Write C++ code to implement model-predictive control for a wind farm.

Model-predictive control (MPC) is a control strategy that uses a mathematical model of the system to predict its behavior over a certain time horizon and then optimizes the control inputs over that horizon to achieve a certain objective. In the case of a wind farm, the objective might be to maximize the energy output while minimizing the loads on the turbines.

Here is an example C++ code that implements MPC for a wind farm:

scss Copy code

```
#include <iostream>
#include <Eigen/Dense>
#include <Eigen/Sparse>
#include <qpOASES.hpp>

using namespace Eigen;
using namespace std;
using namespace qpOASES;

int main() {
    // Define system matrices
    MatrixXd A, B, C;
    A << 0.9, 0, 0, 0.9;
    B << 1, 0, 0, 1;
    C << 1, 0, 0, 1;

    // Define control horizon and prediction horizon
    const int Nc = 10;
    const int Np = 50;

    // Define initial state and control inputs
    VectorXd x0(4);
    x0 << 0, 0, 0, 0;
    VectorXd u(Nc);
    u.setZero();

    // Define constraints
    MatrixXd Umax(Nc, 2);
    Umax << 1, 1, 1, 1;
    MatrixXd Umin(Nc, 2);
    Umin << -1, -1, -1, -1;
    MatrixXd Xmax(Np, 4);
    Xmax.setConstant(1e9);
    MatrixXd Xmin(Np, 4);
    Xmin.setConstant(-1e9);

    // Define objective function
    SparseMatrix<double> Q(Nc+1, Nc+1);
    Q.insert(0,0) = 1;
    SparseMatrix<double> R(Nc, Nc);
    R.insert(0,0) = 1;

    // Define MPC solver
    QProblem problem(Nc+1, Nc);
    Options options;
    options.printLevel = PL_LOW;
    problem.setOptions(options);

    // Loop over time
    for (int i = 0; i < 100; i++) {
        // Predict state over prediction horizon
        MatrixXd Xp(Np, 4);
        Xp.row(0) = x0.transpose();
        for (int j = 1; j < Np; j++) {
            Xp.row(j) = A*Xp.row(j-1) + B*u(j-1);
        }

        // Construct quadratic program
        VectorXd g(Nc+1);
        g.setZero();
        g(0) = C*Xp.transpose()*C.transpose()*x0;
        MatrixXd E(Nc+1, Np);
        E << C*A, C*A*A, C*A*A*A, C*A*A*A*A, C*A*A*A*A*A,
             MatrixXd::Zero(2*Nc, Np);
        MatrixXd F(Nc+1, Nc);
        F << C*B, C*A*B, C*A*A*B, C*A*A*A*B, C*A*A*A*A*B;
        MatrixXd G(Nc, Nc+1);
        G.setZero();
        G.block(0,1,Nc,2*Nc) = MatrixXd::Identity(Nc, 2*Nc);
        VectorXd lb(Nc+1);
        lb << -1e9, Umin.col(0);
        VectorXd ub(Nc+1);
        ub <<
```

continue

c Copy code

```
    1e9, Umax.col(0);

    // Solve quadratic program
    problem.init(Q, g, E, Xmin.transpose(), Xmax.transpose(), F, lb, ub, G)
    VectorXd du(Nc);
    problem.getPrimalSolution(du.data());

    // Update control inputs
    u += du;

    // Apply control inputs
    x0 = A*x0 + B*u(0);

    // Print results
    cout << "Time: " << i << endl;
    cout << "Control inputs: " << u.transpose() << endl;
    cout << "State: " << x0.transpose() << endl;
    cout << endl;
}

return 0;
```

}

css Copy code

```
Note that this is a simplified example, and the actual implementation of MP
```