```cpp
//      Sec. 5.1 of lecture
//      2D arrays dynamic
//      Multiplication Matrix*Vector  f := A * u
//              with A as 2D array   or  B as 1D array
//  g++ -std=c++11 -Wall -Wextra -pedantic bsp514_b.cpp
#include <iostream>
#include <vector>
#include <cassert>                              // assert
using namespace std;

int main()
{
    const int NROW = 4, MCOL = 3;
          vector<double> f(NROW);
    const vector<double> u({1, 2, 3});        // initialize u
    assert( MCOL == static_cast<int>(u.size()) );

//-----------------------------------------------------------------
//      matrix as 2D-Array
    const vector<vector<double>> a({ {4, -1, -0.5}, { -1, 4, -1}, { -0.5, -1, 4}, {3, 0, -1} });
    assert( NROW ==  static_cast<int>(a.size()) );        // check number of rows
    assert( MCOL ==  static_cast<int>(a.back().size()) ); // check number of columns in last row

//              matrix times vector : f := A * u
    for (int i = 0; i < NROW; ++i) {
        f[i] = 0.0;                                          // initialize f[i]
        for (int j = 0; j < MCOL; ++j) {
            f[i] = f[i] + a[i][j] * u[j];
        }
        cout << a[i].data() << endl;           // Address of a[i][0]
    }
//                      output result
    cout << endl;
    for (int i = 0; i < NROW; ++i) {
        cout << "  " << f[i];
    }
    cout << endl << endl;

//-----------------------------------------------------------------
//      matrix as 1D array with index calculation
    const vector<double> b({4, -1, -0.5, -1, 4, -1, -0.5, -1, 4, 3, 0, -1 });
    assert( NROW * MCOL == static_cast<int>(b.size()) );

//              matrix times vector : f := B * u
    for (int i = 0; i < NROW; ++i) {
        f[i] = 0.0;                                          // initialize f[i]
        for (int j = 0; j < MCOL; ++j) {
            f[i] = f[i] + b[i * MCOL + j] * u[j];
        }
    }
//                      output result
    cout << endl;
    for (int i = 0; i < NROW; ++i) {
        cout << "  " << f[i];
    }
    cout << endl << endl;

    return 0;
}
```