.

Student Satellite Project
Indian Institute of Technology, Bombay
Powai, Mumbai - 400076, INDIA

Website: www.aero.iitb.ac.in/satlab

# README - LQR Controller

Guidance, Navigation and Controls Subsystem

---

## v_B ()

**Code author:** Ronit
**Created on:** 4/10/2022
**Last modified:** 4/10/2022
**Reviewed by:** NA
**Description:**
Computes the magnetic field vector using a simplified model
**Formula & References:**

$$\mathbf{b} = \frac{\mu_f}{a^3}\begin{bmatrix}\cos(\omega_o t)\sin(i_m) \\ -\cos(i_m) \\ 2\sin(\omega_o t)\sin(t)\end{bmatrix}$$

Here $\mu_f$ is the fields dipole strength, $a$ is the length of semi major axis of orbit, $\omega_o$ is the orbit angular velocity, $i_m$ is the orbit inclination
refer for more details.
**Input parameters:**

1. **time** : (float) - Time at which magnetic field is computed. *seconds*

**Output:**
Returns the value of magnetic field at that time instant as a numpy array.

## nonlinear_dynamics ()

**Code author:** Ronit
**Created on:** 15/07/2022
**Last modified:** 15/07/2022
**Reviewed by:** NA
**Description:**
Computes the derivative of state using nonlinear attitude dynamics. It has been used for conducting the simulations.
**Formula & References:**

$$\dot{q} = -\frac{1}{2}\omega \times q + \frac{1}{2}q_0\omega$$

$$\dot{\omega} = \mathbf{I}^{-1}(\mathbf{I}\omega \times \omega + u)$$

Here $q$ represents vector part of quaternion, $q_0$ represents scalar part of quaternion, $\omega$ represents angular velocity in body frame, **I** represents inertia matrix of satellite in body frame.
**Input parameters:**

1. **time** : (float) - Time at which derivative is computed. *seconds*

2. **state** : (numpy array) - State at which derivative is computed. *SI units for all*

**Output:**
It returns the $\dot{x}$ value mentioned earlier as a numpy array.

## initialize_gain ()

**Code author:** Ronit
**Created on:** 15/07/2022
**Last modified:** 4/10/2022
**Reviwed by:** NA
**Description:**
Computes the gain matrix for finding control.
**Formula & References:** The cost function that will be minimized by the controller is given by

$$\frac{1}{2}\int_0^{\infty}[x^TQx + u^TRu]dt$$

Gain matrix is given by

$$K = -R^{-1}B^TF$$

here $B$ is the defined in the linear_dynamics description, $Q = \begin{bmatrix} Q_1 & \mathbf{O_3} \\ \mathbf{O_3} & Q_2 \end{bmatrix}$, $F = \begin{bmatrix} F_{11} & F_{12} \\ F_{12}^T & F_{22} \end{bmatrix}$

$$F_{11} = \mathbf{I}R^{1/2}\left(Q_1 + \frac{1}{2}(\mathbf{I}R^{1/2}Q_2^{1/2} + Q_2^{1/2}R^{1/2}\mathbf{I})\right)^{1/2}$$

$$F_{12}^T = \mathbf{I}R^{1/2}Q_2^{1/2}$$

$$F_{22}^T = 2Q_2^{1/2}(Q_1 + \mathbf{I}R^{1/2}Q_2^{1/2})^{1/2}$$

For further details here is the link to the paper being referred.
**Input parameters:**
This function has no input parameters. However it uses certain constants relevant to controller that have been defined in the constants file.
**Output:**
It returns the $3 \times 6$ gain matrix as a numpy array.

## control_law ()

**Code author:** Ronit
**Created on:** 15/07/2022
**Last modified:** 4/10/2022
**Reviewed by:** NA
**Description:**
Computes the magnetic moment required for attitude control

**Formula & References:**

$$\tau = Kx$$
$$m = \frac{\mathbf{b} \times \tau}{\|\mathbf{b}\|^2}$$

here $K$ is the gain matrix described in the previous function, $\mathbf{b}$ is the magnetic field at that time.
**Input parameters:**

1. **state** : (numpy array) - State at which control is computed. *SI units for all*

2. **time**: (float) - Time at which control needs to be computed

**Output:**
It returns the magnetic moment that needs to be applied as a numpy array.

## rk4method()

**Code author:** Ronit Chitre
**Created on:** 30/3/2022
**Last modified:** 31/3/2022
**Reviewed by:** Not yet reviewed
**Description:**
This is a numerical ode solver and uses rk4 method as its solving algorithm.
**Formula & References:**
For an ode $\frac{dx}{dt} = f(t, x)$ define $h$ as the length between the time values at which solution is desired. Here $N$ is the number of points over which solution is desired. $w_i$ is the value of $x(t)$ at the 'i'th point. $t_i = t_0 + hi$.

$$w_{i+1} = w_i + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$$

Here $k_1 = hf(t_i, w_i)$, $k_2 = hf(t_i + \frac{h}{2}, w_i + \frac{k_1}{2})$, $k_3 = hf(t_i + \frac{h}{2}, w_i + \frac{k_2}{2})$, $k_4 = hf(t_{i+1}, w_i + k_3)$
Here the error is of the order $O(h^4)$

**Input parameters:**

1. **function of ode** : (function) - This is the function $f(t, x)$ that defines the ode. *value per time*

2. **initial conditions** : (numpy array) - This array will define the initial conditions or $w_0$. *value*

3. **time values** : (numpy array) - This array will contain all the time values on which value of $x(t)$ is to be found. *time*

**Output:**
If $x$ is an $\mathbf{R}^{n\times 1}$ vector and $m$ time values were given it will return an $(m, n)$ matrix where each row will contain the value of $x$ at that time instant.