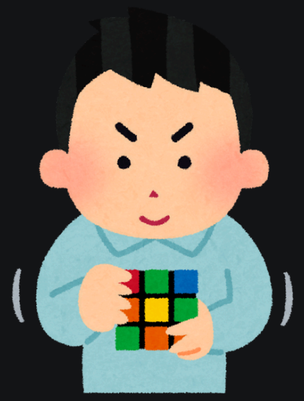
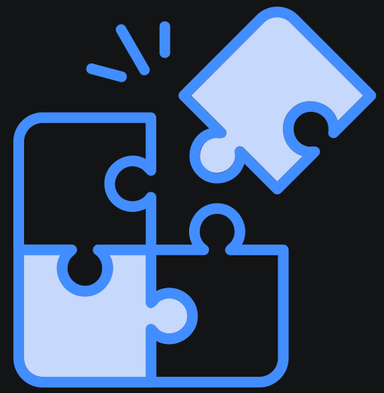


NUMERICAL ALGORITHMS
PHASE 3

LSTD Solvers



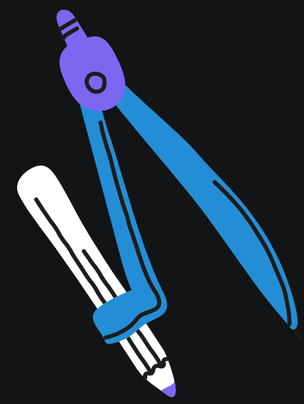
In search for the best Linear Solver across many RL environments

Bhavya Ahuja
2023111035

Mrudani Pimpalkhare
2023101133

Vishal Rao
2023101091

Objective



To investigate the structure of the linear system in LSTD and explore efficient numerical techniques for solving it.

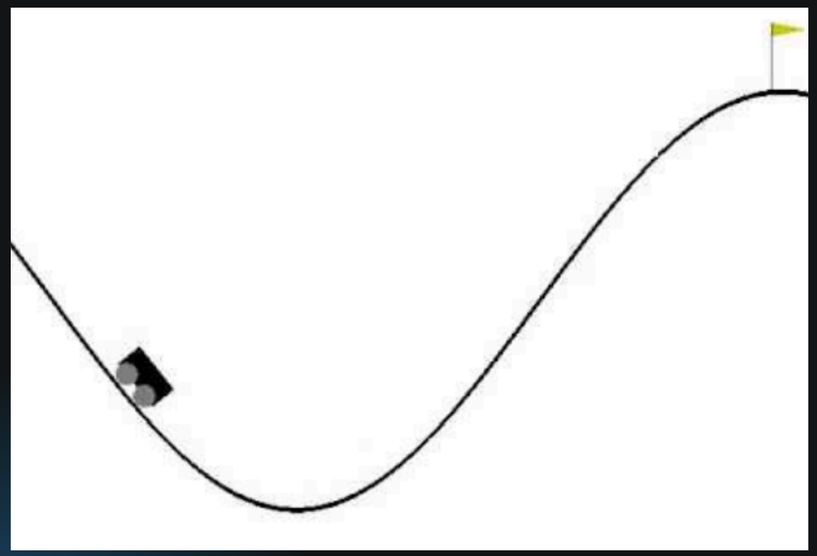
Solvers we tried:

- LU
- QR
- ILU with CG (conjugate gradient)
- pyAMG (multigrid with CG)
- Deflation

Environments we tried:

- MountainCar
- CartPole
- GridWorld
- Scrobot

1. MountainCar (Continuous State, Polynomial Features)



Nature: The state space is continuous (position, velocity). We typically approximate it with polynomial or other basis features, leading to a (potentially) large or ill-conditioned linear system.

Key Challenge: Escaping the negative region (the valley) by building enough momentum—thus sampling can produce tricky transitions, and the resulting feature matrix can be quite large.

MountainCar : Results

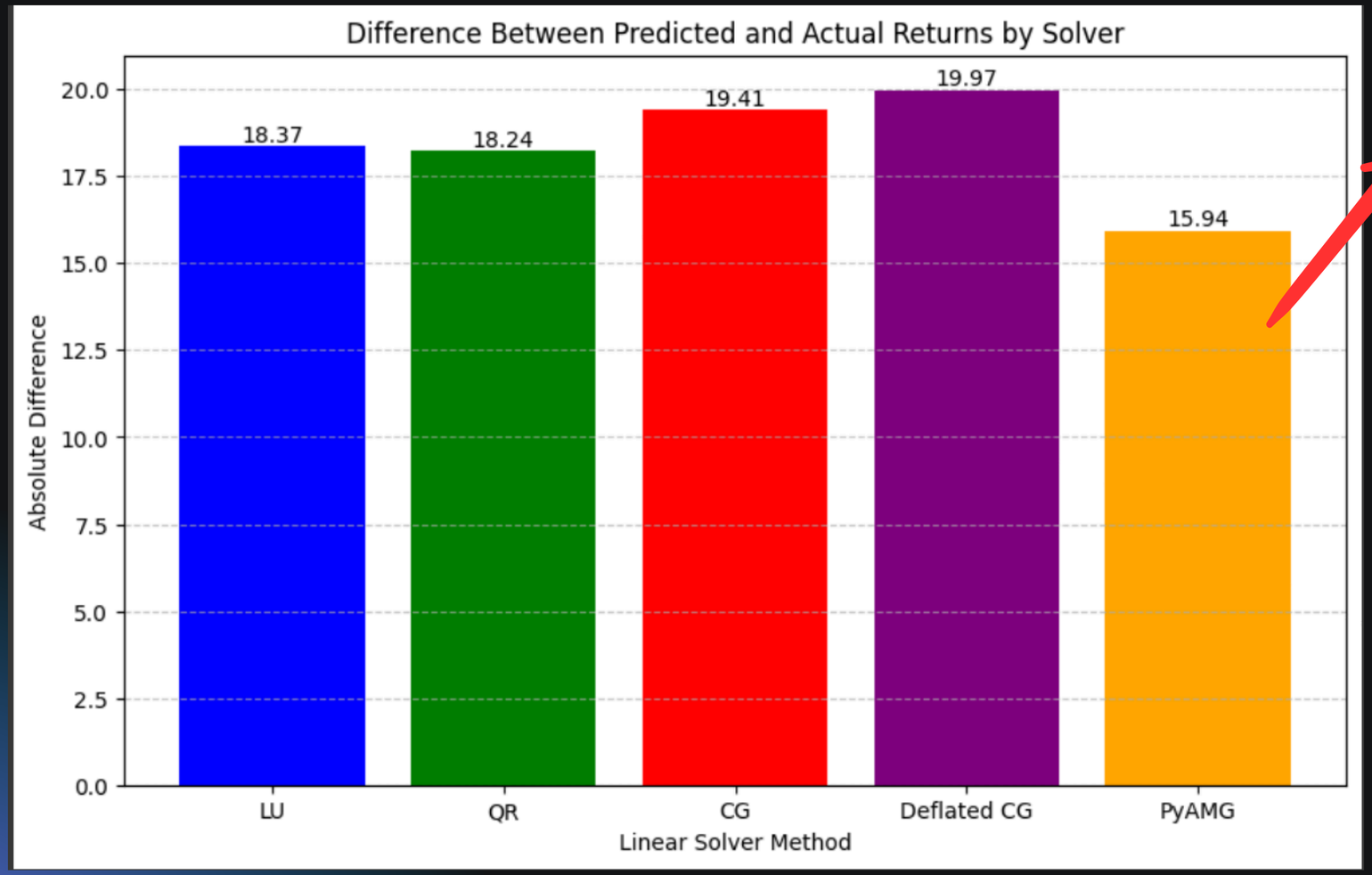
```
Evaluation using PyAMG solution:
/usr/local/lib/python3.11/dist-packages/pyamg/classical/cla
  warn('Implicit conversion of A to CSR',
Average actual return over 20 episodes: -86.60
Average predicted value at initial state: -70.43
difference: 16.173411123848453

Evaluation using LU solution:
Average actual return over 20 episodes: -86.60
Average predicted value at initial state: -68.61
difference: 17.99584417382661

Evaluation using QR solution:
Average actual return over 20 episodes: -86.60
Average predicted value at initial state: -68.56
difference: 18.0440773386931

Evaluation using CG solution:
Average actual return over 20 episodes: -86.60
Average predicted value at initial state: -67.06
difference: 19.544396140608967

Evaluation using Deflated CG solution:
Average actual return over 20 episodes: -86.60
Average predicted value at initial state: -67.15
difference: 19.447166346307725
```

Speed :

Not a surprise that LU and QR methods were the fastest.

They were followed by Deflated CG → PyAMG → CG with ILU

Performance → Visualized

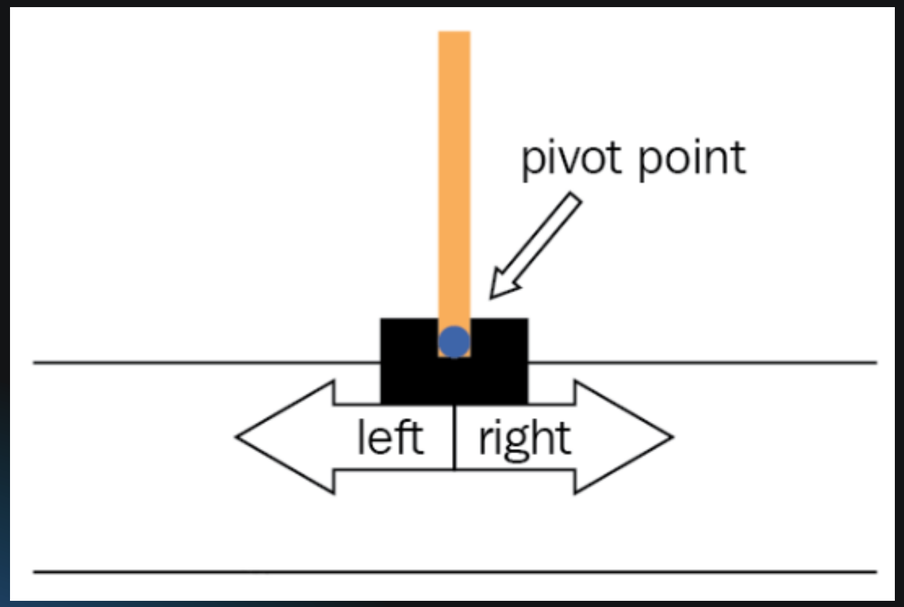


Best!

Why PyAMG Excels Here:
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.

2. CartPole (Continuous State, Low-Dimensional)

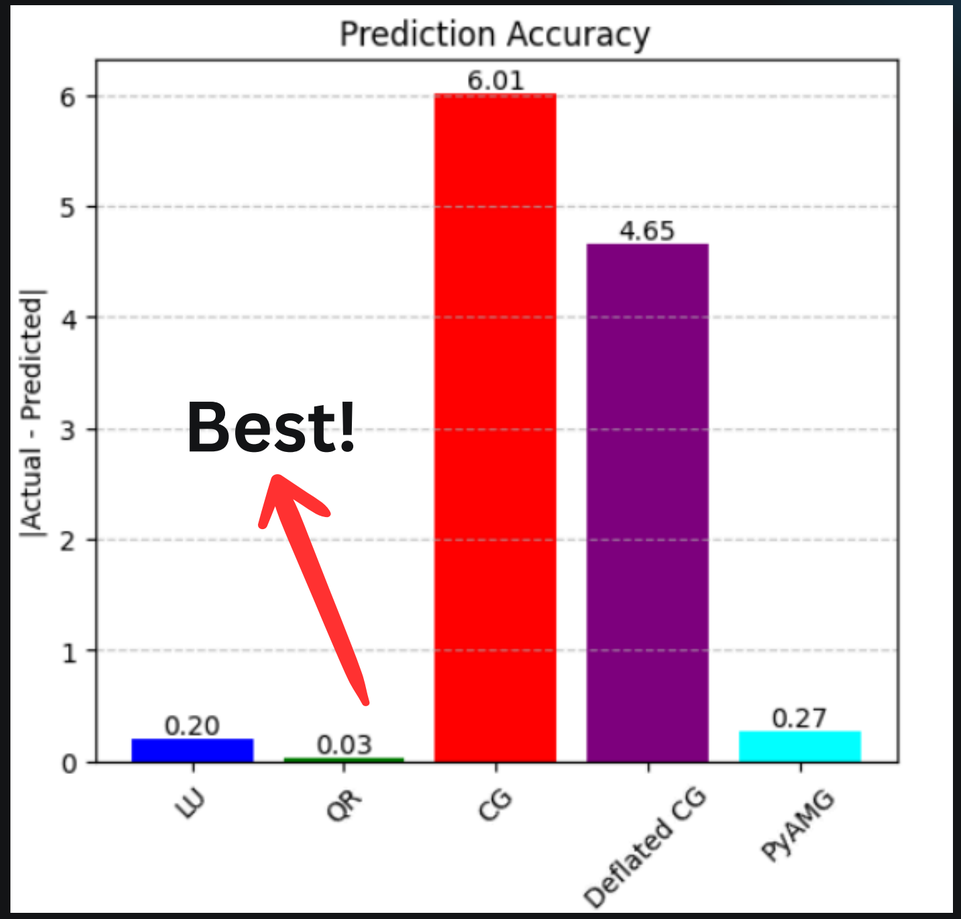


Nature: Balancing a pole hinged on a cart. Features often include position, velocity, angle, angular velocity.

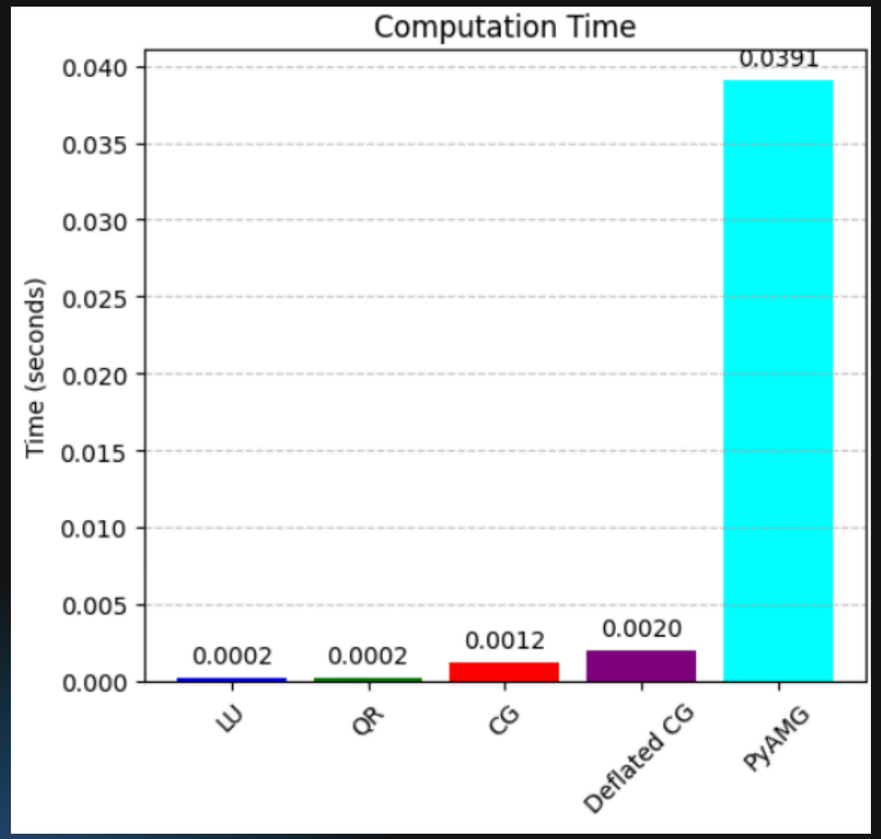
Key Challenge: The environment is relatively small-dimensional, so the feature matrix is not excessively large. However, if the policy is random, the data might still be somewhat noisy.

CartPole : Results

```
Evaluation using PyAMG solution:
Average actual return over 100 episodes: 19.76
Average predicted value at initial state: 19.19
difference: 0.5700363894032492

Evaluation using LU solution:
Average actual return over 100 episodes: 19.57
Average predicted value at initial state: 19.20
difference: 0.36938113581505405

Evaluation using QR solution:
Average actual return over 100 episodes: 20.32
Average predicted value at initial state: 19.23
difference: 1.0846978406669052

Evaluation using CG solution:
Average actual return over 100 episodes: 17.99
Average predicted value at initial state: 24.51
difference: 6.513486514906351

Evaluation using deflated CG solution:
Average actual return over 20 episodes: 20.30
Average predicted value at initial state: 24.21
difference: 3.909496096616021
```



Time :



Why QR is So Good Here:

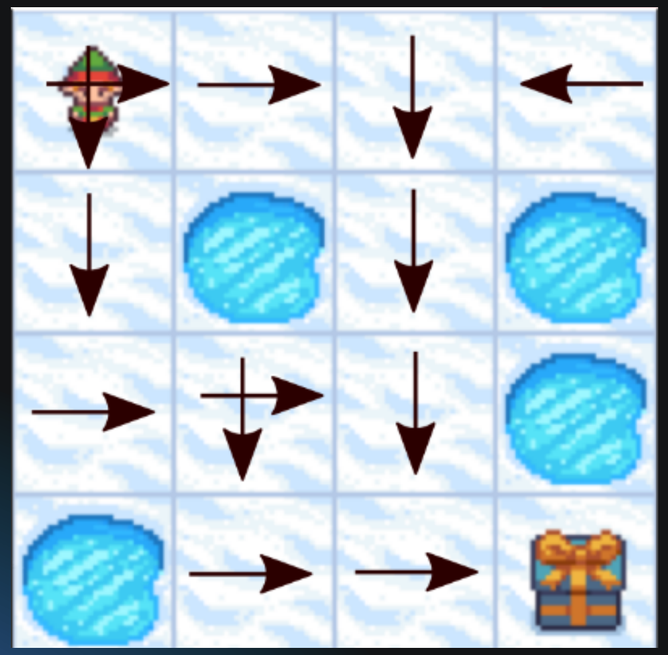
CartPole's state space dimension is relatively small (just four state variables, plus expansions). The matrix is smaller and likely better conditioned.

QR factorization is a robust approach to least-squares for such well-conditioned, moderate-sized problems, hence near-best accuracy.

Why PyAMG is Slower:

Though it can solve large systems effectively, the overhead of building and cycling through a multigrid hierarchy on a smallish system does not pay off. You see a big spike in compute time.

3. GridWorld (Discrete State, One-Hot Features)



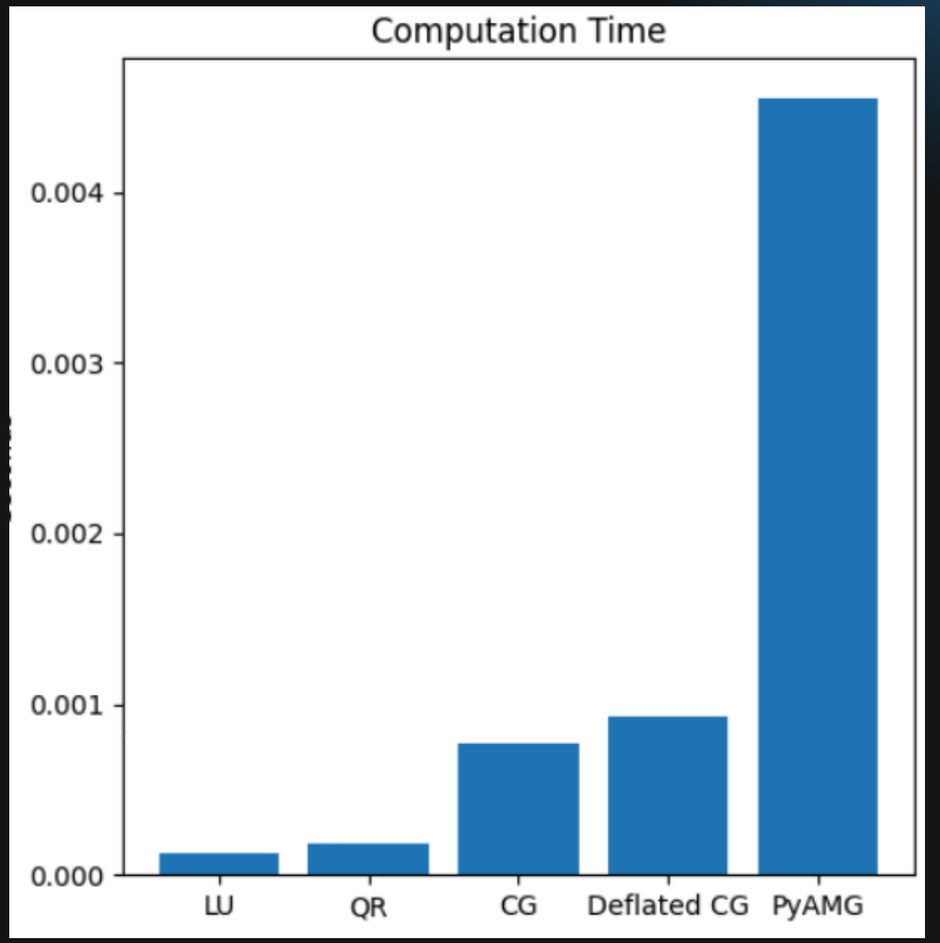
Nature: A tabular, discrete environment (here, FrozenLake). Each cell in the grid is a state, and the agent typically moves up/down/left/right with possible slip or randomness.

Key Challenge:

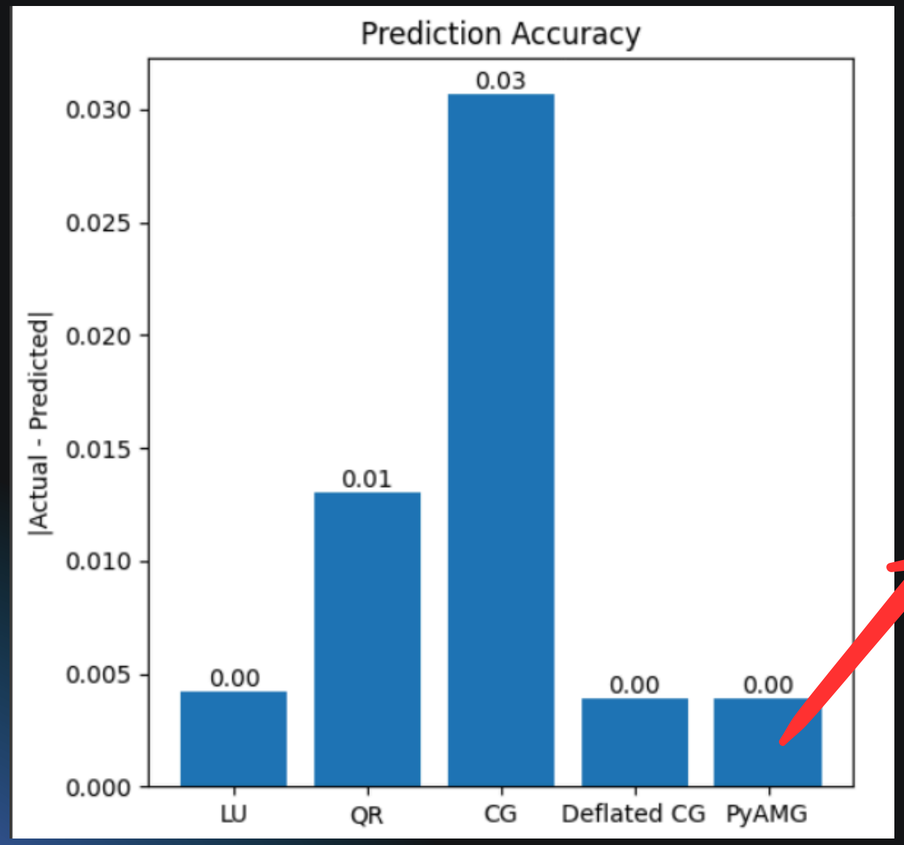
- Extremely sparse rewards.
- Most transitions yield a reward of 0.
- The chance of the agent actually reaching the goal (and earning a nonzero reward) is very low.
- Many episodes end with a reward of 0, which, when averaged over many episodes, gives a very low (or near zero) average return.
- Because rewards are so sparse, a random policy rarely reaches the goal.

GridWorld : Results

```
Evaluation using PyAMG solution:
Average actual return over 100 episodes: 0.02
Average predicted value at initial state: 0.01
difference: 0.0038203145292936575

Evaluation using LU solution:
Average actual return over 100 episodes: 0.02
Average predicted value at initial state: 0.01
difference: 0.004116954182531862

Evaluation using QR solution:
Average actual return over 100 episodes: 0.05
Average predicted value at initial state: 0.01
difference: 0.032149821027569504

Evaluation using CG solution:
Average actual return over 100 episodes: 0.01
Average predicted value at initial state: 0.00
difference: 0.005408256410723939

Evaluation using Deflated CG solution:
Average actual return over 100 episodes: 0.04
Average predicted value at initial state: 0.00
difference: 0.03239846699905807
```



Performance → Visualized

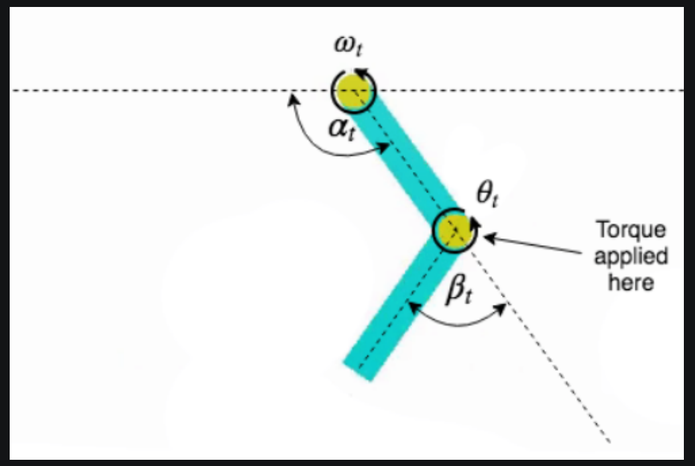


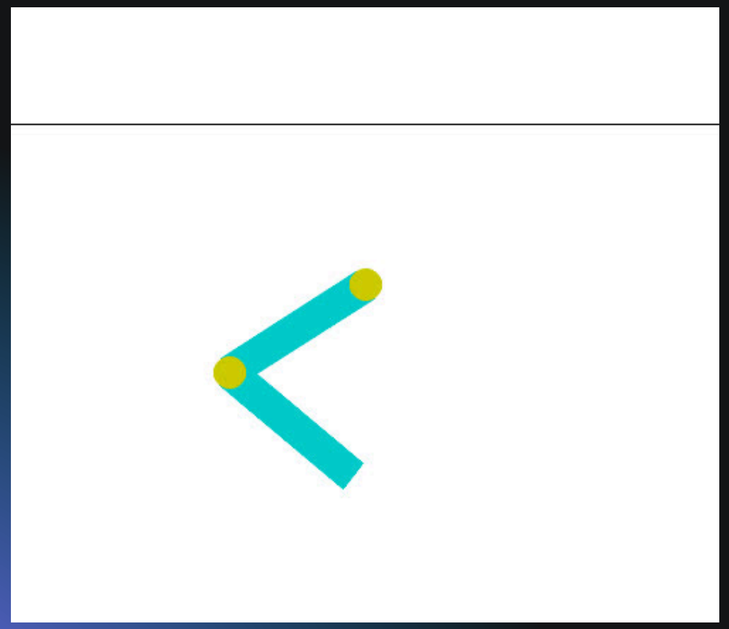
Best!

Why PyAMG excels here?

- GridWorld is literally a grid, and algebraic multigrid methods are designed to handle problems that naturally have a grid or graph-like structure.
- Once the hierarchy is built, PyAMG captures long-range connections in the state space especially well, leading to a near-perfect solution.

4. Acrobot





Nature: A two-link pendulum system where only the joint between the links is actuated. The goal is to swing the lower end above a certain height.

State Space: Includes cosine/sine of both joint angles and their angular velocities (6 values).

Action Space: Discrete torques: -1, 0, or +1.

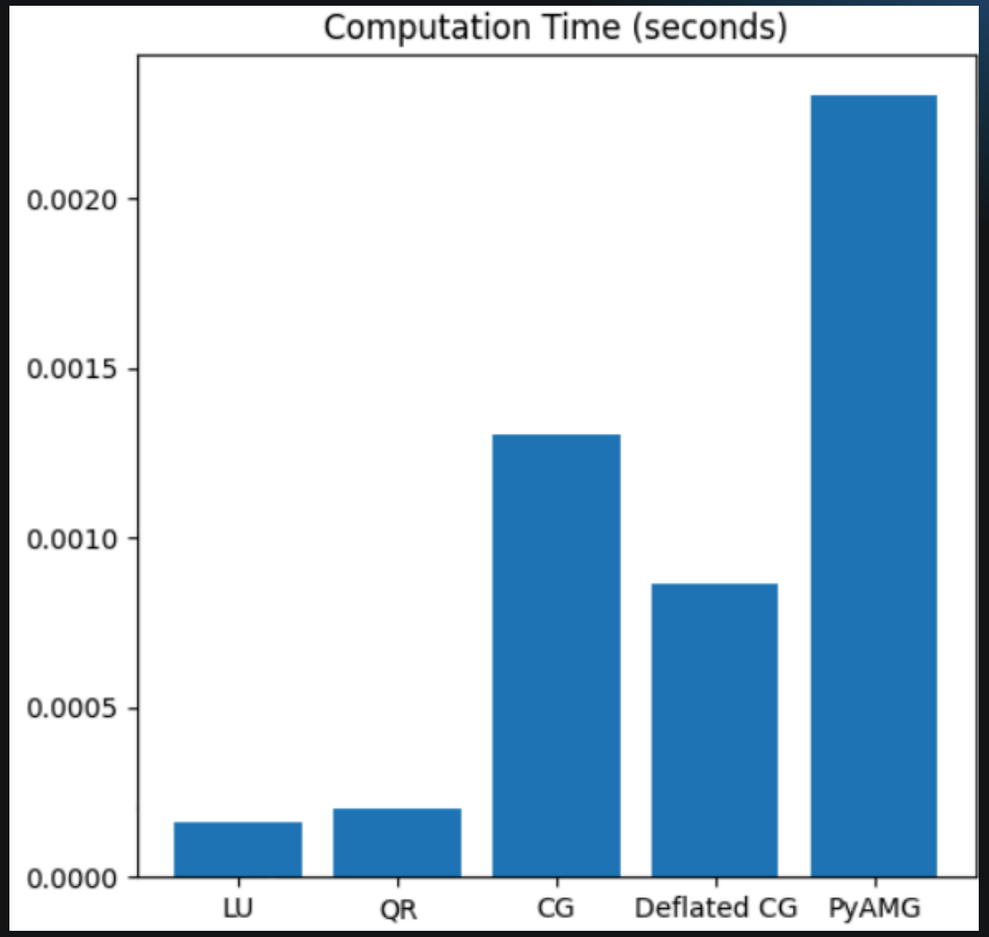
Reward: -1 at each step until the goal is achieved (sparse reward).

Key Challenges

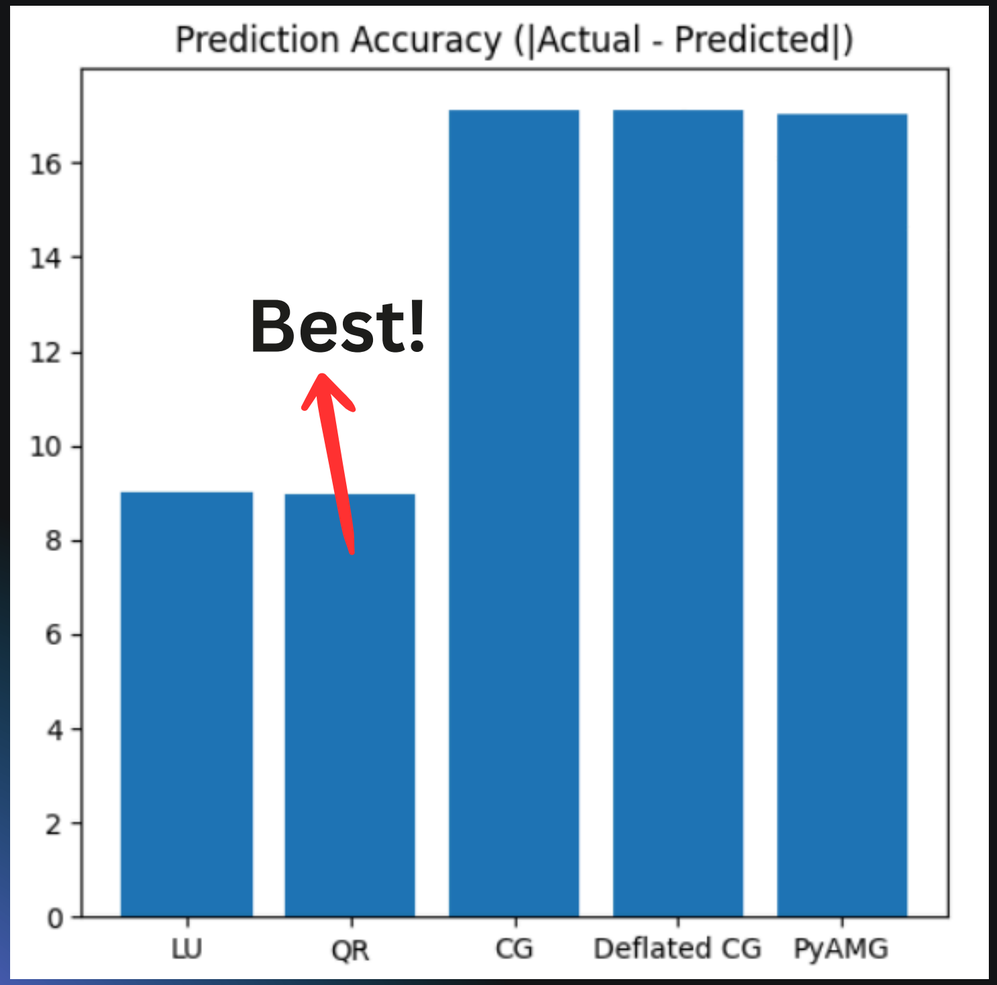
- **Nonlinear Dynamics:** Complex motion that's hard to predict.
- **Sparse Rewards**: Agent gets little feedback until success.
- **Efficient Exploration:** Hard to find the right actions without guidance.

Acrobot : Results

```
Evaluation using LU solution:
/usr/local/lib/python3.11/dist-packages/pyamg/krylov/_cg
Indefinite preconditioner detected in CG, aborting

  warn('\nIndefinite preconditioner detected in CG, abor
Average actual return over 20 episodes: -99.34
Average predicted value at initial state: -90.35
difference: 8.99

Evaluation using QR solution:
Average actual return over 20 episodes: -99.34
Average predicted value at initial state: -90.39
difference: 8.95

Evaluation using CG solution:
Average actual return over 20 episodes: -99.34
Average predicted value at initial state: -82.21
difference: 17.14

Evaluation using Deflated CG solution:
Average actual return over 20 episodes: -99.34
Average predicted value at initial state: -82.20
difference: 17.14

Evaluation using PyAMG solution:
Average actual return over 20 episodes: -99.34
Average predicted value at initial state: -82.32
difference: 17.02
```



Acrobot → Visualized

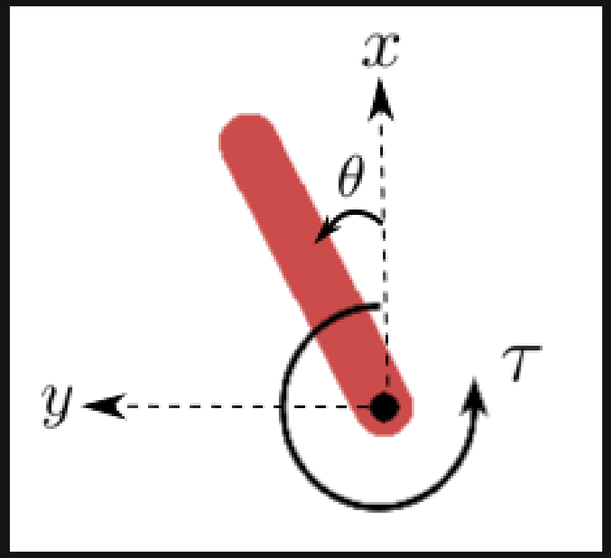


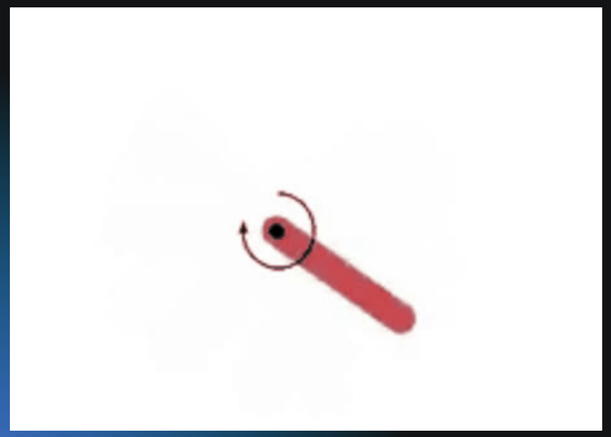
LU and QR superiority

Direct solvers; deterministic; stable for smaller systems.
The Acrobot's system is likely small enough for direct methods to shine.

Iterative solvers like CG and PyAMG underperform, likely because the system is too small for them to be efficient or accurate. (likely due to overhead in building multilevel hierarchies, which pays off only for very large systems)

5. Pendulum-v1





Nature: The agent must swing up and balance an inverted pendulum, starting hanging downwards. The agent must apply continuous torques to bring it upright and keep it there.

State Space: Includes cosine/sine of the angle of the pendulum and its angular velocity(3 values).

Action Space: Continuous, a single torque value.

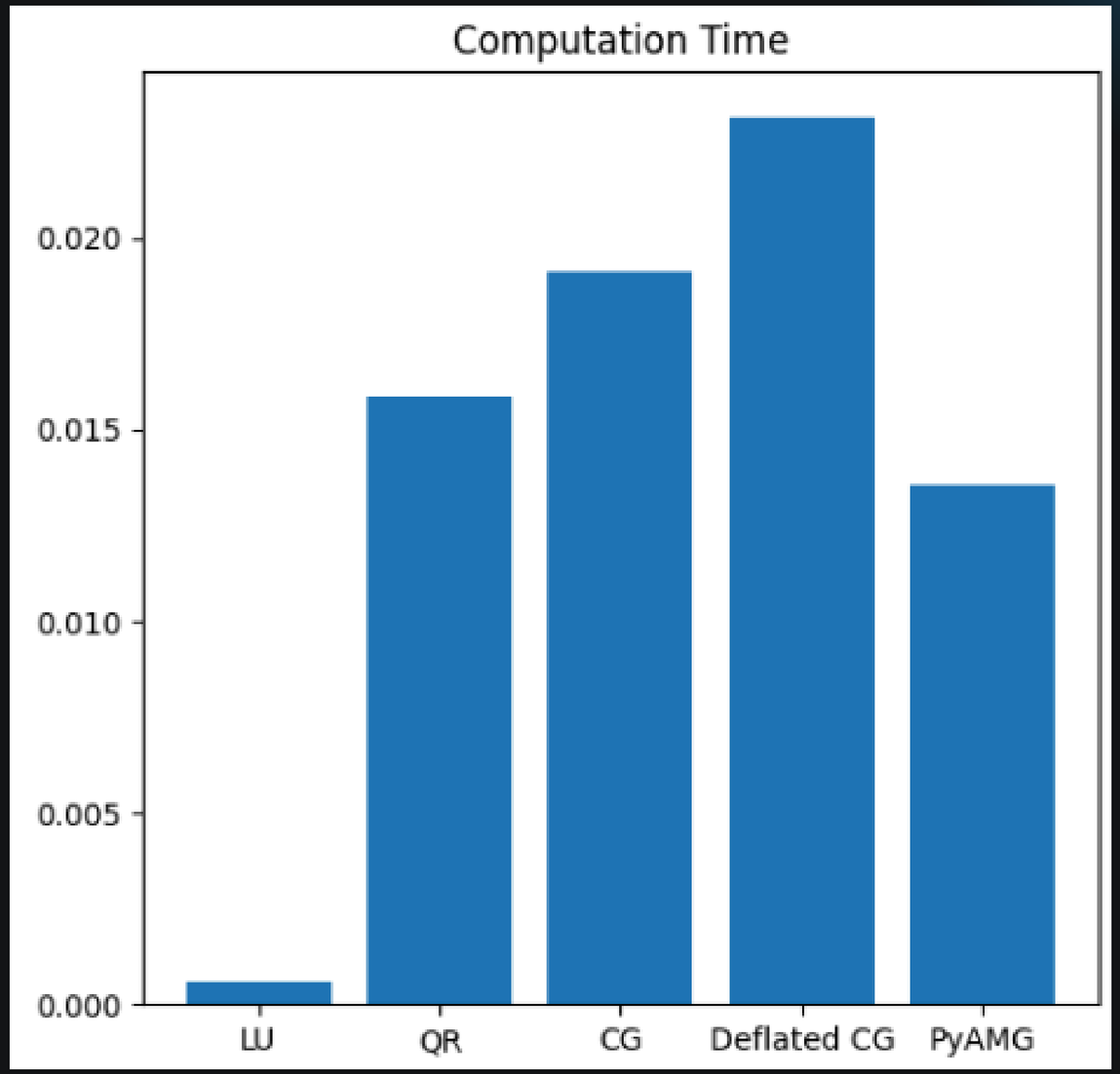
Reward: `reward = -(θ_normalized)^2 + 0.1 * θ̇^2 + 0.001 * torque^2`

Key Challenges

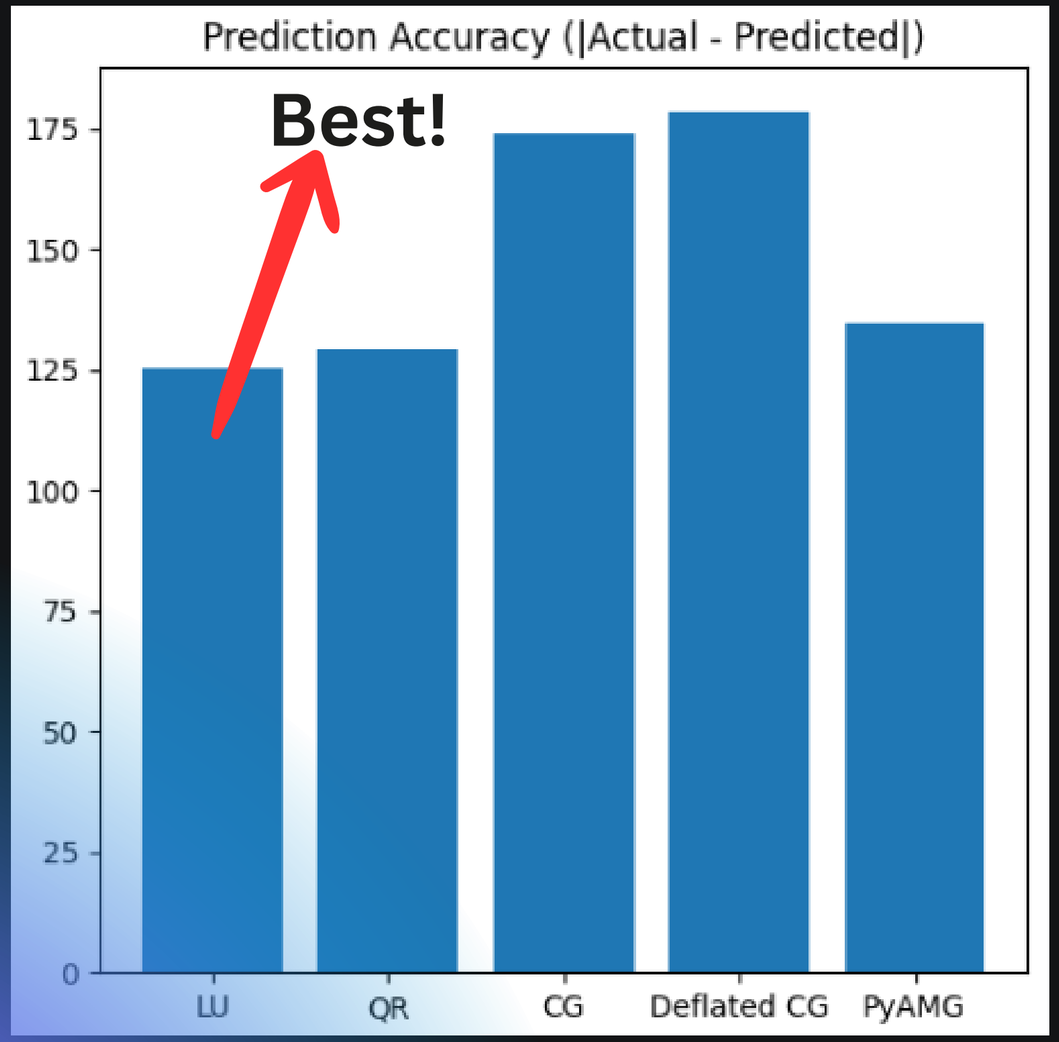
- **Continuous Action Space:** Makes value approximation and solver convergence more delicate
- **Lack of Terminal State**
- **Efficient Exploration:** Hard to find the right actions without guidance.

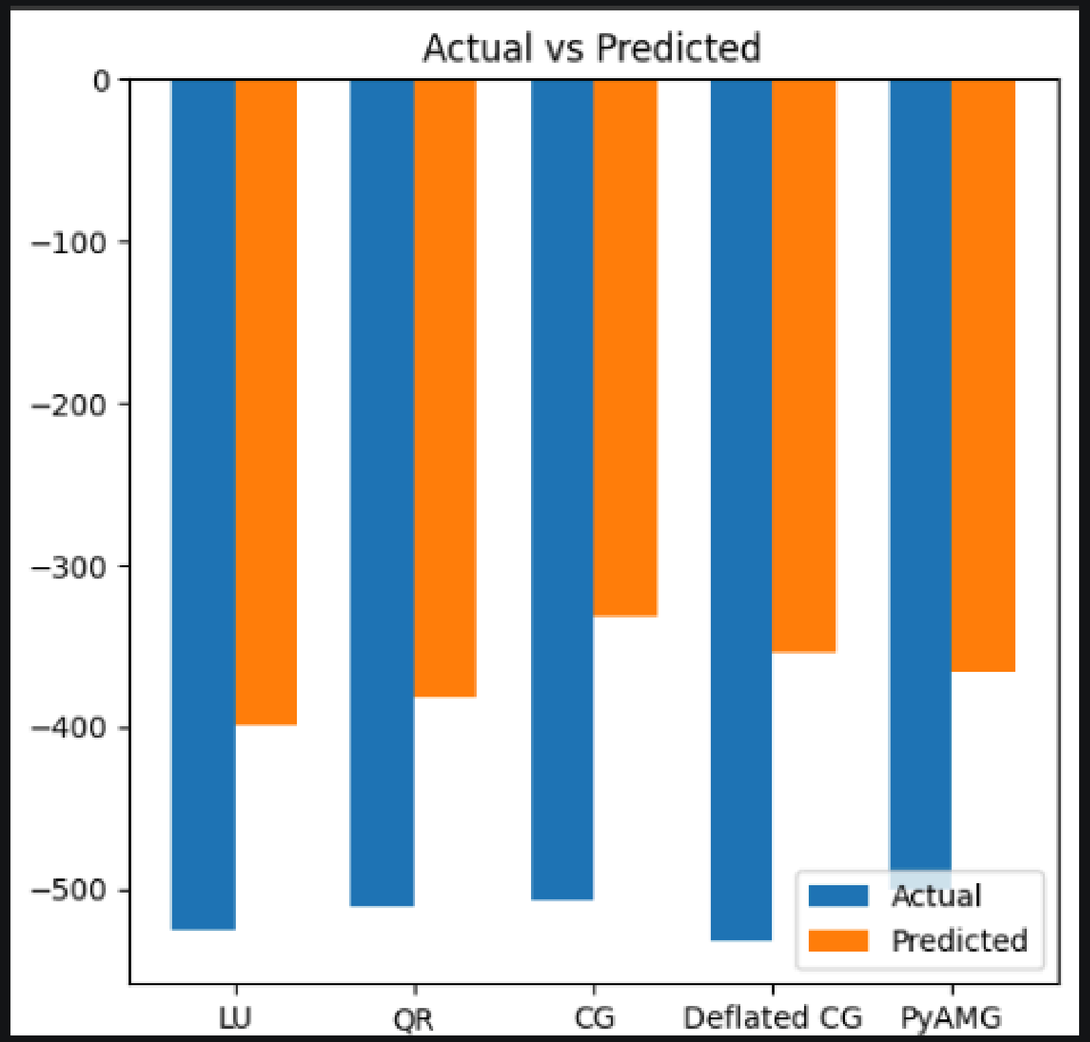
Pendulum-v1 Results

```
Evaluating LU...
Average actual return: -524.91
Average predicted value: -399.39
Difference: 125.51

Evaluating QR...
Average actual return: -510.44
Average predicted value: -381.31
Difference: 129.13

Evaluating CG...
Average actual return: -506.33
Average predicted value: -332.15
Difference: 174.17

Evaluating Deflated CG...
Average actual return: -531.94
Average predicted value: -353.33
Difference: 178.61

Evaluating PyAMG...
Average actual return: -500.55
Average predicted value: -366.13
Difference: 134.42
```



Pendulum-v1 → Visualized





Rank and Sparsity

How the environment and the matrix A are related

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

CartPole, Acrobot:
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

Matrix A, and the environment contd..

Smoother transitions in the environment, like 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 CG with ILU
and also PyAMG due to gradual variation in values across neighboring states

Environments that follow one-hot encoding, eg. GridWorld generate sparse matrices. If all states are visited, then the rank is full

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

each of the phi terms is a one hot encoded feature vector

This leads to sparse matrices, well-conditioned if the transitions are not too chaotic. PyAMG can leverage the grid like structure of the environment to build a multi-grid hierarchy easily.

Summary

Crafting an Ideal Solver Choosing Algorithm