake represents a discrete gridworld with slippery transitions. The agent must navigate from a start position to a goal while avoiding holes. The state space is discrete (usually 4×4 or 8×8), and transitions are stochastic in slippery mode. Rewards are sparse, and movement is constrained to grid directions.

Solver Analysis . PyAMG leverages the grid like structure of the environment to build a multi-grid hierarchy easily, and performs the best. CG-ILU (Incomplete LU) is a preconditioner that approximates LU decomposition, improving convergence — but only when the matrix structure supports it. In this case, it does not.

Matrix Properties Environments that follow one-hot encoding, eg. GridWorld generate sparse matrices. If all states are visited, then the rank is full. This leads to sparse matrices, well conditioned if the transitions are not too chaotic, this helps the pyAMG perform well for larger matrices. CG-ILU performs badly on FrozenLake because the matrix A is small, highly sparse, non-smooth, and non-diagonally dominant. ILU can't construct a good preconditioner, and CG doesn't gain from its usual benefits in this regime.

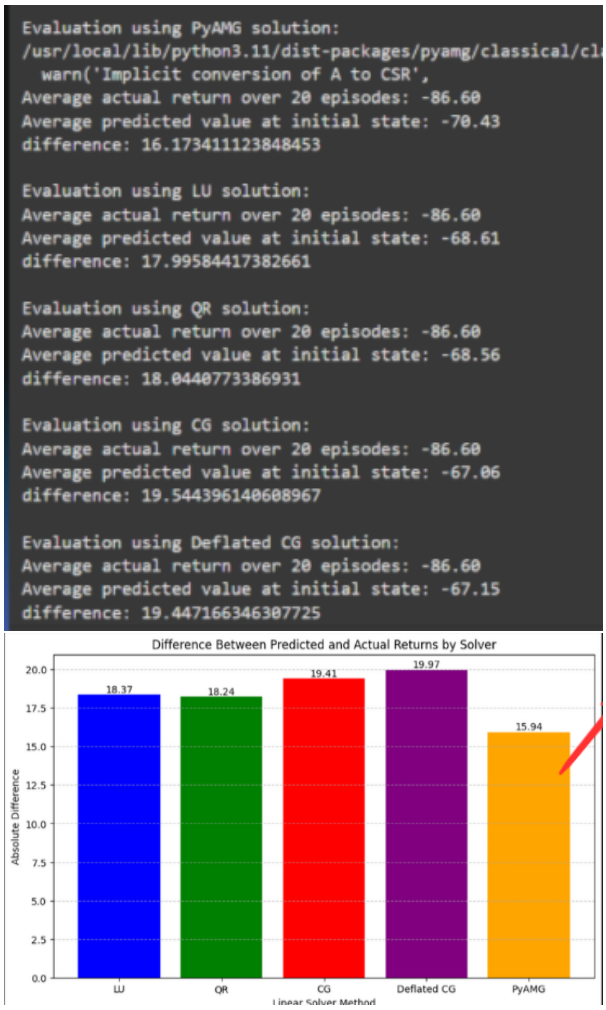

Figure 1. Result metrics and solver performance for MountainCar-v0

2.3 CartPole-v1

In this environment, the agent controls a cart that must balance a pole by applying force to the left or right. The state space is continuous and includes position, velocity, pole angle, and angular velocity. The dynamics are unstable, with frequent early episode termination under random policies.

Solver Analysis For the reason explained below, it was seen that for the small degree of characteristics, LU and QR performed well. Although pyAMG can solve large systems effectively, the overhead of building and cycling through a multigrid hierarchy on a small system does not pay off. You see a big spike in computation time, which is not shown in the report.

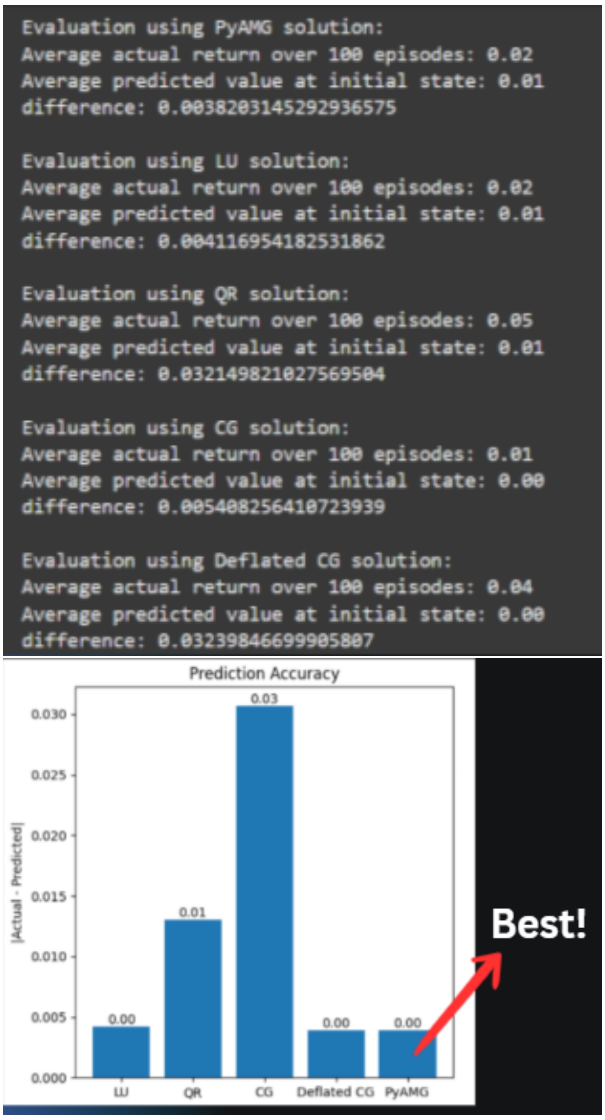

Figure 2. Solver comparison and value estimates on FrozenLake

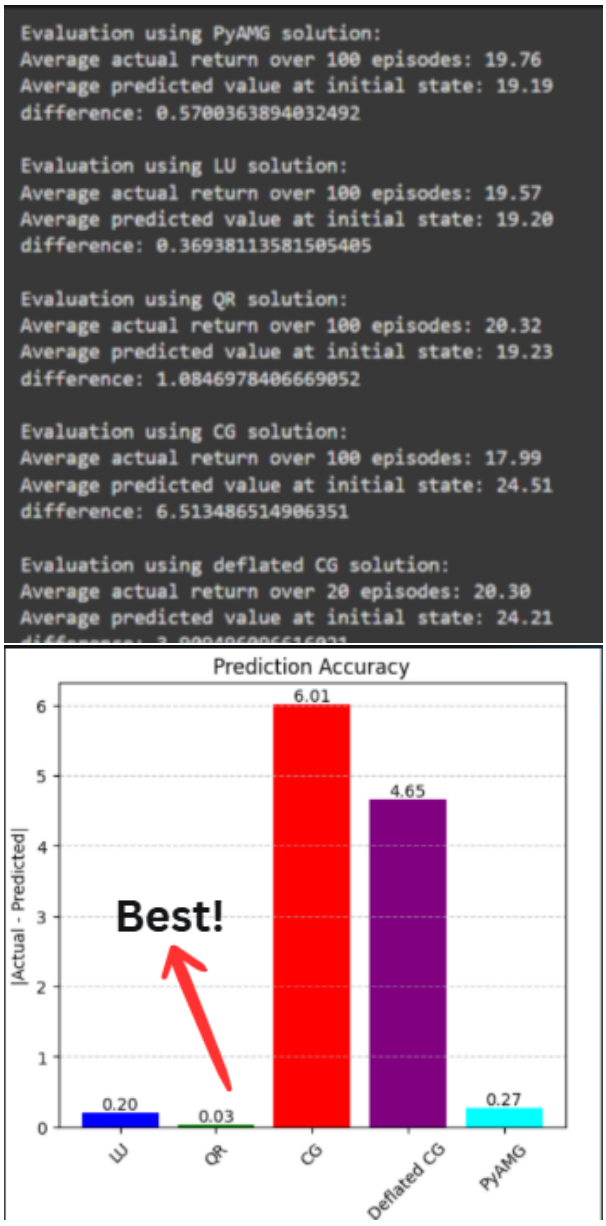

Figure 3. Solver performance and predicted value plots for CartPole-v1

Matrix Properties Chaotic, non-linear, and unstable dynamics lead to ill-conditioned matrices If the transitions are not smooth and the number of episodes are low, often the rank is low too. This also leads to dense and noisy matrices. The advantages of our other Linear Solvers cannot be leveraged, hence for small n, LU and QR suffice. However, when n is large, one may take advantage of the fact that only some eigenvalues dominate, and hence use Delfator CG.

2.4 Acrobot-v1

Acrobot is a two-link, underactuated robot that must swing its end above a threshold. The state space includes joint angles and angular velocities. The dynamics is highly non-linear and chaotic, resulting in noisy feature transitions and difficult exploration.

Solver Analysis Direct solvers like LU and QR exhibit superior performance on Acrobot. This is likely because the system matrix is of moderate size and dense, making deterministic factorization methods efficient and stable. Iterative methods such as CG, Deflated CG, and PyAMG underperform, possibly due to the overhead involved in preconditioning or building multilevel hierarchies, which do not pay off at this scale. Deflated CG shows no clear advantage here, suggesting that low-eigenvalue directions may not be the dominant source of error in this case.

Matrix Properties The matrix A derived from the Acrobot environment is dense and high-dimensional due to the chaotic, non-linear dynamics and complex feature representations. It is also poorly conditioned, making it harder for iterative methods to converge efficiently without precise preconditioning or deflation techniques.

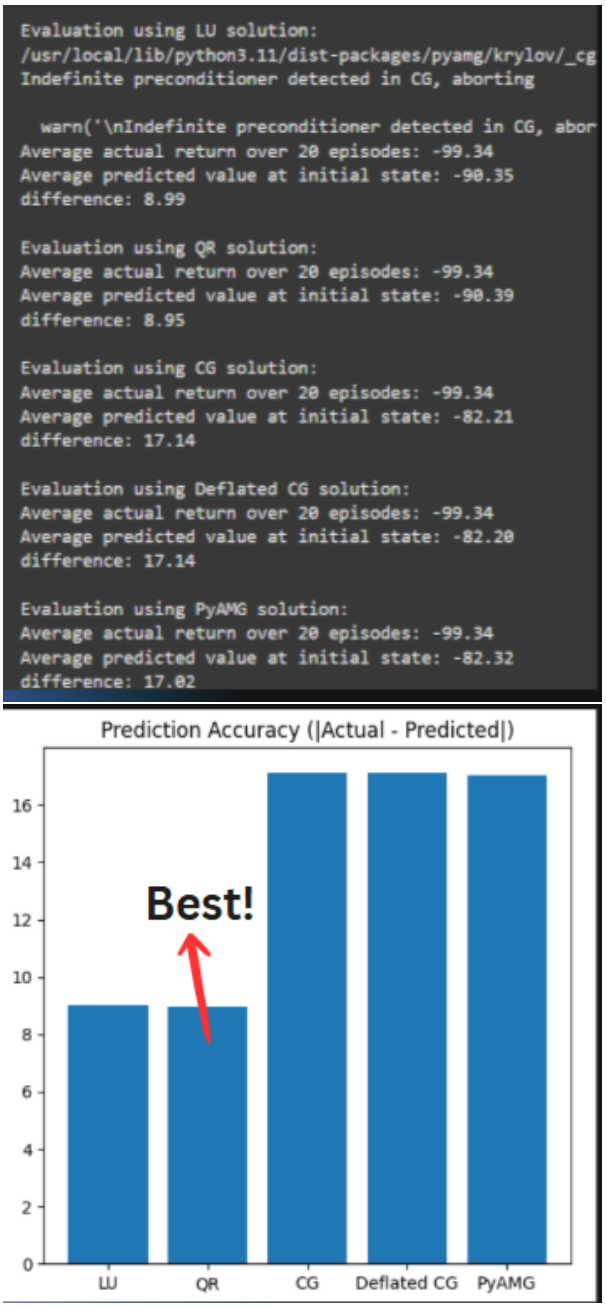

Figure 4. Solver and matrix performance on Acrobot-v1

2.5 Pendulum-v1

The Pendulum environment features a single pendulum that must be swung upright and balanced. It is a continuous control task with a 3D state space (angle, angular velocity, cosine/sine representation). Transitions are smooth, and rewards are shaped.

Solver Analysis LU and QR perform best on Pendulum-v1 in terms of prediction accuracy (smallest difference between actual and predicted returns). This is likely because the feature space is not extremely high-dimensional, and the dynamics are smooth, enabling direct solvers to compute stable and accurate solutions efficiently.

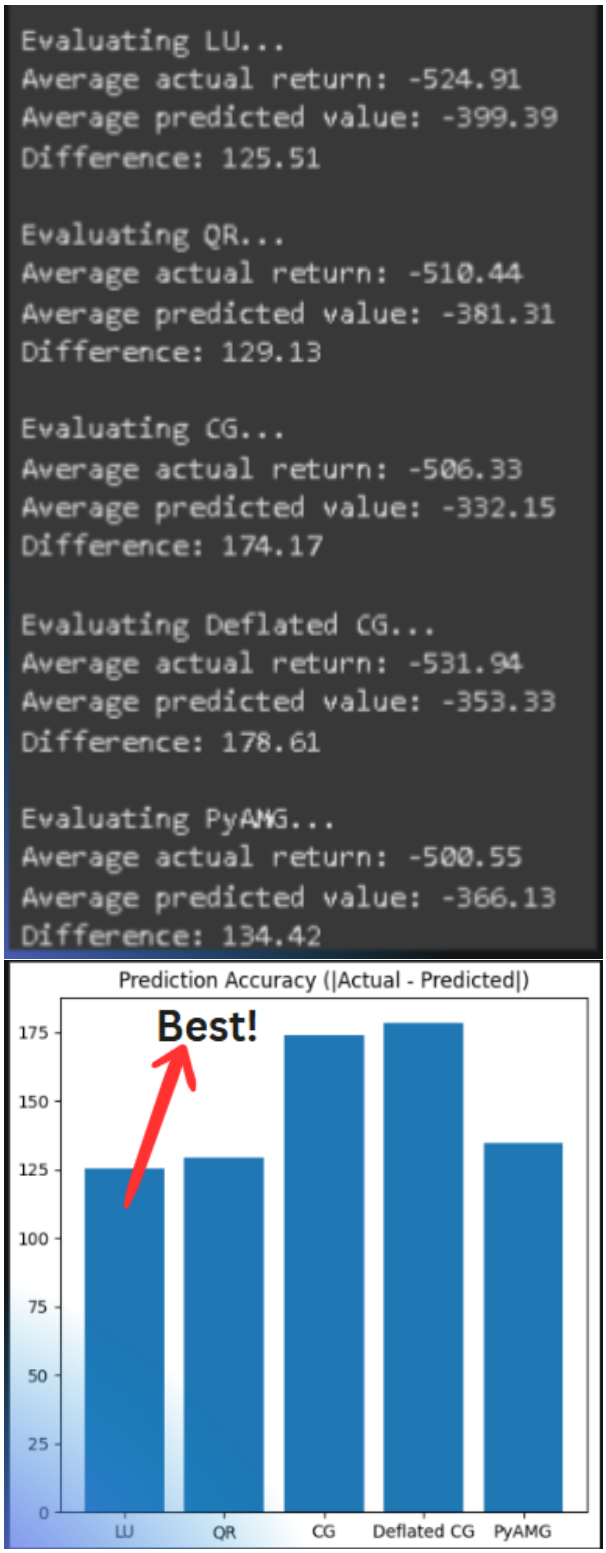

Figure 5. Results and solver timings for Pendulum-v1

Deflated CG and CG perform worse in accuracy and also take longer to compute. Although deflation typically helps with ill-

conditioning, in this case, it adds computational overhead without significant improvement — suggesting that the matrix is not strongly affected by low-eigenvalue directions.

Interestingly, PyAMG performs reasonably well — both in accuracy and time — suggesting that the multigrid hierarchy benefits from Pendulum's smooth, continuous dynamics. However, it still does not outperform LU/QR in precision.

Matrix Properties The matrix A for Pendulum is dense but fairly well-conditioned, likely due to the environment's smooth and continuous transitions. The use of high-degree polynomial features increases dimensionality, but the lack of abrupt transitions results in stable outer products, making A easier to solve with direct methods. While not extremely sparse, the matrix structure is still conducive to PyAMG's hierarchy, albeit not enough to outperform LU/QR in small to medium scales.

.

3 Rank and Sparsity: How the Environment Affects Matrix A

The structure and quality of the matrix A in LSTD are highly dependent on the properties of the environment and the chosen feature representation.

Chaotic and Unstable Environments

In environments such as `CartPole-v1` and `Acrobot-v1`, the dynamics are chaotic and nonlinear. Transitions are often unstable, and episodes terminate early under random policies. These issues result in:

- **Ill-conditioned matrices**: small or zero eigenvalues dominate.
- **Low-rank behavior**: few directions in feature space are explored due to poor coverage.
- **Dense and noisy matrices**: due to high variance in $\phi(s_t) - \gamma\phi(s_{t+1})$.

In such cases, direct solvers like LU and QR perform best due to their robustness, while the benefits of iterative methods are minimal unless n is large and deflation becomes advantageous.

Smoother, Structured Environments

In smoother environments such as `MountainCar-v0` or `Pendulum-v1`, the transitions evolve gradually and the feature space is more continuously covered. With enough episodes, this leads to:

- **Higher effective rank**: more directions in feature space are populated.
- **Better conditioning**: eigenvalues are more evenly distributed.
- **Moderately sparse but structured matrices**: especially with polynomial features.

These properties are ideal for solvers like CG with ILU, and multilevel solvers like PyAMG, which benefit from smoother dynamics and coherent structure.

Sparse and Tabular Environments

Grid-based environments such as `FrozenLake-v1` use tabular (one-hot) features:

$$A = \sum_t \phi(s_t)\,[\phi(s_t) - \gamma\phi(s_{t+1})]^T$$

Here, each $\phi(s_t)$ is a one-hot encoded vector, making A naturally sparse. If all states are visited sufficiently, the rank can be full. The sparsity and grid-like structure make these matrices suitable for solvers like PyAMG, which exploit spatial locality to build efficient multigrid hierarchies.

Key Takeaways

- Low-rank, ill-conditioned, or dense matrices favor direct solvers.
- Well-conditioned, smooth matrices with good state coverage benefit iterative solvers and preconditioning.
- Structured sparsity and grid locality make multigrid-based methods like PyAMG effective.

4 Summary: Crafting an Ideal Solver Selection Strategy

Selecting the appropriate solver for the LSTD linear system depends heavily on the environment's properties and the structure of the resulting matrix A. Based on our evaluations across environments like `MountainCar`, `FrozenLake`, `CartPole`, and `Acrobot`, we propose a decision framework to guide solver selection.

- Use **LU/QR** for small, dense matrices where accuracy and stability are critical.
- Use **Deflated CG** when the matrix is large and ill-conditioned, dominated by a few troublesome eigenvalues.
- Use **PyAMG** when the matrix is sparse and structured, as in environments with smooth transitions or grid-like layouts.

This algorithmic approach allows us to combine structural insight about the environment with empirical solver behavior to make an informed, efficient choice.

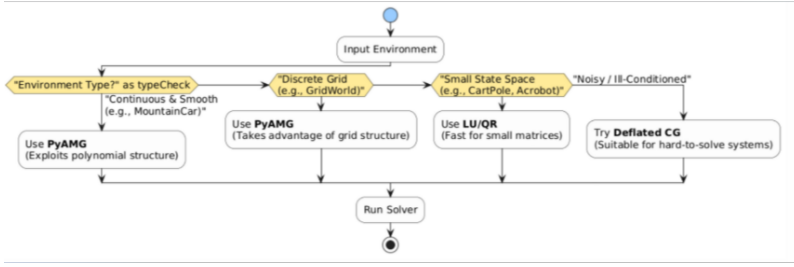

Figure 6. Decision flowchart for choosing the ideal solver based on environment properties.

Acknowledgements

This research received support during the Numerical Algorithms course, instructed by Professor Pawan Kumar