Motion Planning for Mobile Robots -- Assignment 06 Bernstein Basis for Constraints

NOTE Please open this in **VSCode** with **MATLAB plugin**

Solution guide for **Assignment 06, Bernstein Basis for Constraints**.

Introduction

Welcome to **Solution Guide for Assignment 06**! Here I will guide you through the **MATLAB** implementations of both

- **Flight Corridor Quadrotor Navigation with Bernstein Basis**

Q & A

Please send e-mail to alexgecontrol@qq.com with title **Motion-Planning-for-Mobile-Robots--Assignment-05--Q&A-[XXXX]**. I will respond to your questions at my earliest convenience.

NOTE

- I will **NOT** help you debug your code and will only give you suggestions on how should you do it on your own.

QP Solver for Bezier Curve Based Flight Corridor Navigation

Overview

The workflow of **numeric solver** can be summed up as follows:

- Build **objective matrix**
  - It is defined by **minimum-snap** or **minimum-jerk** on **monomial polynomial**
  - Then map the representation to **Bernstein basis**, which can be achieved using **transformation matrix**
- Build **equality constraint matrix**, which is defined by:
  - **Boundary conditions**, **start** / **end** ego states
  - **Continuity on transition waypoint between two consecutive trajectory segments**
- Build **inequality constraint matrix**, which is defined by **the series of bounding boxes** that define the flight corridor:
- Solve the QP problem for the optimal coefficients of **Bernstein polynomial**.
- Map the optimal result back to **monomia polynomial** for easy trajectory generation.

The workflow can be implemented in MATLAB as follows:

```
% ########################################################################################
% numeric solver implementation
% ########################################################################################
function poly_coef = MinimumSnapCorridorBezierSolver(axis, waypoints, corridor, ts, K, t_order, v_max, a_max)
    % TODO -- extract boundary conditions:

    % TODO -- build objective matrix:

    % TODO -- build constraint matrix, equality:

    % TODO -- build constraint matrix, inequality:

    % TODO -- solve optimal coeffs, Bernstein polynomial:

    % TODO -- map optimal Bernstein poly coeffs to optimal monomial coeffs:
end
```

Objective Matrix

Part 1, Optimization Target

The actual objective matrix **Q** is defined by **the L2-norm of the optimization target**:

- **Minimum Snap**, which is equivalent to **t_order = 4** in the implementation below
- **Minimum Jerk**, which is equivalent to **t_order = 3** in the implementation below

```
function Q = getQ(K, t_order, ts)
    % num. of polynomial coeffs:
    N = 2*t_order;

    % ###############################################
    % pre-compute constants used in Q construction
    % ###############################################
    % factorial from derivative
    Q_k = zeros(N - t_order);
    Q_v = zeros(N - t_order);
    for n = t_order:(N - 1)
        Q_k(n - t_order + 1) = n;
        Q_v(n - t_order + 1) = factorial(n) / factorial(n - t_order);
    end
    Q_factorial = containers.Map(Q_k, Q_v);

    % time power:
    ts_power = ts .^ t_order;

    % ###############################################
    % populate Q
    % ###############################################
    Q_i = [];
    Q_j = [];
    Q_v = [];

    index = 1;

    for k = 1:K
        for m = t_order:(N - 1)
            for n = t_order:(N - 1)
                % TODO - define elements in Q:

                index = index + 1;
            end
        end
    end

    Q = sparse(Q_i, Q_j, Q_v);
end
```

Part 2, Transformation Matrix, Bernstein to Monomial

The transformation matrix **M** is defined by **the L2-norm of the optimization target**:

```
function M = getM(K, t_order)
    % num. of polynomial coeffs:
    N = 2*t_order;

    % num. of non-zero M elements:
    E = K*N*(N + 1)/2;

    % ###############################################
    % build M
    % ###############################################
    M_i = zeros(E, 1);
    M_j = zeros(E, 1);
    M_v = zeros(E, 1);
    b = zeros(N, 1);

    index = 1;

    for n = 1: N
        % TODO -- binomial coefficient from bernstein polynomial:
    end

    for n = 1: N
        for m = 1:n
            for k = 1:K
                % TODO -- calculate binomial coefficient from t^(i)*(1-t)^(n-i)

                index = index + 1;
            end
        end
    end

    % done:
    M = sparse(M_i, M_j, M_v);
end
```

Part 3, Done!

Finally, create **objective matrix** for **Bernstein polynomial** as follows:

```
function [P, M] = getPM(K, t_order, ts)
    % num. of polynomial coeffs:
    N = 2*t_order;

    % ###############################################
    % TODO -- get Q, objective matrix for minimum snap
    % ###############################################

    % ###############################################
    % TODO -- get M, mapping from bernstein to monomial
    % ###############################################

    % ###############################################
    % TODO -- build objective matrix for control points
    % ###############################################

end
```

Equality Constraint Matrix

The equality constraint matrix is defined as follows:

- **Boundary Conditions**, which are defined by **start** and **end** states of target trajectory
- **Continuity on transition waypoint between two consecutive trajectory segments**, which requires the planned trajectory should be **t_order - 1** order continuous at each transition waypoint between two segments.

```
function [Aeq, beq] = getAbeq(K, t_order, ts, start_cond, end_cond)
    % num. of polynomial coeffs:
    N = 2*t_order;

    % num. of equality constraints:
    D = (K + 1)*t_order;

    % TODO -- num. of non-zero Aieq elements:
    E = 0;

    % ###############################################
    % pre-compute constants used in A construction
    % ###############################################
    % factorial from derivative
    Aeq_factorial_k = [];
    Aeq_factorial_v = [];

    index = 1;

    for c = 1:t_order
        Aeq_factorial_k(index) = c;
        Aeq_factorial_v(index) = factorial(N - 1) / factorial(N - c);

        index = index + 1;
    end
    Aeq_factorial = containers.Map(Aeq_factorial_k, Aeq_factorial_v);

    % ###############################################
    % build constraint matrix
    % ###############################################
    index = 1;

    Aeq_i = zeros(E, 1);
    Aeq_j = zeros(E, 1);
    Aeq_v = zeros(E, 1);

    beq = zeros(D, 1);

    c_index = 1;

    % ###############################################
    % start & end conditions
    % ###############################################
    for c = 1:t_order
        for i = 1:c
            % TODO -- set start condition:

            % TODO -- set end condition:

            index = index + 2;
        end

        % start condition:
        beq(c_index) = start_cond(c);
        % end condition:
        beq(c_index + 1) = end_cond(c);

        % move to next constraint:
        c_index = c_index + 2;
    end

    % ###############################################
    % transition waypoint continuity constraints
    % ###############################################
    for k = 1:(K - 1)
```

```
        for c = 1:t_order
            for i = 1:c
                % TODO -- end state of current trajectory segment:

                % TODO -- should equal to start state of next trajectory segment:

                index = index + 2;
            end

            % move to next constraint:
            c_index = c_index + 1;
        end
    end

    % done:
    Aeq = sparse(Aeq_i, Aeq_j, Aeq_v);
end
```

Implementation Nodes

- According to the original paper, if your objective function is the L2-norm of **t_order** trajectory derivative, then the trajectory should be **t_order - 1** order continuous at each intermediate waypoint.

Inequality Constraint Matrix

The inequality constraint matrix defined by **the series of bounding boxes** that define the flight corridor

```
function [Aieq, bieq] = getAbieq(K, t_order, ts, corridor_range, v_max, a_max)
    % num. of polynomial coeffs:
    N = 2*t_order;

    % num. of inequality constraints:
    D = K*(2*N - t_order + 1)*t_order;

    % TODO -- num. of non-zero Aieq elements:
    E = 0;

    % ###############################################
    % pre-compute constants used in Aieq construction
    % ###############################################
    % factorial from derivative
    Aieq_factorial_k = [];
    Aieq_factorial_v = [];

    index = 1;

    for c = 1:t_order
        Aieq_factorial_k(index) = c;
        Aieq_factorial_v(index) = factorial(N - 1) / factorial(N - c);

        index = index + 1;
    end
    Aieq_factorial = containers.Map(Aieq_factorial_k, Aieq_factorial_v);

    % ###############################################
    % build constraint matrix
    % ###############################################
    index = 1;

    Aieq_i = zeros(E, 1);
    Aieq_j = zeros(E, 1);
    Aieq_v = zeros(E, 1);

    bieq = zeros(D, 1);

    c_index = 1;

    for c = 1:t_order
        for k = 1:K
            for n = c:N
                % TODO -- set derivative:
                for i = 1:c
                    index = index + 2;
                end

                % TODO -- set limit:

                % move to next constraint:
                c_index = c_index + 2;
            end
        end
    end

    % done:
    Aieq = sparse(Aieq_i, Aieq_j, Aieq_v);
end
```

Wrap-Up

Happy Learning & Happy Coding!

Yao

- GitHub
- LinkedIn