HW3 约束优化

Task I 严格凸的等式约束 QP 的 KKT 推导和求解
见第 3 章第 9 节视频最后

Task II 低维度严格凸 QP 线性时间复杂度算法的补全（投影子问题的构建，Householder 反射）
在 sdqp.hpp 文件中，按照提示补全以下代码。

```
        for (int i = 0; i != m; i = next[i])
        {
            const double *plane_i = halves + (d + 1) * i;

            if (dot<d>(opt, plane_i) + plane_i[d] > (d + 1) * eps)
            {
                const double s = sqr_norm<d>(plane_i);

                if (s < (d + 1) * eps * eps)
                {
                    return INFEASIBLE;
                }

                mul<d>(plane_i, -plane_i[d] / s, new_origin);

                if (i == 0)
                {
                    continue;
                }

                ////////////////////////////// HOMEWORK START ////////////////////////////
                //
                // MISSION TO BE ACCOMPLISHED:
                //
                // now we know the best solution "opt" violates the i-th halfspace. Therefore,
                // please project all previous i halfspaces (from the 0-th to the (i-1)-th one)
                // onto the boundary of the i-th halfspace, then store all projected halfspaces
                // in the double-type-c-array "new_halves".
                // If you successfully complete the mission, the sdqp_example should prints
                //     optimal sol: 4.11111 9.15556 4.50022
                //     optimal obj: 201.14
                //     cons precision: *.*******e-16
                // This means you obtained the correct exact solution (precision near DBL_EPSILON)
                //
                // VARIABLES YOU NEED TO ACCESS:
                //
                // opt is a d-dimensional double-type-c-array
                // opt contains an optimal solution that meets all linear constraints from the 0-th
                // to the (i-1)-th one, but it is known to violate the i-th halfspace here
                //
                // new_origin is also a d-dimensional double-type-c-array
                // new_origin contains the minimum norm point on the boundary of the i-th plane
                //
                // you can read all previous halfspaces via the iteration below
                //
                //     for (int j = 0; j != i; j = next[j])
                //     {
                //         const double *halfspace = halves + (d + 1) * j;
                //         // thus the j-th halfspace is the inequality below
                //         // halfspace[0] * x1 + halfspace[1] * x2 + ... + halfspace[d-1] * xd + halfspace[d] <= 0
                //     }
                //
                // you can write or store all your projected halfspaces via the iteration below
                //
                //     for (int j = 0; j != i; j = next[j])
                //     {
                //         double *proj_halfspace = new_halves + d * j;
                //         // thus the j-th projected halfspace is the inequality below
                //         // proj_halfspace[0] * y1 + proj_halfspace[1] * y2 + ... + proj_halfspace[d-2] * y(d-1) + proj_halfspace[d-1] <= 0
                //         // y1 to y(d-1) is the new coordinate constructed on the boundary of the i-th halfspace
                //     }
                //
```

Task III 用 PHR-ALM 方法求解 NMPC（已给出 penalty 方法求解的范例，修改求解器即可）
Step1: Run the solveNMPC example in mpc-car-tutorial-master.zip;
Step2: Use PHR-ALM to replace the Penalty Method in solveNMPC;
Step3: Compare the constrain violation in Penalty Method and PHR-ALM.