Reinforcement Learning Methods

Applications of Policy Iteration, Q-learning and REINFORCE in Agent Training

Eduardo Joaquin Castillo
Electrical and Computer Engineering
Cornell Tech at Cornell University
New York, NY 10044
`ec833@cornell.edu`

1 Introduction

This report documents the development of a set of Reinforcement Learning (RL) models based on well-known machine learning algorithms.

The objective of these models is to demonstrate how both on-policy and off-policy methods can be applied to agent training in the context of Markov Decision Processes (MDP) to produce equivalent optimal policies for a given reward scheme.

In particular, Policy Iteration and Q-learning are implemented to find an optimal policy in a simulated maze environment given a set of constraints. In addition, Q-learning and REINFORCE models are developed for two off-the-shelf OpenAI gym environments, MountainCar-v0 and Acrobot-v1.

2 Problem Formulation

2.1 The Maze Environment

2.1.1 General Description

The task in the maze environment is to maximize the total discounted reward obtained while traversing a static grid-like maze through a discrete state-action space. Figure 1 shows the 20-cell structure of the maze environment.

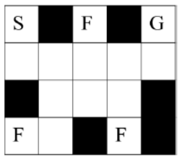

Figure 1: Maze Environment [1]

2.1.2 Reward Scheme

The following reward scheme is applied to the environment:

1. The agent starts from cell **S**, performs a sequence of actions S_A, $\forall$ $a \in$ A = {UP, DOWN, LEFT, RIGHT} until cell **G** is reached, ending the episode.

2. Upon reaching the goal cell **G**, the agent receives a reward equal to the total number of flags collected during the episode.

2.1.3 Action-State Uncertainty

A probabilistic action-state transition distribution is applied to the maze environment, as described below.

1. With 90% probability, an action $a \in$ A = {UP, DOWN, LEFT, RIGHT} results in a motion in direction a.

2. With 10% probability, the same action a results in the agent ”slipping”, and moving in the corresponding direction a_{prime} in the set A_{prime} = {RIGHT, LEFT, UP, DOWN}.

2.2 The Mountain Car Environment

2.2.1 General Description

The Mountain Car environment consists of a simulated hill environment traversed by an under-powered car. Figure 2 shows the general layout of the simulated environment.

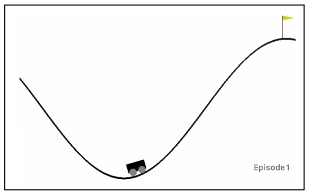

Figure 2: Mountain Car Environment [2]

The Mountain Car environment features a continuous two-dimensional state space consisting of the velocity and position of the agent. In addition, it features a discrete action space with actions {PUSH LEFT, DO NOTHING, PUSH RIGHT}.

The task is to use the car's simulated momentum along with its power to climb the hill in the lowest possible number of time steps.

2.2.2 Reward Scheme

The agent gets a reward of -1 for each time step during an episode (regardless of action taken) until the goal is reached. Therefore, the best policy minimizes the number of time steps before the top of the hill is reached.

2.3 The Acrobot Environment

2.3.1 General Description

The Acrobot environment consists of a simulated two-bar linkage powered at a single joint. In addition, it features a discrete action space with actions {APPLY NEGATIVE TORQUE, DO NOTHING, APPLY POSITIVE TORQUE}. The task at hand is to get the tip of the second linkage above a pre-established height. Figure 3 shows the general layout of the simulated environment.

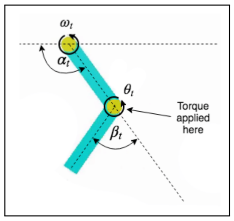

Figure 3: Acrobot Environment [3]

2.3.2 Reward Scheme

As with Mountain Car, the agent gets a reward of -1 for each time step during an episode (regardless of action taken) until the goal is reached. Therefore, the best policy minimizes the number of time steps before the established height is reached.

3 Technical Approach

3.1 Maze

3.1.1 Policy Iteration

As a basic solution approach, Policy Iteration (PI) is implemented to extract individual state-action values and the associated optimal policy.

The Policy Evaluation step of PI is implemented in a recursive fashion, with a root mean square error (RMSE) convergence criterion between corresponding state-action value elements at successive recursive layers.

The Policy Improvement step of PI is implemented following the conventional greedy improvement approach to converge to the optimal policy π^* and approximate the closed-form state-action value pairs.

3.1.2 Q-learning

The basic on-policy approach is contrasted with Q-learning, an off-policy solution approach. Similar to Policy Iteration, the objective of the Q-learning model is to extract individual state-action values and directly learn the optimal policy Q*.

An ϵ-greedy exploration-exploitation approach is implemented to specify the exploration strategy. Hyperparameter sweeps are completed across a set of Learning Rates, Learning Rate decay rates and ϵ-greedy exploration policies.

During training, policies are sampled every 50 epochs, in order to provide a real-time window to the effectiveness of the training routine.

3.2 The Mountain Car Environment

3.2.1 REINFORCE Implementation

A shallow feed-forward neural network (NN) is implemented as a function approximator to the ground-truth optimal decision values applicable to this MDP.

The shallow feed-forward NN features a two-hidden-layer fully-connected architecture with a rectifying linear unit (RELU) activation function on the first hidden layer and a softmax output-layer activation function.

The three outputs of the softmax layer are used as pseudo-probability approximators for the appropriate action at each iteration. These pseudo probabilities are used to generate approximate categorical distributions. The associated log probability values of randomly sampled actions are in turn fed to the policy gradient routine.

The position and velocity parameters are normalized to range between zero and one, in order to promote numerical stability during training of the shallow NN.

In addition to these parameters, the baseline state space is lifted by three dimensions. Namely, velocity squared, per-step velocity change and a position-velocity inner product are computed and passed to the network at each iteration.

It is noted that aside from the per-step velocity change parameter, no additional temporal information is passed to the network by the addition of these engineered features. Albeit NNs do not depend on feature engineering to produce adequate function approximation, these additional features are found to reduce the required complexity and convergence time of the shallow feed-forward NN.

3.2.2 Q-learning Implementation

A state discretization approach is followed in order to iteratively approximate the state-action values applicable to this MDP environment. A similar ϵ-greedy exploration strategy to the maze MDP is applied to this exercise in order to effectively map the state-action space.

In addition, the same training supervision scheme is applied to this exercise, with rewards being recorded every 50 epochs throughout the training routine.

3.3 The Acrobot Environment

3.3.1 REINFORCE Implementation

A shallow feed-forward neural network (NN) is also implemented as a function approximator to the ground-truth optimal actions applicable to the Acrobot MDP.

Similar to Mountain Car environment, the shallow NN featured a two-hidden-layer fully-connected architecture with a rectifying linear unit (RELU) activation function on the first hidden layer and a softmax output-layer activation function.

Given the relative simplicity of the network, the baseline over-parameterized state variable set (including both sines and cosines of each joint angle) is fed to the network. The same pseudo probability regression approach described for the Mountain Car problem is followed to obtain applicable actions and the associated log probabilities.

3.3.2 Q-learning Implementation

Similar to the Mountain Car environment, a state discretization approach is followed in order to iteratively approximate the state-action values applicable to this MDP environment. The same ϵ-greedy exploration and training supervision schemes are also applied.

In this case, the given state space is over-parameterized, with four sinusoidal functions computed at each time step for just two unique angles, α_t and β_t (see Figure 3). To reduce the dimensionality of the state space, angles are computed at each iteration and feed into a four-parameter state-action value estimation routine.

The final four parameters are simply the angles and angular velocities of both joints. This reduction is beneficial in reducing model complexity as well as computation and memory requirements.

4 Results and Discussion

4.1 Maze Optimal Policy

The optimal policies obtained from the Policy Iteration and Q-learning scopes of the project are fully equivalent and may be described as follows:

1. The Robot enters the maze at state 0 —no flags at cell (0,0).
2. The Robot moves down one step and enters State 8.
3. The Robot moves to the right **twice**, passing through State 24 and reaching State 56.
4. The Robot moves up once, capturing one flag and entering State 49.
5. The Robot moves down once, entering State 57.
6. The Robot moves right once, entering State 73.
7. The Robot moves down **twice**, passing through State 81 and reaching state 93 after capturing another flag.
8. The Robot moves up **twice**, passing through state 83 and reaching state 77.
9. The Robot moves right once, entering State 109.
10. The Robot moves up once, entering State 101. This is the end of the path —the goal cell.

During this optimal journey, it is assumed that the robot does not slip at any step, completing the maze through the path that maximizes the total discounted reward. Appendix A contains a detailed table, containing all computed state-action value pairs for both the Policy Iteration and Q-learning methods.

4.2 Maze Optimization and Tuning

4.2.1 Learning Rate and ϵ Tuning

Experiments are performed using Learning Rates from 0.5 to 1.0, in increments of 0.1. Learning Rates are evaluated using both fixed and linearly-decaying values over epochs. In addition, different exploration-exploitation strategies are evaluated using a range of ϵ-greedy ϵ values.

Figure 4 shows the results of the performance evaluation when the policy is sampled every 50 epochs for an epsilon of 1.0 (continuous random exploration) during a 5000-epoch training run.

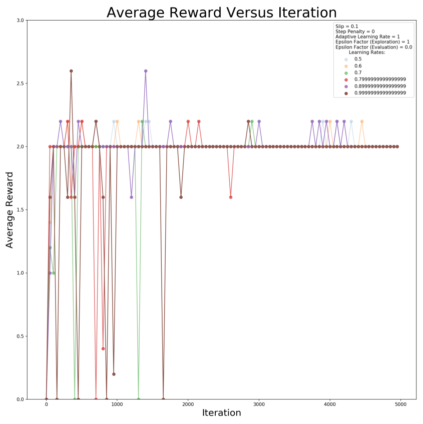

Figure 4: Rewards for Learning Rate Sweep

As shown, a linearly-decaying Learning Rate (LR) of 1.0 offered the best trade-off of training convergence and variance minimization across all training rates.

4.3 Q-learning versus PI Performance

Root mean squared error (RMSE) of state-action values is used as a comparison metric for the Q-learning and Policy Iteration methods in the maze environment. Figure 5 shows the corresponding RMSE error during the training run described in Figure 4.

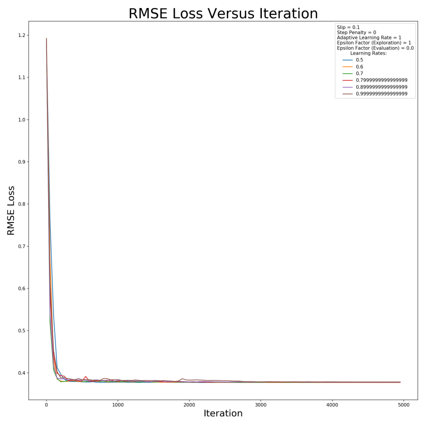

Figure 5: RMSE - Decaying Learning Rates

It should be noted that the reported RMSE (approximately 0.4 after convergence) does not approach zero due to the existence of superfluous states which are never visited or trained.

An instance of such is state 16, which corresponds to a flag cell where the agent is said to hold zero flags. Although harmless for practical purposes, this insight is useful in explaining the 0.4 RMSE convergence value.

4.3.1 Additional Hyperparameter Tuning

Model performance is also evaluated using fixed Learning Rates as well as reduced epsilons. As documented in RL literature [4], large fixed Learning Rates lead to increased RMSEs during the late stages of training. Figure 6 illustrates this behavior.

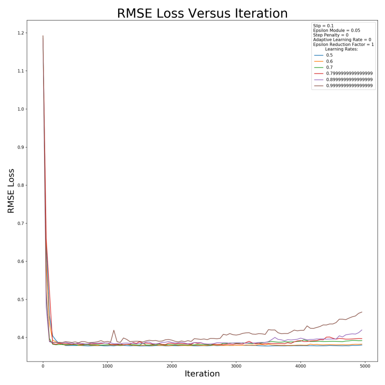

Figure 6: RMSE - Fixed Learning Rates

Likewise, a reduced epsilon routine leads to slower rates of RMSE reduction, due to the slower rate of environment mapping when early exploitation is favored over exploration. Figure 7 illustrates this behavior for an $\epsilon = 0.1$.

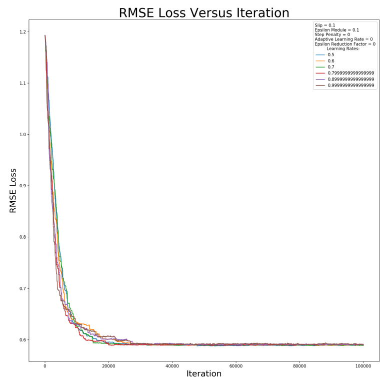

Figure 7: RMSE Values for $\epsilon = 0.1$

4.4 Mountain Car Results

4.4.1 REINFORCE Results

The final REINFORCE results for the Mountain Car (MC) environment are in the range of 105 to 115 steps per episode after a 30,000 epoch training run. Figure 8 shows the performance evaluation curve sampled every 50 epochs for a Learning Rate of 0.001 with 10-fold Learning Rate reductions applied at 10,000 and 20,000 epochs.

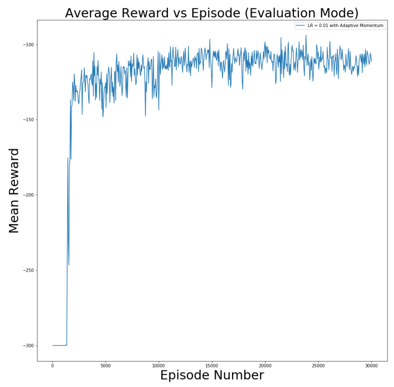

Figure 8: Reinforce Cost-to-go - Mountain Car

4.4.2 Q-Learning Results

The final Mountain Car Q-learning results are in the range of 120 to 140 steps per episode after a 20,000 epoch training run. Figure 9 shows the performance evaluation curve sampled every 50 epochs over a 10 fold Learning Rate sweep.

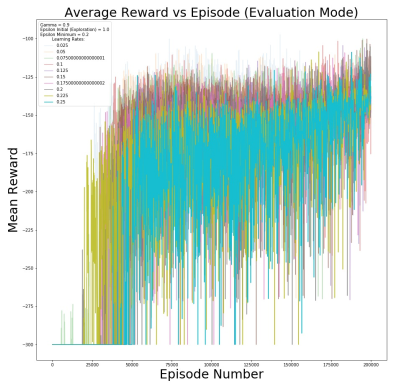

Figure 9: Q-learning Cost-to-go - Mountain Car

4.5 Acrobot Results

4.5.1 REINFORCE Results

The final Acrobot REINFORCE results are in the range of 90 to 95 steps per episode after a 5,000 training run. Figure 8 shows the performance evaluation curve sampled every 50 epochs for a Learning Rate of 0.001 with a 50% Learning Rate reduction applied at 4,000 epochs.

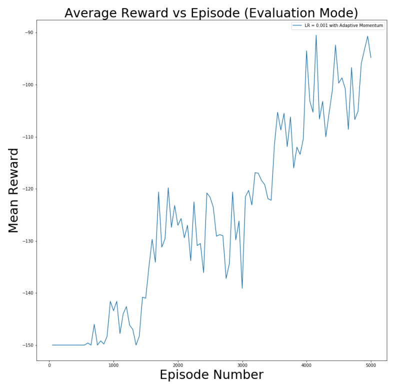

Figure 10: Reinforce Cost-to-go - Acrobot

4.5.2 Q-Learning Results

The final Acrobot Q-learning results are in the range of 120 to 140 steps per episode after a 20,000 epoch training run. Figure 11 shows the performance evaluation curve sampled every

50 epochs over a 5 fold Learning Rate sweep. The epsilon applied to the ϵ-greedy policy is 1.0 (strongly favoring exploration), with a linear epsilon decay to 0.5 over 20,000 epochs.

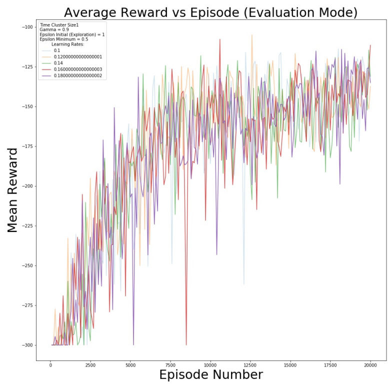

Figure 11: Q-learning Cost-to-go - Acrobot

4.6 General Discussion and Future Works

The algorithms implemented through this project are an effective introduction to mainstream Reinforcement Learning methods.

In the case of Policy Iteration and Q-learning in the Maze environment, adequate convergence to equivalent optimal policies as well as state-action value pairs are effectively demonstrated.

The REINFORCE and Q-learning implementations within the OpenAI gym environment are quite useful in demonstrating how cross-domain knowledge (i.e.: Deep Learning, Temporal Difference and Monte Carlo methods) can be leveraged to solve simulated and real-world problems involving MDPs.

Moreover, the use of adaptive parameterization (i.e. monotonically-decreasing Learning Rates) is generally found to be an important factor in improving numerical stability. This is true for methods of guaranteed convergence (i.e. Q-learning) as well as policy gradient methods based on function approximation.

Future works might include the application of these methods, along with past works in object segmentation and SLAM to the design of functional robots in simulated or real-world environments.

References

[1] Cornell Tech Electrical and Computer Engineering Department. ECE - 5242 Project 4 - Reinforcement Learning, 05 2020.

[2] Gym. `https://gym.openai.com/envs/MountainCar-v0/`. (Accessed on 05/03/2020).

[3] Gym. `https://gym.openai.com/envs/Acrobot-v1/`. (Accessed on 05/05/2020).

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Appendices

Appendix A State Action Value Pairs Table

The table presented below is indexed by state and contains the computed state-action value pairs for the maze environment. For each three-line block, the first line contains the state. The second and third lines contain the state-action value pairs obtained using Policy Iteration and Q-learning, respectively.

0			
0.47627188	0.52919103	0.47627188	0.48215179
0.48352212	0.53610826	0.48350986	0.48960479
1			
0.61934328	0.68815921	0.61934328	0.6269895
0.622336	0.69098822	0.6224298	0.62917451
2			
0.77679472	0.86310525	0.77679472	0.78638478
0.78251451	0.86715811	0.78254684	0.79225186
3			
0.75627459	0.8403051	0.75627459	0.76561131
0.78158893	0.86827467	0.78134824	0.78829271
4			
1.00233621	1.1137069	1.00233621	1.01471073
0.99931058	1.11054174	0.99878679	1.01034527
5			
0.96607901	1.07342113	0.96607901	0.97800591
0.97270359	1.08587808	0.97235153	0.98737901
6			
1.14211398	1.26901553	1.14211398	1.15621415
1.14647595	1.27426621	1.14435416	1.15727332
7			
1.44764354	1.60849283	1.44764354	1.46551569
1.45310288	1.61218832	1.45273362	1.46980954
8			
0.4887576	0.53507094	0.52919103	0.59452328
0.49997301	0.54192609	0.53554496	0.6009044
9			
0.63557969	0.69580542	0.68815921	0.77311713
0.64203118	0.69968621	0.69083743	0.7798842
10			
0.79715879	0.87269531	0.86310525	0.96966146
0.80075808	0.87276359	0.86433148	0.96977949
11			
0.77610072	0.84964182	0.8403051	0.94404647
0.80153703	0.87580872	0.86290288	0.97319053
12			
1.02861297	1.12608142	1.1137069	1.25120158

```
1.03394363    1.12801354    1.11750699    1.25377647

13
0.99140527    1.08534803    1.07342113    1.20594225
0.99001212    1.09752709    1.09176955    1.22189495

14
1.17205509    1.28311571    1.26901553    1.42568412
1.16482146    1.27886297    1.26277307    1.41525417

15
1.48559427    1.62636497    1.60849283    1.80707219
1.49289082    1.62883379    1.60787254    1.80242599

16
0.62908612    0.70047474    0.69346999    0.62536203
0             0             0             0

17
0.9154561     1.01866227    1.00847565    0.91492322
0             0             0             0

18
0.70912258    0.70047474    0.70047474    0.77830528
0.70680364    0.69956738    0.69950808    0.77646868

19
0.92414844    1.02852597    1.01824071    0.92223429
0             0             0             0

20
1.03123835    1.01866227    1.01866227    1.13184697
1.03432969    1.02288725    1.02325082    1.13948847

21
1.1734217     1.30573441    1.29267706    1.17256494
0             0             0             0

22
1.04122382    1.02852597    1.02852597    1.14280663
1.04542908    1.02560034    1.02639636    1.13530183

23
1.32185458    1.30573441    1.30573441    1.45081601
1.32642296    1.31161802    1.31154152    1.45628085

24
0.60910229    0.54982378    0.54167675    0.66792122
0.61978657    0.56001227    0.55076825    0.67585947

25
0.79037944    0.85238273    0.70439561    0.86856369
0.80186771    0.86773632    0.71406233    0.87828384

26
0.99360992    0.88296658    0.88346933    1.08937275
0.99469398    0.8832066     0.8853959     1.08681821

27
```

```
0.96738078    0.85814486    0.86013123    1.06059542
1.00369123    0.88679791    0.88528135    1.08763011

28
1.28049343    1.269747      1.13998366    1.40567091
1.2784567     1.27050801    1.14000903    1.40705427

29
1.23558253    1.10976943    1.09874738    1.35482401
1.23933273    1.11990218    1.11749157    1.35847036

30
1.46091461    1.29665499    1.29895664    1.6016945
1.46496902    1.2912917     1.29805839    1.58791169

31
1.85169831    1.64587197    1.64644355    2.03016752
1.84774252    1.64790596    1.65378748    2.03458565

32
0.6022423     0.61046085    0.5564296     0.61273667
0.61121343    0.61844276    0.5683147     0.62287016

33
0.80087912    0.90003501    0.86097292    0.96642246
0.8002365     0.90262216    0.87285335    0.97916227

34
0.98234204    0.79666859    0.89374059    0.9782463
0.97943275    0.79462456    0.895072      0.98069916

35
0.95454405    0.90529065    0.86863427    0.95019828
0.98157959    0.91824869    0.90448881    0.96017116

36
1.28460724    1.15855173    1.28364924    1.42856649
1.28539357    1.16099986    1.28323552    1.42369251

37
1.22188693    1.15323979    1.12316879    1.23609213
1.23201245    1.14807486    1.1272954     1.25089321

38
1.44239928    1.16976997    1.31249592    1.42079106
1.42876883    1.16375933    1.29452605    1.42212373

39
1.83115491    1.48504651    1.66595055    1.82716657
1.83336272    1.4852713     1.66863414    1.82106893

40
0.55801831    0.62536203    0.68557356    0.61701617
0.56764217    0.63478217    0.69011663    0.6241258

41
0.87314186    0.91492322    1.00377407    0.90339666
0.87933678    0.91463466    1.00560613    0.9024243
```

```
42
0.87439236    0.77830528    0.71883805    0.78695312
0.87453001    0.77790844    0.72339684    0.78686918

43
0.86422309    0.92223429    1.01158233    0.9104241
0.89430544    0.94844242    1.02278568    0.92359447

44
1.27158116    1.13184697    1.04536703    1.14442305
1.27516685    1.13422505    1.04485766    1.14831441

45
1.11701146    1.17256494    1.28640926    1.15776833
1.12401505    1.16975573    1.27907894    1.15169125

46
1.28389387    1.14280663    1.0554893     1.15550448
1.2746174     1.13718443    1.05655874    1.15192621

47
1.62992909    1.45081601    1.33996491    1.46693618
1.62765199    1.44997096    1.35108005    1.46608604

48
0.76450235    0.68158014    0.69654658    0.687333
0             0             0             0

49
0.77299682    0.85888535    0.77299682    0.78253999
0.78531956    0.87490758    0.78524762    0.79207168

50
1.24918602    1.11338871    1.13814726    1.12305867
0             0             0             0

51
1.21639456    1.08416196    1.1082706     1.09357809
0             0             0             0

52
1.26306586    1.40340651    1.26306586    1.27865927
1.2651659     1.40225814    1.26511058    1.28333376

53
1.22991005    1.36656672    1.22991005    1.24509413
1.24945898    1.39013319    1.24922116    1.2655865

54
1.82459184    1.63609183    1.66240589    1.64146145
0             0             0             0

55
1.84486508    2.04985009    1.84486508    1.86764119
1.84533658    2.04801122    1.84520137    1.86992765

56
0.75651225    0.61114796    0.61831587    0.60856216
0.7628018     0.62116795    0.62389231    0.62351633
```

```
57
0.7920947     0.95425352    0.78083627    0.96492058
0.80360867    0.9715004     0.78845726    0.98003063

58
1.23575551    0.99759454    1.00869851    0.99091625
1.23996172    0.99524851    1.00467967    0.99091776

59
1.20331661    0.95460946    0.98207329    0.96303981
1.2219177     0.96294757    1.00267553    0.97549384

60
1.29572069    1.44063457    1.26490002    1.57666658
1.29616345    1.42969475    1.26742822    1.57378341

61
1.26367451    1.24224946    1.22039845    1.53527867
1.27415084    1.25618365    1.23190081    1.54840604

62
1.81713404    1.44939311    1.48185906    1.55582592
1.8003472     1.44545003    1.46361803    1.56240094

63
1.89551177    1.86318803    1.8289222     2.302918
1.90135129    1.8631724     1.83216824    2.30085268

64
0.68029019    0.60618135    0.56440279    0.66886353
0.68966217    0.61807909    0.57289795    0.68231784

65
0.8909458     0.96306081    0.86964504    1.08158369
0.89495699    0.97672817    0.87840916    1.09925238

66
1.11055678    0.98796177    0.90691504    1.08630341
1.11061581    0.98829249    0.8941426     1.07229399

67
1.06068626    0.94506484    0.88147918    0.86946004
1.06460038    0.95009427    0.91595113    0.87595062

68
1.44114012    1.44269517    1.29903885    1.62237554
1.43311256    1.43193253    1.29891855    1.61317406

69
1.38310531    1.23156359    1.13940971    1.38024579
1.39907567    1.24436795    1.15062036    1.39012838

70
1.61140816    1.43505654    1.33188548    1.40079304
1.60317807    1.43865977    1.31829459    1.39370745

71
2.07465796    1.84527689    1.6904981     2.07036868
```

```
2.07457498    1.83898115    1.70215543    2.0469355

72
0.59659629    0.67572351    0.67359004    0.51085746
0.61143032    0.6900313     0.68410889    0.53069676

73
0.95658904    1.07108401    0.87798323    0.91045903
0.97001539    1.09400426    0.88613076    0.90735973

74
0.97997713    1.09757129    1.0999582     0.91069371
0.98491184    1.08240167    1.10512044    0.90921433

75
0.95658904    0.88229678    1.07108401    0.88709871
0.96157876    0.88692922    1.08842887    0.88842259

76
1.60867475    1.61826173    1.43606135    1.766238
1.60534005    1.5953644     1.44045118    1.76707049

77
1.58882117    1.39394139    1.40033118    1.74172739
1.59204293    1.40152535    1.41043596    1.74176923

78
1.58882117    1.41930837    1.62863403    1.74172739
1.60502505    1.41611591    1.61748219    1.74933581

79
2.38323175    2.09091208    2.10049678    2.61259109
2.3720744     2.07964477    2.09526407    2.61020335

80
0.61485131    0.75016964    0.61185017    0.68418669
0.62879847    0.76217008    0.62415089    0.69811872

81
0.97693818    1.21511254    0.97248035    1.10843783
0.9943716     1.24065814    0.98235351    1.11621629

82
1.00056096    1.21772012    0.99854723    1.11054997
0.99457949    1.20730232    0.99626424    1.09457743

83
0.95357786    0.78440529    0.95555343    0.85054756
0.96241215    0.79165195    0.96318903    0.85886842

84
1.59469297    1.82266882    1.47308561    1.66265674
1.59216181    1.82638541    1.45413767    1.64877741

85
1.55032878    1.24224949    1.27707077    1.37996298
1.56230286    1.24604099    1.29191346    1.38792679

86
```

```
1.55032878   1.26279675   1.46199607   1.37996298
1.55544593   1.26198408   1.45915962   1.38601209

87
2.32549317   1.86337423   1.91560615   2.06994447
2.31275233   1.8554528    1.90957372   2.05745214

88
0.67577468   0.7570808    0.68075071   0.68978472
0            0            0            0

89
1.09478984   1.22831869   1.10429827   1.11913481
0            0            0            0

90
1.09690198   1.22831869   1.10453295   1.11913481
0            0            0            0

91
0.85054756   0.7654928    0.77494333   0.7654928
0.85471822   0.76916672   0.78134476   0.76877587

92
1.64218477   1.84247804   1.65644741   1.67870222
0            0            0            0

93
1.37996298   1.24196668   1.2572996    1.24196668
1.38948284   1.24367108   1.26302481   1.24347166

94
1.37996298   1.24196668   1.2572996    1.24196668
1.3863737    1.23570488   1.25494507   1.23657224

95
2.06994447   1.86295002   1.8859494    1.86295002
2.05285805   1.85300096   1.87613354   1.85226471

96
0            0            0            0
0            0            0            0

97
0            0            0            0
0            0            0            0

98
0            0            0            0
0            0            0            0

99
0            0            0            0
0            0            0            0

100
0            0            0            0
0            0            0            0
```

```
101
0           0           0           0
0           0           0           0

102
0           0           0           0
0           0           0           0

103
0           0           0           0
0           0           0           0

104
0.04926024  0.50415731  0.54733604  0.49260243
0.0483798   0.53588396  0.55599184  0.51035155

105
0.98901099  0.89749646  0.96757805  0.89010989
0.98866103  0.90036143  0.97633727  0.89143689

106
0.98901099  0.90009514  0.99096614  0.89010989
0.9907417   0.9083252   0.99123894  0.89594103

107
0.98901099  0.89749646  0.96757805  0.89010989
0.9899801   0.90147376  0.9815799   0.89242632

108
1.97802198  1.76115922  1.63065278  1.78021978
1.97963781  1.75887956  1.59935596  1.77567408

109
1.97802198  1.75895327  1.61079919  1.78021978
1.97466463  1.76630454  1.63061305  1.78683377

110
1.97802198  1.75895327  1.61079919  1.78021978
1.98925551  1.73852692  1.63482685  1.78093177

111
2.96703297  2.6384299   2.41619878  2.67032967
2.96834088  2.63171597  2.40103442  2.65989171
```