****

IEOR E4540 Data Mining

Final Project

**Comparing Different Evolution Strategy Methods**

| *Zhaoyang Liu*  *Kaixuan Ding*  *Yecheng Ma* | *Ding Yuan*  *Tianqi Li*  *Chunxue Chen* |
| --- | --- |

Supervised by

Professor Krzysztof Choromanski

Fall 2021

**Abstract**

This paper performed Evolution Strategies (ES) methods to solve BlackBox optimization problems. We first experimented with a generic ES method on a preset function, including control variate terms to reduce sampling variance, and regression-based methods with regularization to estimate the gradient of our objective function. Once we are confident with our framework, we tested it using several reinforcement learning tasks from the OpenAI Gym Suite. After careful implementation, we proved that ES methods worked well to solve a BlackBox optimization problem, without explicitly computing gradients of the objective function. Furthermore, adding control variate techniques and regressions-based methods indeed improved our gradient estimation results and efficiency.

|  | |
| --- | --- |
|  |  |

**Pendulum**

The inverted pendulum swing-up problem is a classic problem in the control literature. In this problem, the pendulum starts in a random position, and the goal is to swing it up so it stays upright. (Please refer to the video in the drive regarding ES optimization performance.)

| States:  cos(theta): the angle of the pendulum from -1 to 1  sin(theta): the angle of the pendulum from -1 to 1  Theta dot: the angular velocity from -8 to 8 | |
| --- | --- |
| Reward:  -(theta^2 + 0.1\*theta\_dt^2 + 0.001\*action^2)  Termination:  There is no specified termination. | Actions:  The joint effort is a value between -2.0 and 2.0, representing the amount of left or right force on the pendulum. |

1. **Mathematical Principles and Algorithm**

To show our understanding of the Evolution Strategy topics, we wrote a separate Jupyter Notebook to perform ES methods on a preset function without explicitly computing for its gradient. Please refer to *example\_ES.ipynb[[1]](#footnote-0)* for more details.

To solve an optimization problem, most of the time, we will have a differentiable objective function and we can reach an explicit form of its gradient. Once we have its gradient, we can proceed to shift our parameters in the gradient’s direction for a maximization problem or the reverse for minimization. However, sometimes our objective function’s gradient is too complicated to compute, or we don’t even know what the objective function is. For this case, we can apply the ES method with gaussian smoothing.

Suppose we have a BlackBox function , and we want to find an optimal solution that maximizes this BlackBox function. Since we don’t know the gradient of , we can first generate gaussian noises and move the initial to those noisy directions. Next, we need to plug those updated ’s into the BlackBox function to compare the results, and we will move in the direction of that has the highest . Finally, we can repeat this process until a stopping condition is met or an optimal solution is found.

We can also use a regression-based method to estimate the gradient of our BlackBox function. For instance, after generating a set of gaussian noises , we estimate the gradient component on the direction of noise as , and we can apply a regression-based model with regularization to estimate the gradient in the direction of those noises. Then we update by . By doing so iteratively, we can reach the optimal value. Below is the Pseudocode to explain this method

\_\_\_\_\_\_\_\_\_\_

# **Pseudocode:**

1. Initialize a random
2. Around , generate random gaussian noises
3. Use finite difference method, set , which is the gradient in the direction of those
4. Use regression method, set , where is the norm input, is the regularization constant, and = is a matrix with each row to be gaussian noise . Depending on the regularization model we picked, can be set correspondingly.
5. Update with . (“minus” for minimization problem)
6. Repeat steps 2~5 until an optimal solution is found or the stopping condition is met.

\_\_\_\_\_\_\_\_\_\_

For our project, we consider this BlackBox function as a hidden reward function for each game in the Gym environment. We came up with two different ES methods as shown below:

# **ES Method I**

For the first method, we consider our parameter as a sequence of action . We perturb this sequence of action by adding different gaussian noises . In order to keep the underlying BlackBox function stable and make sure we can get back to the right starting point after perturbation, we used env.seed() to reset the environment. Below are our function specifications:

| initialize() | Given the max step of a game and the name of the game, we will use the OpenAI Gym package to generate a series of initial actions in a 1-D array format. |
| --- | --- |
| generate\_z() | This function will generate noises using three variance reduction sampling methods – Vanilla, Antithetic, and Forward-FD, where the input is the number of samplings, control variate method, and max step of the game, and the output is a matrix with each row being gaussian noise directions. |
| convertToAction() | The perturbed action is likely to be not legitimate. For instance, in the Cart-Pole example, actions are either 0 or 1 integer. However, after gaussian perturbation, actions can be floating-point numbers. Therefore, we used convertToAction() function to convert perturbed actions back to legitimate actions. |
| get\_reward() | This function inputs an action and current game in the gym environment, and the result will be the corresponding rewards. |
| regression() | This function uses a regression-based method to estimate the gradient in the direction of generated noises. |
| get\_new\_action() | This function inputs gradient, current action, learning rate, and the game, and it outputs perturbed legitimate action. |

# **ES Method II**

For the second method, we reference from MorvanZhou’s Evolutionary-Algorithm. The overall method would be similar to the ES Method I. We created a neural network that inputs a current state in the game, and the network output will be the optimal policy or action that maximizes the final reward. Our job for this method is to perturb and add noises on the neural network parameters and to find the optimal network parameters.

Our neural network is constructed with 3 hidden layers. To make it easy with adding noises, we convert the 2-D dimensional matrix and into the 1-D array. Otherwise, the skeleton is pretty similar to ES Method I as discussed above.

1. **Variance Reduction & Sampling Techniques**

# **Vanilla Sampling**

Using the Vanilla sampling, we generate gaussian noises . The gradient can be estimated using the formula below. Once we have the gradient, we can update our current parameter in the way specified above. Hence, the Vanilla gradient estimator has a standard Monte Carlo estimator of the expectation.

# **Antithetic Variate**

The antithetic sampling, also known as mirrored sampling, is a variance reduction technique used in Monte Carlo methods. In addition to the vanilla sampling, it adds the antithetic variables in opposite directions, so the direction of the second half of the sampling is the opposite of the first half, and as a result, it reduces the variance of simulations/sampling.

# **Forward-FD (Forward finite difference)**

The Forward FD sampling is a variance reduction technique used in Monte Carlo methods based on the Vanilla sampling. It controls variance through adding generated samples with unperturbed actions, which can be regarded as a shift on F of the Vanilla sampling.

# **Gaussian Orthogonal Matrices**

One problem with our vanilla ES method is that if we generate N noises completely random, it is possible that noises are biased towards some directions. To make sure that we span the whole space, we need to make our noise matrix Z to have orthogonal rows.

# **Random Hadamard Matrices**

Hadamard Matrix is a special case of Gaussian Orthogonal Matrix. The key is to make sure that every perturbation we applied to the original sequence is orthogonal to each other. Instead of using the Hadamard matrix directly, which is composed of +1 and 0, we multiply each element of each row with a number generated from the standard normal distribution. Therefore, the perturbation remains orthogonal, and the matrix can be directly used as a gaussian noise. Then, we randomly choose different rows from the matrix as our perturbation matrix.

# **Givens Random Rotations**

We tried implementing Givens random rotations to the random Hadamard matrix to further improve its performance. It is a low-dimensional random rotation inside the Hadamard matrix. We have space and time complexity gains from it. The algorithm looks like the following:

1. **Regression-based gradient estimator**

With generated actions space and the corresponding reward scores, we can set up the following optimization task to solve for the estimated gradient that minimizes the Empirical Risk.

**LP-decoding**

Lp-decoding applies a linear regression model to minimize the residual sum of squares. In this case, the alpha parameter above is zero. With the estimated v, we could predict the reward scores based on its actions. However, this method may have an overfitting issue that weakens our results. We may need to explore alternative methods that apply regularization to reduce the scale and/or numbers of gradients to reduce variance.

**Lasso**

Lasso method allows us to improve estimated gradients through bias-variance tradeoff. With a chosen alpha (0.05), we add an additional l1-norm to the objective, which will punish large estimators to reduce variance. Lasso tends to shrink estimators to zero, which will reduce the change in actions per iteration.

**Ridge**

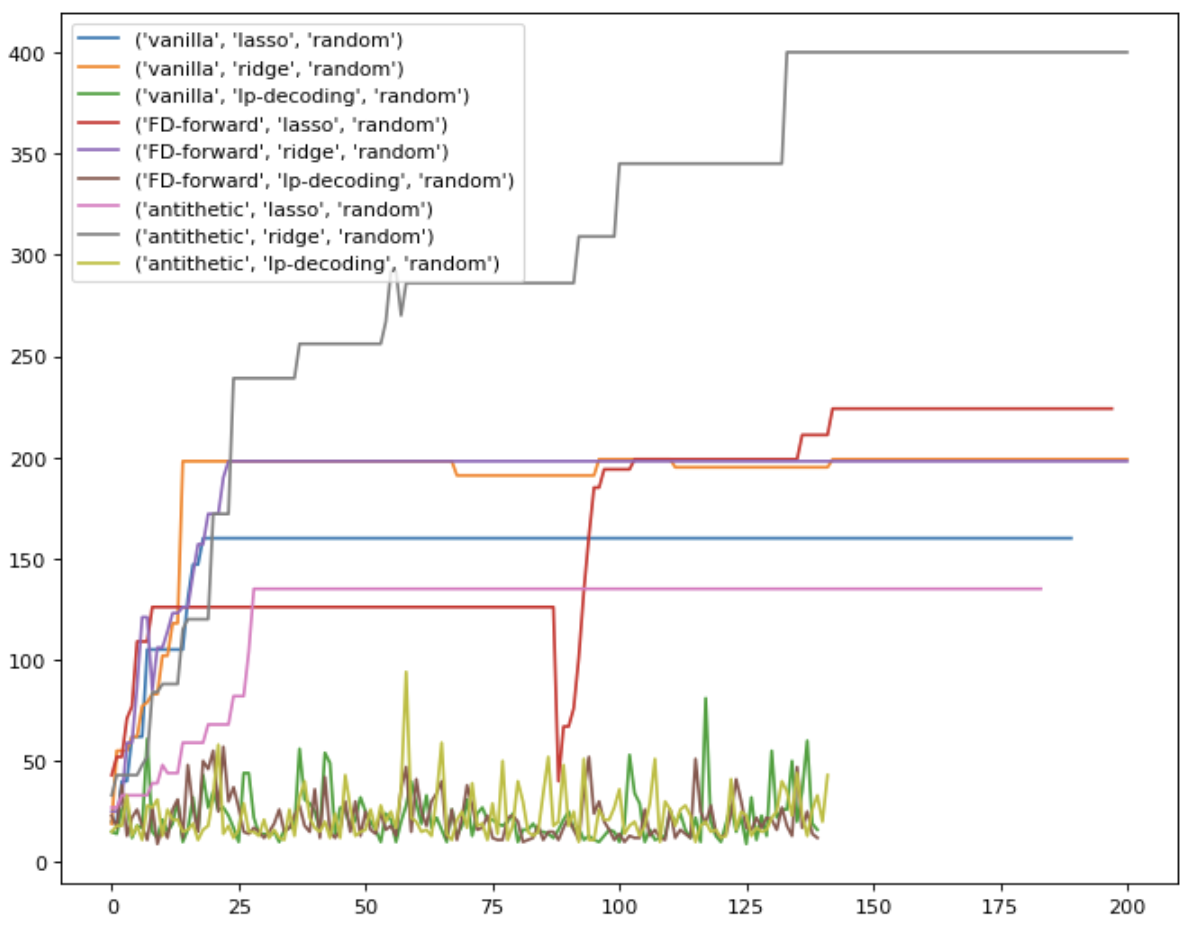
The Ridge method allows us to improve estimated gradients through a bias-variance tradeoff. With a chosen alpha (0.05), we add l2-norm to the objective, which will punish large estimators to reduce variance. Ridge tends to shrink estimators to small values, but not zeros, which will change in actions per iteration.

1. **Observation and Conclusion**

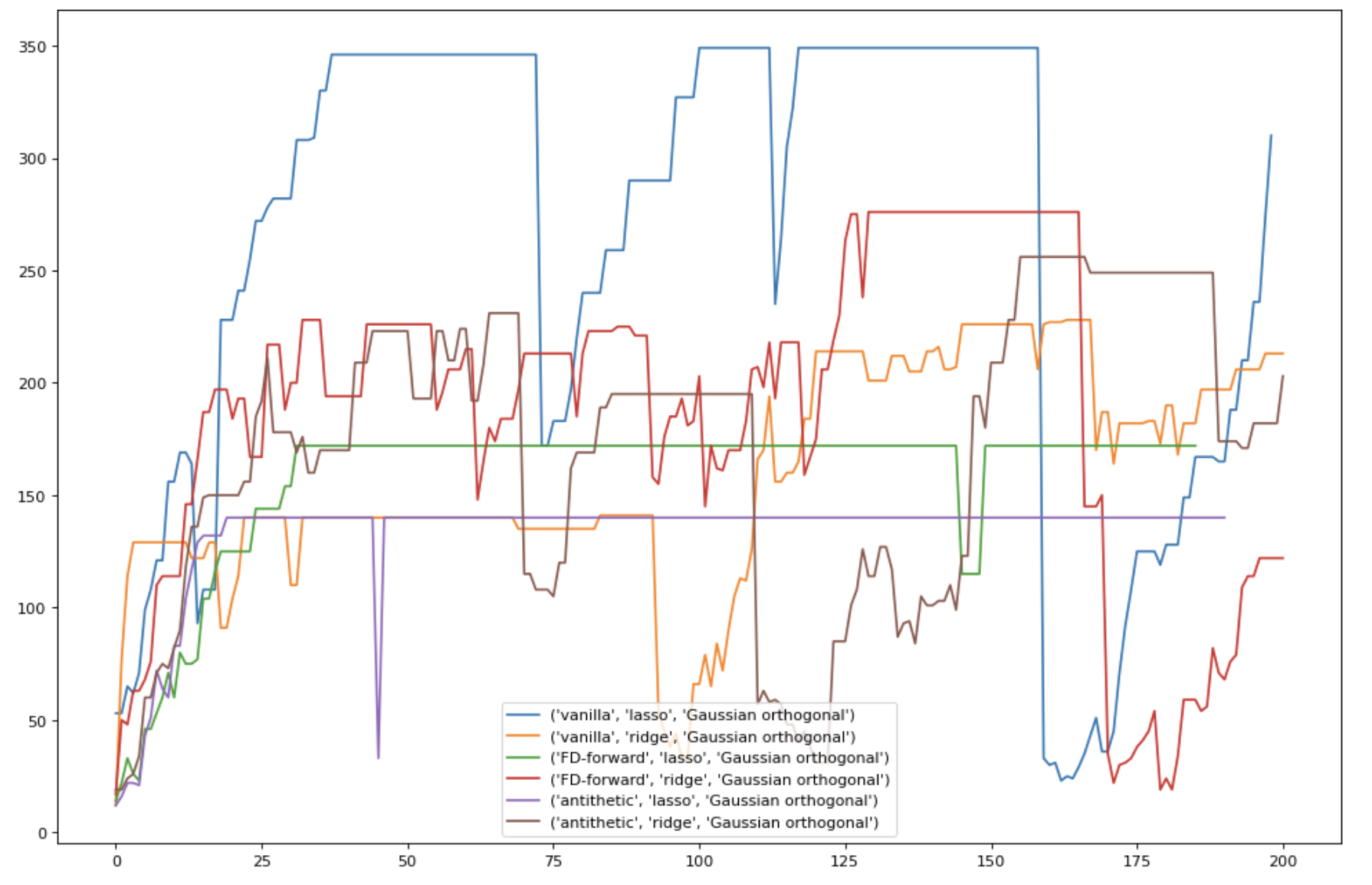
# **ES Method I**

In this section, we will cross-compare three different types of variance reduction sampling methods with three regression-based gradient estimator methods, along with the random sampling matrix, Gaussian orthogonal matrix, rotations, and Random Hadamard matrices. From the graph, the x-axis is the number of iterations, and the y-axis is the reward scores. Due to the limitations in the computing power, we chose not to set the number of iterations too large. We will take both the converging speed, average score levels, and converged score levels into consideration.

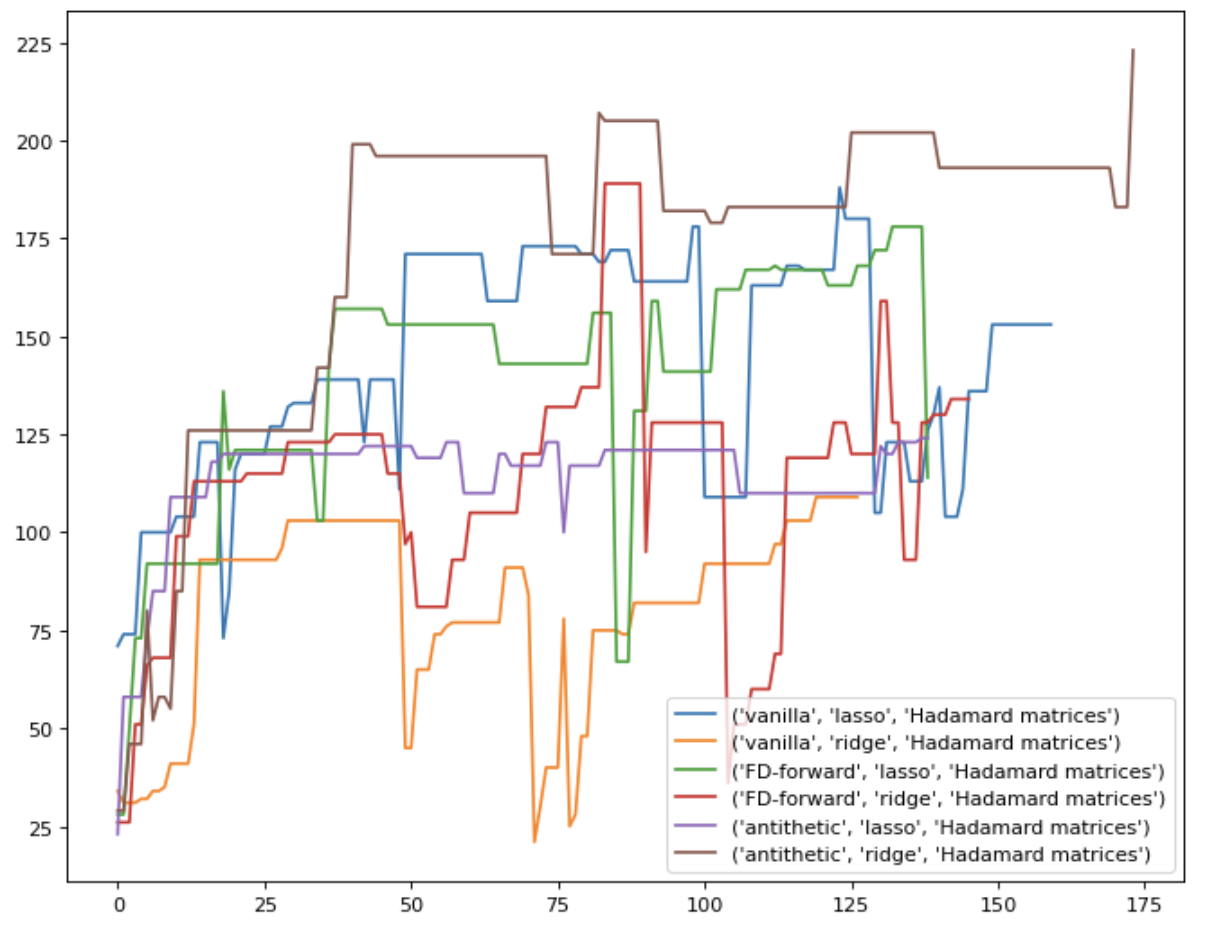
The graph below generates results from the normal sampling matrix. The pair antithetic and ridge performs significantly better than the rest. Also for the following analysis, > means outperform. Given Lasso regression method, Forward-FD > Vanilla > Antithetic; given Ridge regression method, Antithetic > Vanilla > Forward-FD. LP-decoding performs the worst because it has an overfitting issue, which cannot be mitigated by adjusting parameters.



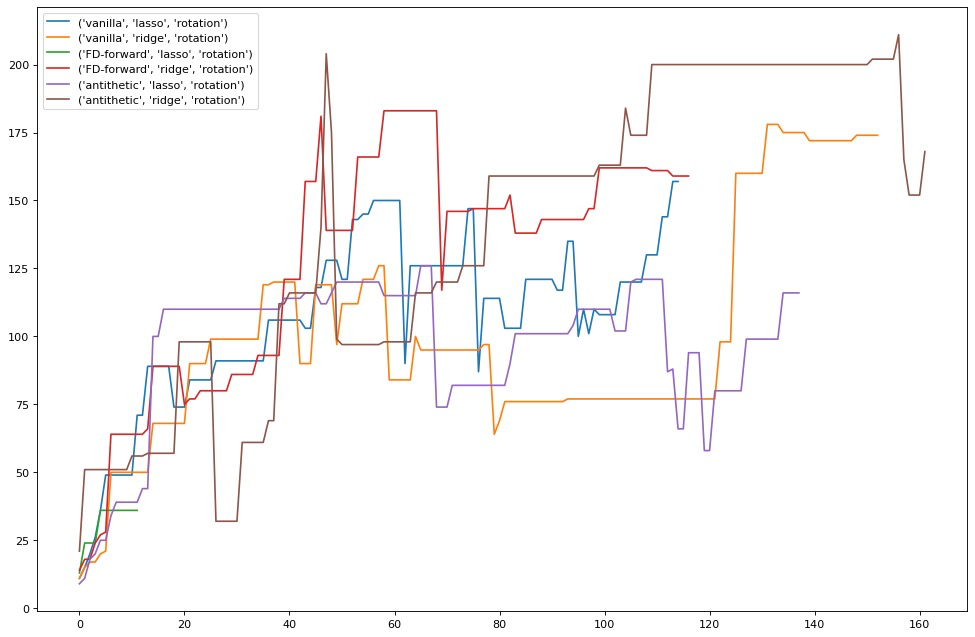
The graph below generates results from the Gaussian orthogonal sampling matrix. The pair of Vanilla and Lasso performs the best compared to others. We didn't use LP-decoding in this graph again because as shown from the graph above, the result is significantly worse than the others. Given Lasso method, Vanilla > Forward-FD > Antithetic; given Ridge method, Forward-FD > Antithetic > Vanilla.



The graph below generates results from the random Hadamard matrix. The pair of Antithetic and Ridge performs the best compared to others. We didn't use LP-decoding in this graph again because from the graph above, the result is significantly worse than the others. Compared to other Monte Carlo simulation methods, the best result of Hadamard is significantly lower. The reason is that Hadamard is composed of a large number of 0 in it, so the regression based on that sampling will take in fewer variables. Therefore, it may need more iterations to achieve better results. Given Lasso method, Vanilla > Forward-FD > Antithetic; given Ridge method, Antithetic > Forward-FD> Vanilla.



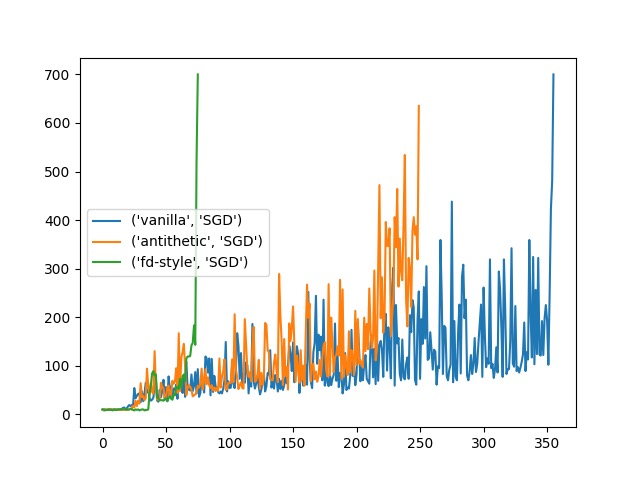
The graph below generates results from a Givens rotation matrix. The pair of Antithetic and Ridge performs the best compared to others. We didn't use LP-decoding in this graph again because from the graph above, the result is significantly worse than the others. During implementation, we noticed that a significant number of iterations result in no change in the score for each perturbation. That led to some lines being shorter than others. Given Lasso method, Vanilla > Antithetic > Forward-FD; given Ridge method, Antithetic > Vanilla > Forward-FD.



There remain some considerations from our graphic results. Ideally, there shouldn’t be a significant drop after hitting a relatively great score, but this situation still reflects from our graphs. This is because 1) step size issue and 2) there are random factors when choosing samplings. An overestimated step size will make the new sequence of action away from the optimal even though it’s moving in the direction with a better gradient. While sampling, if the samples are chosen from someplace that deviates from the original position too much, the estimated gradient will be inaccurate and lead to a sudden drop in the plot.

# **ES Method II**

In ES Method II, because we get a new action from a neural network, the variance is significantly higher than ES Method I. Hence, we see a more fluctuating graph below. Forward-FD hits the score limits the fastest and then is the Antithetic method. Vanilla performs the worst because the variant control is the most limited.



**Conclusion and Future Guidance**

Based on the observations above, we can conclude that the ES optimization algorithm can help us estimate the gradient of an unknown function. Reward scores consistently improve across multiple scenarios. With variance reduction techniques, the overall performance on regularized regression methods efficiently improves, especially for antithetic variates. According to our trials, LP-Decoding is comparably weak compared with other methods due to its over-fitting issue. Regularization methods are necessary to ensure good performance. Also, we need to make sure to reset the environment to a fixed state during perturbation. This is important because we once encountered an issue that final rewards were not going up after iterations. If we forgot to reset the environment, we are essentially training on the different BlackBox functions.

Admittedly, several improvements can be made to our algorithm. First of all, the size of each perturbation, epi in the code, can be more accurately tuned. A suitable perturbation will sample near the original position so that the gradient estimate can be accurate. Secondly, the step size needs more review. Good step size is expected to move toward the gradient without surpassing the optimal point.

**Works Cited**

​​Fan Mo. “Evolution Strategy 强化学习” *mofanpython*, https://mofanpy.com/tutorials/machine-learning/evolutionary-algorithm/evolution-strategy-reinforcement-learning/. Accessed 2 Dec. 2021.

1. Please refer to *example\_ES.ipynb* in the drive. We also have this in the Appendix section below. [↑](#footnote-ref-0)