Computer Aided Development of State Models for Linear Systems

P-195

Project Report 1993-94

Submitted by

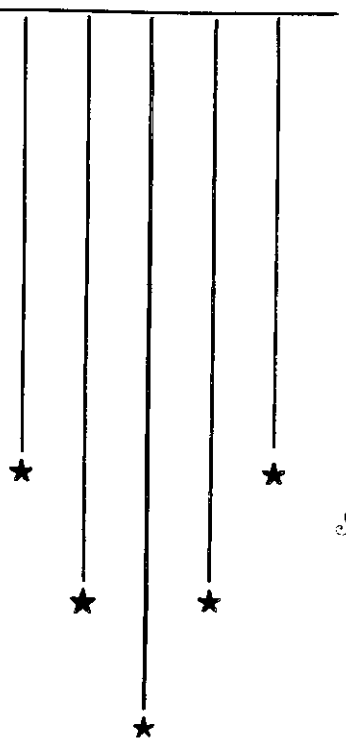
Bharath .C

Nilan .M

Pugazanthi .D

Under the Guidance of

Mrs. R. Umamaheswari



In partial fulfilment of the requirements for the award of Degree of

Bachelor of Engineering

in Electrical and Electronics Engineering

of the Bharathiar University, Coimbatore

Department of Electrical and Electronics Engineering

Kumaraguru College of Technology

Coimbatore-641 006

Department of Electrical and Electronics Engineering

Kumaraguru College of Technology

Coimbatore-641 006

Certificate

This is to Certify that the Report entitled

Computer Aided Development of
State Models for Linear Systems

has been Submitted by

Mr. .....

in partial fulfilment for the award of Bachelor of Engineering in Electrical and Electronics Engineering Branch of the Bharathiar University, Coimbatore-641 046 during the academic year 1993-94.

Faculty Guide

Dr. K. A. PALANISWAMY

Head of the Department

Certified that the candidate was Examined by us in the Project Work Viva-Voce Examination held on ..... and the University Register Number was .....

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We are grateful to our guide Mrs. B.Uma Maheshwari, Associate Lecturer, Department of Electrical and Electronics Engineering, for her guidance and whole hearted co-operation to make this project a success.

We also express our sincere gratitude to Dr.K.A.Palaniswamy B.E., M.Sc(Engg.), Ph.D., C.Eng(I), MISTE, FIE, the Professor and Head, Department of Electrical and Electronics Engineering for his constant encouragement throughout the course of the project.

The team members are greatly indebted to Dr.S.Subramanian Ph.D., SMIEEE, the principal for providing facilities to carry out the project in the college.

Our sincere thanks are due, to all the members of staff, Department of Electrical and Electronics Engineering. and to all our student friends.

SYNOPSIS

Computer Aided Analysis of Linear Systems is basically a software which is used to develop state models of linear time-invariant systems.

The state models which can be developed using this software are of two types, namely, Phase variable form and Canonical form. Single input and single output systems are dealt with in this software. The algorithms have been developed to tackle systems of `n'th order. The maximum order of the system, which this software can accept is 65535, but for ease of use it has been curtailed to 10. This software is also capable of dealing with systems having imaginary poles.

C++ is the language with which the software was developed. This is a flexible language which involves Object Oriented Programming, the use of which simplified many of the algorithms used in the software. Test results are presented.

CONTENTS

INTRODUCTION

State variable analysis involves a general mathematical representation of a system which, along with the output, yields information about the state of the system variables at some predetermined points along the flow of signals. Classical methods like root locus and frequency of a system require that the physical system be modelled in the form of a tranfer function. Though this technique furnishes a simple and powerful analysis of the system under study, it has its own drawbacks. This is where the state variable representation makes its mark. This is basically a time domain approach unlike the classical methods which are essentially frequency domain methods.

The software presented in this project develops state models for linear single-input-single-output systems. The state models which this software can develop are the phase variable representation and the canonical variable representation. Each system is identified by two polynomials, one for input and one for output. The entire project deals with various polynomial algorithms and operations, so a good understanding of polynomial operations like their addition, subtraction, multiplication, division, factorizing and many more is needed. Not only an

understanding of their operations is important, but their software implementation must also be taken into consideration.

The language which was used to develop the software is C++. C++ provides many enhancements to C which greatly simplyfy this software. This language provides the flexibility of object oriented programming, which means new features like operator overloading, polymorphism, inheritance are at the disposal of the programmer. The one aspect of this language which aided the development of this software is operator overloading, which greatly simplified the algorithms used.

STATE VARIABLE ANALYSIS

2.1 INTRODUCTION

The classical methods like root locus and frequency of a system require that the physical system be modelled in the form of a transfer function. Though the transfer function model provides us with simple and powerful analysis and design techniques, it suffers from certain drawbacks. A transfer function is only defined under zero initial conditions. Further, it has certain limitations due to the fact that the transfer function model is only applicable to linear time-invariant systems and there too it is generally restricted to single-input-single-output systems as this approach becomes highly cumbersome for use in multiple-input-mulitple-output systems. Another limitation of the transfer function technique is that it reveals only the system output for a given input and provides no information regarding the internal state of the system. There may be situations where the output of the system is stable and yet the system elements may have a tendency to exceed their specified ratings. In addition to this, it may sometimes be necessary and advantageous to provide a feedback proportiional to some of the internal variables of a system rather than the output alone, for the purpose of stabilizing and improving the performance of a system. It is further

observed that the classical design methods based on the transfer function model are essentially trial and error procedures. Such procedures are difficult to visualize and organize even in moderately comlex systems and may not lead to a control system which yields an optimum performance in some defined sense.

The need of a more general mathematical representation of a system which, along with the output, yields information about the state of the system variables at some predetermined points along the flow of signals, is felt. Such considerations have led to the development of the state variable approach. It is a direct time domain approach which provides a basis for modern control theory and system optimization. It is a very powerful technique for the analysis and design of linear and nonlinear, time-invariant or time-varying multiple-input-multiple-output systems. The organization of the state variable approach is such that it is easily amenable to solution through digital computers.

2.2 CONCEPTS OF STATE, STATE VARIABLES AND STATE MODEL

A mathematical abstraction to represent or model the dynamics of a system utilizes three types of variables called the input, the output and the state variables.

The state of a dynamic system is a minimal set of variables (known as state variables) such that the knowledge if these variables at t=to together with the knowledge of the inputs for t>=to, completely determines the behaviour of the system for t>to.

For notational economy, the different variables may be represented by the input vector U(t), output vector Y(t) and state vector X(t). The state equations for time-invariant and time varying systems are as given below.

$\dot{X}(t) = F(X(t), U(t))$ - time-invariant systems

$\dot{X}(t) = F(X(t), U(t), t)$ - time varying systems

The output equations for time-invarinat and time varying sytems are as follows.

$Y(t) = G(X(t), U(t))$ - time-inariant systems

$Y(t) = G(X(t), U(t), t)$ - time varying systems

The state equations along with the output equations constitute the state model of a system.

2.3 STATE MODEL OF LINEAR SYSTEMS

The state model of linear time-invariant systems is depicted below.

$$\dot{X} = AX(t) + BU(t) \tag{2.1}$$

$$Y = CX(t) + DU(t) \tag{2.2}$$

where A and C are matrices of order n x n, B is of order n x m and D is of order p x m. `n' is the number of state equations, `p' is the number of output equations and `m' is the number of inputs. A is called the system matrix, B is called the input matrix, C is called the output matrix and C is called the transmission matrix.

For single-input-single-output systems B is of order n x 1, C is of order 1 x n and D is constant.

2.4 PHASE VARIABLE REPRESENTATION

For a system defined by the transfer function

$$T(s) = \frac{Y(s)}{U(s)} = \frac{b_0 s^n + b_1 s^{n-1} + \ldots + b_{n-1} s + b_n}{s^n + a_1 s^{n-1} + \ldots + a_{n-1} s + a_n} \tag{2.3}$$

the phase model is as given below

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} -a_1 & 1 & 0 & \dots & 0 \\ -a_2 & 0 & 1 & \dots & 0 \\ & & & & \\ -a_n & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 - a_1 b_0 \\ b_2 - a_2 b_0 \\ \\ b_n - a_n b_0 \end{bmatrix} \tag{2.4}$$

$$y = x_1 + b_0 u \tag{2.5}$$

2.5 CANONICAL VARIABLE REPRESENTATION

For the same system considered above the Canonical model is given by

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} r_1 & 0 & 0 & \dots & 0 \\ 0 & r_2 & 0 & \dots & 0 \\ & & & & \\ 0 & 0 & 0 & \dots & r_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ \\ 1 \end{bmatrix} \tag{2.6}$$

$$y = \begin{bmatrix} c_1 & c_2 & c_3 \dots & c_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + b_0 u \qquad (2.7)$$

Having discussed the various representations of system models in this chapter we go to find how the software can be implemented to get the same.

SOFTWARE IMPLEMENTATION

The software implementation of state models involves quite a lot of polynomil arithmetic since each transfer function usually involves two polynomials.

In the case of the Phase variable state model the elements of the system matrix can be determined from the equation directly. The elements of the input matrix are calculated from the different coefficients of both the input and output equations.

Canonical model of system requires the poles of the system. To determine the various poles of the system Lin's method is used. This method is a iterative polynomial division process. Consider the algebric equation

$$s^{n} + a_{n-1} s^{n-1} + a_{n-2} s^{n-2} + \ldots + a_{2} s^{2} + a_{1} s + a_{0} = 0 \qquad (3.1)$$

The first trial factor uses the three lowest order terms of the original equation. For the above equation the trial factor is

$$s^{2} + \frac{a_{1}}{a_{2}} s + \frac{a_{0}}{a_{2}} \qquad (3.2)$$

The original equation is divided by this first trial factor. If the remainder is too large, the next trial factor is used. This procedure is continued until the remainder is negligible. The last trial factor is a quadratic factor of the original equation. The quotient polynomial, which is of order (n-2) contains the remaining factors of the original equation. Lin's method is then applied to the quotient polynomial to obtain the other quadratic factor of the original equation. Each quadratic factor is then factorized to obtain the roots of the equation.

This process requires an algorithm for general polynomial division which in turn requires algorithms for subtraction and multiplication. This is where the usage of C++ facilitates. It provides operator overloading which allows such arithmetic to be done easily. However the operators must be overloaded properly. In addition to this there is the possiblity of complex roots, which by itself requires a lot of algorithms. Complex roots pose a problem during the partial fractioning phase of the program. They also require that the operators be overloaded for them also in order for easy program development.

Canonical model also requires the transfer function to be in a partial fractioned form. This process is no mean

task when the system under study has a lot of poles. If there are repeated poles then there is a need for differentiation also. As mentioned above, the presence of complex roots presents hinderances for calculation. The numerator of each such complex root also tends to be a complex one. Complex roots and numerators create problems during displays also.

PROGRAMMING LANGUAGE USED FOR DEVELOPMENT OF SOFTWARE.

4.2 INTRODUCTION·

C++, blending the C language with support for object oriented programing, seems destined to be one of the most important programming languages of the 1990s. Its C ancestry brings C++ the tradition of an efficient, compact, fast and portable language. Its object oriented heritage brings C++ a fresh programming methodology designed to cope with the escalating complexity of mordern programming tasks. Another notable aspect of this language is the supporting cast of new jargons-objects, classes, encapsulation, data hiding, polymorphism and inheritance, just to name a few.

C++ joins together two separate programming traditions- the procedural tradition, represented by C, and the object oriented language tradition, represented by the enhancements C++ adds to C.

4.2 OBJECT ORIENTED PROGRAMMING

Although the principles of structured programming improved the clarity, reliability and ease of maintenance of programs, large scale programming still remains a challenge. Object oriented programming (OOP) brings a new approach to that challenge. Unlike procedural programming, which

emphasizes algorithms, OOP emphasizes the data. Rather than trying to fit a problem to the procedural approach of a language, OOP attempts to fit the language to the problem. The idea is to design a data form that corresponds to the essential features of a problem. In C++, a class is a specification describing such a new data form, and an object is a particular data structure constructed according to that plan. In general, a class defines what data is used to represent an object and the operations that can be performed upon that data.

The OOP approach to program design is to first design classes that accurately represent those things with which the program deals. Then you proceed to design a program using objects of those classes. The process of going from a lower level of organization, such as classes, to higher level, such as program design, is called Bottom-up programming.

There's more to OOP programming than the binding of data and methods into a class definition. OOP, for example, facilitates creating reusable code, and that eventually can save a lot of work. Information hiding safeguards data from improper access. Polymorphism lets you create multiple definitions for operators and functions, with the

programming context determining which definition is used. Inheritance lets you derive new classes from old ones. As you can see, object oriented programming introduces many new ideas and involves a different approach to programming than does procedural programming. Instead of concentrating on tasks, you concentrate on representing concepts. Instead of taking a top-down progarmming approach, you sometimes take a bottom-up approach.

4.3 C++

Like C, C++ began its life at Bell labs, where Bjarne Strousrup developed the language in the early 1980s. In his own words, "C++ was primarily designed so that the author and his friends would not have to program in assembler, C, or various modern high-level languages. Its main purpose was to make writing good programs easier and more pleasant for the individual programmer." Strousrup based C++ on C because of C's brevity, its suitability to system programming, its widespread availability, and its close ties to the UNIX operatin system. C++'s OOP aspect was inspired by a computer simulation language called Simula67. Strousrup added OOP features to C without significantly changing the C component. Thus C++ is a superset of C, meaning that any valid C program is a valid C++ program, too. C++ programs can use existing C software libraries. This has helped the

spreading of C++.

The name C++ comes from the C increment operator ++, which adds 1 to the value of a variable. The name C++ correctly suggests an augmented version of C. A computer program translates a real life problem to series of actions to be taken by a computer. While the OOP aspect of C++ gives the language the ability to relate to concepts involved in the problem, the C part of C++ gives the language the ability to get close to the hardware. This combination of abilities also has helped the spread of C++.

DEVELOPMENT AND TESTING

This chapter deals with the various flowcharts, source listings and sample outputs of the project discussed here.

There are three main flowcharts. The first one is the system flowchart and the other two represent the subroutines for the two phase models.

Four source listings are presented, two of which are header files. The other two are contain the code for the various function and class declerations used.

The output is presented for two systems. The order of the systems being 4 and 5.

5.1 FLOWCHARTS

System Flowchart

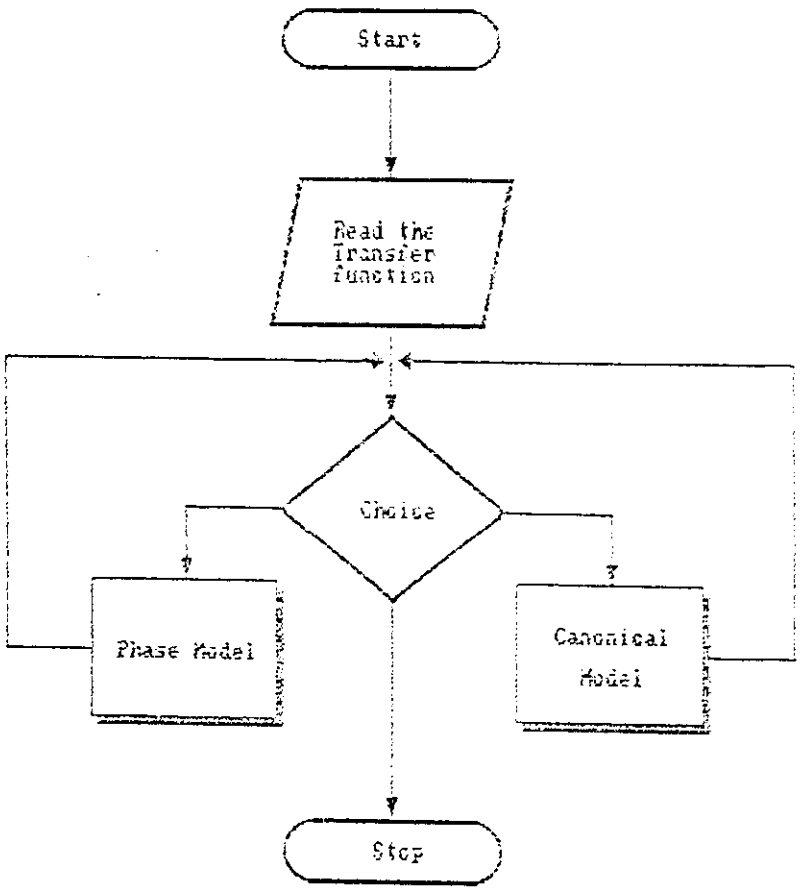


Fig 5.1

Flowchart for Phase Model

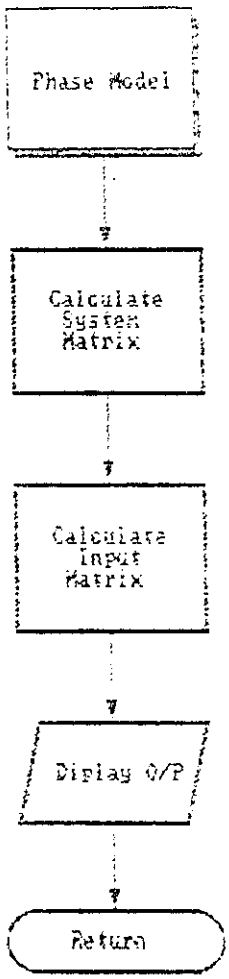


Fig 5.2

Flowchart for Canonical Model

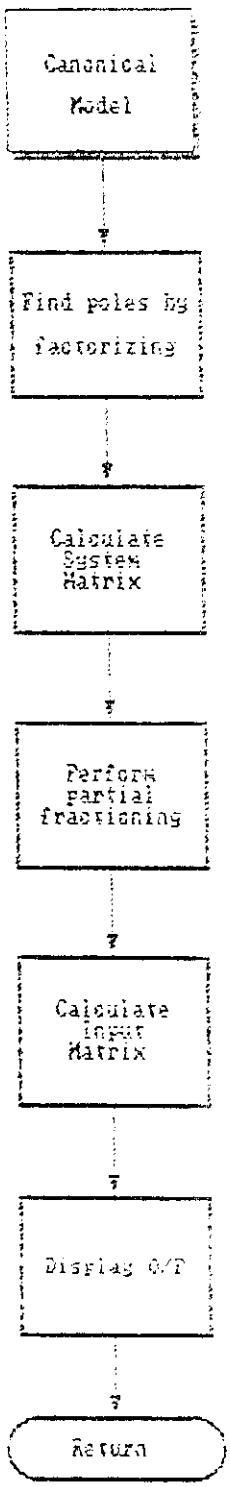


Fig 5.3

5.2 SOURCE LISTING

SOURCE LISTING : POLYN.H

```
class polynomial
{
private:
  unsigned int degree;        //  degree of the polynomial.
  double  *coeffs;            //  coefficients  of  various
                              //      elemnts,
                              //      polynomial    size     being
                              //      (degree+1).
  //double  *rreal;           //  roots of  the  polynomial
                              //      both
  //double *rimag;            //  real and imaginary.
  complex addi( double, double, double, double );

public:

  polynomial(unsigned int, double*);
  polynomial(unsigned int);
  polynomial( const polynomial& );
  ~polynomial(void);
  polynomial& operator=( polynomial& );
  friend  polynomial  operator+ ( const  polynomial&,  const
    polynomial& );
  friend  polynomial  operator- ( const  polynomial&,  const
```

```
    polynomial& );

  friend  polynomial  operator* ( const  polynomial&,  const
    polynomial& );
  friend  polynomial  operator/ ( const  polynomial&,  const
    polynomial& );
  friend  polynomial  operator% ( const  polynomial&,  const
    polynomial& );
  friend   int   operator   >   (   const   polynomial&,   const
    polynomial& );
  friend void factorize( const polynomial& );
  friend   void   partial_fract(   const   polynomial&,   const
    polynomial&, double*, double* );
  friend   void   phase_model(   const   polynomial&,   const
    polynomial& );
  complex evaluvate( double, double );
  void change_values( double* );
  void show();
  void trim();
  void read();
};
```

SOURCE LISTING : COMPLEX.H

```
/* complex.h

        Complex Number Library - Include File
        class complex:  declarations for complex numbers.

        friend complex _Cdecl acos(complex&);
        friend complex _Cdecl asin(complex&);
        friend complex _Cdecl atan(complex&);
        friend complex _Cdecl log10(complex&);
        friend complex _Cdecl tan(complex&);
        friend complex _Cdecl tanh(complex&);
        complex _Cdecl operator+();
        complex _Cdecl operator-();
*/

#ifndef __cplusplus
#error Must use C++ for the type complex.
#endif

#if     !defined(_COMPLEX_H)
#define _COMPLEX_H      1

#include <math.h>

class complex {
```

```
public:
        // constructors
        complex(double __re_val, double __im_val=0);
        complex();

        // complex manipulations
        friend  double _Cdecl real(complex&);  // the  real
                                                //     part
        friend   double  _Cdecl  imag(complex&);     //    the
                                                imaginary part
        friend   complex  _Cdecl  conj(complex&);   //    the
                                            complex conjugate
        friend double _Cdecl norm(complex&);   // the square
                                            of the magnitude
        friend double _Cdecl arg(complex&);     // the  angle
                                            in the plane
        friend  complex  _Cdecl polar(double  __mag,  double
          __angle=0);
                        //  Create  a complex  object  given
                          polar coordinates

        // Overloaded ANSI C math functions
        friend double  _Cdecl abs(complex&);
        friend complex _Cdecl acos(complex&);
        friend complex _Cdecl asin(complex&);
        friend complex _Cdecl atan(complex&);
```

```
friend complex _Cdecl cos(complex&);
friend complex _Cdecl cosh(complex&);
friend complex _Cdecl exp(complex&);
friend complex _Cdecl log(complex&);
friend complex _Cdecl log10(complex&);
friend  complex _Cdecl pow(complex&  __base,  double
   __expon);
friend  complex _Cdecl pow(double  __base,  complex&
  __expon);
friend complex _Cdecl pow(complex& __base,  complex&
  __expon);
friend complex _Cdecl sin(complex&);
friend complex _Cdecl sinh(complex&);
friend complex _Cdecl sqrt(complex&);
friend complex _Cdecl tan(complex&);
friend complex _Cdecl tanh(complex&);

// Binary Operator Functions
friend complex _Cdecl operator+(complex&, complex&);
friend complex _Cdecl operator+(double, complex&);
friend complex _Cdecl operator+(complex&, double);
friend complex _Cdecl operator-(complex&, complex&);
friend complex _Cdecl operator-(double, complex&);
friend complex _Cdecl operator-(complex&, double);
friend complex _Cdecl operator*(complex&, complex&);
```

```
        friend complex _Cdecl operator*(complex&, double);
        friend complex _Cdecl operator*(double, complex&);
        friend complex _Cdecl operator/(complex&, complex&);
        friend complex _Cdecl operator/(complex&, double);
        friend complex _Cdecl operator/(double, complex&);
        friend int _Cdecl operator==(complex&, complex&);
        friend int _Cdecl operator!=(complex&, complex&);
        complex& _Cdecl operator+=(complex&);
        complex& _Cdecl operator+=(double);
        complex& _Cdecl operator-=(complex&);
        complex& _Cdecl operator-=(double);
        complex& _Cdecl operator*=(complex&);
        complex& _Cdecl operator*=(double);
        complex& _Cdecl operator/=(complex&);
        complex& _Cdecl operator/=(double);
        complex _Cdecl operator+();
        complex _Cdecl operator-();

// Implementation

private:
        double re, im;
};

// inline complex functions
```

```
inline complex::complex(double __re_val, double __im_val)
{
        re = __re_val;
        im = __im_val;
}

inline complex::complex()
{
/* if you want your complex numbers initialized ...
        re = im = 0;
*/
}

inline complex _Cdecl complex::operator+()
{
        return *this;
}

inline complex _Cdecl complex::operator-()
{
        return complex(-re, -im);
}

//   Definitions of compound-assignment operator member
    functions
```

```
inline complex& _Cdecl complex::operator+=(complex& __z2)
{
        re += __z2.re;
        im += __z2.im;
        return *this;
}

inline complex& _Cdecl complex::operator+=(double __re_val2)
{
        re += __re_val2;
        return *this;
}

inline complex& _Cdecl complex::operator-=(complex& __z2)
{
        re -= __z2.re;
        im -= __z2.im;
        return *this;
}

inline complex& _Cdecl complex::operator-=(double __re_val2)
{
        re -= __re_val2;
        return *this;
}

inline complex& _Cdecl complex::operator*=(double __re_val2)
{
```

```
        re *= __re_val2;
        im *= __re_val2;
        return *this;
}

inline complex& _Cdecl complex::operator/=(double __re_val2)
{
        re /= __re_val2;
        im /= __re_val2;
        return *this;
}

// Definitions of non-member complex functions

inline double _Cdecl real(complex& __z)
{
        return __z.re;
}

inline double _Cdecl imag(complex& __z)
{
        return __z.im;
}

inline complex _Cdecl conj(complex& __z)
{
        return complex(__z.re, -__z.im);
```

```
}

inline complex _Cdecl polar(double __mag, double __angle)
{
        return                          complex(__mag*cos(__angle),
                __mag*sin(__angle));
}

// Definitions of non-member binary operator functions

inline  complex  _Cdecl  operator+(complex&  __z1,  complex&
        __z2)
{
        return   complex(__z1.re   +   __z2.re,   __z1.im   +
                __z2.im);
}

inline  complex _Cdecl operator+(double __re_val1,  complex&
     __z2)
{
        return complex(__re_val1 + __z2.re, __z2.im);
}

inline   complex  _Cdecl  operator+(complex&  __z1,   double
     __re_val2)
{
        return complex(__z1.re + __re_val2, __z1.im);
}
```

```
inline complex _Cdecl operator-(complex& __z1, complex&
    __z2)
{
        return complex(__z1.re - __z2.re, __z1.im -
                __z2.im);
}

inline complex _Cdecl operator-(double __re_val1, complex&
__z2)
{
        return complex(__re_val1 - __z2.re, -__z2.im);
}

inline complex _Cdecl operator-(complex& __z1, double
    __re_val2)
{
        return complex(__z1.re - __re_val2, __z1.im);
}

inline complex _Cdecl operator*(complex& __z1, double
    __re_val2)
{
        return complex(__z1.re*__re_val2,
                __z1.im*__re_val2);
}
```

```
inline  complex _Cdecl operator*(double __re_val1,  complex&
    __z2)
{
        return                          complex(__z2.re*__re_val1,
                __z2.im*__re_val1);
}

inline  complex  _Cdecl  operator/(complex&  __z1,  double
    __re_val2)
{
        return                          complex(__z1.re/__re_val2,
                __z1.im/__re_val2);
}

inline int _Cdecl operator==(complex& __z1, complex& __z2)
{
        return __z1.re == __z2.re && __z1.im == __z2.im;
}

inline int _Cdecl operator!=(complex& __z1, complex& __z2)
{
        return __z1.re != __z2.re || __z1.im != __z2.im;
}

// Complex stream I/O

#include <iostream.h>
```

```
ostream& _Cdecl operator<<(ostream&, complex&);
istream& _Cdecl operator>>(istream&, complex&);

#endif
```

SOURCE LISTING : LINSA.CPP

```
#include<iostream.h>
#include<complex.h>
#include<stdio.h>
#include<conio.h>
#include"c:\tcp\bharath\polyn.h"

double* read_coeffs( int* );
int main()
{
  int dg;
  double* ar;

  clrscr();

  double* a;
  double* b;
  double na[] = {2,6,2};
  double da[] = {6,11,6,1};
  polynomial n(2,na);
  polynomial d(3,da);

  //factorize(d);
  partial_fract(n,d,a,b);

  /*ar = read_coeffs( &dg );
  polynomial nu( dg, ar );
```

```
    nu.show();
    delete ar;
    ar = read_coeffs( &dg );
    polynomial do( dg, ar );
    do.show();
    phase_model( nu, do );*/
}

double* read_coeffs( int* dog )
{
     int dg,i;
     double* arr;

     clrscr();
     cout<<"Enter the degree : ";
     scanf("%d",&dg);
     *deg = dg;
     arr = new double[ dg + 1 ];
     for( i = dg; i >= 0; i-- )
     {
        cout<<"Enter the coefficient of degree "<<i<<" : ";
        scanf("%lf",&arr[i]);
        cout<<"\n";
     }
     return arr;
}
```

SOURCE LISTING : POLYN.CPP

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<complex.h>
#include"c:\tcp\bharath\polyn.h"

polynomial::polynomial( unsigned deg, double *cof )
{
  int i;

  degree = deg;
  coeffs = new double[ deg + 1 ];
  for( i = 0; i < deg + 1; i++)
    coeffs[ i ] = cof[ i ];
}

polynomial::polynomial(unsigned deg)
{
  int i;

  degree = deg;
  coeffs = new double[ degree + 1 ];
  if( coeffs == NULL ) cout<<"Not enough memory";
  for( i = 0; i < deg + 1; i++ )
    coeffs[ i ] = 0;
}
```

```
polynomial::polynomial( const polynomial& a)
{
  int i;

  degree = a.degree;
  coeffs = new double[ degree + 1 ];
  for( i = 0; i <= degree; i++ )
    coeffs[ i ] = a.coeffs[ i ];
}

polynomial::~polynomial()
{
  delete coeffs;
}

complex  polynomial::addi( double r1, double i1, double  r2,
  double i2 )
{
  double r, i;
  r = r1 + r2;
  i = i1 + i2;
  if( r == 0 && i == 0 )
    r = 1;
  complex rs( r, i );
  return rs;
}
```

```
void polynomial::change_values( double* val )
{
   int i;

   for( i = 0; i < degree + 1; i++ )
     coeffs[ i ] = val[ i ];
}

void polynomial::show()
{
  int i;
  cout<<"\n(";
  for( i = degree; i >= 0; i-- )
    cout<<"  "<<coeffs[i]<<":"<<i;
  cout<<"  )";
}

void polynomial::trim()
{
   int i;
   long cents;
   double round;

   for( i = 0; i <= degree; i++)
   {
     cents = long(1000.0*coeffs[i]+0.5);
     round = double(cents)/1000.0;
```

```
    coeffs[i] = round;
  }
  for( i = degree; coeffs[i] == 0; i-- );
  degree = i;
  if( i < 0 )
    degree = 0;
  *this = *this;
}

polynomial& polynomial::operator=( polynomial& py )
{
  int i;
  if( this == &py ) cout<<"Yes";
  double* temp = new double[ py.degree + 1 ];
  for( i = 0; i <= py.degree; i++ )
    temp[ i ]  = py.coeffs[ i ];

  delete coeffs;
  degree = py.degree;
  coeffs = new double[ degree + 1 ];
  for( i = 0; i <= degree; i++ )
    coeffs[ i ] = temp[ i ];
  delete temp;
  return *this;

}
```

```
polynomial operator+( const polynomial& a, const polynomial&
  b )
{
  int i;
  polynomial  rs(  (  a.degree > b.degree  )  ?  a.degree  :
    b.degree );
  for( i = 0; i <= rs.degree; i++ )
  {
    rs.coeffs[i] += ( a.degree >= i ) ? a.coeffs[i] : 0;
    rs.coeffs[i] += ( b.degree >= i ) ? b.coeffs[i] : 0;
  }
  return rs;
}

polynomial operator-( const polynomial& a, const polynomial&
  b )
{
  int i;
  polynomial  rs(  (  a.degree > b.degree  )  ?  a.degree  :
    b.degree );
  for( i = 0; i <= rs.degree; i++ )
  {
    rs.coeffs[i] += ( a.degree >= i ) ? a.coeffs[i] : 0;
    rs.coeffs[i] -= ( b.degree >= i ) ? b.coeffs[i] : 0;
```

```
    }
    for( i = rs.degree; rs.coeffs[i] == 0; i-- );
    rs.degree = i;
    if( i < 0 )
      rs.degree = 0;
    //rs = rs;
    return rs;
  }

  polynomial operator* ( const polynomial& mr, const
    polynomial& mt )
  {
    int i, j;
    polynomial rs ( mr.degree + mt.degree );
    for( i = 0; i <= mr.degree; i++ )
      for( j = 0; j <= mt.degree; j++ )
        rs.coeffs[ i + j ] += mr.coeffs[ i ] * mt.coeffs[ j ];
    for( i = rs.degree; rs.coeffs[i] == 0; i-- );
    rs.degree = i;
    if( i < 0 )
      rs.degree = 0;
    //rs = rs;
    return rs;
  }
```

```
polynomial operator/ ( const polynomial& d, const
  polynomial& v )
{
  int i, j, quo_deg = d.degree - v.degree;
  double* ta = new double [ quo_deg + 1];

  polynomial q ( quo_deg );
  polynomial tq ( quo_deg );
  polynomial td = d;

  //td = d;
  for( i = 0; i <= quo_deg; i++ )
  {
    for( j = 0; j <= quo_deg; j++ )
      ta[ j ] = 0;
    ta[ quo_deg - i ] = td.coeffs[ td.degree ] / v.coeffs[
      v.degree ];
    tq.change_values( ta );
    td = td - tq * v;
    q = q + tq;
  }
  delete ta;
  return q;
}
```

```
polynomial operator% ( const polynomial& d, const
  polynomial& v)
{
  polynomial rs( v.degree - 1 );

  rs = d - d / v * v;
  rs.trim();

  return rs;
}

complex polynomial::evaluvate( double r1, double im )
{
  int i;
  double r1, r2;
  complex rs( 0 );
  complex a( r1, im );

  if( r1 == 0 && im == 0 )
    rs = coeffs[ 0 ];
  else
    for( i = 0; i <= degree; i++)
      rs += pow( a, i ) * coeffs[ i ];
  r1  = double ( long( real( rs ) * 1000.0 + 0.5 ) /  1000.0
    );
  r2  = double ( long( imag( rs ) * 1000.0 + 0.5 ) /  1000.0
    );
```

```
  complex ret( r1, r2 );
  return ret;
}

void factorize( const polynomial& a, double* r, double* i )
{
  int ts, rc = 0, ic = 0;
  double temp[ 2 ];
  double dd, r1, r2, i1, i2;
  //double* r = new double[ a.degree ];
  //double* i = new double[ a.degree ];

  temp[ 0 ] = a.coeffs[ 0 ];
  temp[ 1 ] = a.coeffs[ 1 ];
  polynomial rr( 1, temp );
  temp[ 1 ]++;
  polynomial pr( 1, temp );
  polynomial v( 2 );
  polynomial q( 2 );
  polynomial nul( 1 );
  polynomial d = a;

  while( d.degree > 2 )
  {
    v.coeffs[ 2 ] = 1;
    v.coeffs[ 1 ] = d.coeffs[ 1 ] / d.coeffs[ 2 ];
    v.coeffs[ 0 ] = d.coeffs[ 0 ] / d.coeffs[ 2 ];
```

```
while( pr > rr )
{
  ic++;
  v.show();
  pr = rr + nul;
  rr = d % v;
  q = d / v;
  ts = q.degree;
  q.degree = 0;
  v = v * q + rr;
  v.coeffs[ 1 ] = v.coeffs[ 1 ] / v.coeffs[ 2 ];
  v.coeffs[ 0 ] = v.coeffs[ 0 ] / v.coeffs[ 2 ];
  v.coeffs[ 2 ] = 1;
  q.degree = ts;
}
d = q + nul;
dd = v.coeffs[ 1 ] * v.coeffs[ 1 ] - 4 * v.coeffs[ 2 ] *
  v.coeffs[ 0 ];
if( dd >= 0 )
{
  r1 = ( -v.coeffs[ 1 ] + sqrt( dd ) ) / ( 2 * v.coeffs[
    2 ] );
  r2 = ( -v.coeffs[ 1 ] - sqrt( dd ) ) / ( 2 * v.coeffs[
    2 ] );
  i1 = i2 = 0;
```

```
    }
    else
    {
      dd = -dd;
      r1 = r2 = ( - v.coeffs[ 1 ] ) / ( 2 * v.coeffs[ 2 ] );
      i1 = sqrt( dd ) / ( 2 * v.coeffs[ 2 ] );
      i2 = - i1;
    }
    r[ rc ] = r1;
    i[ rc ] = i1;
    cout<<"\n"<<r[ rc ]<<" + "<<i[ rc ];
    rc++;
    r[ rc ] = r2;
    i[ rc ] = i2;
    rc++;
    cout<<"\n"<<r[ rc ]<<" + "<<i[ rc ];
    temp[ 0 ] = d.coeffs[ 0 ];
    temp[ 1 ] = d.coeffs[ 1 ];
    rr.change_values( temp );
    temp[ 1 ]++;
    pr.change_values( temp );
  }
  if( d.degree == 1 )
  {
    r[ rc ] = -d.coeffs[ 0 ] / d.coeffs[ 1 ];
```

```
  i[ rc ] = 0;
  cout<<"\n"<<r[ rc ]<<" + "<<i[ rc ];
}
else
{
  dd = d.coeffs[ 1 ] * d.coeffs[ 1 ] - 4 * d.coeffs[ 2 ] *
    d.coeffs[ 0 ];
  if( dd >= 0 )
  {
    r1 = ( -d.coeffs[ 1 ] + sqrt( dd ) ) / ( 2 * d.coeffs[
      2 ] );
    r2 = ( -d.coeffs[ 1 ] - sqrt( dd ) ) / ( 2 * d.coeffs[
      2 ] );
    i1 = i2 = 0;
  }
  else
  {
    dd = -dd;
    r1 = r2 = ( - d.coeffs[ 1 ] ) / ( 2 * d.coeffs[ 2 ] );
    i1 = sqrt( dd ) / ( 2 * d.coeffs[ 2 ] );
    i2 = -i1;
  }
  r[ rc ] = r1;
  i[ rc ] = i1;
  cout<<"\n"<<r[ rc ]<<" + "<<i[ rc ];
```

```
    rc++;
    r[ rc ] = r2;
    i[ rc ] = i2;
    cout<<"\n"<<r[ rc ]<<" + "<<i[ rc ];
  }
  int l;
  for(l = 2;l>=0 ; l--)
    cout<<r[l]<<";"<<i[l];

}

int operator > ( const polynomial& a, const polynomial& b)
{
  if( a.coeffs[ 1 ] == b.coeffs[ 1 ] )
    return ( a.coeffs[ 0 ] > b.coeffs[ 0 ] ) ? 1 : 0;
  return ( a.coeffs[ 1 ] > b.coeffs[ 1 ] ) ? 1 : 0;
}

void partial_fract( const polynomial& nu, const  polynomial&
  de, double* cr, double* ci)
{
  int i,j;
  double* rr;
  double* ri;
  rr = new double[ de.degree ];
  ri = new double[ de.degree ];
```

```
factorize( de, rr, ri );
for(i = de.degree-1;i>=0 ; i--)
   cout<<rr[i]<<":"<<ri[i];
//double rr[] = { -2, -2, -1 };
//double ri[] = {  1,  1,  0 };
cr = new double[ de.degree ];
ci = new double[ de.degree ];
complex ac( 1 );
//complex nl( 0, 0 );
//complex* rs = new complex[ de.degree ];
//complex retval();
//factorize( de );

for( i = 0; i < de.degree; i++)
{
   ac = ac - ac + 1;
   for( j = 0; j < de.degree; j++)
      ac *= de.addi( rr[i], ri[i], -rr[j], -ri[j] );
   //cout<<"\n"<<ac;
   ac = nu.evaluvate( rr[i], ri[i] ) / ac;
   //cout<<ac;
   cr[i]=real(ac);
   ci[i]=imag(ac);
   cout<<"\n"<<cr[i]<<"   "<<ci[i];
}
clrscr();
```

```
  cout<<"System matrix\n\n";
  for( i = de.degree - 1; i >= 0; i -- )
  {
    for( j = de.degree - 1; j >= 0; j --
    {
      if( i == j )
        printf("%5.2lf:%5.2lf",rr[i],ri[i]);
      else
        printf("       0       ");
      printf("       ");
    }
    cout<<"\n";
  }
  //getch();
  cout<<"\n\nOutput matrix\n\n";
  for( i = de.degree - 1; i >= 0; i -- )
    printf("%5.2lf:%5.2lf\n",cr[i],ci[i]);
}

void polynomial::read()
{
  int i;
  unsigned deg;

  clrscr();
  cout<<"Enter the degree : ";
```

```
    scanf("%d",&deg);
    degree = deg;
    delete coeffs;
    coeffs = new double[ degree + 1 ];
    cout<<"\n\n";
    for( i = degree; i >= 0; i-- )
    {
      cout<<" Enter the coefficient of the "<<i<<" degree : ";
      scanf("%f",coeffs[i]);
      cout<<"\n";
    }
}

void phase_model( const polynomial& n, const polynomial& d )
{
   int i, j;
   clrscr();
   cout<<"System matrix\n\n";
   for( i = n.degree - 1; i >= 0; i-- )
   {
      for( j = n.degree - 1; j >= 0; j-- )
      {
        if( j == n.degree - i )
          printf("%5.2lf", -d.coeffs[i] );
        else if( i - j == 1 )
```

```
        printf("  1  ");
      else
        printf("  0  ");
      printf("    ");
    }
    cout<<"\n";
  }
  //getch();
  cout<<"\n\nInput matrix\n\n";
  for( i = n.degree - 1; i >= 0; i-- )
    cout<<n.coeffs[i]
d.coeffs[i]*n.coeffs[n.degree]<<"\n";
  }
```

5.3 SAMPLE OUTPUT

```
Enter the degree : 4
Enter the coefficient of degree 4 : 0
Enter the coefficient of degree 3 : 0
Enter the coefficient of degree 2 : 2
Enter the coefficient of degree 1 : 6
Enter the coefficient of degree 0 : 2
```

```
Enter the degree : 4
Enter the coefficient of degree 4 : 1
Enter the coefficient of degree 3 : 9
Enter the coefficient of degree 2 : 30
Enter the coefficient of degree 1 : 42
Enter the coefficient of degree 0 : 20
```

System matrix

-9.00	1	0	0
-30.00	0	1	0
-42.00	0	0	1
-20.00	0	0	0

Input matrix

0
2
6
2

PHASE MARGIN

System matrix

-3.00: -1.00	0	0	0
0	-3.00: 1.00	0	0
0	0	-2.00: 0.00	0
0	0	0	-1.00: 0.00

Output matrix

-0.30: 0.90
-0.30: -0.00
1.00: 0.00
-0.40: -0.00

CANONICAL MODEL

```
Enter the degree : 5
Enter the coefficient of degree 5 : 0
Enter the coefficient of degree 4 : 15
Enter the coefficient of degree 3 : 170
Enter the coefficient of degree 2 : 675
Enter the coefficient of degree 1 : 1098
Enter the coefficient of degree 0 : 630
```

```
Enter the degree : 5
Enter the coefficient of degree 5 : 1
Enter the coefficient of degree 4 : 15
Enter the coefficient of degree 3 : 85
Enter the coefficient of degree 2 : 225
Enter the coefficient of degree 1 : 274
Enter the coefficient of degree 0 : 120
```

System matrix

-15.00	1	0	0	0
-85.00	0	1	0	0
-225.00	0	0	1	0
-274.00	0	0	0	1
-120.00	0	0	0	0

Input matrix

1
170
675
1000
600

PHASE MODEL

System matrix

-5,00: 0,00	0	0	0	0
0	4,00: 0,00	0	0	0
0	0	-3,00: 0,00	0	0
0	0	0	-2,00: 0,00	0
0	0	0	0	-1,00: 0,00

Output matrix

5,00: 0,00
-5,00: 0,00
4,00: 0,00
-4,00: 0,00
-3,00: 0,00

CANONICAL MODEL

CONCLUSION

A software has been developed in C++ to generate two different types of state models for linear systems. A SISO systems was considered for the development of the software. Systems upto 5th order have been tested successfully. However it can be extended to systems of higher orders.

There are a few limitations as far as this software is concerned. The computational time required increases considerably with increase in the order of the system.

This program does not work efficiently when the poles of the system are repetitive in nature. This problem can be over come by using a different algorithm for factorization.

Another limitation is that only SISO systems can be analysed. Further improvements in this sofrware can be made to make it handle MIMO systems.

This software is quite useful for control system engineers.

REFERENCES

1. Kuo, B.C., "Automatic Control Systems", Prentice-Hall, Englewood Cliffs, N.J., 1975.

2. Ogata, K., "Modern Control Engineering", Prentice-Hall, Englewood Cliffs, N.J., 1970.

3. Nagrath, I.J. and Gopal, M., "Control Systems Engineering", Wiley Eastern Limited, New Delhi, 1981.

4. DeRusso,M.P., R.J.Roj and C.M.Close, "State Variables for Engineers", John Wiley, New York, 1965.

5. Schultz,D.M. and J.L.Melsa, "State Functions and Linear Control Systems", McGraw-Hill, New York, 1967.

6. Dorf,R.C., "Time Domain Analysis and Design of Control Systems", Addison-Wesley, Reading, Mass., 1965.

7. Stephen Prata, "The Waite Group's C++ Primer Plus", Waite Group Press, California, 1991.

8. Neil Graham, "Learning C++", McGraw Hill, Singapore, 1991.