```cpp
#include <stdexcept>
#include "Matrix.h"

Matrix::Matrix(const Matrix &m) : N(m.rows()), M(m.columns()), matrix(m.matrix) {
    std::cerr << "Copiando matriz de " << this->rows() << "x" << this->columns() << std::endl;
}

Matrix::Matrix(int N, int M)
        : N(N), M(M), matrix(N, std::vector<double>(M, 0.0)) {
    if (this->rows() < 0 || this->columns() < 0) {
        throw new std::out_of_range("Invalid matrix dimension");
    }

    std::cerr << "Creando matriz de " << this->rows() << "x" << this->columns() << std::endl;

    if (this->matrix.size() > 0) {
        std::cerr << "Dimensiones del vector de salida: " << this->matrix.size() << "x" << this-
>matrix[0].size() << std::endl;
    }
}

Matrix &Matrix::operator=(const Matrix &m) {
    if (*this != m) {
        // Poner información de la representación interna
        this->N = m.rows();
        this->M = m.columns();

        // Crear matriz nueva
        this->matrix = m.matrix;
    }

    return *this;
}

bool Matrix::operator==(const Matrix &m) const {
    if (this->rows() != m.rows() || this->columns() != m.columns()) {
        return false;
    } else {
        for (int i = 0; i < this->rows(); i++) {
            for (int j = 0; j < this->columns(); j++) {
                if ((*this)(i, j) != m(i, j)) {
                    return false;
                }
            }
        }

        return true;
    }
}

bool Matrix::operator!=(const Matrix &m) const {
    return !(*this == m);
}

Matrix &Matrix::operator+=(const Matrix &m) {
    if (this->rows() == m.rows() && this->columns() == m.columns()) {
        for (int i = 0; i < m.rows(); ++i) {
            for (int j = 0; j < m.columns(); ++j) {
                this->matrix[i][j] += m.matrix[i][j];
            }
        }
    } else {
        throw new std::out_of_range("Different dimensions for matrix sum");
    }

    return *this;
}

Matrix &Matrix::operator*=(const double &c) {
    for (int i = 0; i < this->rows(); ++i) {
        for (int j = 0; j < this->columns(); ++j) {
            this->matrix[i][j] *= c;
        }
    }

    return *this;
}

```

```cpp
std::ostream &operator<<(std::ostream &os, const Matrix &m) {
    for (int i = 0; i < m.rows(); ++i) {
        for (int j = 0; j < m.columns(); ++j) {
            os << (double)m(i, j) << " ";
        }

        os << std::endl;
    }

    os << std::endl;

    return os;
}

std::istream &operator>>(std::istream &is, Matrix &m) {
    for (int i = 0; i < m.rows(); ++i) {
        for (int j = 0; j < m.columns(); ++j) {
            is >> m(i, j);
        }
    }

    return is;
}

Matrix operator+(const Matrix &m, const Matrix &n) {
    Matrix output(m);
    output += n;
    return output;
}

Matrix operator*(const Matrix &m, const double &c) {
    Matrix output(m);
    output *= c;
    return output;
}

Matrix operator*(const Matrix &m, const Matrix &n) {
    if (m.columns() == n.rows()) {
        Matrix output(m.rows(), n.columns());

        for (int i = 0; i < output.columns(); ++i) {
            for (int j = 0; j < output.columns(); ++j) {
                for (int k = 0; k < m.columns(); ++k) {
                    output(i, j) += m(i, k) * n(k, j);
                }
            }
        }

        return output;
    } else {
        throw new std::out_of_range("Matrix product between incompatible matrices.");
    }
}
```