DOI 10.4010/2016.1860
ISSN 2321 3361 © 2016 IJESC

Research Article Volume 6 Issue No. 6

C++ Codes for the Implementation of Determining and Controllability Matrices, Cardinality and Controllability Rank Conditions for A Class of Double - Delay Control Systems

Ukwu Chukwunenye[1], Obiorah Mmachi[2]
Department of Mathematics[1], Department of Computer Science[2]
University of Jos, P.M.B. 2084, Jos, Plateau State, Nigeria
cukwu@hotmail.com[1], obiorahm@gmail.com[2]

Abstract:
As dictated by practical exigencies, this research article designed and developed extensive C++ codes for the implementation of determining matrices, controllability matrices, cardinality and controllability rank conditions, for a class of double - delay control systems, thereby averting the prohibitive manual computations associated with such mathematical objects. With these results, the interrogation of the controllability disposition of this class of functional differential control systems can be accomplished with astounding swiftness, in turn, providing the much desired implementation paradigm shift.

Keywords: C++ codes, Cardinality, Determining matrices, Euclidean controllability, Computational feasibility, Electronic implementation, Permutations, Double-delay control systems.

1. INTRODUCTION

Determining matrices derive their importance from the fact that they constitute the optimal platform for the determination of Euclidean controllability and compactness of cores of Euclidean targets [1], [2], [3] and [4]. In sharp contrast to determining matrices, the use of indices of control systems on the one hand and the application of controllability Grammians on the other, for the investigation of the Euclidean controllability of systems can at the very best be quite computationally challenging and at the worst, mathematically intractable [5]. Thus, determining matrices are beautiful brides for the interrogation of the controllability disposition of delay control systems [4]. However, up-to-date review of literature on this subject reveals that until [6], there had been no result on the structure of determining matrices for double-delay systems. This could be attributed to the severe difficulty in identifying recognizable mathematical patterns needed for inductive proof. Furthermore, thus far, there has been no reported investigation on the computational feasibility of determining matrices in general. By an appeal to the multinomial distribution, the cardinality of the determining matrix, $Q_k(jh)$, in [7] can be seen to be

$$\sum_{r=0}^{\left[\left[\frac{j}{2}\right]\right]} \frac{k!}{(r+k-j)!(j-2r)!r!}, \text{ for } 0 \le j \le k \ne 0,\ h>0,$$

where $[[.]]$ denotes the greatest integer function (the floor function). In particular the cardinality leaps from 25,653 for $j=k=11$, to 1,105,350,729, for $j=k=21$. How in the world could one manually handle over one billion additions, even for a moderately-sized practical problem! Clearly the curse of dimensionality cannot be mitigated in manual computations of determining matrices; needless to say, they are inherently computationally intractable.

Thus, this paper makes a positive contribution to knowledge by overcoming the manual computational complexity through the automation of the determining matrices, controllability matrices, cardinality and controllability rank conditions on the C++ platform. It is noteworthy that due to the limitations in the functionality of C++, it was initially impossible to obtain the cardinalities of $Q_k(jh)$ for $j+k \ge 23$. C++ has overflow problems for $i!$ when $i \ge 23$. The above limitations were eventually overcome by the incorporation of a C++ Boost Library into Visual Studio 2012, [8]. This guaranteed the computational feasibility of $Q_k(jh)$ for arbitrary j and k, and hence the suitability and appropriateness of $Q_k(jh)$ even for large-scale applications. Another challenge encountered was the problem associated with generating extensive and well-tested code for the implementation of ranks of controllability matrices. This was eventually resolved through the inclusion of Armadillo Library [9]. This Library has well-tested and robust functionality for obtaining the ranks of matrices of any structure. Thus, the major impediments in this area of acute research need have been eliminated.

2. MATERIALS AND METHODS

2.1 Identification of Work-Based Double-Delay Autonomous Control System

We consider the double-delay autonomous control system

$$\dot{x}(t) = A_0 x(t) + A_1 x(t-h) + A_2 x(t-2h) + B\,u(t);\, t \ge 0$$
$$x(t) = \phi(t),\, t \in [-2h, 0],\, h>0$$

where A_0, A_1, A_2 are $n \times n$ constant matrices with real entries, B is an $n \times m$ constant matrix with real entries. The initial function ϕ is in $C\left(\left[-2h,0\right],\mathbf{R}^n\right)$, the space of continuous functions from $[-2h,0]$ into the real n-dimension Euclidean space, $\mathbf{R}^n$ with norm defined by $\|\phi\| = \sup_{t\in[-2h,0]} |\phi(t)|$, (the sup norm). The control u is in the space $L_\infty\left(\left[0,t_1\right],\mathbf{R}^n\right)$, the space of essentially bounded measurable functions taking $\left[0,t_1\right]$ into $\mathbf{R}^n$ with norm $\|\phi\| = ess\sup_{t\in[0,t_1]} |u(t)|$.

Any control $u \in L_\infty\left(\left[0,t_1\right],\mathbf{R}^n\right)$ will be referred to as an admissible control. For full discussion on the spaces C^{p-1} and $L_p(\text{or } L^p)$, $p\in\{1,2,\ldots,\infty\}$, see [10], [11] and [12].

Let r_0, r_1, r_2 be nonnegative integers and let $P_{0(r_0),1(r_1),2(r_2)}$ denote the set of all permutations of $\underbrace{0,0,\ldots0}_{r_0\text{ times}}\ \underbrace{1,1,\ldots1}_{r_1\text{ times}}\ \underbrace{2,2,\ldots2}_{r_2\text{ times}}$: the permutations of the objects $0,1,2$ in which i appears r_i times, $i\in\{0,1,2\}$.

The results to be automated are the following:

2.2 Theorem 1: The structure of $Q_k(jh)$, [7]

$Q_0(0) = I_n$, the identity matrix of order n.

For $0\le j\le k$, j, k integers, $k\ne 0$,

$$Q_k(jh) = \sum_{r=0}^{\left[\left[\frac{j}{2}\right]\right]} \sum_{(v_1,\cdots,v_k)\in P_{0(r+k-j),1(j-2r),2(r)}} A_{v_1}\cdots A_{v_k}$$

For $j\ge k\ge 1$, j, k integers,

$$Q_k(jh) = \begin{cases} \displaystyle\sum_{r=0}^{\left[\left[\frac{2k-j}{2}\right]\right]} \sum_{(v_1,\cdots,v_k)\in P_{0(r),1(2k-j-2r),2(r+j-k)}} A_{v_1}\cdots A_{v_k}, & 1\le j\le 2k \\ 0, & j\ge 2k+1 \end{cases}$$

Table 1: Computing Complexity Table For $Q_k(jh)$

	Number of nonzero terms = Number of nonzero products = Cardinality of $Q_k(jh)$.	Number of additions	Size of permutation = sum of powers of the A_is
$Q_k(jh)$, $0\le j\le k\ne 0$	$\displaystyle\sum_{r=0}^{\left[\left[\frac{j}{2}\right]\right]} \frac{k!}{(r+k-j)!(j-2r)!r!}$	$\left[\left[\frac{j}{2}\right]\right]$	k
$Q_k(jh)$, $1\le k\le j\le 2k$	$\displaystyle\sum_{r=0}^{\left[\left[\frac{2k-j}{2}\right]\right]} \frac{k!}{r!(2k-j-2r)!(r+j-k)!}$	$\left[\left[\frac{2k-j}{2}\right]\right]$	k

The complexity table for $Q_k(jh)$, $k\ge j$ cannot be obtained by swapping j and k, in $Q_k(jh)$, $j\ge k$. $Q_k([k+p]h)$ and $Q_{k+p}(kh)$ do not have the same complexity, for any positive integer, p.

2.3 Theorem 2: Controllability Matrix for Controllability investigation via rank conditions

The controllability matrix is given by

$\hat{Q}_n(t_1) = \left[Q_0(s)B, Q_1(s)B,\cdots,Q_{n-1}(s)B : s\in\{0,h,\cdots,\text{n}\right.$

an n by $mn\left(1+\left[\left[\left[\min\left\{\frac{t_1}{h}, n-1\right\}\right]\right]\right]\right)$ concatenated matrix of $n\left(1+\left[\left[\left[\min\left\{\frac{t_1}{h}, n-1\right\}\right]\right]\right]\right)$ matrix product objects, each of dimension n by m. Here, $[[[.]]]$ denotes the least integer function (the ceiling function).

2.4 Rank Condition for Controllability

System (1) is Euclidean controllable on the interval $[0,\ t_1]$ if and only if $\text{rank}\left[\hat{Q}_n(t_1)\right] = n.$

2.5 Visual C^{++} Implementation Codes

2.5.1 Source Code for the Header File Q.h (C:\Users\ENGR\Desktop\ProjectQarma\Q.h)

```
#ifndef Q_1
#define Q_1
#include "math.h"
#include "QException.h"
#include "algorithm"
#include <armadillo>
#include <string>
#include <fstream>
#include <boost\multiprecision\cpp_int.hpp>

using boost::multiprecision::cpp_int;
using namespace arma;
using namespace std;
class Q
{
    private:
        int n,m ,r, k, j,summation;
        int beta, h;
            string a1mat, a2mat, a3mat, bmat, outputmat;
        mat *A[3] ;
        mat *B;
        mat *C;
        mat *SS;

        //vector<vector<vector<int> > >J;
            mat J;
            //mat J replaces vector<vector<vector<int> > >J; and we use the join function
        //Matrix SS;
        int A0appears;
        int A1appears;
        int A2appears;
            ofstream outputfile;

    public:
        Q(int n1, int m1, int beta1 = 1, string matA1="A1.mat", string matA2="A2.mat", string matA3="A3.mat", string
outputmat1="fstream.dat", string matB="B.mat", int h1 = 1, int k1=0, int j1=0);
          void timesToSum(int j1, int k1);
          void Matrixappears();
          void Summation(int j1, int k1);
          void Summation2(int j1, int k1);
          mat* SumofPermutations(int A0a = 0, int A1a = 0,int A2a = 0);
          void PopulateQ(int a,int b, int c);
          void printJ();
                          cpp_int factorial(int n);
                          void Cardinality(int j2, int k2);
                         void Cardinality0(int j3, int k3);
        ~Q();
};
#endif
```

2.5.2 Source Code for the Implementation File Q.cpp (C:\Users\ENGR\Desktop\ProjectQarma\ Q.cpp)

```
#include "Q.h"
#include <cmath>
/*Constructor for Q*/
        Q::Q(int n1, int m1, int beta1, string matA1, string matA2, string matA3, string outputmat1, string matB, int h1, int k1, int
j1):n(n1),m(m1),beta(beta1),a1mat(matA1),a2mat(matA2),a3mat(matA3),outputmat(outputmat1),bmat(matB),h(h1),k(k1),j(j1){}
```

```
/*times to sum chooses the range for r; r ranges from zero to summation based on the conditions below*/
void Q::timesToSum(int j1, int k1)
{
    j = j1;
    k = k1;
  if ((j>=0) && (j <= k))
  {
      summation = floor(j/2);
  }
  else if((j>=k) && (k >= 1))
  {
      summation = floor((2*k-j)/2);
  }
  else
  {
  }
}

/*Matrix appears calculates the number of times each matrix appears */
 void Q::Matrixappears()
{
  if ((j>=0) && (j <= k))
  {
     A0appears = r +k - j;
     A1appears = j - 2*r;
     A2appears = r;
   }
  else if((j>=k) && (k >= 1))
  {
     A0appears = r;
     A1appears = 2*k-j - 2*r;
     A2appears = r +j - k;
  }
  else
  {
  }
  if (A0appears < 0 || A1appears < 0 || A2appears <0)
    throw(QlessThanZero("One of the matrices is to be permuted for a negative number of times"));

}
/*Populates Qs matrices*/
void Q::PopulateQ(int a, int b, int c)
{
     A[0] = new mat(n,m);
     A[1] = new mat(n,m);
     A[2] = new mat(n,m);
     B = new mat(m,beta);
  //Populate matrix
     A[0]->load(a1mat);
     A[1]->load(a2mat);
     A[2]->load(a3mat);
     B->load(bmat);

          //Prepare output files
          outputfile.open(outputmat.c_str());
          outputfile<<"FIXED PARAMETER MATRICES"<<endl
                   <<"Matrix A["<<a<<"]"<<endl
                   <<*A[0]<<endl
                   <<"Matrix A["<<b<<"]"<<endl
                   <<*A[1]<<endl
                   <<"Matrix A["<<c<<"]"<<endl
                   <<*A[2]<<endl
                   <<"Matrix B"<<endl
                   <<*B<<endl
                   <<"DETERMINING MATRICES FOR THE VARIABLES J AND K"<<endl;
 }
```

```
void Q::Summation2(int j1, int k1)
{
    j = j1;
    k = k1;
    SS = new mat(n,m);
      mat *G = new mat(m ,beta);

if (k == 0)
{
                            if(j == 0)
                            {
               SS->eye();
                            }
                            else
                            {
                                     *SS = *(A[0]);
                                     for(int i = 0; i < j-1; i++)
                                     {
                                              *SS = *SS * *(A[0]);
                                     }
                            }
 }
                   else if((j == 0) &&  (k!=0))
                   {
                   *SS = *(A[1]) ;
                   for(int i = 0; i < k-1; i++)
                            {
                                     *SS = *SS * *(A[1]);
                            }
                   }
else if((k >= j) && (j>=1))
{
      A0appears = j;
      A1appears = k;
      A2appears = 0;
      *SS = *(SumofPermutations(A0appears,A1appears, A2appears));

      A0appears = 0;
      A1appears = k-j;
      A2appears = j;
      *SS = *SS + *(SumofPermutations(A0appears,A1appears, A2appears));
                                      if (j >= 2)
                                      {
           for (r = 1; r <= j-1; r++)
           {
                A0appears = r;
                A1appears = r + k -j;
                A2appears = j -r;
                *SS = *SS + *(SumofPermutations(A0appears,A1appears, A2appears));
           }
                                      }
   }
   else if((j >= k) &&(k>=1))
   {
      A0appears = j;
      A1appears = k;
      A2appears = 0;
      *SS = *(SumofPermutations(A0appears,A1appears, A2appears));

      A0appears = j - k;
      A1appears = 0;
      A2appears = k;
      *SS = *SS + *(SumofPermutations(A0appears,A1appears, A2appears));
      if(k >= 2)
                                      {
          for (r = 1; r <= k-1; r++)
```

```
                {
                    A0appears = r + j -k;
                    A1appears = r;
                    A2appears = k - r;
                    *SS = *SS + *(SumofPermutations(A0appears,A1appears, A2appears));
                }
                                              }
      }
    cout << fixed << fpreset;
                  cout <<"for k = "<<k<<" and j = "<<j <<"\n ";
    *G = *SS * *B;
    J = join_rows(J,*G);
    cout <<*G;
     //Continue Outputstream
            outputfile<<"for k = "<<k<<" and j = "<<j <<endl
            <<*G<<endl;
}
    void Q::Summation(int j1, int k1)
    {
        j = j1;
        k = k1;
        SS = new mat(n,m);
        mat *G = new mat(m,beta);
            mat *T = new mat(n,m);

     if (k == 0)
     {
        if (j==0)
        {
            SS->eye();
        }
        else
        {
          SS->zeros();
        }
     }
      else if(j >= 2*k + 1)
      {
         SS->zeros();
      }
      else
      {
                                   SS->zeros();
        for ( int r = 0; r <= summation; r++)
        {
            if ((j>=0) && (j <= k))
            {
              A0appears = r +k - j;
              A1appears = j - 2*r;
              A2appears = r;
             }
            else if((j>=k) && (k >= 1))
            {
              A0appears = r;
              A1appears = 2*k-j - 2*r;
              A2appears = r +j - k;
            }
            else
            {
              A0appears = 0;
              A1appears = 0;
              A2appears = 0;
            }
      if (A0appears < 0 || A1appears <  0 || A2appears <0)
     throw(QlessThanZero("One of the matrices is to be permuted for a negative number of times"));
            *T = *(SumofPermutations(A0appears,A1appears, A2appears));
```

```
                    *SS = *SS + *T;
                }//end for
            }//end if
            cout <<"for k = "<<k<<"and j = "<<j <<"\n ";
            *G = *SS * *B;
            J = join_rows(J,*G);
            cout <<*G;

                    //Continue Outputstream
                    outputfile<<"for k = "<<k<<" and j = "<<j <<endl
                    <<*G<<endl;
}
                mat* Q::SumofPermutations(int A0a, int A1a,int A2a)
            {
                int TotalObjects = A0a +A1a + A2a;
                int CountObjects = 0;
                            mat* C = new mat(n,m);
              //C is initialized to the identity matrix
                    mat* S = new mat(n,m);//Sum matrix; we initialize to zero
                            S->zeros();
                    vector<int> MatricestoPermute(TotalObjects);
              for(int i = 0; i < A0a; i++)
              {
                    MatricestoPermute[CountObjects] = 0;
                    CountObjects++;
              }
              for(int i = 0; i < A1a; i++)
              {
                    MatricestoPermute[CountObjects] = 1;
                    CountObjects++;
              }
              for(int i = 0; i <A2a; i++)
              {
                    MatricestoPermute[CountObjects] = 2;
                    CountObjects++;
              }
             for(int i = 0; i < TotalObjects; i++)
             {
                  cout<<" matrix:" <<MatricestoPermute[i]<<"\n";
             }
             cout <<"the total number of objects:"<<TotalObjects<<"\n";
             do
             {
                    //We multiply each matrix with a for loop
                    C->eye();
                    for (int i = 0; i < TotalObjects; i++)
                    {
                        *C = (*C)*(*(A[MatricestoPermute[i]]));
                    }
                    *S = *S + *C;
}while(next_permutation(&MatricestoPermute[0],&MatricestoPermute[TotalObjects]));
             return S;
            }
            void Q::printJ()
            {
                    cout<< J<<"\n";
                    cout<<"The rank of the above matrix is "<<arma::rank(J);
                    J.save("Newmat.mat",raw_ascii);
                    //Place output matrix in output file
                    outputfile<<"THE IMPLEMENTATION OF RANK CONDITION FOR THE CONCATENATED
MATRICES"<<endl
                            <<" "<<endl
                            <<J<< endl
                            <<"THE RANK FOR THE ABOVE MATRIX IS "<<arma::rank(J)<<endl;
                    outputfile.close();
            }
```

```
cpp_int Q::factorial(int n)
{
    if ((n==0) || (n==1))
    {
        return 1;
    }
    else
        return n * factorial(n -1);
}
void Q::Cardinality(int j2, int k2)
{
    outputfile.open(outputmat.c_str());
    j =  j2;
    k = k2;
    cpp_int  a , b, c;
    c = 0;
if (k == 0)
{
    if(j == 0)
    {
        a = 1;b = 0;
    }
    else
    {
        a = 1; b = 0;
    }
 }
else if((j == 0) &&  (k!=0))
{
    a = 1;b =0;
}
else if((j >= k) && (k>=1))
{
        a = factorial(j+k)/(factorial(j)*factorial(k));
        b = factorial(j)/(factorial(k)*factorial(j-k));
        if (k >=2)
        {
            for(int r = 1; r <= (k-1); r++)
c = c + factorial(j+r)/(factorial(r +j -k)* factorial(r)*factorial(k - r));
        }
}
else if((k >= j) &&(j>=1))
{
    a = factorial(j+k)/(factorial(j)*factorial(k));
    b = factorial(k)/(factorial(j)*factorial(k-j));
        if (j >=2)
        {
            for(int r = 1; r <= (j-1); r++)
    c = c + factorial(k+r)/(factorial(r +k -j)* factorial(r)*factorial(j - r));
        }
  }
    outputfile<<"For j = "<<j<<" and k = "<<k<<endl
        <<"CARDINALITY = "<<a +b+c <<endl
                <<" "<<endl;
    cout<<"\n"<<"\tFor j = "<<j<<" and k = "<<k<<endl
        <<"\n"<<"\tCARDINALITY = "<<a +b+c <<endl
                <<" "<<endl;
}
void Q::Cardinality0(int j3, int k3)
{
        outputfile.open(outputmat.c_str());
  j = j3;
  k = k3;
        cpp_int a = 0;

        if (k == 0)
```

```
                                    {
              if (j==0)
              {
                  a=1;
              }
              else
              {
                a=0;
              }
          }
          else if(j >= 2*k + 1)
          {
               a = 0;
          }
          else
          {
             for ( int r = 0; r <= summation; r++)
             {
                   if ((j>=0) && (j <= k))
                   {
                            a = a + factorial(k)/(factorial(r + k-j)*factorial(j-2*r)*factorial(r));
                    }
                   else if((j>=k) && (k >= 1))
                   {
                            a = a + factorial(k)/(factorial(r + j-k)*factorial(2*k - j-2*r)*factorial(r));
                   }
             }//end for
        }//end if
                              outputfile<<"\n\nFor j = "<<j<<"and k = "<<k<<endl
                                                  <<"CARDINALITY = "<<a <<endl
                                                  <<" "<<endl;
                              cout<<"\n"<<"\tFor j = "<<j<<" and k = "<<k<<endl
                                    <<"\n"<<"\tCARDINALITY = "<<a <<endl
                                                  <<" "<<endl;
}
Q::~Q()
{
     for (int i = 0; i < 3; i++)
     {
          delete A[i];
     }
     delete SS;
     delete B;
     delete C;
}
```

2.5.3 QException.h: Header file

```
#ifndef QEXCEPTION
#define QEXCEPTION
#include <iostream>
#include <string>
using namespace std;
class QException
{
     private:
          std::string errmessage;
     public:
          QException(const std::string& mssg);
          std::string getmssg();
};

class QlessThanZero: public QException
{
     public:
          QlessThanZero(const std::string& mssg);
```

```
};
```

2.5.4 QException.h: Header file

```
class QLOEZero: public QException
{
    public:
          QLOEZero(const string& mssg);
};
class QUnEqualSizedMatrix: public QException
{
    public:
          QUnEqualSizedMatrix(const string& mssg);
};

class QUnusualEntry: public QException
{
    public:
          QUnusualEntry(const string& mssg);
};
#endif
```

2.5.5 QException.cpp

```
#include "QException.h"
            QException::QException(const std::string& mssg): errmessage(mssg){}
            string QException::getmssg()
           {
                 return errmessage;
           }

           QlessThanZero::QlessThanZero(const string& mssg): QException(mssg){}
           QLOEZero::QLOEZero(const string& mssg): QException(mssg){}
           QUnEqualSizedMatrix::QUnEqualSizedMatrix(const string& mssg): QException(mssg){}
           QUnusualEntry::QUnusualEntry(const string& mssg): QException(mssg){}
```

2.5.6 main.cpp

```
#include <cstdlib>
#include <typeinfo>
#include <iostream>
#include "Q.h"
#include "QException.h"

using namespace std;
string A1mat, A2mat,A3mat, Bmat, Fstream;
int m, n;
void checkType(int Qvariable)
{
    if ((typeid(Qvariable) != typeid(int))&&(typeid(Qvariable) != typeid(float)))
        {
                 throw(QUnusualEntry("This is an unexpected entry"  ));
        }
}
void NameFixedParameterMatrix(int a,int b, int c)
{
            cout<<"Enter Matrix "<<a<<" file name: ";
            cin>>A1mat;
            cout<<"Enter Matrix "<<b<<" file name: ";
            cin>>A2mat;
            cout<<"Enter Matrix "<<c<<" file name: ";
   cin>>A3mat;
            cout<<"Enter Matrix B file name: ";
```

```
   cin>>Bmat;
        cout<<"Enter Output Matrix file name: ";
   cin>>Fstream;
     if (m<= 0 ||n <=0)
       throw (QlessThanZero("the column or row size is less than zero"));
     if (m != n)
       throw (QUnEqualSizedMatrix("Unequal column and row size"));
}
int main(int argc, char *argv[])
{
   try
   {
     int Bm,k,choose;
                   float h,t1;
cout<<"WELCOME TO VISUAL C++ IMPLEMENTATION OF DETERMINING MATRICES";
                   cout<<"\n      FOR HEREDITARY SYSTEMS AND RELATED PROBLEMS        \n";
                   cout<<"********************************************************* \n";
                   cout<<"\n ";
     cout <<"Enter 1 for Controllability Matrices and Ranks (Double-Delay Systems) \n\n";
     cout <<" Enter 2 for Cardinalities of Determining Matrices (Double-Delay Systems)\n";
     cout<<"\n\t:";cin >>choose;
                         Q *firstQ;
                         Q *Q1= new Q(3,3);
                         int a,b;
                   if (choose == 1 )
                   {
                         checkType(choose);
                         cout << "Please enter the number of rows for A_matices: ";
                         cin >> n;
                         checkType(n);
                         cout << "Please enter the number of columns for A_matrices: ";
                         cin >> m;
                         checkType(m);
                         cout << "Please enter the number of columns for matrix B: ";
                         cin >>Bm;
                         checkType(Bm);
                         cout << "Please enter a value for t1: ";
                         cin >>t1;
                         checkType(t1);
                         cout << "Please enter a value for h: ";
                         cin >>h;
                         checkType(h);
                   }
                   else if(choose ==2)
                   {
                               cout<<" Enter a value for j : "<<"\t";
                               cin>>a;
                               cout<<"\n "<<"Enter a value for k : \t";
                               cin>>b;
                               cout<<"\n"<< " Factorials 0 t0 40:\n \n";
                   }
     switch (choose)
     {
        case 1:
                               NameFixedParameterMatrix(0,1,2);
                               firstQ = new Q(n,m,Bm,A1mat,A2mat,A3mat,Fstream,Bmat);
                               firstQ->PopulateQ(0,1,2);
                               for (int j = 0; ((j*h < t1) && (j<=n-1)); j++)
           {
             for (int k = 0; k <= (n - 1); k++)
             {
                firstQ->timesToSum(j,k);
                firstQ->Summation(j,k);
                                                   //firstQ->Cardinality0(j,k);
             }
           }
```

```
                                        firstQ->printJ();
            break;
                              case 2:
                                        cout<<"\n" <<"\t"<<"i"<<"\t"<<"i!"<<"\n";//Factorials 0 to 40 and Cardinality for Double-
Delay System
                                        for (int i = 0; i <= 40; i++)
                                        {
                                                cout<<"\n"<<"\t"<<i<<"\t"<<Q1->factorial(i)<<"\n";
                                        }
                                        Q1->timesToSum(a,b);
                                        Q1->Cardinality0(a,b);
                                        break;
                    }
  }
  catch(QlessThanZero& error)
  {
     cout <<error.getmssg();
  }
  catch(QLOEZero& error)
  {
          cout <<error.getmssg();
  }
  catch(QUnEqualSizedMatrix& error)
  {
          cout <<error.getmssg();
  }
  catch(QUnusualEntry& error)
  {
      cout<<error.getmssg();
  }
       //   system("PAUSE");
//   return EXIT_SUCCESS;*/
       return 0;
}
```

Excellent sources of information on C++ programming include Hubbard (1996), Cohoon & Davidson (2003), D'Orazio (2004) and Deitel & Deitel (2011).

3. RESULTS AND DISCUSSIONS

3.1 Illustration of Electronic C++ Implementation of Determining matrices, Controllability Matrix and Controllability Rank Condition for System (1)

Let $A_0 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 3 & 2 \end{pmatrix}, A_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & -1 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}, B = \begin{pmatrix} 1 & -1 \\ 2 & 1 \\ 1 & 2 \end{pmatrix}, t_1 = 3,\ h = 0.5.$

Then,

$$\begin{aligned} \operatorname{rank} \hat{Q}_3(3) &= \operatorname{rank}\left[Q_0(0)B, Q_1(0)B, Q_2(0)B, Q_1(h)B, Q_2(h)B, Q_1(2h)B, Q_2(2h)B\right] \\ &= \operatorname{rank}\left[B, A_0B, A_0^2B, Q_1(h)B, Q_2(h)B, Q_1(2h)B, Q_2(2h)B\right]. \end{aligned}$$

The components $Q_0(h) = Q_0(2h) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$ do not affect the rank, and hence those four columns may be deleted, reducing $\hat{Q}_3(3)$ to the following 3 by 14 matrix:

$$\begin{matrix} 1 & -1 & 6 & 3 & 20 & 10 & 4 & 2 & 38 & 25 & 4 & 2 & 48 & 33 \\ 2 & 1 & 2 & 1 & 2 & 1 & 8 & 7 & 48 & 27 & 5 & 4 & 53 & 34 \\ 1 & 2 & 10 & 5 & 38 & 19 & 0 & 0 & 46 & 32 & 4 & 5 & 49 & 32 \end{matrix}$$

of rank 3.

4. CONCLUSION

The article addressed the issue of computational feasibility and large–scale industrial applicability of determining matrices and related concepts and pioneered the implementation of the determining matrices, controllability matrices, cardinality and controllability rank conditions for a class of double-delay control systems, on the C++ platform, followed with an illustrative example. The automation of the results has implicitly eliminated all computational and implementation constraints and has placed the generally neglected implementation aspect of mathematical results on the front burner.

REFERENCES

[1] Gabasov, R. & Kirillova, F. *The qualitative theory of optimal processes*. Marcel Dekker Inc., New York, 1976.

[2] Ukwu, C. Euclidean controllability and cores of Euclidean targets for differential difference systems. *Master of Science Thesis in Applied Math. with O.R. (Unpublished), North Carolina State University, Raleigh, N. C. U.S.A*. 1992.

[3] Ukwu, C. An exposition on Cores and Controllability of differential difference systems, *ABACUS*. ***24****(2), 276-285*, 1996.

[4] Ukwu, Chukwunenye. Controllability, Cores of Targets and matrix structures of certain classes of functional differential systems. Ph.D. Thesis in Mathematics. (Unpublished), University of Jos, Jos, Plateau State, Nigeria, 2015.

[5] Ukwu, C. & E.J.D. Garba (2014a). A derivation of an optimal expression for the index of control systems matrices for single – delay differential systems. *Journal of Mathematical Sciences*. ***25****(1)*, 2014.

[6] Ukwu Chukwunenye (2014c). The structure of determining matrices for single-delay autonomous linear neutral control systems. *International Journal of Mathematics and Statistics Invention (IJMSI). 2(3), 31 – 47*, 2014.

[7] Ukwu Chukwunenye (2014b). The structure of determining matrices for a class of double delay control systems. *International Journal of Mathematics and Statistics Invention (IJMSI). 2(3), 14–30*, 2014.

[8] "Boost C++ Libraries Release 1.54.0: *An Open Source C++ Software, Boost, Beman Dawes, Rene Rivera"*, 2008.

[9] Cohoon J. P. & Davidson J. W. C++ Program design: An introduction to programming and object-oriented design, third edition. 3rd edition, Tata *McGraw–Hill*, *New Delhi*, 2003.

[10] Chidume, C. An Introduction to Metric Spaces. *The Abdus Salam, International Centre for Theoretical Physics, Trieste, Italy*, 2003.

[11] Chidume, C. Applicable Functional Analysis. *The Abdus Salam, International Centre for Theoretical Physics, Trieste, Italy*, 2007.

[13] Hubbard, J.R. Theory and Problems of programming with C++. Schaum's Outline Series, *McGraw-Hill*, *New York*, 1996.

[14] Conrad Sanderson. Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. Technical Report, NICTA, 2010.

[15] D'Orazio B. T. Programming in C++: Lessons and Applications. McGraw-Hill, New York, 2004.

[16] Deitel P. & Deitel H. C++: How to program, Late objects version. 7th Ed. Pearson International edition, New Jersey, 2011.