Proceedings of the International Conference on Neural Networks, Orlando Florida, June 1994.

REINFORCEMENT LEARNING IN CONTINUOUS TIME: ADVANTAGE UPDATING

Leemon C. Baird III

bairdlc@wL.wpafb.af.mil
Wright Laboratory
WL/AAAT, Bldg 635
2185 Avionics Circle
Wright-Patterson Air Force Base, OH 45433-7301

ABSTRACT

A new algorithm for reinforcement learning, *advantage updating*, is described. Advantage updating is a direct learning technique; it does not require a model to be given or learned. It is incremental, requiring only a constant amount of calculation per time step, independent of the number of possible actions, possible outcomes from a given action, or number of states. Analysis and simulation indicate that advantage updating is applicable to reinforcement learning systems working in continuous time (or discrete time with small time steps) for which standard algorithms such as *Q*-learning are not applicable. Simulation results are presented indicating that for a simple linear quadratic regulator (LQR) problem, advantage updating learns more quickly than *Q*-learning by a factor of 100,000 when the time step is small. Even for large time steps, advantage updating is never slower than *Q*-learning, and advantage updating is more resistant to noise than is *Q*-learning. Convergence properties are discussed. It is proved that the learning rule for advantage updating converges to the optimal policy with probability one.

REINFORCEMENT LEARNING

A *reinforcement learning problem* is an optimal control problem where the controller is given a scalar reinforcement signal (or cost function) indicating how well it is performing. The reinforcement signal is a function of the state of the system being controlled and the control signals chosen by the controller. The goal is to maximize the *expected total discounted reinforcement*, which for continuous time is defined as

$$E\left(\int_0^\infty \gamma^t r(x_t,u_t)dt\right) \quad (1)$$

where $E(\cdot)$ denotes expected value, and where $0<\gamma\_1$ is the *discount factor* which determines the relative significance of early versus later reinforcement. The controller is required to learn which control actions are best in each state in order to maximize the expected total discounted reinforcement. For a continuous-time system, the control signal is constantly changing as the state of the system being controlled changes. For a discrete-time system, the controller must choose a new control action every Δt seconds. For discrete time, the total discounted reinforcement received during one time step of duration Δt when performing action u_t in state x_t is defined as:

$$R_{\Delta t}(x_t,u_t) = \int_t^{t+\Delta t} \gamma^{\tau-t} r(x_\tau,u_\tau)d\tau \quad (2)$$

The goal for a discrete-time controller is to find actions that maximize the expected total discounted reinforcement:

$$E\left(\sum_{i=0}^{\infty} \left(\gamma^{\Delta t}\right)^i R_{\Delta t}(x_{i\cdot\Delta t},u_{i\cdot\Delta t})\right) \quad (3)$$

This expression is often written with Δt not shown and with γ chosen to implicitly reflect Δt, but is written here with the Δt shown explicitly so that expression (3) will reduce to expression (1) in the limit as Δt goes to zero. A *policy*, $\_(x)$, is a function that specifies a particular action for the controller to perform in each state x. The *optimal policy*, $\_^*(x)$, is a policy such that choosing $u_t=\_^*(x_t)$ results in maximizing the total discounted reinforcement for any choice of starting state. A reinforcement learning algorithm should modify the policy in the controller based on experience, so that the policy eventually converges to the optimal policy. The total discounted reinforcement

received by starting in state x and following the optimal policy is called the *value* of state x.

A reinforcement learning system typically uses a set of real-valued parameters to store the information that is learned. When a parameter is updated during learning, the notation

$$W \longleftarrow K \tag{4}$$

represents the operation of instantaneously changing the parameter W so that its new value is K, whereas

$$W \xleftarrow{\alpha} K \tag{5}$$

represents the operation of moving the value of W toward K. This is equivalent to

$$W_{new} \longleftarrow (1 - \alpha) W_{old} + \alpha K \tag{6}$$

where the learning rate α is a small positive number.

Q-LEARNING

One popular reinforcement learning algorithm is Q-learning (Watkins, 1989, Watkins and Dayan, 1992). A controller using Q-learning must store a number called a Q *value* for each possible action in each state. For a given state x and action u, the optimal Q value, $Q^*(x,u)$, is the expected total discounted reinforcement that is received by starting in state x, performing action u on the first time step, then performing optimal actions thereafter. The maximum Q value in a state is the value of that state. The action associated with the maximum Q value in a state is the policy for that state. Initially, all Q values are set to arbitrary numbers. During learning, an action u is performed in state x, resulting in a transition to state x', and a receipt of reinforcement $R_\Delta(x,u)$. The Q value is then updated:

$$Q(x,u) \xleftarrow{\alpha} R_{\Delta t}(x,u) + \gamma^{\Delta t} \max_u Q(x',u) \tag{7}$$

If the Q values are stored in a lookup table, then update (7) is very easy to implement. If the Q values are represented by a function approximation system, such as a multilayer perceptron network, then the network would take x and u as inputs, would give $Q(x,u)$ as an output, and would use the expression on the right side of the arrow in (7) as the "desired output" for the network during training. The parameter α would be the learning rate for the network. In this case, it is not obvious how to find the maximum Q value required for update (7). One algorithm that does this is described in Baird (1992). Another method, *wire fitting*, is described in Baird and Klopf (1993b). Reinforcement learning systems based on discrete Q-learning are described in Baird and Klopf (1993a), and Klopf, Morgan, and Weaver (1993).

Q-learning requires relatively little computation per update, but it is useful to consider how the number of updates required scales with noise or with the duration of a time step, Δt. An important consideration is the relationship between Q values for the same state, and between Q values for the same action. The Q values $Q(x,u_1)$ and $Q(x,u_2)$ represent the long-term reinforcement received when starting in state x and performing action u_1 or u_2 respectively, followed by optimal actions thereafter. In a typical reinforcement learning problem with continuous states and actions, it is frequently the case that performing one wrong action in a long sequence of optimal actions will have little effect on the total reinforcement. In such a case, $Q(x,u_1)$ and $Q(x,u_2)$ will have relatively close values. On the other hand, the values of widely separated states will typically not be close to each other. Therefore $Q(x_1,u)$ and $Q(x_2,u)$ may differ greatly for some choices of x_1 and x_2. Therefore, if the network representing the Q function makes even small errors, the policy derived from it will have large errors. As the time step duration Δt approaches zero, the penalty for one wrong action in a sequence decreases, the Q values for different actions in a given state become closer, and the implied policy becomes even more sensitive to noise or function approximation error. In the limit, for continuous time, the Q function contains no information about the policy. Therefore, Q-learning would be expected to learn slowly when the time steps are of short duration, due to the sensitivity to errors, and it is incapable of learning in continuous time. This problem is not a property of any particular function approximation system; rather, it is inherent in the definition of Q values.

ADVANTAGE UPDATING

Reinforcement learning in continuous time is possible through the use of *advantage updating*. The advantage updating algorithm is a reinforcement learning algorithm in which two types of information are stored. For each

state x, the value $V(x)$ is stored, representing the total discounted return expected when starting in state x and performing optimal actions. For each state x and action u, the *advantage,* $A(x,u)$, is stored, representing the degree to which the expected total discounted reinforcement is increased by performing action u (followed by optimal actions thereafter) relative to the action currently considered best. After convergence to optimality, the value function $V^*(x)$ represents the true value of each state. The advantage function $A^*(x,u)$ will be zero if u is the optimal action (because u confers no advantage relative to itself) and $A^*(x,u)$ will be negative for any suboptimal u (because a suboptimal action has a negative advantage relative to the best action). For a given action u, the Q value $Q^*(x,u)$ represents the utility of that action, and the advantage $A^*(x,u)$ represents the utility of that action *relative* to the optimal action. The optimal advantage function A^* can be defined in terms of the optimal value function V^*:

$$A^*(x,u) = \frac{1}{\Delta t}\left[R_{\Delta t}(x,u) - V^*(x) + \gamma^{\Delta t}\sum_{x'} P(u,x,x')V^*(x')\right] \quad (8)$$

where $P(u,x,x')$ is the probability of transitioning from state x to state x' when performing action u. The definition of an advantage includes a $1/\Delta t$ term to ensure that, for small time step duration Δt, the advantages will not all go to zero. Advantages are related to Q values by:

$$A^*(x,u) = \frac{1}{\Delta t}\left[Q^*(x,u) - \max_{u'} Q^*(x,u')\right] \quad (9)$$

Both the value function and the advantage function are needed during learning, but after convergence to optimality, the policy can be extracted from the advantage function alone. The optimal policy for state x is any u that maximizes $A^*(x,u)$. All of the advantages in a state are relative to a reference advantage, $A_{\text{ref}}(x)$, which is defined by equation (10).

$$A_{\text{ref}}(x) = \max_u A(x,u) \quad (10)$$

If A_{ref} is zero in every state, then the advantage function is said to be *normalized*. A_{ref} should eventually converge to zero in every state. The update rules for advantage updating for optimal control in discrete time are given by updates (11), (12), and (13).

The Advantage Updating Algorithm

LEARN:perform action u_t in state x_t

$$A(x_t,u_t) \xleftarrow{\alpha} A_{\text{ref}}(x_t) + \frac{R_{\Delta t}(x_t,u_t) + \gamma^{\Delta t}V(x_{t+\Delta t}) - V(x_t)}{\Delta t} \quad (11)$$

$$V(x_t) \xleftarrow{\beta} V(x_t) + \left[A_{\text{ref}_{new}}(x_t) - A_{\text{ref}_{old}}(x_t)\right]/\alpha \quad (12)$$

NORMALIZE:pick an arbitrary state x and pick an action u randomly with uniform probability

$$A(x,u) \xleftarrow{\omega} A(x,u) - A_{\text{ref}}(x) \quad (13)$$

For the learning updates, the system performs action u_t in state x_t and observes the reinforcement received, $R_{\Delta t}(x_t,u_t)$, and the next state, $x_{t+\Delta t}$. The advantage and value functions are then updated according to updates (11) and (12). Update (11) modifies the advantage function $A(x,u)$. The maximum advantage in state x prior to applying update (11) is $A_{\text{ref}_{old}}(x)$. After applying update (11) the maximum is $A_{\text{ref}_{new}}(x)$. If these are different, then update (12) changes the value $V(x)$ by a proportional amount. Advantage updating can be applied to continuous-time systems by taking the limit as Δt goes to zero in updates (11), (12), and (13). For (12) and (13), Δt can be replaced with zero. Substituting equation (2) into update (11) and taking the limit as Δt goes to zero yields:

$$A(x_t,u_t) \xleftarrow{\alpha} A_{\max}(x_t) + V(x_t)\ln\gamma + \dot{V}(x_t) + r(x_t,u_t) \quad (14)$$

Learning is done to find the correct policy. Normalization is done to ensure that after convergence $A_{\text{ref}}(x)=0$ in every state, and so the function will remain in a form that can be represented easily. This avoids the representation problem noted above for Q-learning, where the Q function differs greatly between states but differs little between actions in the same state. Learning and normalizing can be performed asynchronously. For example, a system might perform a learning update once per time step, in states along a particular trajectory through state space, and perform a normalizing update multiple times per time step in states scattered randomly throughout the state space. The advantage updating algorithm is referred to as "advantage updating" rather than "advantage learning"

	Information stored for state x, action u	Update rules	Bellman equation	Direct	Converge to *	Cont. time
Q-learning	Q(x,u)	$Q \xleftarrow{\alpha} R + \gamma^{\Delta t} \max Q'$	$Q = E[R + \gamma^{\Delta t} \max Q']$	yes	yes	no
Value iteration	V(x)	$V \xleftarrow{\alpha} R + \gamma^{\Delta t} \max V'$	$V = E[R + \gamma^{\Delta t} \max V']$	no	yes	yes
Advantage updating	V(x) A(x,u)	$A \xleftarrow{\alpha} A_{ref} + (R + \gamma^{\Delta t} V' - V)/\Delta t$ $V \xleftarrow{\beta} V + \Delta A_{ref}/\alpha$ For a randomly, uniformly chosen action: $A \xleftarrow{\omega} A - A_{ref}$	$V = E[R + \gamma^{\Delta t} V'] - A\Delta t$ $A_{ref} = 0$	yes	yes	yes

Table 1

because it includes both learning and normalizing updates.

Advantage updating is *direct*; it learns to control a system without learning to predict its behavior. This is particularly useful if the system being controlled is stochastic. Traditional dynamic programming algorithms, such as *value iteration*, (Bertsekas, 1987) require an amount of calculation per update proportional to the number of possible outcomes from performing that action. If an action leads stochastically to a continuum of states, then value iteration would require complicated integrals to be calculated numerically for each update. For this reason, direct learning systems, such as *Q*-learning and advantage updating, can require much less computation per update. If a model of the system being controlled is known or learned, then the system can learn by interacting with the model as in the Dyna system (Sutton, 1990). Table 1 compares advantage updating, *Q*-learning, and value iteration. All three learning algorithms are guaranteed to converge to the optimal policy in the discrete case. Value iteration is not direct because it must learn a model of the system being controlled. Q-learning does not work in continuous time. Advantage updating has all three properties. An equation called the *Bellman Equation* (Bertsekas, 1987) indirectly defines the optimal value function. It is an equation containing the function $V(x)$ that is only satisfied everywhere if $V(x)$ is the optimal value function. The chart shows this equation, and the analogous equations for the *Q* function and advantage function, where $E[]$ denotes the expected value.

Analysis has shown that many reasonable-looking algorithms for reinforcement learning can fail to converge in some situations (Williams and Baird, 1990, 1993). When Q-learning is applied to a finite, discrete system, and when a lookup table is used to store the Q values, Q-learning is guaranteed to converge to the optimal policy. The learning rule for advantage updating can be shown to also have this desirable property. Equation (9) showed a relationship between *Q* values and advantages. Given an advantage function, solving this equation for *Q* gives a *Q* function that is implied by the advantage function. The learning part of the advantage updating algorithm, updates (11) and (12), cause the implied *Q* function to change as if it were being updated by the Q-learning algorithm. Because Q-learning is guaranteed to converge to the optimal policy, the learning rule for advantage updating is also guaranteed to converge. This trivial proof does not consider the convergence of the full advantage updating algorithm, including both learning and normalizing. This more general case can be addressed using the theorems of Jaakkola, Jordan, and Singh (1993), and will be considered in a forthcoming paper.

LINEAR QUADRATIC REGULATOR

Linear Quadratic Regulator (LQR) problems are commonly used as test beds for control systems, and are useful benchmarks for reinforcement learning systems (Bradtke, 1993). The following linear quadratic regulator (LQR) control problem can serve as a benchmark for comparing *Q*-learning to advantage updating in the presence of noise or small time steps. At a given time t, the state of the system being controlled is the real value x_t. The controller chooses a control action u_t which

is also a real value. The dynamics of the system are:

$$\dot{x}_t = u_t \tag{15}$$

The rate of reinforcement to the learning system, $r(x_t,u_t)$, is

$$r(x_t,u_t) = -x_t^2 - u_t^2 \tag{16}$$

Given some positive discount factor γ_1, the goal is to maximize the total discounted reinforcement:

$$\int_0^\infty \gamma^t r(x_t,u_t)dt \tag{17}$$

A discrete-time controller can change its output every Δt seconds, and its output is constant between changes. The discounted reinforcement received during a single time step is

$$\begin{aligned} R_{\Delta t}(x_t,u_t) &= \int_t^{t+\Delta t} \gamma^{\tau-t} r(x_\tau,u_\tau)d\tau \\ &= \int_t^{t+\Delta t} \gamma^{\tau-t}(-(x_\tau+\tau u_\tau)^2 - u_\tau^2)d\tau \end{aligned} \tag{18}$$

and the total reinforcement to be maximized is

$$\sum_{i=0}^{\infty} (\gamma^{\Delta t})^i R_{\Delta t}(x_{i\Delta t},u_{i\Delta t}) \tag{19}$$

Given this control problem, it is possible to calculate the optimal policy $\_^*(x)$, value function $V^*(x)$, Q value function $Q^*(x,u)$, and advantage function $A^*(x,u)$. These functions are linear or quadratic for all Δt and γ_1.

$$\pi^*(x) = -k_1 x \tag{20}$$

(21)

$$A^*(x,u) = -k_3(k_1 x + u)^2 \tag{22}$$

$$\begin{aligned} Q^*(x,u) = \\ -(k_2 + \Delta t k_1^2 k_3)x^2 - 2\Delta t k_1 k_3 xu - \Delta t k_3 u^2 \end{aligned} \tag{23}$$

The constants k_i are positive for all nonnegative values of Δt and γ_1. For Δt=0 and γ=1, all k_i=1. The appendix gives the general formula for each k_i as a function of Δt and γ. Note that for continuous time (Δt=0), the optimal Q function is a function of x only, not u, and so the Q function no longer contains any information about the policy.

SIMULATION RESULTS

Advantage updating and Q-learning were compared on the LQR problem described in the previous section. In the simulations, the V function was approximated by the expression w_1x^2, and the A and Q functions were approximated by $w_2x^2+w_3xu+w_4u^2$. All weights, w_i, were initialized to random values between $\pm 10^{-4}$, and were updated by simple gradient descent. Each Q function was initialized with the same weights as the corresponding advantage function to ensure a fair comparison. The control action chosen by the learning system was constrained to lie in the range [-1,1]. When calculating the maximum A or Q value in a given state, only actions in this range were considered. On each time step, a state was chosen randomly from the interval [-1,1]. With probability 0.5, an action was also chosen randomly and uniformly from that interval. With probability 0.5, the learning system chose an action according to its current policy. The advantage updating system also performed one normalization step on each time step in a state chosen randomly and uniformly from [-1,1]. A set of 100 Q-learning systems and 100 advantage updating systems were allowed to run in parallel, initialized with different random weights, and all exploring with different random states and actions. At any given time, the policy of each system was a linear function. The absolute value of the difference between the constant in the current policy and the constant in the optimal policy was calculated for each of the 200 learning systems. For Q-learning and advantage updating, the solution was said to have been learned when the mean absolute error for the 100 learning systems running in parallel fell below 0.001. Figure 1 shows the number of time steps required for learning when various amounts of noise were added to the reinforcement signal. Figure 2 shows the number of time steps required for learning with various time step durations.

Figure 1

Figure 2

For the simulations described here, normalization was done once after each learning update, and both types of update used the same learning rate. Advantage updating could be optimized by changing the number of normalizing updates performed per learning update, but this was not done here. To ensure a fair comparison for the two learning algorithms, the learning rate for Q-learning was optimized for each simulation. Rates were found by exhaustive search that were optimal to two significant digits. The rates for advantage updating had only a single significant digit, and were not exhaustively optimized. The rates used were sufficiently good to demonstrate that advantage updating learned faster than Q-learning in every simulation. Advantage updating appears more resistant to noise than Q-learning, with learning times that are shorter by a factor of up to seven. This may be due to the fact that noise introduces errors into the stored function, and the policy for advantage updating is less sensitive to errors in the stored functions than for Q-learning. All of figure 1, and the leftmost points of figure 2, represent simulations with large time steps. When the time step duration is small, the difference between the two algorithms is more dramatic. In figure 2, as the time step duration Δt approaches zero (continuous time), advantage updating is able to solve the LQR problem in a constant 216 time steps. Q-learning, however, requires approximately $10/\Delta t$ time steps. Simulation showed a speed increase for advantage updating by a factor of over 160,000. Smaller time steps might have resulted in a larger factor, but Q-learning would have learned too slowly for the simulations to be practical. Even for a fairly large time step of Δt=0.03, advantage updating learned twice as quickly as Q-learning. When Δt=0.03, the optimal policy reduces x by 90% in 81 time steps. This suggests that if a controller updates its outputs 50 times per second, then advantage updating will learn significantly faster than Q-learning for operations that require at least 2 seconds (100 time steps) to perform. Further research is necessary to determine whether this is true for systems other than a simple LQR problem.

CONCLUSION

Advantage updating is shown to learn much faster than Q-learning for problems with small time steps or noise, and no slower than Q-learning for other problems. Advantage updating works in continuous time, which Q-learning cannot do. Unlike value iteration, it is possible for advantage updating to learn the policy without learning a model. If a model is known or learned, advantage updating may be combined with the model as in Dyna (Sutton, 1990). The learning rule for advantage updating is guaranteed to converge to the optimal policy for systems with discrete states and actions when the values and advantages are stored in a lookup table.

ACKNOWLEDGMENTS

This research was supported under Task 2312R1 by the Life and Environmental Sciences Directorate of the United States Air Force Office of Scientific Research. The author gratefully acknowledges the contributions of Harry Klopf, Jim Morgan, Gábor Bartha, Scott Weaver, Mance Harmon, and Tommi Jaakkola.

REFERENCES

Baird, L. C. (1992). Function minimization for dynamic programming using connectionist networks. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics* (pp. 19-24). Chicago, IL.

Baird, L. C., & Klopf, A. H. (1993a). A hierarchical network of provably optimal learning control systems: Extensions of the associative control process (ACP) network. *Adaptive Behavior*, **1**(6), 321-352.

Baird, L. C., & Klopf, A. H. (1993b). *Reinforcement Learning with High-Dimensional, Continuous Actions*. To appear as a United States Air Force technical report.

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall.

Bradtke, S. J (1993). Reinforcement learning applied to linear quadratic regulation. *Proceedings of the Fifth Conference on Neural Information Processing Systems* (pp. 295-302). Morgan Kaufmann.

Jaakkola, T., Jordan, M. I., & Singh, S. P. (1993). *On the Convergence of Stochastic Iterative Dynamic Programming Algorithms* (Tech. Rep. 9307). Department of Brain and Cognitive Sciences, Massachusetts Institute of Technology, Cambridge, MA.

Klopf, A. H., Morgan, J. S., & Weaver, S. E. (1993). A hierarchical network of control systems that learn: Modeling nervous system function during classical and instrumental conditioning. *Adaptive Behavior*, **1**(6), 263-319.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* . Doctoral thesis, Cambridge University, Cambridge, England.

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note: *Q*-learning. *Machine Learning*, **8**(3/4), 279-292.

Williams, R. J., & Baird, L. C. (1990). A mathematical analysis of actor-critic architectures for learning optimal control through incremental dynamic programming. *Proceedings of the Sixth Yale Workshop on Adaptive and Learning Systems* (pp. 96-101). New Haven, CN.

Williams, R. J., & Baird, L. C. (1993). *Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems*. (Tech. Rep. NU-CCS-93-11). Boston, MA: Northeastern University, College of Computer Science.

APPENDIX: LQR CONSTANTS

For Δt_0 and γ_1, the following 3 equations give the constants, k_i, for the optimal controller for the LQR problem. If $\Delta t=0$, or $\gamma=1$, or both, the constants are calculated by evaluating the limit of the right side the equations as Δt goes to zero, or γ goes to one, or both. The constants found by these equations can then be used to check whether a learning system has learned the correct policy. This was used in the simulation to determine how long it took for Q-learning and advantage updating to find the correct policy in each case.

$$k_1 = \left(\frac{1-\gamma^{\Delta t}}{\Delta t}\right)\frac{2\gamma^{\Delta t} - 2\Delta t \ln\gamma - 2 - (1-\gamma^{\Delta t})\ln^2\gamma + \sqrt{(2+\ln^2\gamma)^2(1-\gamma^{\Delta t})^2 - 4\Delta t^2\gamma^{\Delta t}\ln^2\gamma}}{2 - 2\gamma^{2\Delta t} + 4\Delta t\gamma^{\Delta t}\ln\gamma + (1-\gamma^{2\Delta t})\ln^2\gamma + (1-\gamma^{\Delta t})\sqrt{(2+\ln^2\gamma)^2(1-\gamma^{\Delta t})^2 - 4\Delta t^2\gamma^{\Delta t}\ln^2\gamma}} \qquad (24)$$

$$k_2 = \frac{(2+\ln^2\gamma)(1-\gamma^{\Delta t})^2 - 2\Delta t^2\gamma^{\Delta t}\ln^2\gamma - (1-\gamma^{\Delta t})\sqrt{(2+\ln^2\gamma)^2(1-\gamma^{\Delta t})^2 - 4\Delta t^2\gamma^{\Delta t}\ln^2\gamma}}{2\Delta t^2\gamma^{\Delta t}\ln^3\gamma} \qquad (25)$$

$$k_3 = \frac{(2 + \ln^2 \gamma)(\gamma^{2\Delta t} - 1) - 4\Delta t \gamma^{\Delta t} \ln \gamma - (1 - \gamma^{\Delta t})\sqrt{(2 + \ln^2 \gamma)^2 (1 - \gamma^{\Delta t})^2 - 4\Delta t^2 \gamma^{\Delta t} \ln^2 \gamma}}{2\Delta t \gamma \ln^3 \gamma} \quad (26)$$