C#编程与.Net 框架课程作业报告

基于 C#的控制系统仿真设计工具包（类 MATLAB）

类型：C# .net core 控制台应用程序

姓名：梁子

学号：20163933

班级：自动化 1609 班

学院：信息科学与工程学院

日期：2019 年 6 月 22 日

摘要

本次课程作业的设计内容为《C#控制系统工具包》。该包基于 C# 的.Net Core，可在 Windows 系统、Linux 系统和 Mac 系统上跨平台部署和使用。在使用该安装包前需要安装 dotnet 核心中的基本数学库 MathNet，以提供基本的线性代数运算。

由于时间问题，本工具包目前仅仅封装了基本控制系统工具(C#代码量约 1500 行)，包括：

1）基本的控制系统模型建立。包括以传递函数矩阵（包含传递函数）为基础的经典控制理论模型和以状态空间矩阵方程为基础的现代控制理论模型，并提供了二者的非双射转换；

2）基本的控制系统分析。包括稳定性分析、极点获取（即传函的零点获取）、能控性分析、能观测行分析、时滞、连续、采样周期等；

3）基本的控制系统搭建和组合。针对线性系统模型的串联连接、并联连接、反馈连接等基本连接的传递函数矩阵模型实现和状态空间模型实现。

除此之外，工具包中还包含对类似 MATLAB 中 SIMULINK 控制系统仿真功能是实现的设计准类。包含对状态空间模型的可逆变换的封装设计、动态反馈的封装设计。

本工具包将来会增添以下功能：

1. 增加基于最小二乘法的系统辨识功能，支持导入基本格式的数据进行辨识；
2. 增加绘图。绘制阶跃响应、绘制波德图、绘制奈氏图、绘制根轨迹等基本图像；
3. 实现图形界面的 Simulink 功能。

本报告将根据上述实现功能展开，由于是课程报告，所以重点描述如何使用课堂讲述的语法知识（尤其是面向对象）实现整个工具包的设计，对算法部分不进行重点的阐述。

目录

C#编程与.Net 框架课程作业报告 1
摘要 2
1 工具包概析 4
1.1 模块划分和结构概述 4
1.1.1 文件结构 4
1.1.2 命名空间包含关系 5
1.2 利用抽象类 ControlModel 实现对控制系统基本属性功能封装 5
1.2.1 使用 abstract 修饰的抽象方法 5
1.2.2 使用 virtual 修饰的虚方法 7
1.3 使用接口 ISimu 实现对 Simulink 仿真模块的基本约束 7
2 经典控制和现代控制 8
2.1 使用 override：派生类 TransferFunction 和 StatementSpace 的具体化和实现 8
2.1.1 传递函数模型 8
2.1.2 状态空间模型 10
2.2 自实现和调用现有库，MathNet 和 Array 类型 15
2.2.1 使用数组来描述张量（多边矩阵） 15
2.2.2 使用已有函数库 MathNet.Numerics.LinearAlgebra 16
2.3 运算符重载：实现控制系统模型的加、减和等于 18
2.4 自定义异常处理命名空间 ExceptionSelf 进行常见异常处理 21
3 如何使用 C#写一个简化版 SIMULINK？ 21
3.1 继承接口的子模块定义和实现 22
3.2 一个模块应该怎么组织他的子模块 23
3.3 使用 XML 格式进行 Simulink 模型的保存和导入 24
3.4 泛型与否，论 ArrayList 的优与劣 25
附录及参考文献 25
工具包未来情况说明 25
参考文献和网站 25

1 工具包概析

1.1 模块划分和结构概述

1.1.1 文件结构

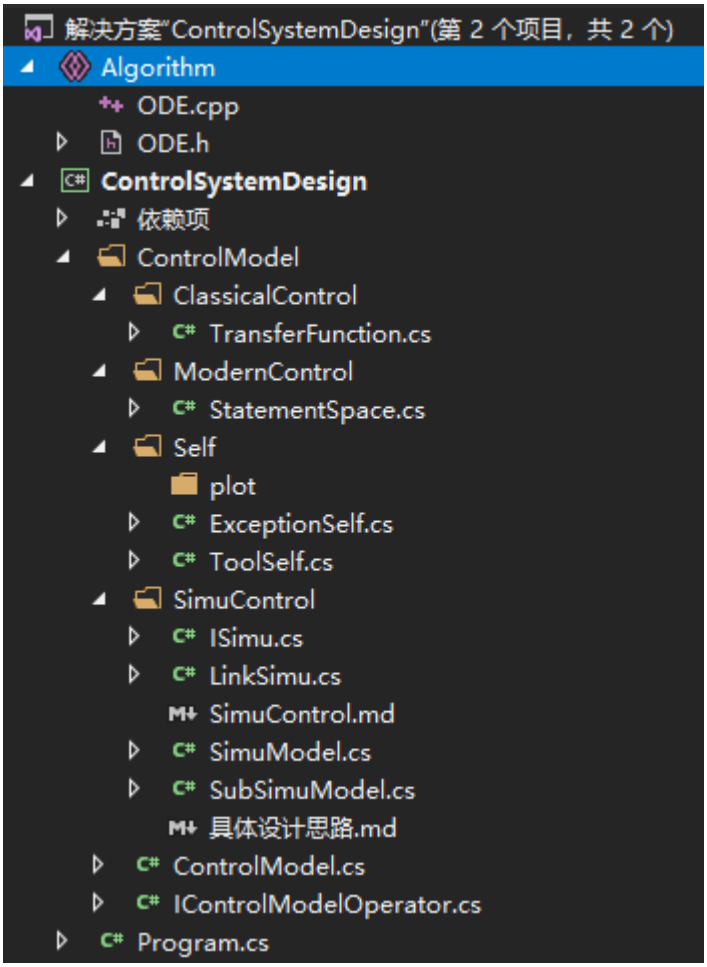

图 1.1

图 1.1 所示为目前工具包设置的整体架构，可见所有的控制相关封装都在 ControlModel 主文件下，该文件夹包括抽象类 ControlModel 来对一个控制系统模型的基本属性进行封装，该类被 ClassicalControl 文件夹下的 TransferFunction 类和 ModernControl 文件夹下的 StatementSpace 类进行继承实现，二者分别构造了传递函数矩阵模型类和状态空间模型类。

除此之外，该文件夹下还有 Self 文件夹，里面用于定义工具包自身使用用于构建自身算法或完善功能的工具，包括：plot 文件夹下的绘图工具（暂无），ExceptionSelf 文件所代表的控制系统常见异常的命名空间，以及 TooSelf 类内的自定义工具（如矩阵增广等功能封装的静态方法。

此处之外，学习 MATLAB 中的 SIMULINK 功能，SimuControl 文件夹下提供了

相关的实现。其中 SimuModel 类提供了使用对整个 Simulink 模型的封装，SubSimuModel 类提供了对每一个模块的封装和自定义。LinkSimu 类提供了对子模块之间相互连接的描述。接口 ISimu 提供了对模块和子模块应该具有的输入输出接口的描述。通过这些类和方法，就可以逐步实现最终完成类似 SIMULINK 功能的设计。

Program 文件是作为检验的一个主文件，里面提供了基本的 Demo 演示，尽管 Debug 使用，在实际发行时应该删去。

1.1.2 命名空间包含关系

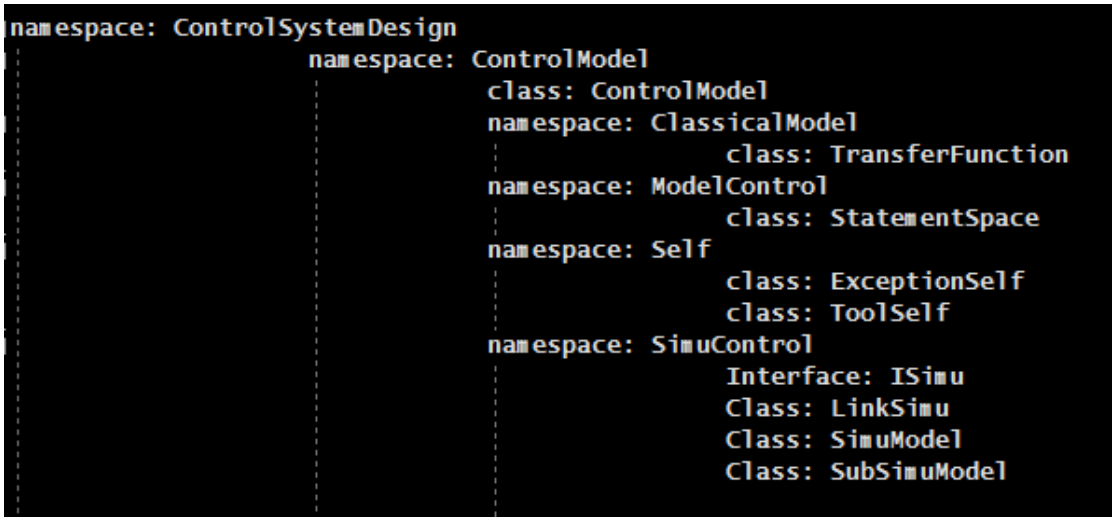

图 1.2

如图 1.2 所示。

1.2 利用抽象类 ControlModel 实现对控制系统基本属性功能封装

为了对传统控制理论模型（传递函数矩阵模型）和现代控制理论模型（状态空间模型）进行统一的描述，抓住他们在控制系统描述上的共性，本工具包定义了 ControlModel 抽象类。

1.2.1 使用 abstract 修饰的抽象方法

```
/// <summary>
        /// 判断系统是否稳定的函数，必须被描述
        /// </summary>
        /// <returns>
        /// 布尔变量，描述是否稳定
```

```
        /// </returns>
        public abstract bool IsStable();

        /// <summary>
        /// 获取系统的所有极点，必须被描述
        /// </summary>
        /// <returns></returns>
        public abstract Vector<MathNet.Numerics.Complex32> GetPole();

        /// <summary>
        /// 返回系统的时滞系数，必须被描述
        /// </summary>
        /// <returns></returns>
        public abstract double GetTimeDelay();

        /// <summary>
        /// 判断系统连续或离散，必须被描述
        /// </summary>
        /// <returns></returns>
        public abstract bool IsContinous();

        /// <summary>
        /// 获取系统的采样时间，如果是连续系统，返回值为0
        /// </summary>
        /// <returns></returns>
        public abstract double GetTs();
/// <summary>
        /// 方法：实现将两个系统进行并联，返回并联之后的系统
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public abstract ControlModel ParallelConnection(ControlModel a, ControlModel b);

        /// <summary>
        /// 方法：实现将两个系统进行串联，返回串联之后的系统
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <returns></returns>
        public abstract ControlModel SeriesConnectioin(ControlModel a, ControlModel b);

        /// <summary>
        /// 方法：实现将两个系统进行反馈连接，返回之后得到的系统
```

```
        /// 其中：a为正向通道系统模型，b为反向通道系统模型，isPositive检测是否为正反馈
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <param name="isPositive"></param>
        /// <returns></returns>
        public abstract ControlModel FeedbackConnection(ControlModel a, ControlModel b,
bool isPositive);
```

1.2.2 使用 virtual 修饰的虚方法

打印函数使用 virtual 修饰。

```
        /// <summary>
        /// 打印系统模型信息
        /// </summary>
        public virtual void PrintModel()
        {
            Console.WriteLine("这是一个控制系统模型！");
        }
```

1.3 使用接口 ISimu 实现对 Simulink 仿真模块的基本约束

主要以下方法：
1）获取目标信息；
2）获取目标值；
3）获取目标模型标签；
4）获取输入输出标签；

```
interface ISimu
    {
        /// <summary>
        /// 返回目标输入输出的信息
        /// </summary>
        /// <returns></returns>
        string GetInfo();

        /// <summary>
        /// 得到目标地方的值
        /// </summary>
        /// <returns></returns>
        ArrayList GetValue();
```

```
        /// <summary>
        /// 返回对应的模型
        /// </summary>
        /// <returns></returns>
        SubSimuModel GetModel();

        /// <summary>
        /// 返回每个子模块独一无二的tag
        /// </summary>
        /// <returns></returns>
        string GetMainTag();

        /// <summary>
        /// 获取每个模块中输入输出对应的接口标签名字，在同一个模块中输出输入的标签名字都是确定的
        /// </summary>
        /// <returns></returns>
        ArrayList GetSubTags();

    }
```

2 经典控制和现代控制

2.1 使用 override：派生类 TransferFunction 和 StatementSpace 的具体化和实现

2.1.1 传递函数模型

根据控制理论知识，传递函数模型应该具有的私有字段如图 2.1 所示。

```
private double[,,] Den;//分母
private double[,,] Num;//分子
private double[,] TimeDelay;
private double Ts;
```

图 2.1 私有字段图

可以看出，我们使用一个三维张量来存储传递函数模型的分母分子系数，其中第一个维度代表传递函数矩阵的行的索引，第二个维度代表传递函数矩阵中的列的索引，第三个维度代表该位置传函的分子分母系数组成的向量。除此之外，还有时间延迟（对于传递函数矩阵中的每一个元素都应有一个时滞系数）以及采

样周期。

2.1.1.1 构造函数重载

```
/// <summary>
/// 根据传递函数矩阵中各个元素分子分母从高阶到低阶排列的顺序向量 时滞 采样周期 来确定传递函数矩阵的表达式
/// </summary>
/// <param name="den"></param>
/// <param name="num"></param>
/// <param name="timeDelay"></param>
/// <param name="ts"></param>
1 reference
public TransferFunction(double[,,]den,double[,,]num,double[,]timeDelay,double ts)
{
    for (int i = 0; i < den.GetLength(0); i++)
        for (int j = 0; j < den.GetLength(1); j++)
            for (int k = 0; k < den.GetLength(2); k++)
            {
                Den[i,j,k] = den[i,j,k];
                Num[i,j,k] = num[i,j,k];
            }
    for(int i=0;i<timeDelay.GetLength(0);i++)
        for(int j=0;j<timeDelay.GetLength(1);j++)
        {
            TimeDelay[i,j] = timeDelay[i,j];
        }
    Ts = ts;
}
```

```
/// <summary> 默认构造出来的传递函数矩阵为1*1的单位传递函数，传函为1，即不做任何变换．属于连续系统
2 references
public TransferFunction()...

/// <summary> 根据传递函数矩阵中各个元素分子分母从高阶到低阶排列的顺序向量来确定传递函数矩阵的表达式
0 references
public TransferFunction(double[,,] den,double[,,] num)...

/// <summary>
/// 根据传递函数矩阵中各个元素分子分母从高阶到低阶排列的顺序向量 时滞 采样周期 来确定传递函数矩阵的表达
/// </summary>
/// <param name="den"></param>
/// <param name="num"></param>
/// <param name="timeDelay"></param>
/// <param name="ts"></param>
1 reference
public TransferFunction(double[,,]den,double[,,]num,double[,]timeDelay,double ts)
{
```

图 2.2 三种不同的构造函数剪影

如图 2.2 所示，针对用户输入了不同的数据做出相应的初始化手段，因而需要采用构造函数的重载。

2.1.1.2 抽象函数的实现

由于 Transfer Function 类实现了对 ControlModel 抽象类的继承，所以必须要覆盖它所有的抽象方法，此处给出中不同类型的抽象方法的实现剪影（如图 2.3 所示），其他见源代码。

```
/// <summary>
/// 判断系统是否稳定的函数，必须被描述
/// </summary>
/// <returns></returns>
2 references
public override bool IsStable()
{
    var poles = GetPole();
    foreach(var pole in poles)
    {
        if (pole.Real >= 0)//如果实部大于0，则系统不稳定
            return false;
    }
    //如果所有极点都小于0
    return true;
}
```

```
public override ControlModel ParallelConnection(ControlModel aa, ControlModel bb)
{
    var a = (TransferFunction)aa;//拆箱操作
    var b = (TransferFunction)bb;
    int highestOrder = Math.Max(a.Den.GetLength(2), b.Den.GetLength(2));
    //本质上就是一个输入可以进入到两个子模型的输入中，两个子模型的输出为总的输出，即行不变，列相加
    double[,,] den=new double[a.Den.GetLength(0),a.Den.GetLength(1)+b.Den.GetLength(1),highestOrder];
    double[,,] num=new double[a.Den.GetLength(0), a.Den.GetLength(1) + b.Den.GetLength(1), highestOrder];
    double[,] TimeDelay = new double[a.Den.GetLength(0), a.Den.GetLength(1) + b.Den.GetLength(1)];

    for (int i=0;i<a.Den.GetLength(0);i++)
        for (int j = 0; j < a.Den.GetLength(1) + b.Den.GetLength(1); j++)
            for (int k = 0; k < Math.Max(a.Den.GetLength(2),b.Den.GetLength(2)); k++)
            {

                if(j<a.Den.GetLength(1))
                {
                    //if (a.Den.GetLength(2) == highestOrder)
                    //    den[i, j, k] = a.Den[i, j, k];
                    //else
                    den[i, j, k] = a.Den[i, j, k];//哈哈，后来换了一种想法

                    num[i, j, k] = a.Num[i, j, k];
                    TimeDelay[i, j] = a.TimeDelay[i, j];
                }
                else
                {
                    den[i,j,k]=b.Den[i, j - a.Den.GetLength(1), k];
                    num[i, j, k] = b.Den[i, j - a.Den.GetLength(1), k];
```

图 2.3 override 函数操作

2.1.2 状态空间模型

根据状态空间模型知识，一个状态空间模型应该具有图 2.4 所示的私有字段。

```
///该系统的ss模型为:
///diff(x)=A*x+B*u;
///y=C*x+D*u;
private Matrix<double> A;
private Matrix<double> B;
private Matrix<double> C;
private Matrix<double> D;
private double Ts; //采样时间
private double TimeDelay;//，这一部分代码暂时没有设计
```

图 2.4 状态空间模型私有字段

其中 A 是系统矩阵，B 是输入矩阵（控制矩阵），C 是输出矩阵，D 是直接传递矩阵。尽管里面包含时滞变量，但是此时的系统并不具有时滞特点（这部分知识暂时还没掌握，理论上应该是一个向量类型）。

2.1.2.1 构造函数重载

图 2.5 表示的是状态空间模型类下的所有构造函数的剪影。

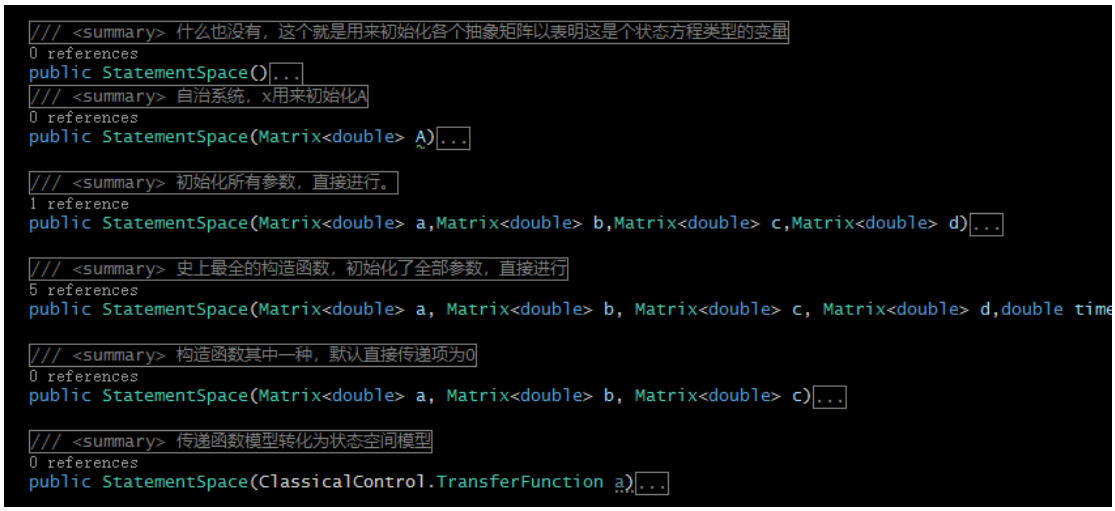

图 2.5 构造函数示例

其中，第四个构造函数的表述为：

```
/// <summary>
        /// 史上最全的构造函数，初始化了全部参数，直接进行
        /// </summary>
        /// <param name="a"></param>
        /// <param name="b"></param>
        /// <param name="c"></param>
        /// <param name="d"></param>
        /// <param name="timeDelay"></param>
        /// <param name="ts"></param>
        public StatementSpace(Matrix<double> a, Matrix<double> b, Matrix<double> c,
Matrix<double> d,double timeDelay,double ts)
        {
            A = a;
            B = b;
            C = c;
            D = d;

            if (A.RowCount != B.RowCount)
            {
                Console.WriteLine("错误。矩阵尺度不一致。A矩阵行数{0}不等于B矩阵行数{1}",
A.RowCount, B.RowCount);
```

```
                throw new Self.ExceptionSelf.MatrixNotTheSameRowCol("矩阵尺度不一致");
            }
            if (A.ColumnCount != C.ColumnCount)
            {
                Console.WriteLine("错误。矩阵尺度不一致。A矩阵列数{0}不等于B矩阵列数{1}",
A.ColumnCount, B.ColumnCount);
                throw new Self.ExceptionSelf.MatrixNotTheSameRowCol("矩阵尺度不一致");
            }
            if (D.RowCount != C.RowCount)
            {
                Console.WriteLine("错误。矩阵尺度不一致。D矩阵行数{0}不等于C矩阵行数{1}",
D.RowCount, C.RowCount);
                throw new Self.ExceptionSelf.MatrixNotTheSameRowCol("矩阵尺度不一致");
            }
            if (D.ColumnCount != B.ColumnCount)
            {
                Console.WriteLine("错误。矩阵尺度不一致。D矩阵列数{0}不等于B矩阵列数{1}",
D.ColumnCount, B.ColumnCount);
                throw new Self.ExceptionSelf.MatrixNotTheSameRowCol("矩阵尺度不一致");
            }
            TimeDelay = timeDelay;
            Ts = ts;

        }
```

2.1.2.2 抽象函数和虚函数的覆盖重写

在介绍抽象类 ControlModel 时，曾经在里面定义了一系列的抽象方法和虚方法，由于本类继承该类，所以必须实现所有的抽象方法并选择性 override 虚方法。此处给出实现两个模型反馈连接的示例进行解释。

```
public override ControlModel FeedbackConnection(ControlModel aa, ControlModel bb, bool
isPositive)
        {
            var a = (StatementSpace)aa;
            var b = (StatementSpace)bb;//拆箱操作

            int i = isPositive ? (-1) : 1;//正反馈则为-1,负反馈则为1

            ///此处还欠缺两个模型的尺度检查这个问题。

            if(!isPositive)
            {
                //计算出来合适的分块矩阵形式
```

```
                var A11 = a.A - a.B *
                    ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) + b.D *
a.D).Inverse()) * b.D * a.C;
                var A12 = -a.B * ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) +
b.D * a.D).Inverse()) * b.C;
                var A21 = b.B * a.C- b.B * a.D *
                    ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) + b.D *
a.D).Inverse()) * b.D * a.C;
                var A22 = b.A - b.B * a.D *
                    ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) + b.D *
a.D).Inverse()) * b.C;

                var B1 = a.B * ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) + b.D
* a.D).Inverse());
                var B2 = b.B * a.D * ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount)
+ b.D * a.D).Inverse());

                var C1 = a.C - a.D * ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount)
+ b.D * a.D).Inverse())
                    * b.D * a.C;
                var C2 = -a.D * ((Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) +
b.D * a.D).Inverse()) * b.C;

                //计算出ABCD四个矩阵
                var A = Matrix<double>.Build.Dense(A11.RowCount + A21.RowCount,
A11.ColumnCount + A12.ColumnCount, 0);
                var B = Matrix<double>.Build.Dense(B1.RowCount + B2.RowCount,
B1.ColumnCount,0.0);
                var C = Matrix<double>.Build.Dense(C1.RowCount, C1.ColumnCount +
C2.ColumnCount,0.0);
                var D = Matrix<double>.Build.Dense(a.D.RowCount, a.D.ColumnCount,0.0);

                //分块矩阵赋值，得到系统矩阵A
                for(i=0;i<A.RowCount; i++)
                    for(int j=0;j<A.ColumnCount;j++)
                    {
                        if(i<A11.RowCount)
                        {
                            if(j<A11.ColumnCount)
                            {
                                A[i, j] = A11[i, j];
                            }
                            else
                            {
```

```
                A[i, j] = A12[i, j - A11.ColumnCount];
            }
        }
        else
        {
            if (j < A11.ColumnCount)
            {
                A[i, j] = A21[i - A11.RowCount, j];
            }
            else
            {
                A[i, j] = A22[i - A11.RowCount, j - A11.ColumnCount];
            }
        }
    }
//分块矩阵赋值操作，得到输入（控制）矩阵B
for(i=0;i<B.RowCount;i++)
    for(int j=0;j<B.ColumnCount;j++)
    {
        if(i<B1.RowCount)
        {
            B[i, j] = B1[i, j];
        }
        else
        {
            B[i, j] = B2[i - B1.RowCount, j];
        }
    }

//分块矩阵赋值操作，得到输出矩阵C
for(i=0;i<C.RowCount;i++)
    for(int j=0;j<C.ColumnCount;j++)
    {
        if(j<C1.ColumnCount)
        {
            C[i, j] = C1[i, j];
        }
        else
        {
            C[i, j] = C2[i, j - C1.ColumnCount];

        }
    }
```

```
                D = a.D * (Matrix<double>.Build.DiagonalIdentity(b.D.RowCount) + b.D *
a.D).Inverse();

                return new StatementSpace(A, B, C, D, a.TimeDelay, a.Ts);
            }
            else//正反馈，雷同负反馈，此处不再占用空间
            {
                ;
            }
        }
```

2.2 自实现和调用现有库，MathNet 和 Array 类型

为了将所学知识尽可能使用，在进行线性代数运算时，库中给出了两种不同的实现方法，并分别在传递函数模型和状态空间模型种使用了上述两种方法，并给出了转化的枢纽。

其中，在传递函数模型中，最基本的“数组”被用来作为参数存储和交互的数据结构，即用数组实现简单而基本的数据运算属性。而在状态空间模型种，则直接采用泛型化的 MathNet.Numerics.LinearAlgebra.Matrix<T>类型。

2.2.1 使用数组来描述张量（多边矩阵）

采用多维数组来描述张量是图像处理中的通识，而这种实现放在此处也非常合理。如何描述一个线性的传递函数矩阵模型？无论是使用多项式描述、增益-时间常数描述还是零极点描述，都可以理解为三层：首先需要在传递函数矩阵中确定某一个传递函数，这即需要行和列的索引，除此之外，还需要确定该传递函数模型中每个子分量在该特征里的索引（例如多项式中的分子分母多项式中每一项的系数，或者因式分解后的时间常数组成的向量中的某一项，或者零极点数组中的某一个等），所以即为一个三维张量。关于其基础操作的一个举例如图 2.6 所示。

```
for (int i=0;i<a.Den.GetLength(0);i++)
    for (int j = 0; j < a.Den.GetLength(1) + b.Den.GetLength(1); j++)
        for (int k = 0; k < Math.Max(a.Den.GetLength(2),b.Den.GetLength(2)); k++)
        {

            if(j<a.Den.GetLength(1))
            {
                //if (a.Den.GetLength(2) == highestOrder)
                //    den[i, j, k] = a.Den[i, j, k];
                //else
                den[i, j, k] = a.Den[i, j, k];//哈哈，后来换了一种想法

                num[i, j, k] = a.Num[i, j, k];
                TimeDelay[i, j] = a.TimeDelay[i, j];
            }
            else
            {
                den[i,j,k]=b.Den[i, j - a.Den.GetLength(1), k];
                num[i, j, k] = b.Den[i, j - a.Den.GetLength(1), k];
                TimeDelay[i, j] = b.TimeDelay[i, j - a.Den.GetLength(1)];
            }
```

图 2.6 基于数组的传递函数矩阵串联操作

2.2.2 使用已有函数库 MathNet.Numerics.LinearAlgebra

本着不要重复造轮子的理念，本文作者查询了 C#中已有的官方数学库，最终发现并使用了 MathNet 中的线性代数运算库 MathNet.Numerics.LinearAlgebra。根据笔者在各大语言上使用矩阵运算的体验，该库并不算好用（譬如：不支持切片操作!），算法也封装不够全面（譬如：求取特征值的方法都没有!），但是官方库给出了基本的线性运算雏形，当然仍然需要很大的努力（比如目前只支持向量和矩阵，不支持张量运算的问题）去完善。此处可以给出一个不官方的个人体验的排序： Matlab>numpy(python 库)>Eigen3(C++库)>BLAS(C&C++库)>MathNet.Numerics.LinearAlgebra(C#库)。

如果在 C#中使用 MathNet，需要配置相关环境。方法有二：一是引入动态链接库 MathNet.Numerics.dll ， 第二种方法是使用 Install-Package MathNet.Numerics 指令在控制台（比如 PowerShell）安装。

官方库的优点是对底层的运算进行了足够的优化，当矩阵的维度较高时，其计算所需时间会小很多。当然由于该库语法糖不足的问题，缺少的切片操作导致需要循环赋值（而这又是非常耗费时间的），所以效率上可能并不会提升太高。不过，对于控制系统而言，几百个状态变量的模型基本是不存在的（即使是航空航天这种多状态的过程也难以存在，更别说日常控制），所以本工具包对于绝大多数过程控制和时间要求不过于高的运动控制仍然适用。

事实上，关于该库的基本操作再之前贴过的代码里已经有展示了。下面给出一个自己实现的矩阵增广的功能（该功能在其他语言的线性函数库里都是内置的）。

```
/// <summary>
        /// 自己写的小工具，用来实现增广矩阵的功能，此处的函数只支持double型数据
        /// axis==0为横向增广，axis==1为纵向增广
        /// </summary>
        /// <param name="matrix1"></param>
        /// <param name="matrix2"></param>
        /// <param name="axis"></param>
        /// <returns></returns>
        public static Matrix<double> MatrixZengGuang(Matrix<double> matrix1,
Matrix<double> matrix2, int axis)
        {
            if (axis == 0)
            {
                Matrix<double> matrix = Matrix<double>.Build.Dense(matrix1.RowCount,
matrix1.ColumnCount + matrix2.ColumnCount);
                for (int i = 0; i < matrix1.RowCount; i++)
                    for (int j = 0; j < matrix1.ColumnCount + matrix2.ColumnCount; j++)
                    {
                        if (j < matrix1.ColumnCount)
                            matrix[i, j] = matrix1[i, j];
                        else
                            matrix[i, j] = matrix2[i, j - matrix1.ColumnCount];
```

```
                }
            return matrix;
        }
        else if (axis == 1)
        {
            Matrix<double> matrix = Matrix<double>.Build.Dense(matrix1.RowCount +
matrix2.RowCount, matrix1.ColumnCount);
            for (int i = 0; i < matrix1.RowCount + matrix2.RowCount; i++)
                for (int j = 0; j < matrix1.ColumnCount; j++)
                {
                    if (i < matrix1.RowCount)
                        matrix[i, j] = matrix1[i, j];
                    else
                        matrix[i, j] = matrix2[i - matrix1.RowCount, j];
                }
            return matrix;
        }
        else
        {
            Console.WriteLine("错误的输入尺度。矩阵只能在行或者列的方向上进行增广");
            //报出异常
            throw new ExceptionSelf.WrongInputArgument("输入尺度错误");
            return Matrix<double>.Build.Dense(1, 1, 0);
        }
    }
```

此外，该库的泛型是一个略显鸡肋的操作。比如下面的一个实现任意类型矩阵增广的函数，由于类型非空的要求，导致该函数无法实现。而类似操作在 C++ 确是可以的。

```
    /// <summary>
    /// 实现矩阵的增广功能的函数，返回一个增光的数值 C#无法实现泛型编程，所以这里就只能写一个自己用到的但是通用意义不算特别强的
    /// 这里面的axis是进行增广的尺度，如果为0，表示横向增广，如果为1，表示纵向增广。
    ///
    /// </summary>
    private static Matrix<T> MatrixZengGuang<T>(Matrix<T> matrix1, Matrix<T> matrix2,
int axis)
    {
        if (axis == 0)
        {
            Matrix<T> matrix = Matrix<T>.Build.Dense(matrix1.RowCount,
matrix1.ColumnCount + matrix2.ColumnCount);
            for (int i = 0; i < matrix1.RowCount; i++)
                for (int j = 0; j < matrix1.ColumnCount + matrix2.ColumnCount; j++)
                {
```

```
                    if (j < matrix1.ColumnCount)
                        matrix(i, j) = matrix1(i, j);
                    else
                        matrix(i, j) = matrix2(i, j - matrix1.ColumnCount);
                }
            return matrix;
        }
        else if (axis == 1)
        {
            Matrix<T> matrix = Matrix<T>.Build.Dense(matrix1.RowCount +
matrix2.RowCount, matrix1.ColumnCount);
            for (int i = 0; i < matrix1.RowCount + matrix2.RowCount; i++)
                for (int j = 0; j < matrix1.ColumnCount; j++)
                {
                    if (i < matrix1.RowCount)
                        matrix(i, j) = matrix1(i, j);
                    else
                        matrix(i, j) = matrix2(i - matrix1.RowCount, j);
                }
            return matrix;
        }
        else
        {
            Console.WriteLine("错误的输入尺度。矩阵只能在行或者列的方向上进行增广");
            //报出异常

        }
    }
```

2.3 运算符重载：实现控制系统模型的加、减和等于

为了简化用户使用，工具包为运算符在模型之间的运算定义了一定的规则。这种定义是通过运算符重载实现的。遗憾的是，运算符重载无法在抽象类（没有实现就没有重载？）中被定义，因而分别需要在状态空间模型和传递函数模型中定义。此处给出加号在控制系统模型连接中的定义（并联）。

```
        /// <summary>
        /// 运算符重载，实现两个模型并联的操作
        /// </summary>
        /// <param name="aa"></param>
        /// <param name="BB"></param>
        /// <returns></returns>
        public static  StatementSpace operator + (StatementSpace a,StatementSpace b)
        {
            //初始化
```

```
            var A = Matrix<double>.Build.Dense(a.A.RowCount + b.A.RowCount, a.A.RowCount
+ b.A.RowCount, 0);
            var B = Matrix<double>.Build.Dense(a.A.RowCount + b.A.RowCount,
a.B.ColumnCount, 0);
            var C = Matrix<double>.Build.Dense(a.C.RowCount, a.A.RowCount + b.A.RowCount,
0);
            var D = Matrix<double>.Build.Dense(a.C.RowCount, a.C.ColumnCount);

            //赋值操作
            for (int i = 0; i < A.RowCount; i++)
                for (int j = 0; j < A.ColumnCount; j++)
                {
                    if (i < a.A.RowCount && j < a.A.ColumnCount)
                    {
                        A[i, j] = a.A[i, j];
                    }
                    else if (i > a.A.RowCount && j > a.A.ColumnCount)
                    {
                        A[i, j] = b.A[i - a.A.RowCount, j - a.A.ColumnCount];

                    }
                }

            for (int i = 0; i < B.RowCount; i++)
                for (int j = 0; j < B.ColumnCount; j++)
                {
                    if (i < a.B.RowCount)
                    {
                        B[i, j] = a.B[i, j];
                    }
                    else
                    {
                        B[i, j] = b.B[i - a.B.RowCount, j];

                    }
                }

            for (int i = 0; i < C.RowCount; i++)
                for (int j = 0; j < C.ColumnCount; j++)
                {
                    if (j < a.C.ColumnCount)
                    {
                        C[i, j] = a.C[i, j];
                    }
```

```
                else
                {
                    C[i, j] = b.C[i, j - a.C.ColumnCount];
                }
            }

        D = a.D + b.D;

        return new StatementSpace(A, B, C, D, a.TimeDelay, a.Ts);
    }
```

再比如，定义两个模型相同的==和不相同的!=运算符的定义。在编写程序时，系统会警告，这两个必须被同时定义，不能只出现一个。

```
/// <summary>
    /// 定义相等
    /// </summary>
    /// <param name="a"></param>
    /// <param name="b"></param>
    /// <returns></returns>
    public static bool operator == (StatementSpace a,StatementSpace b)
    {
        bool equalA = (a.A == b.A);
        bool equalB = (a.B == b.B);
        bool equalC = (a.C == b.C);
        bool equalD = (a.D == b.D);

        bool equalTimeDelay = (a.TimeDelay == b.TimeDelay);
        bool equalTs = (a.Ts == b.Ts);
        if(equalA&&equalB&&equalC&&equalD&&equalTimeDelay&&equalTs)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    /// <summary>
    /// 定义不相等
    /// </summary>
    /// <param name="a"></param>
    /// <param name="b"></param>
    /// <returns></returns>
```

```
public static bool operator != (StatementSpace a,StatementSpace b)
{
    return (!(a == b));
}
```

2.4 自定义异常处理命名空间 ExceptionSelf 进行常见异常处理

在控制系统中存在很多异常，比如两个状态空间方程进行串联连接，必须要求前面的输出的维度等于后面的模型的输入维度，如果不满足这个条件，就应当抛出一个异常。相关的问题非常多，因而很有必要自定义一些异常类来对这些问题进行具体化描述。命名空间 ExceptionSelf 就是这些异常类集合而来。

如图 2.7 所示。

```
/// <summary>
/// 命名空间，里面存放着所有自定义的异常类
/// </summary>
namespace ExceptionSelf
{

    /// <summary>
    /// 当两个模型特性不一导致无法进行结合操作时需要抛出的异常
    /// </summary>
    3 references
    class NotTheSameModelException...

    /// <summary>
    /// 当两个矩阵的行数和列数不匹配（比如矩阵相乘左边的列数不等于右边的行数）时抛出此错误
    /// </summary>
    12 references
    class MatrixNotTheSameRowCol...


    /// <summary>
    /// 输入参数格式或者范围错误异常
    /// </summary>
    6 references
    class WrongInputArgument...

    /// <summary>
    /// 错误的输入矩阵格式
    /// </summary>
    4 references
    class WrongInputMatrix...
```

图 2.7 常见异常类

3 如何使用 C#写一个简化版 SIMULINK?

首先必须声明：**这一部分并没有完成和实现（没有代码就没有真相）。目前仅仅是在设计和构想阶段。**由于时间问题，本问题可能会在大四闲暇获得一定的进度。

为什么要去制作一个 SIMULINK 呢？抛去别的原因，在精度要求不高的情况下，一个轻量级的、仅仅实现控制仿真绘图而不包含各种冗余功能（比如图像处理、电气、机械等等）、速度不卡顿的软件具有一定的吸引力。除此之外，SIMULINK

的仿真可拓展度非常高，甚至比使用 MATLAB 中的控制系统分析设计指令还要高。所以缺少 SIMULINK（尤其是可以自定义非线性模块的）类似功能的仿真工具包功能上会欠缺很多。

下面的内容由于没有实现，所以更多的是一个设想。且：这种设计思路是在逻辑层进行的，即考虑的是功能的逻辑实现而非图像实现。所以即使是在实现了下述功能后还需要添加一点计算机图形学的组件。

上述功能需要基于以下考虑：如何定义子模块，如何看待连接方式，如何定义模块，如何进行模型保存和加载。SIMULINK 具体的运行机制（数值逼近连续，微分方程解决，初态解决等数学问题）。这些问题我在大二就有一些思索，在深度使用 SIMULINK 仿真后有很多感触。下面重点介绍和编程设计相关的，数学方面的此处略去。

3.1 继承接口的子模块定义和实现

在搭建 SIMULINK 时，可以很明显的感觉到：一个模型就是由子模块和连线表达。而子模块既可以时基础的、不可分割的模块，也可以是多个子模块组合在一起表达的一个新的子模块。所以可以看出：子模块中也有模块的特性。因而：一个模块可以转化为子模块，这个操作即是完全忽略其内部特征，只逗留外部接口，类似于从状态空间方程到传递函数模型的转换。

那么从简单的开始，如何去定义一个子模块呢？子模块的功能肯定是截然不同的，甚至，他们的私有字段、属性、方法都截然不同，但是，他们必须有共性。这些共性抽象出来就是子模块的父类特征（所有的子模块类都要继承于此）。根据，这个想法，可以做图 3．1 所示的父类定义。

```
namespace ControlSystemDesign.ControlModel.SimuControl
{
    /// <summary>
    /// simuControl中对子模块进行描述的类
    /// 子模块的类仅仅支持输入和输出运算，对模型类下的一些功能进行了抽象
    ///
    /// </summary>
    3 references
    class SubSimuModel:ISimu
    {
        private ArrayList Input;
        private ArrayList Output;
        private ArrayList InterValues;



        2 references
        string ISimu.GetInfo()
        {
            throw new NotImplementedException();
        }

        2 references
```

图 3．1　子模块定义

可以看出，我们只要规定好每一个模块的输入和输出，以及模块在内部运行时需要保存和设置的中间变量（比如说：状态初值）就可以了。

同时，可以看见，该类继承了一个接口 ISimu，这个接口的设计是为了实现不同子模块之间连接时的通用协议和要求。即使是总模块，也应该继承该接口，否则它就无法转化为一个子模块了，图 3.2 展示了这个实现。

```
0 references
class SimuModel:ISimu
{

    private ArrayList Input;          //ISimu 输入
    private ArrayList Output;         //ISimu 输出
    private ArrayList Submodules;     //SubSimuModel 子模型
    private ArrayList Link;           //连接关系
```

图 3.2　模块定义

那么接口应该封装出什么功能呢？目前认为它应该包含如下方法（不一定准确，将来可能会修改）：

```
/// <summary>
        /// 返回目标输入输出的信息
        /// </summary>
        /// <returns></returns>
        string GetInfo();

        /// <summary>
        /// 得到目标地方的值
        /// </summary>
        /// <returns></returns>
        ArrayList GetValue();

        /// <summary>
        /// 返回对应的模型
        /// </summary>
        /// <returns></returns>
        SubSimuModel GetModel();

        /// <summary>
        /// 返回每个子模块独一无二的tag
        /// </summary>
        /// <returns></returns>
        string GetMainTag();

        /// <summary>
        /// 获取每个模块中输入输出对应的接口标签名字，在同一个模块中输出输入的标签名字都是确定的
        /// </summary>
        /// <returns></returns>
        ArrayList GetSubTags();
```

3.2 一个模块应该怎么组织他的子模块

有了子模块，下面的问题就是如何组织他们。目前的想法是组装的模块需要

符合接口要求和数据流要求两个层次。在此处，计划采用标签重组的方式。即定义一个二元元组，元组是字符串类型，描述了某个子模块的某个输入输出。这样就可以把连接描述出来了！

```
6 references
class SimuModel:ISimu
{

    private ArrayList Input;        //ISimu 输入
    private ArrayList Output;       //ISimu 输出
    private ArrayList Submodules;   //SubSimuModel 子模型
    private ArrayList Link;         //连接关系
```

图 3.2　模块定义

如果把整个模型使用这种方法进行解析，就可以将微分方程（或者差分方程）的迭代过程一步步实现，最终实现整个系统的仿真。由于这里面有一些数学知识（主要是微分方程数值解法问题）时间仓促未能理解明白，所以此处代码停滞。

3.3 使用 XML 格式进行 Simulink 模型的保存和导入

如果能实现一个模型，那么应该存储和导入它呢？目前简单而直接的想法是使用 XML 语言进行描述。这种描述主要分为两个步骤：

1. 实现每个成员类型到字符串类型的可逆变换；
2. 按照其变量名作为标签生成和解析 XML 文件。

这也并非太难的工作。图 3.3 所描述即为一个 SIMULINK 模型的存储格式，可以看出，它使用的是类似于 JSON 和 XML 等相关格式的描述方法（由于 MATLAB 自己有语言，所以里面很显然是弱类型的），我们可以发现这里面并不存在任何难点。

```
SimulationMode       "normal"
LinearizationMsg     "none"
Profile          off
ParamWorkspaceSource     "MATLABWorkspace"
AccelSystemTargetFile    "accel.tlc"
AccelTemplateMakefile    "accel_default_tmf"
AccelMakeCommand     "make_rtw"
TryForcingSFcnDF     off
Object {
  $PropName            "DataLoggingOverride"
  $ObjectID            2
  $ClassName           "Simulink.SimulationData.ModelLoggingInfo"
  model_           "SCI"
  signals_             []
  overrideMode_        [0.0]
  Array {
    Type               "Cell"
    Dimension              1
    Cell               "SCI"
    PropName               "logAsSpecifiedByModels_"
  }
```

图 3.3 MATLAB 中的.mdl 文件格式一瞥

3.4 泛型与否，论 ArrayList 的优与劣

在进行模块、子模块的定义时，由于输入输出的变量的类型是变化的且不同模块的元素个数不同，所以在对输入和输出进行概括时，使用到了 ArrayList 类型。这种类型有利有弊。由于是课上所学知识，故简单分析之。

ArrayList 类型在此处的优点在于其对数据类型的泛化，即：可以存储不同数据类型的元素。当然，这样的缺点就是索引和重读的时候辨识度太低了。因而设计有新定义的子标签去指引它们。除此之外，另一个问题是：对于某一个特定的子模块——一般而言其输入输出的个数是固定的，换而言之：模块不会在仿真的时候修改其输入输出的变量的个数，所以在此处使用 ArrayList 会有一些大材小用。大材小用的潜台词即是：在性能上存在一定程度的浪费。当然，在极个别特殊情形下这或许也有用武之地。比如 SIMULINK 中司空见惯的加法器，它的输入的个数可通过设置参数来进行调节，这个功能或许就可以对应到这里。然而，一般情形下，使用一个固定长度的数组这个问题也可以解决了。

附录及参考文献

工具包未来情况说明

本次提交的作业可以视作 0.1 版本，将来（辽远的将来，或许到大四上学期不忙的时候）会添加一些基本的其他功能。计划当能实现本摘要里撰写的全部功能时，可以更新到 1.0 版本。关于该项目（已开源）可以到这里（https://github.com/liangzid/ControlPackage）查看。本项目开源，免费，欢迎有兴趣的人一起实现。

参考文献和网站

[1] C#编程和.NET 框架，崔建江，机械工业出版社
[2] 现代控制理论基础，石海彬，清华大学出版社
[3] MathNet 数值运算网址：https://numerics.mathdotnet.com/