Νευρο-Ασαφής Έλεγχος & Εφαρμογές
Εργαστηριακή Άσκηση
Q Learning

Όνομα: Μάριος Παπαχρήστου
Αριθμός Μητρώου: 03115101 Σχολή ΗΜΜΥ
e-mail: papachristoumarios@gmail.com

"Incorporating general intelligence, bodily intelligence, emotional intelligence, spiritual intelligence, political intelligence and social intelligence in AI systems are part of the future deep learning research. " – Amit Ray, Compassionate Artificial Intelligence

1 Σκοπός Εργασίας

Σκοπός της παρούσας εργασίας είναι η ανάπτυξη ενός ελεγκτή state-feedback διακριτού χρόνου για τη σταθεροποίηση ενός ΓΧΑ συστήματος με χρήση Q Learning βάσει τετραγωγνικού κριτηρίου κόστους. Συγκεκριμένα δίνεται το σύστημα

$$x_{k+1} = Ax_k + Bu_k$$

με δυναμικούς πίνακες

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{1}$$

Και το τετραγωνικό κριτήριο κόστους άπειρου ορίζοντα

$$J = \sum_{i=0}^{\infty} (x_i^T x_i + \rho u[i]^2) \qquad \rho > 0$$

2 Q-Learning

Θέλουμε να βρούμε βέλτιστο ελεγκτή $u_k^* = -Kx_k$ τέτοιο ώστε το σύστημα να είναι ασυμπτωτικά ευσταθές και να ελαχιστοποιείται το κριτήριο κόστους J. Με χρήση ενισχυτικής μάθησης, και συγκεκριμένα Q-Learning, μπορούμε να βρούμε αυτή την είσοδο, αγνοώντας τη δυναμική του συστήματος. Θέλουμε να λύσουμε το εξής πρόβλημα βελτιστοποίησης

$$\min_{u_1,\dots,u_T} \sum_{k=1}^{T} r_k(x_k, u_k)$$
$$\text{s.t. } x_{k+1} = f(x_k, u_k)$$

Στην περίπτωση μας η συνάρτηση $r_k(x_k, u_k)$ είναι η $r_k(u_k, x_k) = x_k^T x_k + \rho u_k^2$. Εισάγουμε την Q-function για την αξιολόγηση μιας πολιτικής K ως την cost το go συνάρτηση

$$Q_K(x_k, u_k) \sum_{t\geq k} r_t(x_t, u_t) = r_k(x_k, u_k) + Q_K(x_{k+1}, u_{k+1}) = r_k(x_k, u_k) + x_{k+1}^T P_K x_{k+1} \tag{2}$$

όπου ο όρος $x_{k+1}^T P_K x_{k+1}$ είναι το cost-to-go. Η παραπάνω σχέση μπορεί να λυθεί με δυναμικό προγραμματισμό προκειμένου να λάβουμε την βέλτιση συνάρτηση $Q_K^*(x_k, u_k)$ από την εξίσωση του Bellman

$$Q_K^*(x_k, u_k) = r_k(x_k, u_k) + Q_K^*(x_{k+1}, u_{k+1})$$

Για την ανεύρεση του βέλτιστου κόστους $J^* = x_0^T P^* x_0$ για το πρόβλημα. Η συνάρτηση ποιότητας μπορεί να γραφεί ως

$$Q_K(x_k, u_k) = z_k^T \begin{pmatrix} I + A^T PA & B^T PA \\ A^T PB & \rho + B^T PB \end{pmatrix} z_k = z_k^T H z_k$$

όπου $z_k = \begin{pmatrix} x_k \\ u_k \end{pmatrix}$ το επαυξημένο διάνυσμα. Παρατηρούμε ότι ορίζοντας το διάνυσμα $\bar{z}_k = \mathrm{vec}(z_k z_k^T)$ και το διάνυσμα $\bar{H} = \mathrm{vec}(H)$ η εξίσωση (2) γράφεται

$$\bar{H}^T(\bar{z}_k - z_{k+1}) = r_k$$

Για την εύρεση της H θεωρούμε τους πίνακες

$$R = (r_1, \dots, r_T)^T \qquad Z = \begin{pmatrix} [\bar{z}_0 - \bar{z}_1]^T \\ \vdots \\ [\bar{z}_{T-1} - \bar{z}_T]^T \end{pmatrix}$$

Και η παραπάνω εξίσωση λαμβάνει τη μορφή γραμμικού συστήματος

$$Z\bar{H} = R$$

η οποία έχει λύση την $\bar{H} = Z^+ R$. Στη συνέχεια επαναφέρουμε τον πίνακα στην αρχική του μορφή

$$H = \begin{pmatrix} H_{xx} & H_{xu} \\ H_{ux} & H_{uu} \end{pmatrix} \qquad \text{όπου } H = H^T > 0$$

Ο βέλτιστος νόμος ελέγχου βρίσκεται με το ακρότατο ως προς u_k της συνάρτησης ποιότητας

$$\frac{\partial Q_K(x_k, u_k)}{\partial u_k} = 0 \iff K^* = H_{uu}^{-1} H_{ux}$$

Για τη συλλογή των δειγμάτων χρησιμοποιείται μια τυχαία είσοδος $u_k = -Lx_k,\ L \sim \mathcal{N}(0, 1_{3\times 1})$.

Στην υλοποίησή μας με MATLAB, η υλοποίηση της παραπάνω διαδικασίας γίνεται με τη χρήση της ακόλουθης συνάρτησης

```
function [ H, K ] = q_learning(A, B, K, Q, rho, Niter, x0)

    % Initialize variables
    x = x0;
    u = -K * x;
    Y = [];
    X = [];

    for n = 1 : Niter
```

```
        % Calculate quadratic combinations
        phi = quad_comb(x, u);
        r = x' * Q * x + u' * rho * u;

        % update x_k+1 u_k + 1
        x = A * x + B * u;
        u = -K * x;

        % Calculate new quadratic combinations
        phi_new = quad_comb(x, u);

        % Calculate entry
        dphi = phi' - phi_new';

        % Append to matrices
        X = [X; dphi];
        Y = [Y; r];

    end

    % Solve Least Squares Problem
    psi = pinv(X) * Y;

    dim = length(x) + length(u);
    H = reshape(psi, dim, dim);

    % Get desired elements
    Hux = H(length(x) + 1 : end, 1:length(x));
    Huu = H(length(x) + 1 : end, length(x) + 1 : end);

    % Find optimal value of gain
    K = inv(Huu) * Hux;

end
```

3 Ιδανική Περίπτωση

Η εύρεση ελεκγτή $u_k = K\mathbf{x}_k$ ισοδυναμεί με την επίλυση της Αλγεβρικής Εξίσωσης Ricatti για συστήματα διακριτού χρόνου

$$A^T PA - A^T PB(B^T PB + R)^{-1} B^T PA = -Q$$

όπου $Q = I \geq 0, R = \rho > 0$ για την ανεύρεση της βέλτιστης εισόδου

$$u_k^* = -(R + B^T PB)^{-1}) B^T PAx_k$$

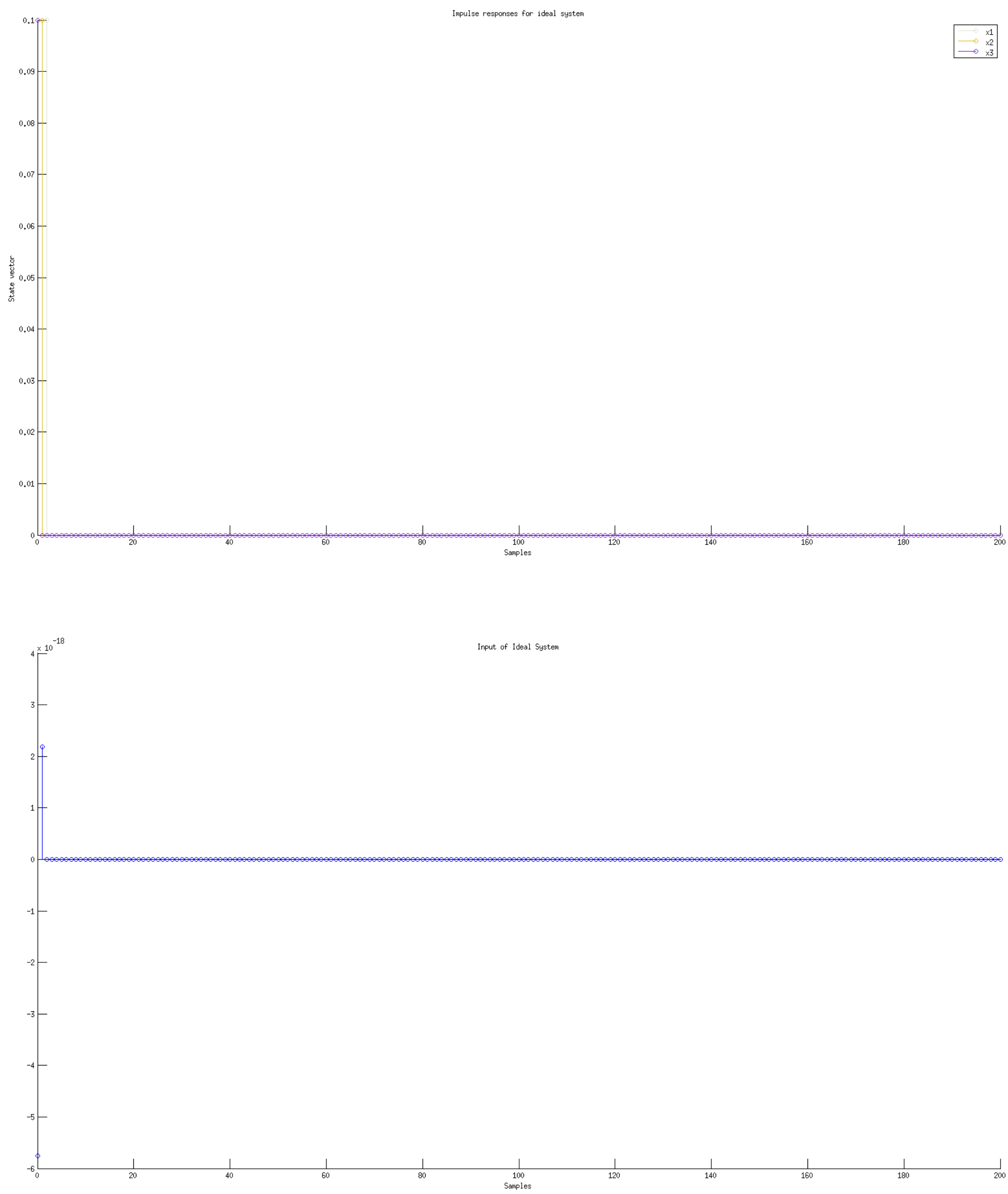
Με χρήση του MATLAB επιλύουμε τη Ricatti και λαμβάνουμε τις τιμές

```
Kid =

   1.0e-16 *
```

```
         0   -0.2184    0.7943

>> Pid

Pid =

    1.0000    0.0000   -0.0000
    0.0000    2.0000    0.0000
   -0.0000    0.0000    3.0000
```

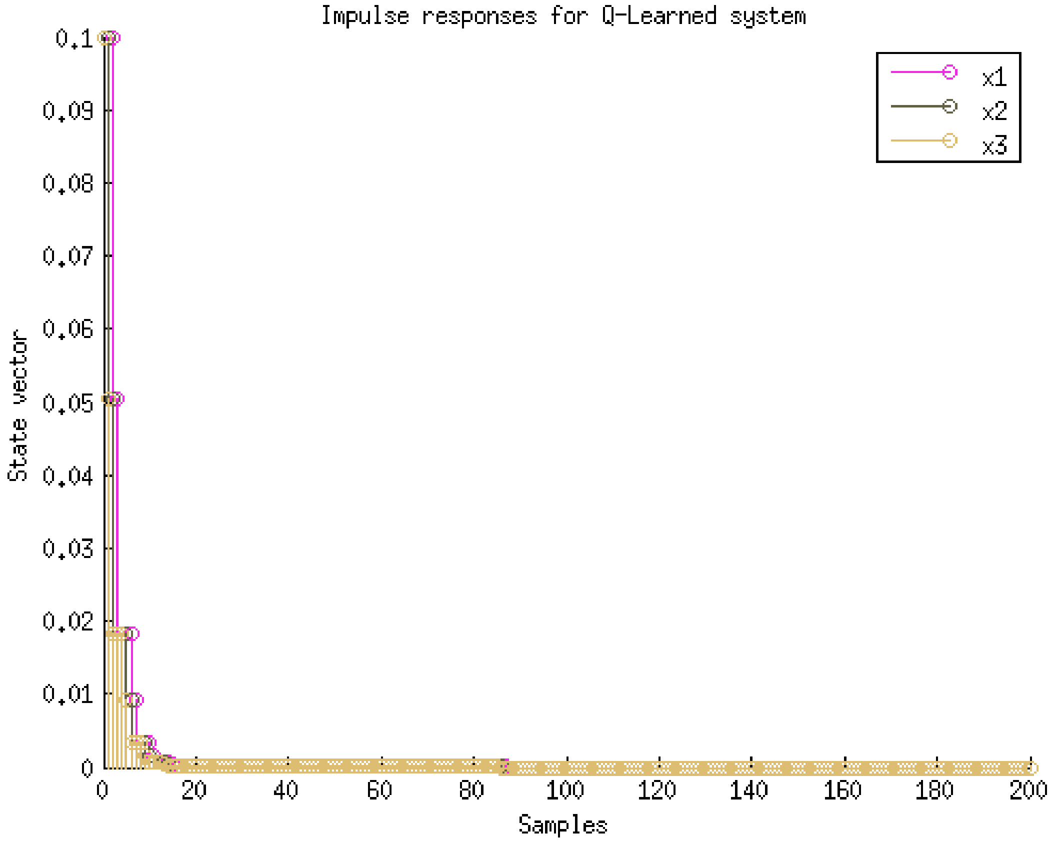
4 Σύγκριση Αποτελεσμάτων

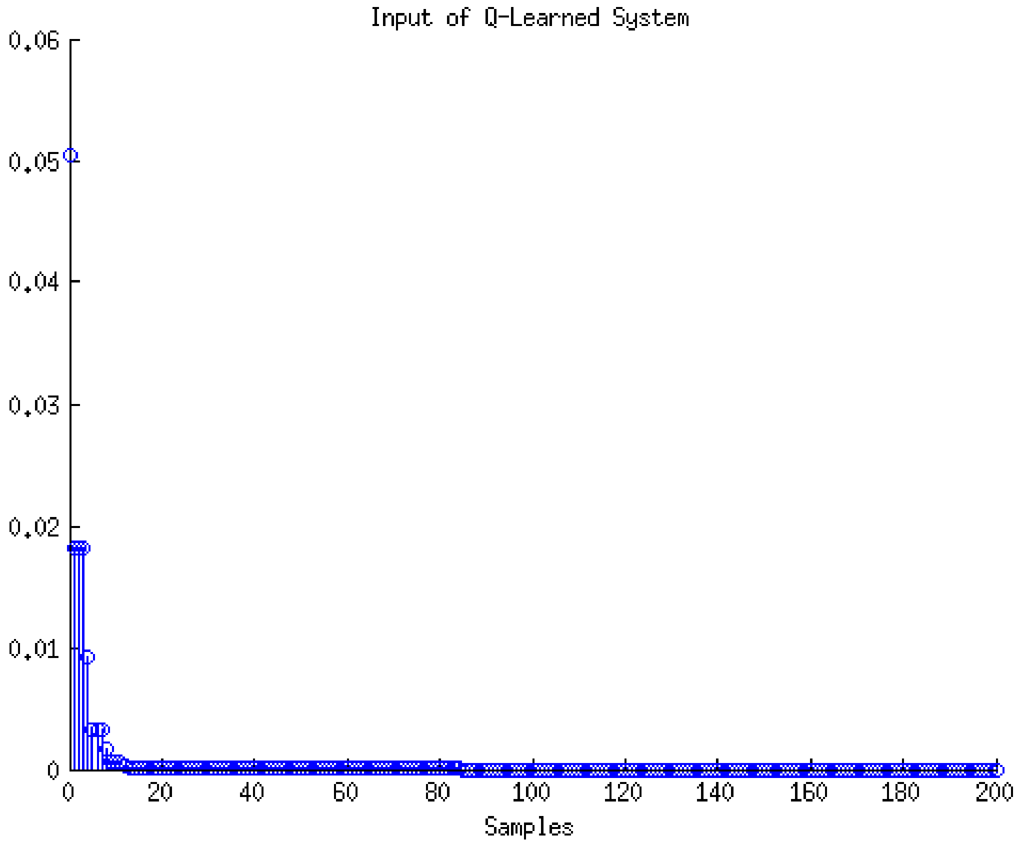
Θεωρήσαμε αρχική συνθήκη $x_0 = (0.1, 0.1)^T$. Για το ιδανικό σύστημα έχουμε τις εξής αποκρίσεις για το διάνυσμα κατάστασης και την είσοδο



Σχήμα 1: Ιδανικό Σύστημα (αναλυτική λύση της Ricatti)

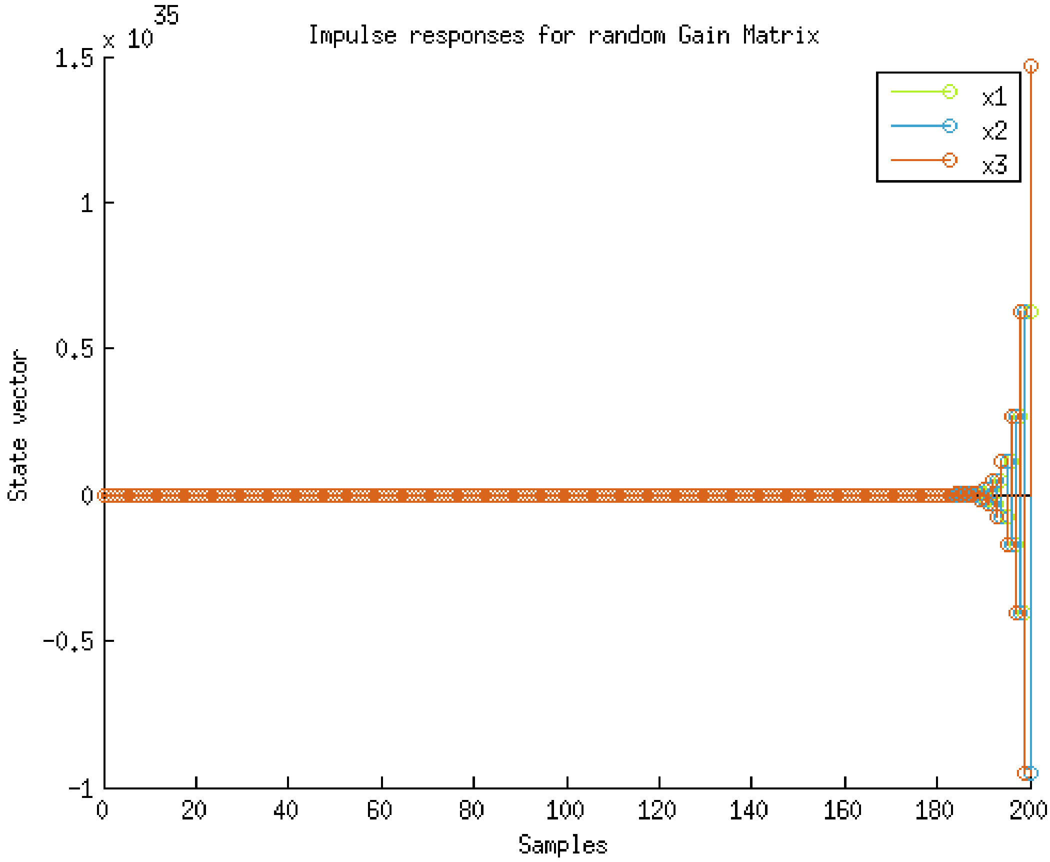
Για το σύστημα που σχεδιάσαμε με Q-Learning

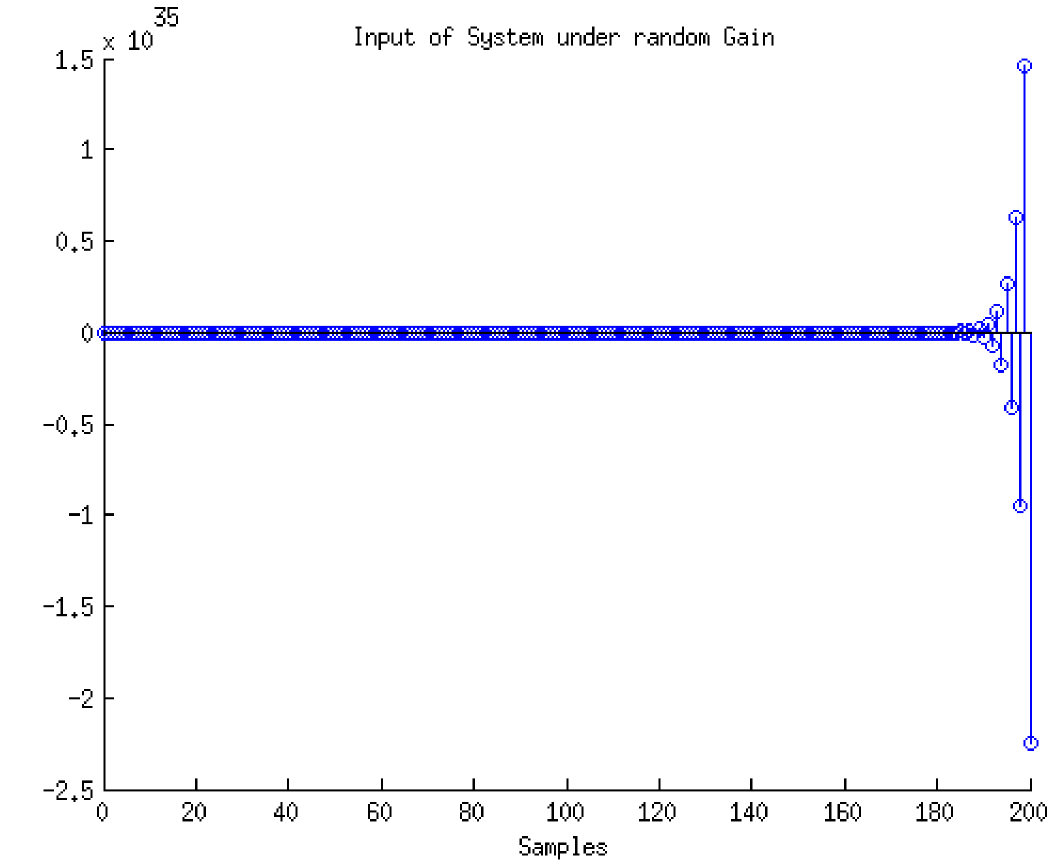




Σχήμα 2: Σύστημα με Q-Learning $K = (-0.2781, 0.4261, -0.6527)$

Για τη χρήση τυχαίων εισόδων





Σχήμα 3: Δείγματα με τυχαία είσοδο $L = (1.2178, -1.4951, 0.0373)$

Παρατηρούμε ότι το σύστημα από το οποίο πήραμε τα δείγματα είχε ασταθή συμπεριφορά, παρόλα αυτά μάθαμε μια τιμή gain η οποία έκανε το σύστημά μας ευσταθές.

5 Πηγαίος Κώδικας

Το κύριο αρχείο main.m

```
% Q Learning for Optimal Control with LQR Criterion
% Author: Marios Papachristou
% AM: 03115101
% email: papachristoumarios@gmail.com

%% Parameters
% Actual System Dynamics
A = [0 1 0; 0 0 1; 0 0 0];
B = [0; 0; 1];

% Sampling Period
Ts = 1;

% Choose initial condition
x0 = 0.1 * ones(length(A), 1);

% LQR Parameters
rho = 1;
Q = eye(size(A));

% Number of iterations
Niter = 200;
time = 0 : Niter;
epochs = 100;

%% Q Learning

for ep = 1 : epochs
    L = randn(size(B'));
    [H, K] = q_learning(A, B, L, Q, rho, Niter, x0);
    % Suppose we put this controller and simulate the model
    % if it is stable, we are good to go
    if max(abs(eig(A - B * K))) < 1
        break
    end
end



%% Ideal LQR
[Kid, Pid, e] = dlqr(A, B, Q, rho);

Aid_c = A - B * Kid;

ideal_model = ss(Aid_c, B, zeros(size(A)), zeros(size(B)), Ts);

[~, ~, x_id] = lsim(ideal_model, zeros(1, Niter + 1), time, x0);
```

```
figure;
hold on;
for i = 1 : length(x0)
    stem(time, x_id(:, i), 'color', rand(1,3));
end
title('Impulse responses for ideal system')
legend('x1', 'x2', 'x3')
xlabel('Samples')
ylabel('State vector')

hold off

hold off

figure;
hold on
stem(time, - Kid * x_id');
title('Input of Ideal System')
xlabel('Samples')

hold off


%% Q-Learned LQR

Aq_c = A - B * K;

q_learning_model = ss(Aq_c, B, zeros(size(A)), zeros(size(B)), Ts);

[~, ~, x_q] = lsim(q_learning_model, zeros(1, Niter + 1), time, x0);


figure;
hold on;
for i = 1 : length(x0)
    stem(time, x_q(:, i), 'color', rand(1,3));
end
title('Impulse responses for Q-Learned system')
legend('x1', 'x2', 'x3')
xlabel('Samples')
ylabel('State vector')

hold off

figure;
hold on
stem(time, - K * x_q');
title('Input of Q-Learned System')
xlabel('Samples')
```

```
hold off

%% Behaviour due to random gain

Aq_r = A - B * L;

q_learning_model = ss(Aq_r, B, zeros(size(A)), zeros(size(B)), Ts);

[~, ~, x_r] = lsim(q_learning_model, zeros(1, Niter + 1), time, x0);


figure;
hold on;
for i = 1 : length(x0)
    stem(time, x_r(:, i), 'color', rand(1,3));
end
title('Impulse␣responses␣for␣random␣Gain␣Matrix')
legend('x1', 'x2', 'x3')
xlabel('Samples')
ylabel('State␣vector')

hold off

hold off

figure;
hold on
stem(time, - L * x_r');
title('Input␣of␣System␣under␣random␣Gain')
xlabel('Samples')

hold off
```

Για τον υπολογισμό των H, K

```
function [ H, K ] = q_learning(A, B, K, Q, rho, Niter, x0)

    % Initialize variables
    x = x0;
    u = -K * x;
    Y = [];
    X = [];

    for n = 1 : Niter

        % Calculate quadratic combinations
        phi = quad_comb(x, u);
        r = x' * Q * x + u' * rho * u;

        % update x_k+1 u_k + 1
        x = A * x + B * u;
        u = -K * x;
```