Esercitazioni di Programmazione Sistemi Robotici

Prof. Corrado Santoro

A.A. 2016-17

1 Esercizio 1

1.1 Testo

Sia dato un sistema dinamico caratterizzato dalla equazione differenziale:

$$\ddot{y} + 0.5\dot{y} + 3y \quad = \quad u$$

dove u è l'ingresso e y è l'uscita;

- Discutere la **stabilità** del sistema;
- Calcolare il **guadagno statico**;
- Implementare il sistema usando un tempo di campionamento di $10\ ms$ e tracciare la risposta al gradino unitario.

1.2 Soluzione

Per **determinare** la stabilità occorre calcolare i modi del sistema, pertanto lo trasformiamo nella forma:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}$$

Si ponga $x_1 = y$, $x_2 = \dot{y}$; otteniamo il seguente sistema:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -3x_1 - 0.5x_2 + u \\ y &= x_1 \end{aligned}$$

da cui:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u$$
$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Le matrici sono:

$$A = \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Determiniamo gli autovalori della matrice A, i quali sono le soluzioni dell'equazione caratteristica:

$$|\lambda I - A| = 0$$

Calcoliamo $[\lambda I - A]$:

$$[\lambda I - A] = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix}$$
$$= \begin{bmatrix} \lambda & -1 \\ 3 & \lambda + 0.5 \end{bmatrix}$$

Calcoliamo il determinante:

$$\lambda(\lambda + 0.5) + 3 = \lambda^2 + 0.5\lambda + 3$$

Gli autovalori sono le soluzioni dell'equazione:

$$\lambda^2 + 0.5\lambda + 3 = 0$$

ossia:

$$\lambda = \frac{-0.5 \pm \sqrt{0.25 - 12}}{2}$$
$$= -\frac{0.5}{2} \pm \frac{\sqrt{12 - 0.25}}{2} i$$

Le radici sono **complesse e coniugate**; la parte reale è **negativa**, pertanto il sistema è **asintoticamente stabile.**

Per il calcolo del **guadagno statico**, occorre determinare la **funzione di trasferimento** $G(s)$, la quale è data da:

$$G(s) = C(sI - A)^{-1}B$$

Determiniamo, in primo luogo, $sI - A$:

$$\begin{aligned}(sI - A) &= \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix} \\ &= \begin{bmatrix} s & -1 \\ 3 & s + 0.5 \end{bmatrix}\end{aligned}$$

quindi $(sI - A)^{-1}$:

$$(sI - A)^{-1} = \frac{\begin{bmatrix} s + 0.5 & 1 \\ -3 & s \end{bmatrix}}{s^2 + 0.5s + 3}$$

Calcoliamo la funzione di trasferimento completa:

$$\begin{aligned}G(s) &= C(sI - A)^{-1}B \\ &= \begin{bmatrix} 1 & 0 \end{bmatrix} \frac{\begin{bmatrix} s + 0.5 & 1 \\ -3 & s \end{bmatrix}}{s^2 + 0.5s + 3} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &= \frac{\begin{bmatrix} s + 0.5 & 1 \end{bmatrix}}{s^2 + 0.5s + 3} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ &= \frac{1}{s^2 + 0.5s + 3}\end{aligned}$$

Il **guadagno statico** è il valore a regime della risposta al gradino $U(s) = \frac{1}{s}$, pertanto, applicando il teorema del valore finale, risulta:

$$\begin{aligned}K &= \lim_{s \to 0} s \frac{1}{s} \frac{1}{s^2 + 0.5s + 3} = \\ &= \lim_{s \to 0} \frac{1}{s^2 + 0.5s + 3} = \\ &= \frac{1}{3}\end{aligned}$$

Per **implementare** il sistema occorre discretizzarlo, pertanto calcoliamo le matrici A' e B' del sistema discretizzato:

$$\begin{aligned}A' &= A\Delta T + I \\ B' &= B\Delta T\end{aligned}$$

$$\begin{aligned}A' &= \begin{bmatrix} 0 & 1 \\ -3 & -0.5 \end{bmatrix} \Delta T + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} 1 & \Delta T \\ -3\Delta T & -0.5\Delta T + 1 \end{bmatrix}\end{aligned}$$

$$B' = \begin{bmatrix} 0 \\ \Delta T \end{bmatrix}$$

Il sistema discretizzato può pertanto essere espresso nel seguente modo:

$$\begin{aligned} x_1(k+1) &= x_1(k) + x_2(k)\Delta T \\ x_2(k+1) &= -3\Delta T x_1(k) + (-0.5\Delta T + 1)x_2(k) + \Delta T u(k) \\ y(k) &= x_1(k) \end{aligned}$$

Per l'implementazione possiamo utilizzare la classe `DynamicSystem` sviluppata durante il corso e creare una nuova classe denominata `G1` nella quale il metodo `evaluate` implementa il sistema desiderato. Il sorgente dell'implementazione risulta pertanto essere:

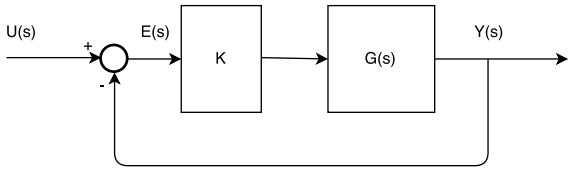
```
#include <fstream>
#include "dynamic_system.h"

class G1 : public DynamicSystem {
public:
    G1(float delta_t) : DynamicSystem(delta_t) { x1 = 0; x2 = 0; };
    float evaluate(float input);
private:
    float x1, x2;
};

float G1::evaluate(float input)
{
    float y, x1_tmp, x2_tmp;

    x1_tmp = x1 + x2 * m_delta_t;
    x2_tmp = -3 * m_delta_t * x1 + (-0.5 * m_delta_t + 1)*x2 + m_delta_t * input;
    y = x1;
    x1 = x1_tmp;
    x2 = x2_tmp;
    return y;
}

int main(int argc, char **argv)
{
    G1 g(0.01);
    // Delta T = 10 ms

    float input = 1; // gradino unitario

    std::ofstream output_file("data.txt");
    // output file dei dati

    // 40 secs di simulazione
    for (int i = 0; i < 40 / 0.01;i++) {
        float output = g.evaluate(input);
        output_file << (i*0.01) << "␣" << output << std::endl;
    }

    output_file.close();
}
```

2 Esercizio 2

2.1 Testo

Dato il sistema con funzione di trasferimento $G(s) = \frac{s+1}{s^2+2s+2}$:

- discutere la **stabilità**;
- implementare il sistema utilizzando un controllore proporzionale in retroazione (vedi figura) e determinare sperimentalmente il valore a regime per $K = 1.5$ e gradino unitario come ingresso. Utilizzare $5\ ms$ come tempo di campionamento.



2.2 Soluzione

Gli autovalori sono le soluzioni dell'equazione:

$$s^2 + 2s + 2 = 0$$

ossia:

$$\begin{aligned} s &= \frac{-4 \pm \sqrt{4-8}}{2} \\ &= -2 \pm \frac{\sqrt{-4}}{2} \\ &= -2 \pm i \end{aligned}$$

Le radici sono **complesse e coniugate**; la parte reale è **negativa**, pertanto il sistema è **asintoticamente stabile.**

Per l'implementazione occorre discretizzare il sistema, pertanto occorre determinare la sua forma nello spazio di stato. Applicando la rappresentazione canonica abbiamo:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -2 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

Per **implementare** il sistema occorre discretizzarlo, pertanto calcoliamo le matrici A' e B' del sistema discretizzato:

$$\begin{aligned} A' &= A\Delta T + I \\ B' &= B\Delta T \end{aligned}$$

$$
\begin{aligned}
A' &= \begin{bmatrix} 0 & 1 \\ -2 & -2 \end{bmatrix} \Delta T + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \\
&= \begin{bmatrix} 1 & \Delta T \\ -2\Delta T & -2\Delta T + 1 \end{bmatrix}
\end{aligned}
$$

$$
B' = \begin{bmatrix} 0 \\ \Delta T \end{bmatrix}
$$

Il sistema discretizzato può pertanto essere espresso nel seguente modo:

$$
\begin{aligned}
x_1(k+1) &= x_1(k) + x_2(k)\Delta T \\
x_2(k+1) &= -2\Delta T x_1(k) + (-2\Delta T + 1)x_2(k) + \Delta T u(k) \\
y(k) &= x_1(k) + x_2(k)
\end{aligned}
$$

Per l'implementazione possiamo utilizzare la classe `DynamicSystem` sviluppata durante il corso e creare una nuova classe denominata `G` nella quale il metodo `evaluate` implementa il sistema desiderato. Il sorgente dell'implementazione del sistema risulta pertanto essere:

```
#include <fstream>
#include "dynamic_system.h"

class G : public DynamicSystem {
public:
    G(float delta_t) : DynamicSystem(delta_t) { x1 = 0; x2 = 0; };
    float evaluate(float input);
private:
    float x1, x2;
};

float G::evaluate(float input)
{
    float y, x1_tmp, x2_tmp;

    x1_tmp = x1 + x2 * m_delta_t;
    x2_tmp = -2 * m_delta_t * x1 + (-2 * m_delta_t + 1)*x2 + m_delta_t * input;
    y = x1 + x2;
    x1 = x1_tmp;
    x2 = x2_tmp;
    return y;
}
```

A questo punto simuliamo, nel `main`, il sistema a ciclo chiuso includendo il controllore:

```
int main(int argc, char **argv)
{
    float delta_t = 0.005;
    // Delta T = 5 ms
    G g(delta_t);

    float input = 1; // gradino unitario
    float y = 0;
```

```
    std::ofstream output_file("data.txt");
    // output file dei dati

    float KP = 1.5;

    // 10 secs di simulazione
    for (int i = 0; i < 10 / delta_t;i++) {
        float err = input - y;
        float output_controller = err * KP;
        y = g.evaluate(output_controller);
        output_file << (i*delta_t) << "␣" << y << std::endl;
    }

    output_file.close();
}
```

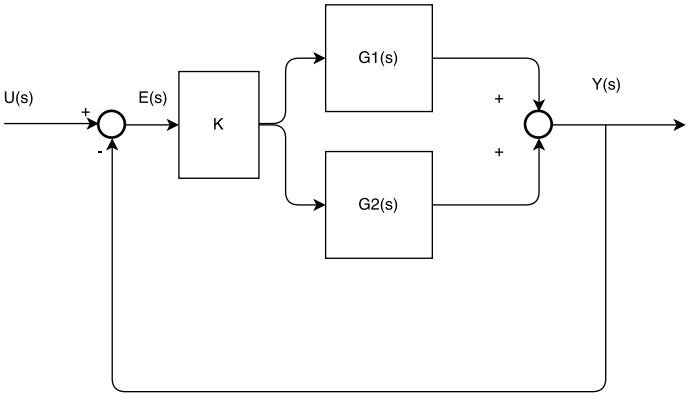
Dalla simulazione si evince che il valore a regime risulta essere $\simeq 0.43$.

3 Esercizio 3

3.1 Testo

Dato il sistema il figura con $G1(s) = \frac{1}{s+5}$ e $G2(s) = \frac{3}{s+2}$, implementare il sistema complessivo (senza determinare la funzione di trasferimento totale) determinando sperimentalmente:

- il tempo di salita per $K = 2.5$;
- il valore a regime per $K = 2.5$ e ingresso pari al gradino unitario. Utilizzare $5\ ms$ come tempo di campionamento.



3.2 Soluzione

Occorre discretizzare $G1(s)$ e $G2(s)$. Essendo due sistemi del primo ordine possiamo passare facilmente al dominio del tempo.

Per il primo sistema abbiamo:

$$\begin{aligned} Y(s) = &= \frac{1}{s+5}U(s) \\ sY(s) + 5Y(s) &= U(s) \\ \dot{y} + 5y &= u \\ \dot{y} &= -5y + u \end{aligned}$$

discretizzando:

$$\begin{aligned} \frac{y(k+1) - y(k)}{\Delta T} &= -5y(k) + u(k) \\ y(k+1) - y(k) &= -5y(k)\Delta T + u(k)\Delta T \\ y(k+1) &= (-5\Delta T + 1)y(k) + u(k)\Delta T \end{aligned}$$

Per il secondo sistema otteniamo:

$$\begin{aligned} Y(s) = &= \frac{3}{s+2}U(s) \\ sY(s) + 2Y(s) &= 3U(s) \\ \dot{y} + 2y &= 3u \\ \dot{y} &= -2y + 3u \end{aligned}$$

discretizzando:

$$\begin{aligned} \frac{y(k+1) - y(k)}{\Delta T} &= -2y(k) + 3u(k) \\ y(k+1) - y(k) &= -2y(k)\Delta T + 3u(k)\Delta T \\ y(k+1) &= (-2\Delta T + 1)y(k) + 3u(k)\Delta T \end{aligned}$$

Implementando il sistema otteniamo:

```
#include <fstream>
#include "dynamic_system.h"

class G1 : public DynamicSystem {
public:
    G1(float delta_t) : DynamicSystem(delta_t) { y = 0; };
    float evaluate(float input);
private:
    float y;
};

float G1::evaluate(float input)
{
    y =(-5 * m_delta_t + 1)*y + m_delta_t * input;
    return y;
}

class G2 : public DynamicSystem {
public:
    G2(float delta_t) : DynamicSystem(delta_t) { y = 0; };
```

```
    float evaluate(float input);
private:
    float y;
};

float G2::evaluate(float input)
{
    y =(-2 * m_delta_t + 1)*y + 3 * m_delta_t * input;
    return y;
}


int main(int argc, char **argv)
{
    float delta_t = 0.005;
    // Delta T = 5 ms
    G1 g1(delta_t);
    G2 g2(delta_t);

    float input = 1; // gradino unitario
    float y = 0;

    std::ofstream output_file("data.txt");
    // output file dei dati

    float KP = 2.5;

    // 10 secs di simulazione
    for (int i = 0; i < 5 / delta_t;i++) {
        float err = input - y;
        float output_controller = err * KP;
        y = g1.evaluate(output_controller) + g2.evaluate(output_controller);
        output_file << (i*delta_t) << "␣" << y << std::endl;
    }

    output_file.close();
}
```

Dal grafico della risposta a gradino otteniamo i valori approssimati del tempo di salita $T_s \simeq 0.5\ s$ e valore a regime $\simeq 0.8$

4 Esercizio 4

4.1 Testo

Considerando il sistema dell'esercizio 3, utilizzare un controllore di tipo proporzionale-integrale (PI) e determinare KP e KI in modo da avere:

- tempo di salita pari almeno a $\simeq 0.25s$;
- tempo di assestamento pari almeno a $\simeq 0.5s$;
- errore a regime nullo;
- assenza di sovraelongazione.

Utilizzare $5\ ms$ come tempo di campionamento.

4.2 Soluzione

Aggiungiamo al sorgente dell'esercizio precedente un controllore PI:

```
class PI : public DynamicSystem {
public:
    PI(float delta_t, float kp, float ki) :
        DynamicSystem(delta_t), m_kp(kp), m_ki(ki), m_integral(0) { };
    float evaluate(float input);
private:
    float m_kp;
    float m_ki;
    float m_integral;
};

float PI::evaluate(float input)
{
    m_integral = m_integral + input * m_delta_t;
    return input * m_kp + m_integral * m_ki;
}
```

Modifichiamo dunque il `main` in modo da includere il controllore PI:

```
int main(int argc, char **argv)
{
    float delta_t = 0.005;
    // Delta T = 5 ms
    G1 g1(delta_t);
    G2 g2(delta_t);
    PI controller(delta_t, 5, 10);

    float input = 1; // gradino unitario
    float y = 0;

    std::ofstream output_file("data.txt");
    // output file dei dati

    // 10 secs di simulazione
    for (int i = 0; i < 5 / delta_t;i++) {
        float err = input - y;
        float output_controller = controller.evaluate(err);
        y = g1.evaluate(output_controller) + g2.evaluate(output_controller);
        output_file << (i*delta_t) << "␣" << y << std::endl;
    }

    output_file.close();
}
```

Effettuando il tuning dei parametri si nota che per $KP = 5$ e $KI = 10$ si ottengono le prestazioni specificate nella consegna.