

第二章 Simulink 建模与调试

Simulink 是动态和嵌入式等系统的建模与仿真工具，也是基于模型设计的基础。对于机电、航空航天、信号处理、自动控制、通讯、音视频处理等众多领域，Simulink 提供了交互式的可视化开发环境和可定制的模块库，对系统进行建模、仿真与调试等。并可实现与 Stateflow 有限状态机的无缝连接，扩展对复杂系统的建模能力。

通过 Simulink 模块库自带的 1000 多个预定义模块，基本上可快速地创建基于 MCU 器件应用的系统模型。运用层次化建模、数据管理，子系统定制等手段，即使是复杂的嵌入式 MCU 应用系统，也能轻松完成简明精确的模型描述。

大量使用 Embedded MATLAB 来创建用户自己的算法模块，可大大加快建模速度。读者在后面的内容中，会经常看到用 Embedded MATLAB 创建的算法模块，加快 MCU 器件开发的实例。模型是基于模型设计的起点，同时也最核心的东西。本章将以基于 PID 控制的直流电机的物理建模与调试为例来介绍 Simulink，更详细的内容请读者参考 MathWorks 公司相关内容的用户手册。

Simulink 的主要特点如下：

- 众多可扩展的模块库
- 利用图形编辑器来组合和管理模块图
- 以系统功能来划分模型，实现对复杂系统的管理
- 利用模型浏览器（Model Explorer）寻找、创建、配置模型组件的参数与属性
- 利用 API 实现与其他仿真程序的连接或集成用户代码
- 用图形化的调试器和剖析器来检查仿真结果，评估模型的性能指标
- 在 MATLAB 命令窗口中，可对仿真结果进行分析与可视化，自定义模型环境、信号参数和测试数据
- 利用模型分析和诊断工具来确保模型的一致性，定位模型中的错误


本章主要内容有：

- Simulink 基本操作
- 搭建直流电机模型
- Simulink 模型调试

2.1 Simulink 基本操作

2.1.1 模块库和编辑窗口

打开模型库浏览器

在 matlab 的命令窗口中输入“simulink”指令或单击 matlab 工具栏上的“simulink”图标  就可以打开模型库浏览器。如图 2.1.1 所示：

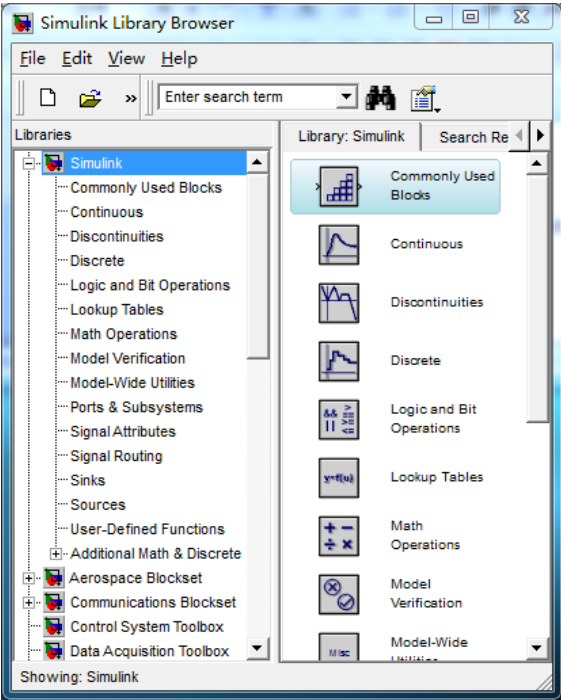


图 2.1.1 模型库浏览器

打开模型编辑窗口

要建立一个新的模型，首先要打开一个模型编辑窗口。可以通过点击模块库浏览器上的 NEW Model 按钮，或 File→NEW→Model 来打开窗口，如图 2.1.2 所示。

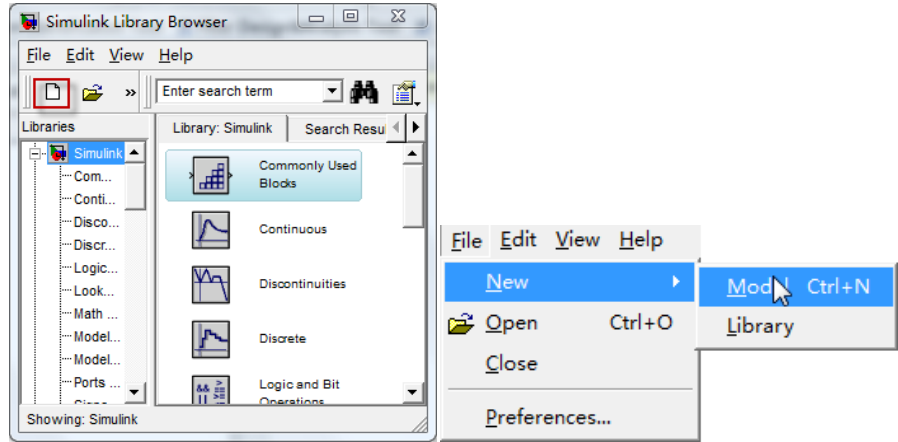


图 2.1.2 打开模型编辑窗口

2.1.2 Simulink 模块库

Simulink 模块库是建立模型的基础，其中囊括了大量的基本功能模块，只有当用户熟练的掌握了模块库，才能快速、高效的建立模型。从图 2.1.1 所示的模型库浏览器可知，在 Simulink 模块库中包含有以下子模块库，如表 2.1.1 所示：

表 2.1.1 模块库列表

常用模块（commonly used block）	连续模块（continuous）
非连续模块（discontinuous）	离散模块（discrete）

逻辑和位操作模块（logic and bit operations）	查找表模块（lookup tables）
数学运算模块（math operations）	模型验证模块（model verification）
模型实用模块（model-wide utilities）	端口与子系统模块（ports & subsystems）
信号属性模块（signal attributes）	信号路由模块（signal routing）
接收器模块（sinks）	源模块（sources）
用户自定义模块（user-defined functions）	附加操作模块（additional math & discrete）

下面将详细介绍几种使用频率较高的模块库。

1. 常用模块库（commonly used block）

常用模块库中的模块是 simulink 所有模块库中使用频率最高模块的合集，主要是为了方便用户以最快的速度建立模型。常用模块包含如图 2.1.3 所示的成员，模块功能如表 2.1.2 所示：

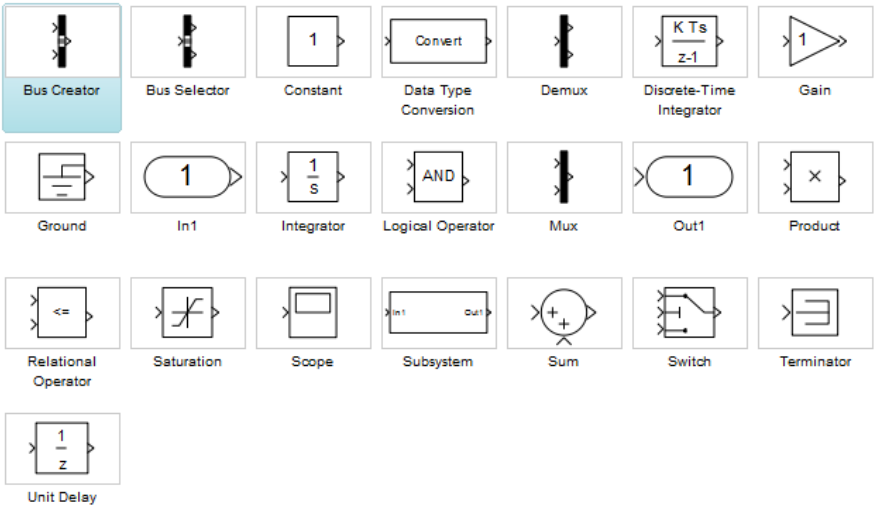


图 2.1.3 常用模块库

表 2.1.2 常用模块库列表

名称	功能	名称	功能
Bus Creator	生成总线	Bus Selector	分离总线
Constant	常量信号	Data Type Conversion	转换数据类型
Demux	抽取向量信号中的元素并输出	Discrete-Time Integrator	时间离散积分
Gain	放大器	Ground	接地
Inport	产生输入口	Integrator,Integrator Limited	信号积分
Logical Operator	逻辑运算	Mux	将输入信号合成为向量
Outport	产生输出口	Product	标量和非标量乘除或矩阵乘法和转置

Relational Operator	对输入做关系运算	Saturation	饱和
Scope and Floating Scope	显示仿真信号	Subsystem,Atomic Subsystem,Nonvirtual Subsystem,CodeReuse Subsystem	以子系统表示其他系统
Sum,Add,Subtract,Sum of Elements	加或减	Switch	通过第二个输入值来输出第一或第二个输入。
Terminator	终止未连接的输出口	Unit Delay	延迟一个采样周期

2. 连续模块库（continuous）

连续模块库中的模块如 2.1.4 图所示，它包含了搭建连续系统所涉及到的绝大部分模块，这些模块的功能如 2.1.3 表所示：

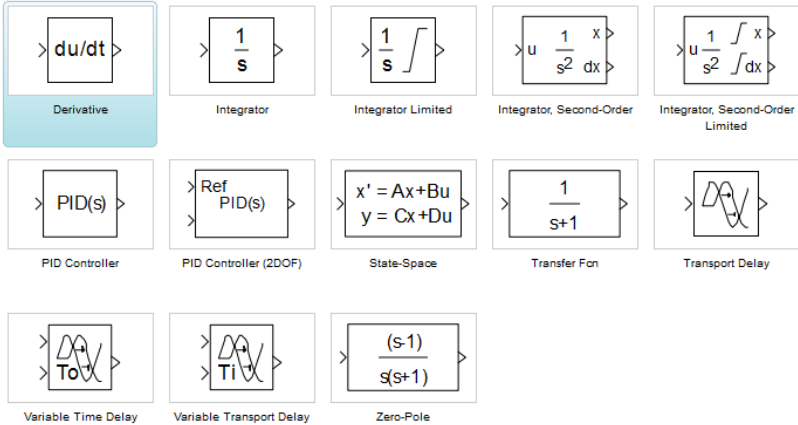


图 2.1.4 连续模块库

表 2.1.3 连续模块库列表

名称	功能	名称	功能
Derivative	微分	Integrator	积分
Integrator Limited	有限积分	Integrator 2 nd -order	二阶积分
Integrator 2 nd -order Limited	二阶有限积分	PID Controller	比例微积分控制器
PID Controller (2DOF)	双自由度比例微积分控制器	State-Space	状态空间
Transfer Fcn	传递函数	Transport Delay	时间延迟
Variable Time Delay	可变时间延迟	Variable Transport Delay	可变时间延迟

Zero-Pole	零极点		
-----------	-----	--	--

结合本书是讲述基于模型设计的思想开发 MCU 器件，本章将以 Simulink 在控制电机中的应用为例，介绍 Simulink 的建模与调试技术。这里值得一提的是 PID 控制模块。PID 控制器就是根据系统的误差，利用比例、积分、微分计算出控制量进行控制的。它是在较新版本的 Simulink 中才新增并逐步完善的一个模块，R2010b 版已经具备自动调节功能。具体原理和使用将在后面分析。

3. 离散模块库（discontinuous）

离散模块在涉及数字信号系统中被广泛使用，基于这种考虑，mathworks 公司单独列出了离散系统模块库。离散模块库中的模块和其功能如 2.1.5 图所列：

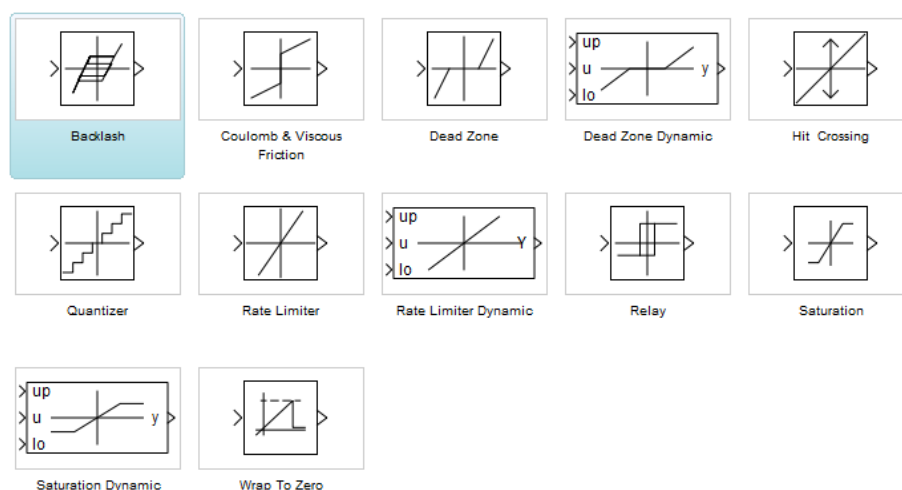


图 2.1.5 离散模块库

其中常用模块的功能如表 2.1.4 所示：

表 2.1.4 离散模块库列表

名称	功能	名称	功能
Difference	差分	Discrete Derivative	离散微分方程
Discrete FIR Filter	离散 FIR 滤波器	Discrete Filter	离散滤波器
Discrete PID Controller	离散 PID 控制器	Discrete PID Controller (2DOF)	离散双自由度 PID 控制器
Discrete State-Space	离散状态空间	Discrete Transfer Fcn	离散传递函数
Discrete Zero-Pole	离散零极点	Discrete Time Integrator	离散时间积分
1 st -order Hold	一阶保持器	Integer Delay	采样保持
Memory	记忆	Tapped Delay	采样周期延迟
Transfer Fcn 1 st -order	一阶传递函数	Transfer Fcn Lead or Lag	传递函数（超前或延迟）

Transfer Fcn	传递函数(有零点无极点)	Unit Delay	单位延迟
Real Zero			
Zero-Order Hold	零阶保持器		

4. 数学运算模块库（math operations）

数学运算模块将很多数学运算封装成模块的形式，使数学运算操作大大简化，减少了很多程序设计上的繁琐过程。此模块库所包含的模块如图 2.1.6 所示：

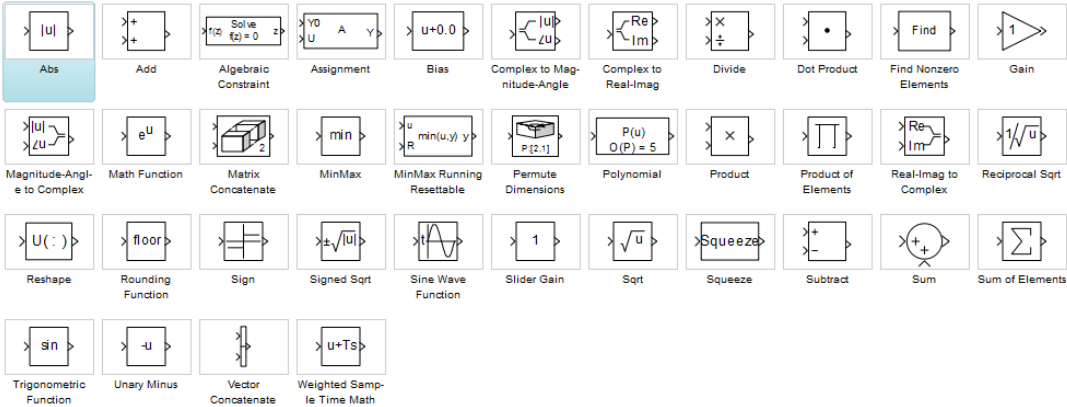


图 2.1.6 数学运算模块库

其中常用模块的功能如表 2.1.5 所示：

表 2.1.5 数学运算模块库列表

名称	功能	名称	功能
Sum	对输入求代数 和	Rounding Function	取整
Gain	常量增益	MinMax	求最值
Slider Gain	可用滑动条改变的增益	Trigonometric Function	三角函数
Product	对输入求积或商	Algebraic Constraint	强制驶入信号为 0
Dot Product	点积	Complex to Magnitude-Angle	复数的幅值相角
Sign	取输入的正负符号	Magnitude-Angle to Complex	根据幅值相角得到复数
Abs	绝对值（模）	Complex to Real-Imag	复数的实部虚部
Math Function	数学运算函数	Real-Imag to complex	由实部虚部求复数

5. 信号源模块库（signal attributes）

信号源模块库如 2.1.7 图所示：

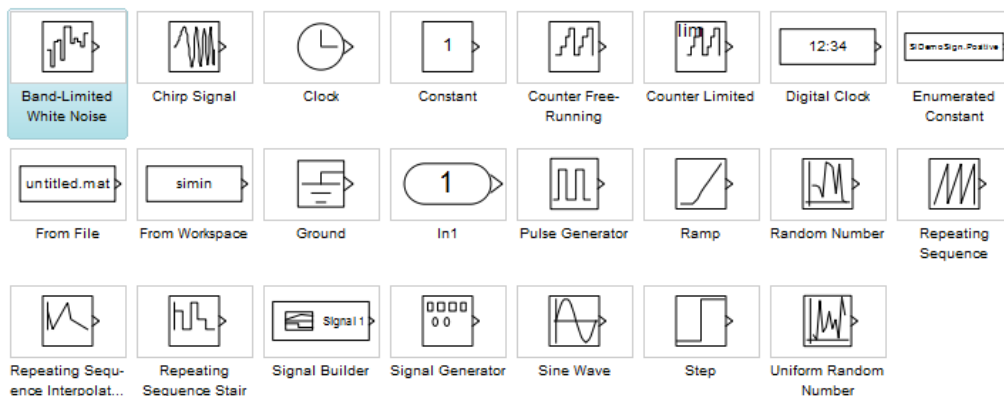


图 2.1.7 信号源模块库

其中常用模块的功能如表 2.1.6 所示：

表 2.1.6 信号源模块库列表

名称	功能	名称	功能
Band-Limited White Noise	限带白噪声	Chirp Signal	频率变化的正弦信号
Clock	时钟信号	Constant	常数
Counter Limited	受限计数器	Digital Clock	数字时钟
Enumerated Constant	枚举常数	From File	从文件读数据
From Workspace	从工作空间读数据	Ground	接地
Inport	输入接口	Pulse Generator	脉冲发生器
Ramp	线性增或减的信号	Random Number	随机数
Repeating Sequence	重复系列	Repeating Sequence Interpolated	重复序列插值
Repeating Sequence Stair	阶梯状重复序列	Signal Builder	产生分段线性的可交替信号
Signal Generator	信号发生器	Sine Wave	正弦信号
Step	阶跃信号	Uniform Random Number	平均分布的随机信号

6. 信号接收模块库（sinks）

信号接收模块库如图 2.1.8 所示：



图 2.1.8 信号接收模块库

其中常用模块的功能如表 2.1.7 所示：

表 2.1.7 信号接收模块库列表

名称	功能	名称	功能
Display	显示输入值	Output	输出端口
Scope and Floating Scope	显示仿真信号	Stop Simulation	非零时停止仿真
Terminator	终止输出信号	To File	输出到文件
To WorkSpace	输出到工作空间	XY Gragh	作图

7. 用户自定义模块库（user-defined functions）

用户自定义模块库如图 2.1.9 所示：

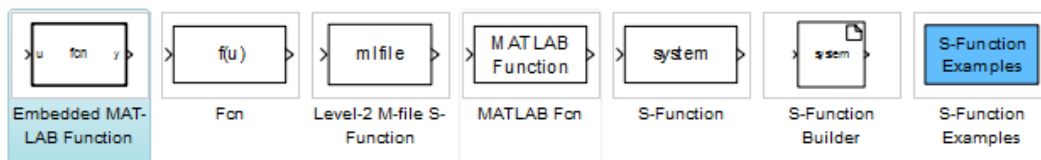


图 2.1.9 用户自定义模块库

其中常用模块的功能如表 2.1.8 所示：

表 2.1.8 用户自定义模块库列表

名称	功能	名称	功能
Embedded MATLAB Function	嵌入式 MATLAB 函数	Fcn	各种函数组合
Level-2 M-File S-Function	二级 M 文件 S 函数	MATLAB Fcn	使用 MATLAB 函数作为输入
S-Function	S 函数	S-Function Builder	S 函数构造器

2.1.3 模块的基本操作

1. 模块的查找

在模块库页面的左侧以树形结构显示了一系列的子库，如连续模块库，离散模块库，数学运算模块库，当用户对这些模块比较熟悉以后，可以非常方便、快捷的找到自己需要的模块。

如果是刚刚接触 Simulink 的用户，这里还提供了查找功能，方便不熟悉它的人使用。如图 2.1.10 所示：

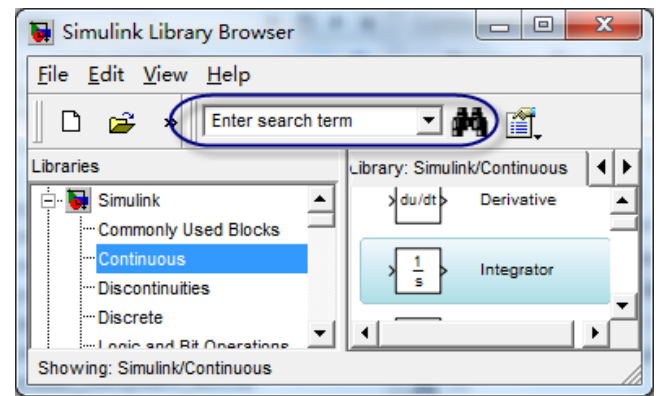


图 2.1.10 模块查找

当用户找到自己所需要的模块后，就要把它复制到模型编辑窗口中以便进行下一步的操作。这里可以直接把模块拖拽到编辑窗口中完成复制，也可以右键单击，在弹出菜单中选择“Add to new model”，如图 2.1.11 所示。

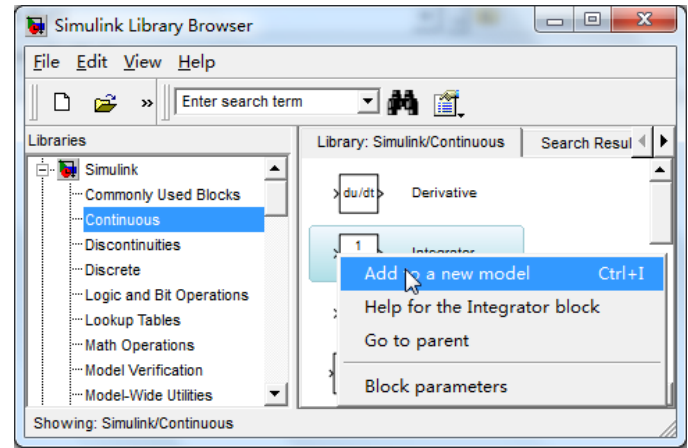


图 2.1.11 添加模块

2. 模块的选定

模块被加入到编辑窗口以后，要通过选定模块才能对其进行操作。选定单个模块可以直接用鼠标左键单击目标模块；也可以按下鼠标任意键拖动，此时会出现一个虚线框，当虚线框包围了目标模块时放开鼠标，即可选中模块。

若是要选中多个模块就需要用上述的虚线框法来操作，如图 2.1.12 所示。选择编辑窗

口中所有对象的方法是单击菜单 Edit→Select All。

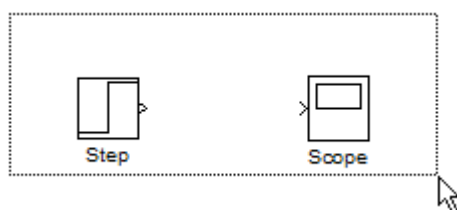


图 2.1.12 选定一组对象

3. 模块的连接和调整

- 模块的调整

为了使模型更加美观和符合逻辑,有时需要对模块的大小、方向等做适当的修改。模块大小的调整。选中模块,用鼠标选中模块周围 4 个黑色方块中的任意一个开始拖动,这时会出现一个虚线框表示的新模块大小示意,调整至需要的大小后释放鼠标即可,如图 2.1.13 所示。



图 2.1.13 调整模块大小

模块的旋转。选中模块想要旋转的模块,在菜单中选择 **Format**,在次级目录中选择 **Flip Block** 可以使模块 180° 旋转;选择 **Rotate Block** 使模块旋转 90°。如图 2.1.14 所示:

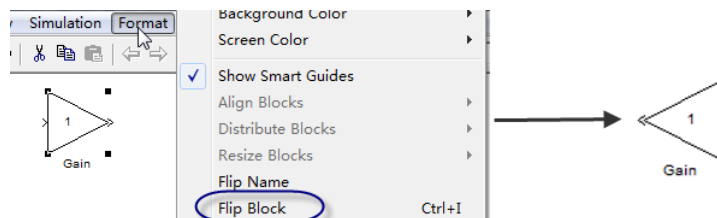


图 2.1.14 模块翻转

- 模块的连接

模型中的信号是从模块经连线传输到下一个模块的,因此模块间的连线被称为信号线,当模块设置好以后,只有将它们按一定的顺序连接起来才能完成一个正确的模型。

自动连线: Simulink 系统具备自动连线的功能,步骤如下:选择一个具有信号输出的模块,按下“ctrl”键,然后用鼠标左键单击要连接到的模块。如图 2.1.15 所示:

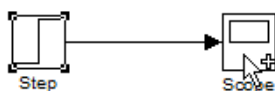


图 2.1.15 自动连线

手动连线.用户也可以选择自己手动连线。把鼠标指针移动到第一个模块的输出端口,这时指针的形状会变为一个十字,如图 2.1.16 所示。

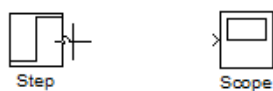


图 2.1.16 手动连线

按下并拖动鼠标至目标模块的输入端口处，这时鼠标指针会变成一个双十字，释放鼠标后两个模块就连接起来了。如图 2.1.17 所示：

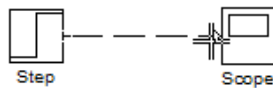


图 2.1.17 手动连线

提示：用手动方法连接模块时是可以从输出到输入口划连接线的。

分支连线：是指从一条已经存在的连线上另外再拉出一条线用来连接一个模块的输入口。这时，原连线 and 分支连线上传输的是同一个信号。

把鼠标指针移动到原有连线上，按下“ctrl”键，拖动鼠标左键到目标模块的端口，释放鼠标，完成连线。如图 2.1.18 所示：

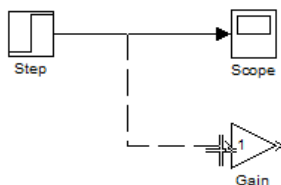


图 2.1.18 分支连线

移动连线：把鼠标移动到想要移动的连线处，按下左键并拖动到目标位置，释放按键，完成移动。如图 2.1.19 所示：

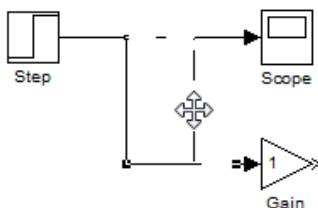


图 2.1.19 移动连线

连线的折曲：选择要折曲的线段，将鼠标移动到要折曲的点上，按下“shift”键，并按下鼠标左键，这时指针会变成一个圆圈状，拖动折曲点至目标位置后释放鼠标，完成折曲。如图 2.1.20 所示：

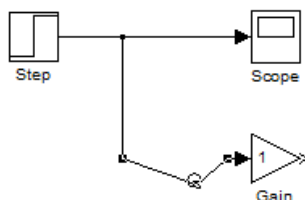


图 2.1.20 曲折连线

连线中插入模块：如果模块只有一个输入口和输出口，那么可以将该模块直接插入到一天连线中。如图 2.1.21 所示：

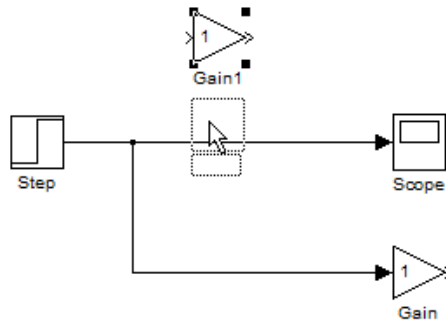


图 2.1.21 连线中插入模块

连线的注释:有时为了增加模型的可读性，常常会给信号线加上注释。在要添加注释的连线附近双击鼠标左键会出现一个文本输入框，用户可以根据需要添加适当的注释，注释的位置是可以通过鼠标拖拽更改位置的。如图 2.1.22 所示：

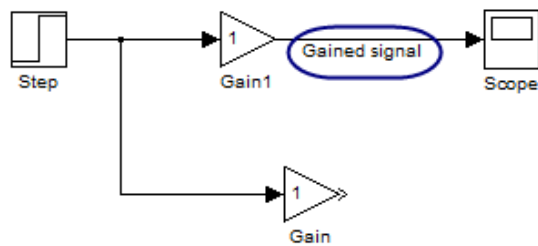


图 2.1.22 连线的注释

2.2 搭建直流电机模型

直流电机是控制系统中的常用组件，它给系统提供旋转动力或者和其他组件配合进行传动。下面将通过一个直流电机模型（DC Motor）来介绍如何在 Simulink 中建立 LTI（LTI Linear time-invariant）系统并实现对其转速的控制。

2.2.1 数学模型分析

电机可分为电气部件和机械部件两部分，电机模型如图 2.2.1 所示。

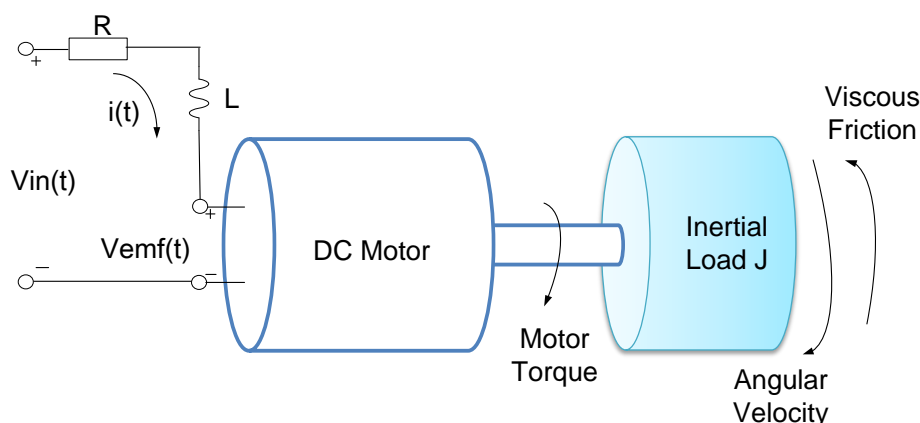


图 2.2.1 电机模型

电机的电气部分和机械部分是通过转子联系起来的，转子即为电能与机械能转化的枢纽。这两部分的方程都要受到转子感应电动势的影响。

对于左边的电气部分，由电磁感应定律可知，转子会产生一个反向的感应电动势来阻止转子的运转，外界电源 V_{in} 要克服反向感应电动势做功，才能使转子转动。我们可以先把其他元件抽象为电阻和电感，把精力集中到主要矛盾上来：电机如何平稳运转。

当电机正常运转时，先分析电气部分回路：外部电源 V_{in} 、电阻和电感上的电压降 V_R 和 V_L 、转子上的反向感应电动势 V_{emf} 在同一个回路中。由电路分析的基础知识 KVL 定律（基尔霍夫电压定律）可以得到：

$$V_{in} = V_R + V_L + V_{emf} \quad (1)$$

对于右边的机械部分，稍复杂一些。当电动机带着负载匀速旋转时，其输出转矩必定与负载转矩相等。在非理想电机模型中（即不忽略机械、电磁损耗），轴承的摩擦、电刷和换向器的摩擦、转子铁心中的涡流、磁滞损耗都要引起阻转矩。此阻转矩用 T_f 表示。这样，电动机的输出转矩便等于电磁转矩 T 减去电机本身的阻转矩 T_f 。所以，当电机克服负载阻转矩 T_L 匀速旋转时，则有：

$$T_L = T_O = T - T_f \quad (2)$$

实际上，电机经常运行在转速变化的情况下，例如启动、停转或反转等，因此必须讨论转速改变时的转矩平衡关系。当电机的转速改变时，由于电机及负载具有转动惯量，将产生惯性转矩 T_j ，则有： $T_j = J * \frac{d\omega}{dt}$ 。其中 J 是负载和电动机的转动惯量， ω 是电动机的角速度， $\frac{d\omega}{dt}$ 是其角加速度。

为了使模型能覆盖这种更一般的情况，可根据牛顿力学定律对式②做一些微调：

为了使模型能覆盖这种更一般的情况，可根据牛顿力学定律对式②做一些微调：

$$T_O - T_L = T_j = J * \frac{d\omega}{dt} \quad (3)$$

得到表示电机基本原理的方程后，需要设置一些参数以备建模使用。如：电阻 R ，电感 L ，转动惯量 J ，转矩 T_m ，旋转角度 θ ，转矩常量 K_m ，感应电动势常量 K_{emf} ，粘滞摩擦系数 K_f 。

因为 $\omega(t) = \dot{\theta}$ ， $V_{emf} = K_{emf} * \omega(t)$ ，式①可化为：

$$L \frac{di}{dt} + Ri = V_{in} - V_{emf} \quad (4)$$

又因为 $\ddot{\theta} = \frac{d\omega}{dt}$ ，式③可化为：

$$J * \ddot{\theta} + K_f \dot{\theta} = T_m \quad (5)$$

将④，⑤两式子用电压 V_{in} 和转子角速度 ω 表示出来可写为：

$$\frac{di}{dt} = \frac{V_{in}(t) - Ri(t) - K_{emf}\omega(t)}{L} \quad (6)$$

$$\frac{d\omega}{dt} = \frac{K_m i(t) - K_f \omega(t)}{J} \quad (7)$$

为了表示方便，我们对上式做 Laplace 变换，转换到 S 域。

根据拉氏变换的规则，在 0 初始条件下，一阶导数 $\rightarrow S$ ；二阶导数 $\rightarrow S^2$ ，则可得到：

$$I(s) = \frac{V_{in}(s) - RI(s) - K_{emf}\omega(s)}{L} \quad (8)$$

$$\omega(s) = \frac{K_m I(s) - K_f \omega(s)}{J} \quad (9)$$

合并化简后得到系统传输函数：

$$\frac{\omega(s)}{V_{in}(s)} = \frac{K_m}{JLS^2 + (K_f L + JR)S + (K_f R + K_{emf} K_m)} \quad (10)$$

至此，分析了直流电机的传递函数，下面将利用 Simulink 建立直流电机模型。

2.2.2 模型搭建与参数设置

模型搭建

直流电机的模型需要用到 source 库中的“step”模块，sinks 库中的“scope”模块，math operations 库中的“add”、“gain”，和 continuous 库中的“integrator”。

1. 添加 source 库中的“step”模块到模型中。如图 2.2.2 所示：

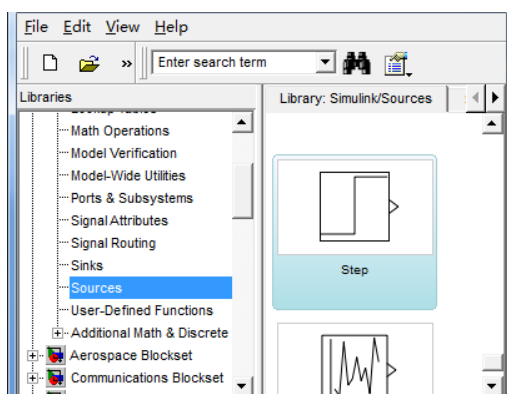


图 2.2.2 step 模块

默认情况下，信号从第一秒处产生跃变。

2. 添加 sinks 库中的“scope”模块到模型中，如图 2.2.3 所示。

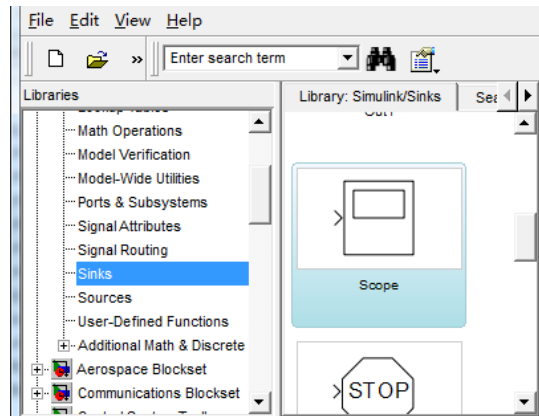



图 2.2.3 scope 模块

示波器模块是可以同时显示多个输入信号的。例如，双击打开示波器模块，单击“”按钮，在 number of axes 中输入参数 3 即可使示波器变为 3 个输入接口，如图 2.2.4。

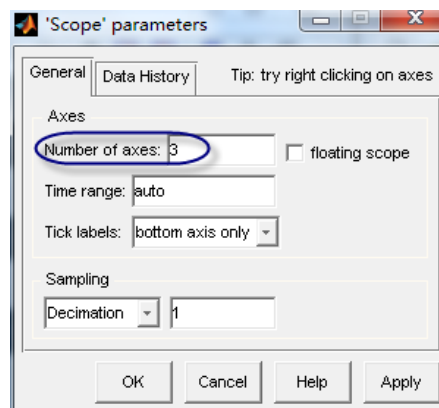


图 2.2.4 设置 scope 模块

3. 添加 math operations 库中的“add”模块到模型中，如图 2.2.5 所示。

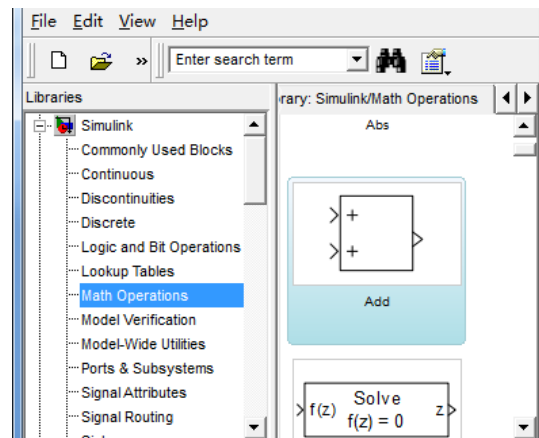


图 2.2.5 add 模块

为了正确表达电机模型的系统方程，需要两个“Add”模块，其符号参数 List of signs 需要分别设置为“+--”和“+-”，这样“Add”模块会分别含有 2 个和 3 个输入接口。如图 2.2.6 所示：

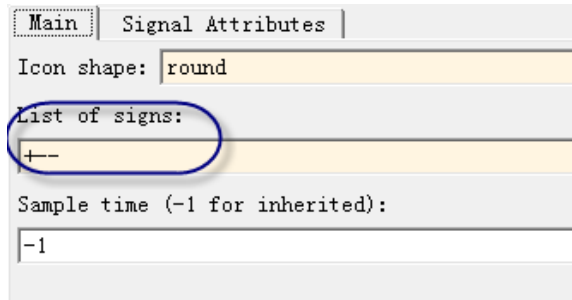


图 2.2.6 设置 add 模块

4. 添加 math operations 库中的“gain”模块到模型中，如图 2.2.7 所示。

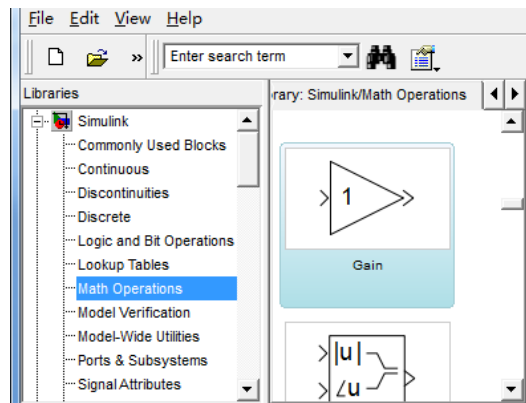


图 2.2.7 gain 模块

“gain”模块的参数需要修改。该模型需要 6 个“gain”模块，参数分别设置为： $1/L$ 、 K_m 、 $1/J$ 、 R 、 K_f 、 K_{emf} 。可以通过双击放大器模块打开其参数设置页面，然后输入参数。如图 2.2.8 所示：

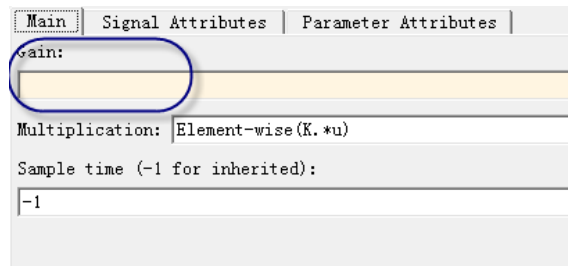


图 2.2.8 设置 gain 模块

5. 添加 continuous 库中的“integrator”模块到模型中，如图 2.2.9 所示。

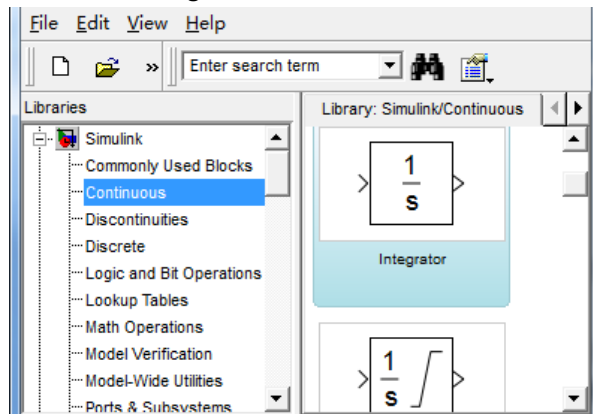


图 2.2.9 integrator 模块

模型中需要两个 “Integrator” 模块，参数可采用默认值。

6.按照式⑩表达的逻辑，在第一个积分器之前算子为 S^2 ，其系数为 JL ；第一个和第二个积分器之间算子为 S ，其系数为 $(K_f L + J R)$ ；第二个积分器之后算子为 1 ，其系数为 $(K_f R + K_{emf} K_m)$ 。由此可以连接其中的各个模块，得到如图 2.2.10 所示的模型。

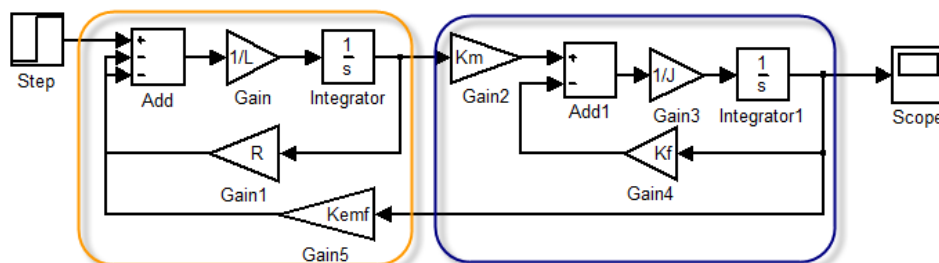


图 2.2.10 Simulink 电机模型

左侧的模块对应于公式⑧，其输出为转矩 $Torque$ ；右侧的模块对应于公式⑨，其输出为转速 $\omega(t)$ 。此模型实现了系统传输函数所描述的由输入电压控制直流电机转速的功能。当输入信号为 “Step” 阶跃信号时，输出为其阶跃响应；当输入为外部实际控制信号时，电机转速 $\omega(t)$ 即由输入信号所控制。

为了后面更好的分析电机模型，在仿真时同时输出其转动控制信号和输出转速、角度和转矩。实现原理如下：

- 控制信号与输出转速比较在添加 Signal Routing 库中的 “Mxx” 模块到模型中，如图 2.2.11 所示。

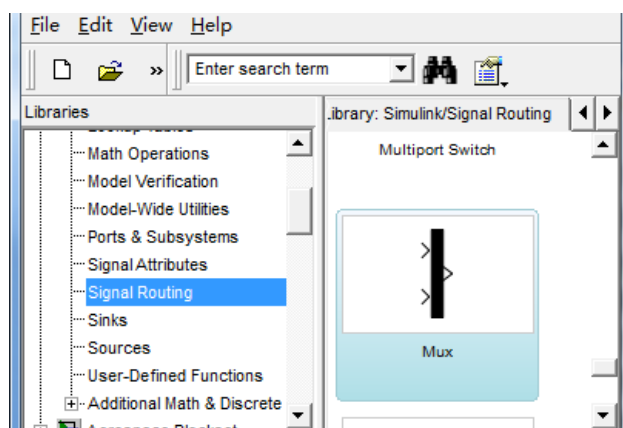


图 2.2.11 mux 模块

修改模型如图 2.2.12 所示：

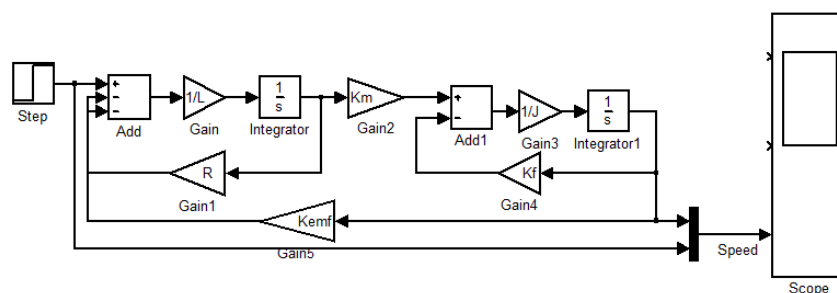


图 2.2.12 Simulink 电机模型

- 扭矩信号输出如图 2.2.10 所示，左边框图模块组的输出既是扭矩信号。
- 角度信号输出由电机数学模型公式

$$\frac{\omega(s)}{V_{in}(s)} = \frac{K_m}{JLS^2 + (K_f L + J R)s + (K_f R + K_{emf} K_m)} \quad (10)$$

可知该模型的输出为 $\omega(t)$ ，即旋转角速度，且 $\theta = \int_0^t \omega(\tau) d\tau$ ，因此，模型的输出经过一个积分器即转化为角度，修改模型如图 2.2.13:

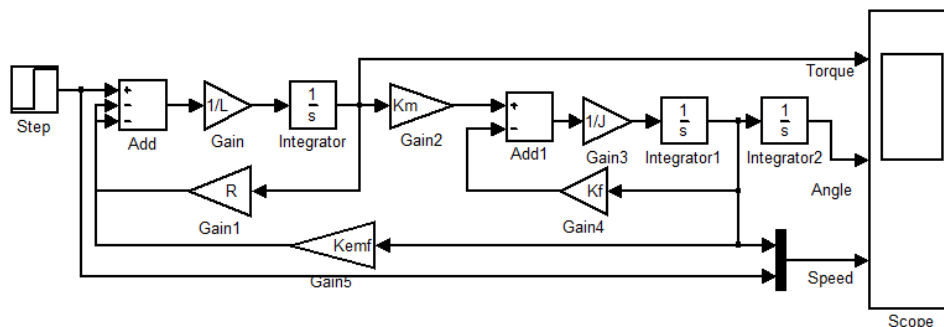


图 2.2.13 Simulink 电机模型

参数设置

最后，在仿真之前一定要设置模型仿真参数，如仿真时间，步长，表达式中定义的 R、L、J 等参数值等。

● 在 MATLAB 中以 M 文件或命令行定义变量

用户可以直接在 MATLAB 命令窗口中敲入以下命令：

```
>>R=2;           %电阻值 Resistor
>> L=0.5;        %电感值 Inductor
>> Km=0.1;       %转矩常量 Torque Constant
>> Kemf=0.1;     %反电动势常量 Back EMF Voltage Constant
>> Kf=0.2;       %阻滞系数常量 Viscous Friction Constant
>> J=0.02;       %惯性载荷 Inertial Load
```

完成后，在 MATLAB 工作空间中可以看到这些变量，如图 2.2.14 所示。

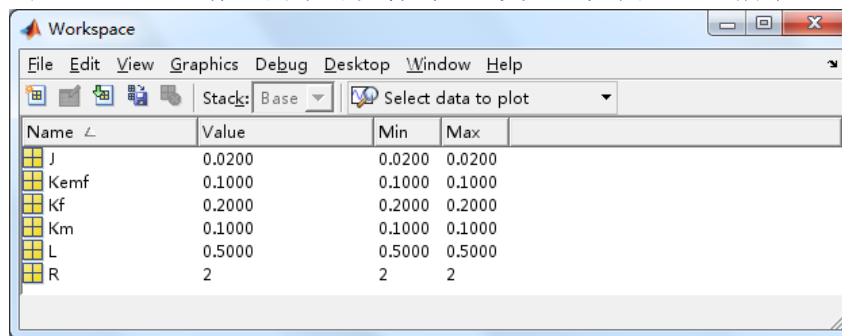


图 2.2.14 仿真参数设置

这样就完成了模型中以表达式形式定义的参数的赋值。为了方便以后再次运行模型，可以把这些数据保存下来，在运行模型之前装载这些变量即可。

全选这些变量，点击“”按钮，输入保存名 DC_motor(用户可自定义)，扩展名为.mat，即把数据保存到了当前目录下，以后可以通过双击直接装载变量。如图 2.2.15 所示：

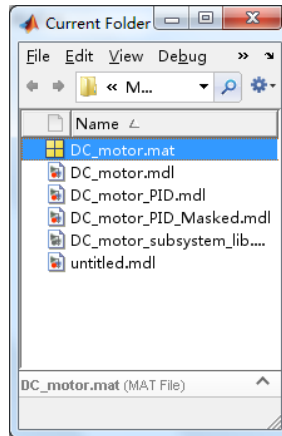


图 2.2.15 保存参数

当然，用户也可以将上述命令窗口中的指令保存为 M 文件的形式，并在运行模型前执行 M 文件来装载变量。如图 2.2.16 所示：

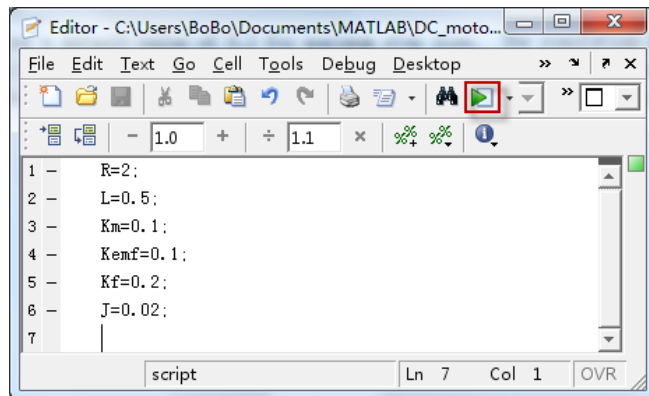


图 2.2.16

● 在 Simulink 中用 Annotation 方式定义变量

通过在 Simulink 中定义 Annotation，可以直接在模型编辑窗口中完成对参数的赋值，而不必再从 MATLAB 中读取 .mat 文件和 M 文件，省去不少麻烦。

在模型的任意空白处双击鼠标左键会出现一个可编辑的文本框，写入相关文字，如“Parameters Configuration”。如图 2.2.17 所示：

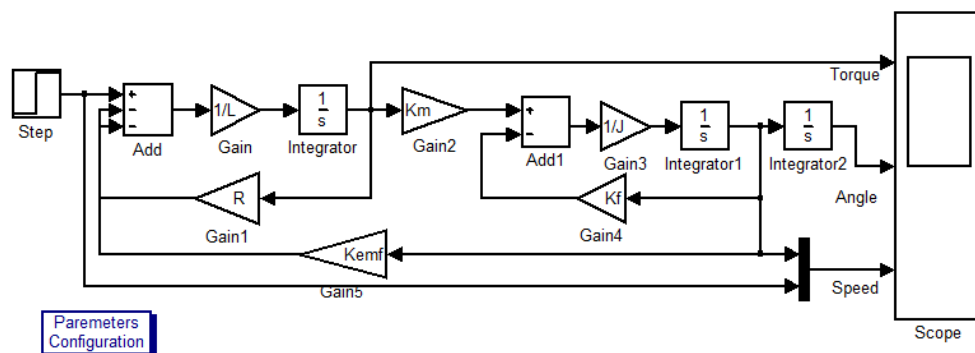


图 2.2.17 添加 Annotation

右键单击文本框会出现弹出菜单，其中较为有用的有如下几项：

- Front... 调节字体字号
- Text Alignment 文本对齐方式
- Hide/Show Drop Shadow 隐藏/显示阴影

- Annotation Properties... Annotation 属性设置
- Foreground Color 前景色
- Background Color 背景色

选择 Foreground/Background Color 调节文本框的颜色，选择 Show Drop Shadow 显示文本框阴影，美化 Annotation。

选择 Annotation Properties...打开其设置页面，如图 2.2.18 所示。

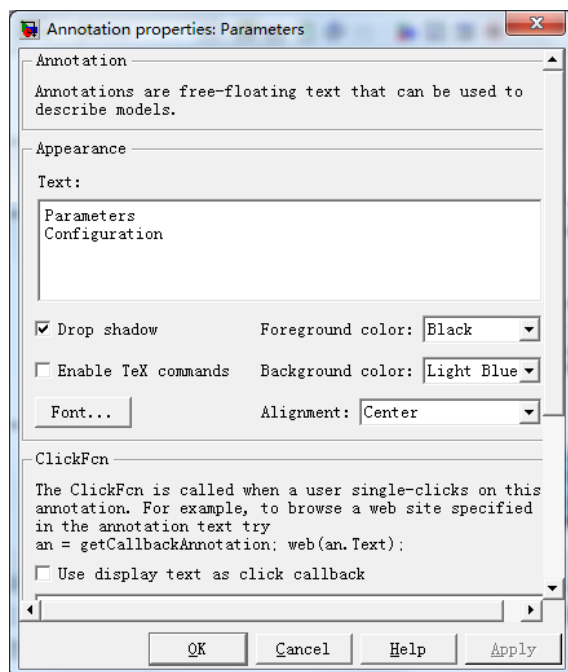


图 2.2.18 设置 Annotation

在 ClickFcn 中输入上文中使用的指令，并点击 OK 确认。这就完成了此 Annotation 的基本设置。在以后重用本模型时，只要点击该文本框就可以执行对模型参数的赋值。如图 2.2.19 所示：

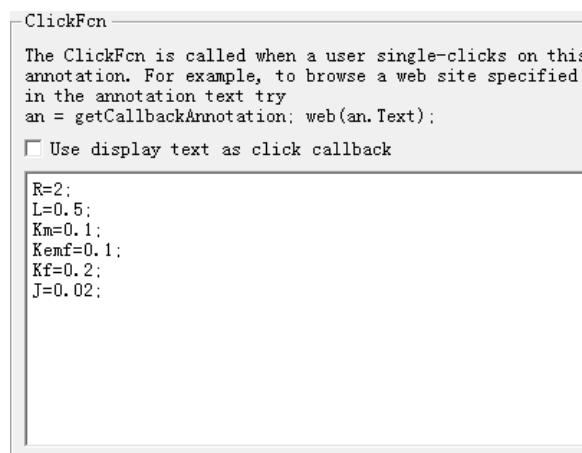



图 2.2.19 添加 Annotation 命令

- Model Explorer

Model Explorer 能够快速浏览、批量修改模型中各种元素，而不用再在图形或表格中一个个寻找这些元素。如 Simulink 模块，StateFlow 状态图，MATLAB 工作空间变量等。

在电机模型编辑窗口右侧点击“”按钮打开 Model Explorer 界面，如图 2.2.20 所示。

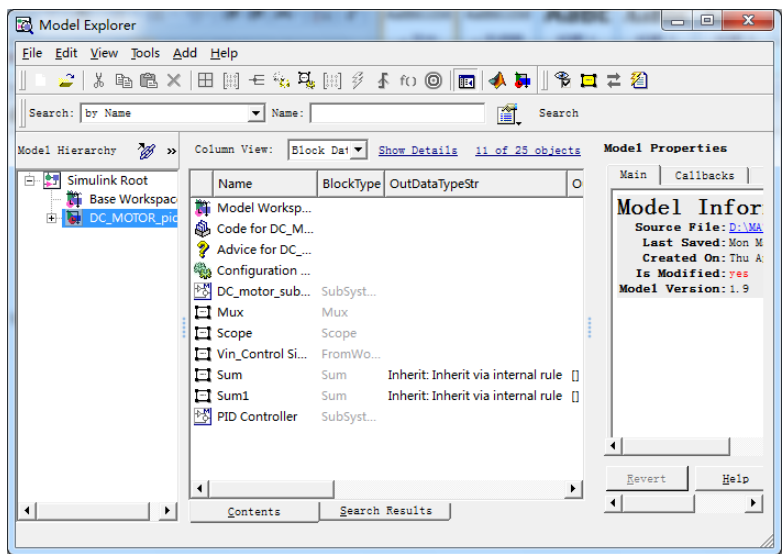


图 2.2.20 模型浏览器

在界面中间的 Content Pane 里列出了模型中的所有模块和变量。选中任意模块可以进行修改参数，数据类型等工作，该模型的所有模块设置都可以在此页面中快速完成。更详细的用法可参考帮助文档。

- 设置 configuration Parameters 中的参数

在 simulation→configuration Parameters 中可进行更多的设置，这里只介绍几种常用的设置。

1. 仿真时间设置。

设置起止时间是在“Start Time”和“Stop Time”的文本框中输入相应的数值，其默认值分别为 0 和 10，单位是秒，如图 2.2.21 所示。

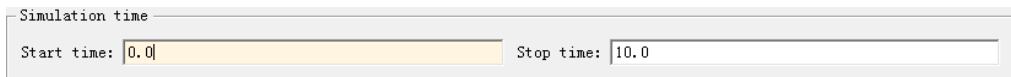


图 2.2.21 仿真时间设置

2. 求解器步长设置

求解器分为变步长和定步长求解器两种，一般情况下可采用默认的“auto”设置。其默认值为：最大步长=（停止时间-起始时间）/50

如果是高级用户，则可做进一步设置。步长是求解器运算时的时间精度，步长越短，精度越高，运算量也更大。

通过下面的例子可以直观的看到步长对仿真结果的影响。

建立如图 2.2.22 所示模型

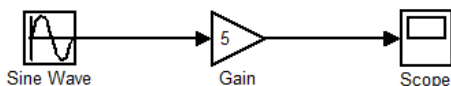


图 2.2.22 求解器步长实验模型

在 Configuration Parameters 中选择使用定步长求解器（Fixed-step），如图 2.2.23 所示，

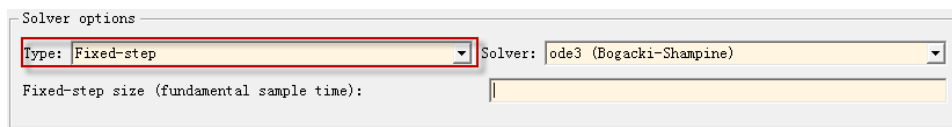


图 2.2.23 求解器设置

则当步长（Fixed-step size）分别选择 1 秒和 0.1 秒时的仿真结果如下（图 2.2.24）：

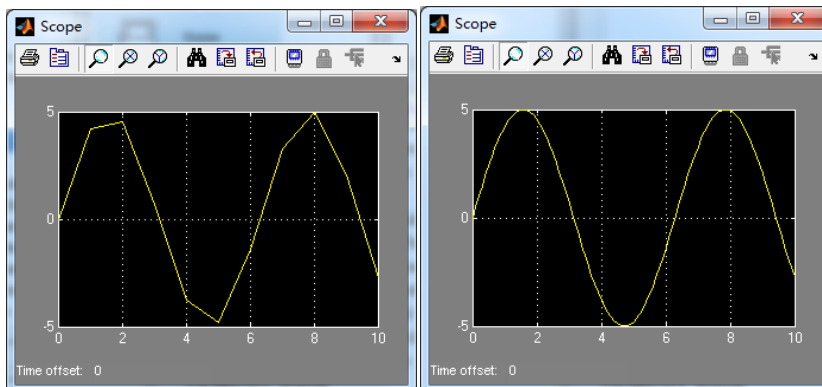


图 2.2.24 仿真结果

可见，当用户对仿真结果的精度要求比较高的时候，应该尽量选择小步长求解。

● 运行模型

点击工具栏上的“▶”按钮运行仿真，双击“Scope”模块即可查看仿真结果（按下“🔍”可以自动调节坐标轴到合适的范围），如图 2.2.25：

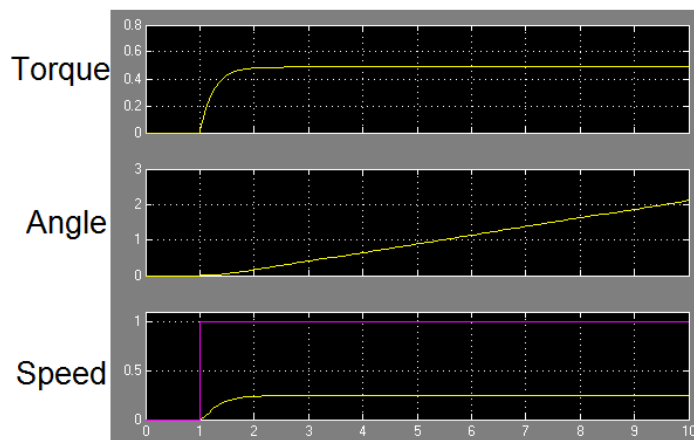


图 2.2.25 电机模型仿真结果

在示波器 Torque 中可见转矩跟随阶跃信号的趋势变化。

在示波器 Angle 中可见，转子转动过的角度随时间变化基本呈线性增长，符合其与 $w(t)$ 的关系 $\theta = \int_0^t w(\tau) d\tau$ 。

在示波器 Speed 中，紫色信号表示阶跃信号；黄色信号为系统响应，在接近 2 秒的时候达到稳定状态，幅度为 0.25 左右。

为了更进一步分析此系统，可以在 MATLAB 中划出其波特图：

```
>> DC_motor=tf([Km],[J*L (Kf*L+J*R) (Kf*R+Kemf*Km)]); %生成电机传输函数
>> bode(DC_motor) %生成波特图
```

在 MATLAB 中生成如图 2.2.26 所示波特图：

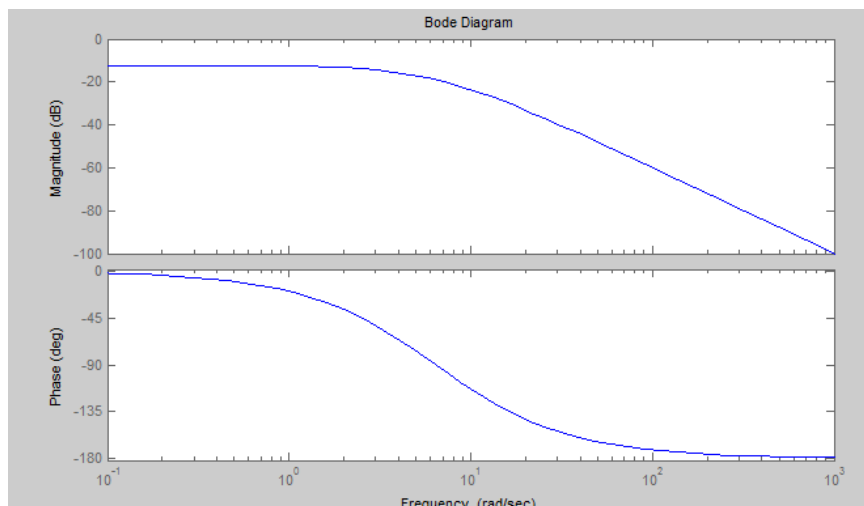


图 2.2.26 波特图

2.2.3 子系统与库

建立子系统可以把一系列的模块表示为一个子系统模块，大大简化模型的复杂度，其具有以下突出优点：减少编辑窗口中模块的数量；可以把功能相关联的模块组合在一起；使模型具有明确的功能层次。

更重要的是在子系统内部，各个模块被认为是一个整体，这样一来，其运算速度和信号传递速度都会大幅度提高，是对模型的一种优化。

1. 建立子系统

如果模型中已经完全包含了你想要创建的子系统的所有模块，可以通过把它们组合起来的方式建立子系统。

打开电机模型，选定想要包含到子系统的所有模块，如图 2.2.27 所示。

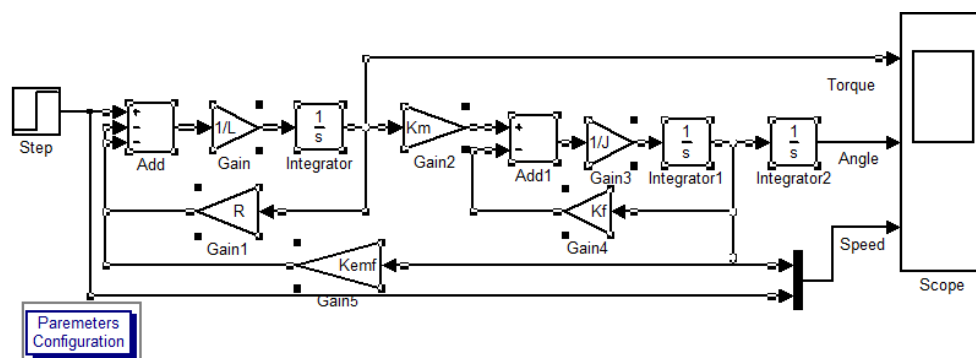


图 2.2.27 选定模块

单击 Edit→Creat Subsystem。选定的模块组即变成了一个子系统模块，如图 2.2.28 所示。

2. 封装子系统

将子系统模块重命名为 DC_motor_subsystem 方便以后使用。右键单击子系统模块，在弹出菜单中选择 Mask Subsystem，打开封装编辑器。

- Icon & Ports 页面

在该页面的 Icon options 区域，可以设置模块外框是否可见，模块是否透明，旋转时图标是否固定或跟随模块旋转，以及图标是否自适应模块的大小，这里均采用默认设置。

Icon Drawing Commands 区域中可以用命令改变模块的端口名称，添加图像，修改颜色等，如图 2.2.32 所示。

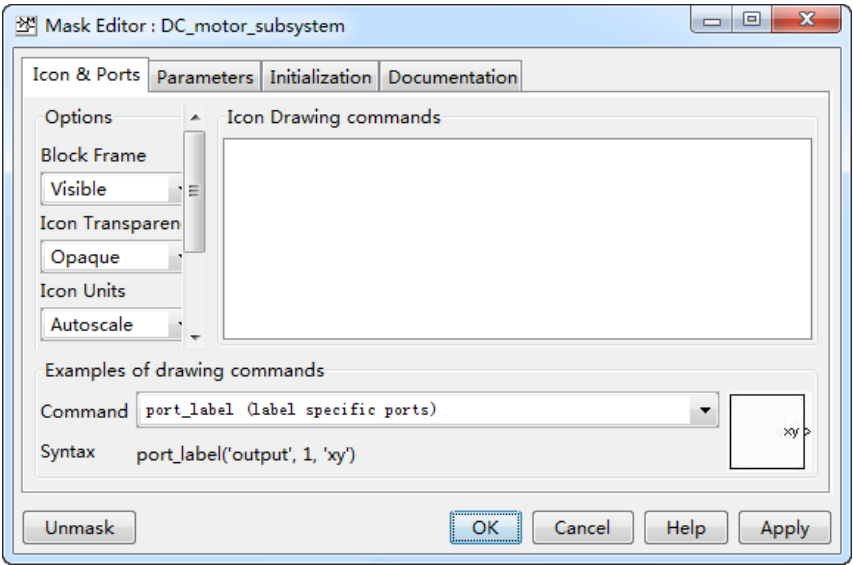


图 2.2.32 Mask 页面设置

按照页面下方的 Example Drawing Commands 给出的语法格式，我们为 DC_motor_subsystem 添加如下命令：

```
color('red');port_label('input', 1, 'Control Signal');
color('red');port_label('output', 1, 'Torque');
color('red');port_label('output', 2, 'Angle');
color('red');port_label('output', 3, 'Step');
color('red');port_label('output', 4, 'Speed');
```

子系统的端口名称的颜色和名称都会随之改变，如图 2.2.33 所示。

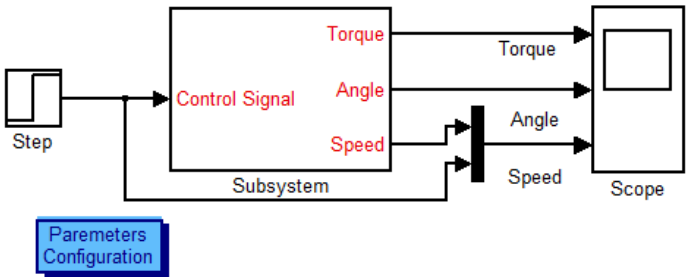


图 2.2.33 封装后的模型

添加命令 `image(imread('dianji.jpg'));`

会将当前工作目录下的“dianji.jpg”图像文件覆盖到模块上，非常直观的说明该子模块的作用如图 2.2.34 所示。

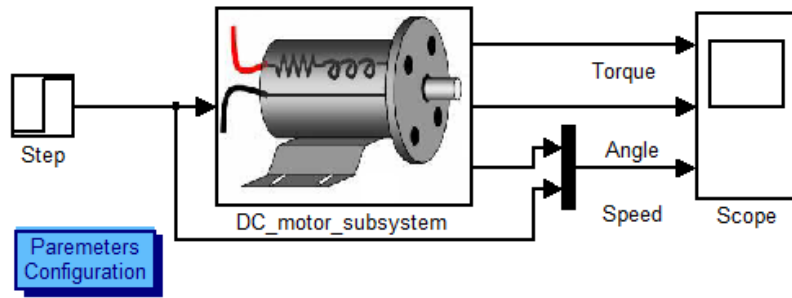



图 2.2.34 封装后的模型

点击 Example Drawing Commands 的下拉菜单可以看到所有的绘图指令和语法格式，用户可根据需要添加相应的命令。

- Parameters 页面

单击左侧工具栏按钮 ，Dialog parameters 区域新增一个参数设置框，各栏的功能如下：

- Prompt 参数的文字描述
- Variable 参数对应的变量名
- Type 参数的输入模式，edit 适用于需要用户具体指定的参数
- Evaluate 选中表示在参数对话框中输入表达式的值赋予指定的变量，否则将输入的表达式看作字符串赋予指定的变量
- Tunable 选中则允许用户在仿真过程中修改该参数值

在本例中，按图 2.2.35 所示方案设定参数：

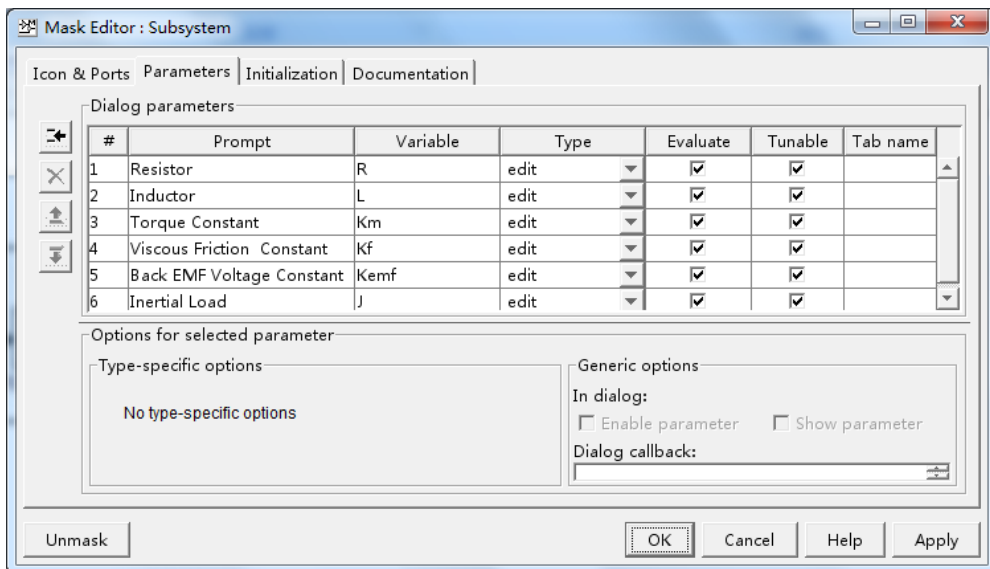


图 2.2.35 Mask 参数设置页面

点击 OK 确认后，若双击子系统模块，将不会再显示子系统底层结构，而会像 Simulink 模块库中自带的模块一样弹出参数设置页面，如图 2.2.36 所示：

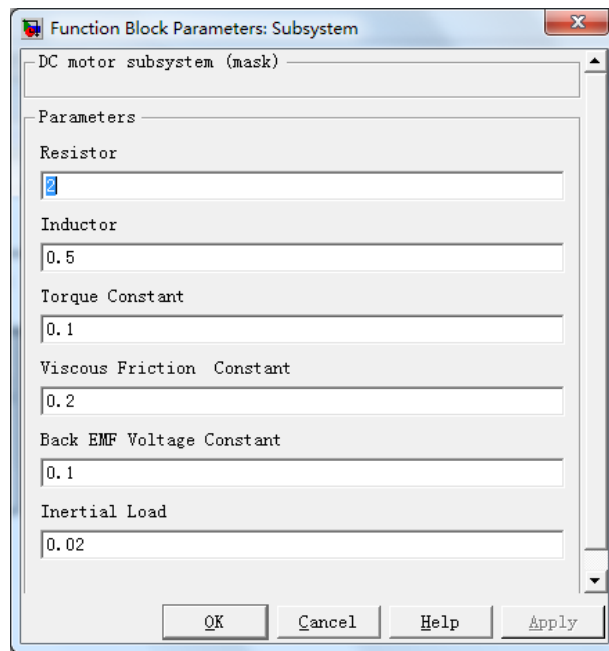


图 2.2.36 封装后的参数设置页面

封装后的系统参数其实是与底层模块的参数相关联的，给封装系统的参数赋值就相当于给定义于封装工作空间中的参数赋值，而这些参数与底层的一个或多个模块相关。以电阻值 R 和扭矩常量 K_m 为例说明封装参数与底层模块参数间的关系，如图 2.2.37 所示。

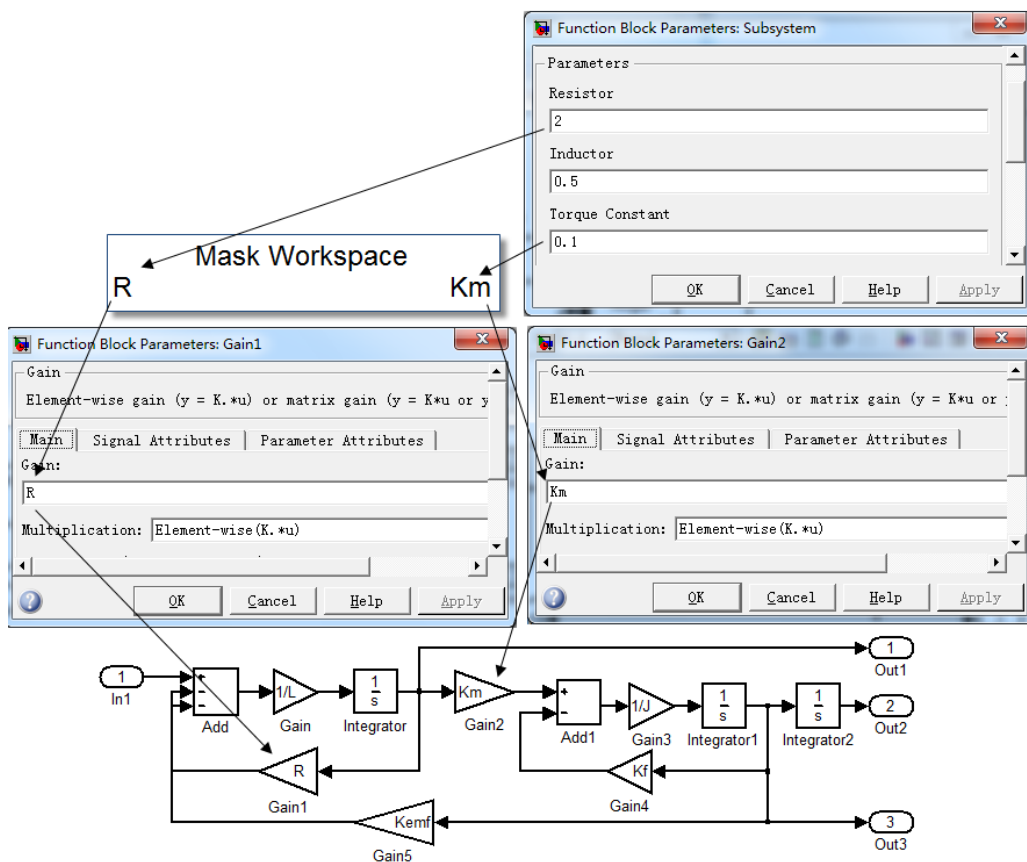


图 2.2.37 参数传递关系

- Initialization 页面

Initialization commands 窗口用于输入用于初始化该模块的 MATLAB 命令。如果在该页面中输入如下命令，同样可以完成参数的赋值工作，如图 2.2.38 所示。

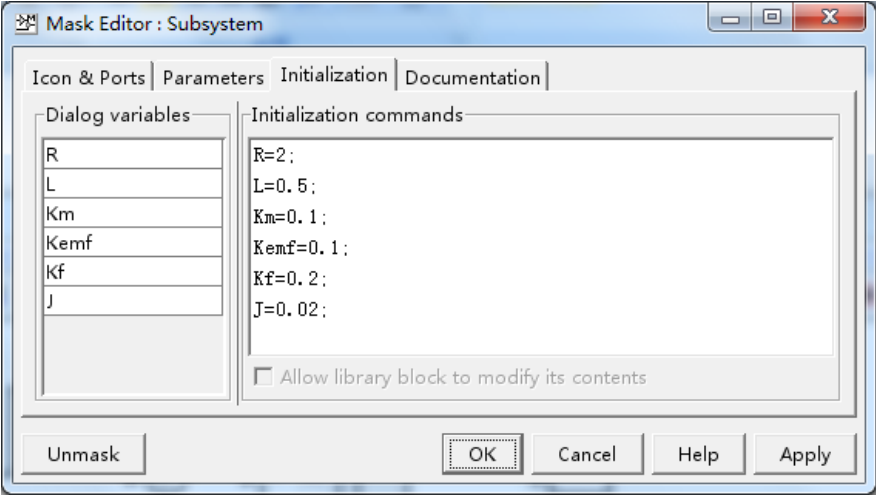


图 2.2.38 Initialization 页面

但是这种方式应慎用，这会导致上文中所讲的 Annotation、M 文件等赋值方法统统失效，这是由于最终在模块执行之前它都会执行此初始化命令，将参数的值重新定义为初始化页面中设置的量。

- Documentation 页面

在此页面中可以注释封装的类型、描述、帮助等信息。

- Mask type 文本框可写入对模块的简单描述
- Mask description 文本框用于详细描述模块的功能：
- Mask help 文本框用于指定模块的帮助文档

用户可根据需要编辑相关信息，如图 2.2.39 所示。

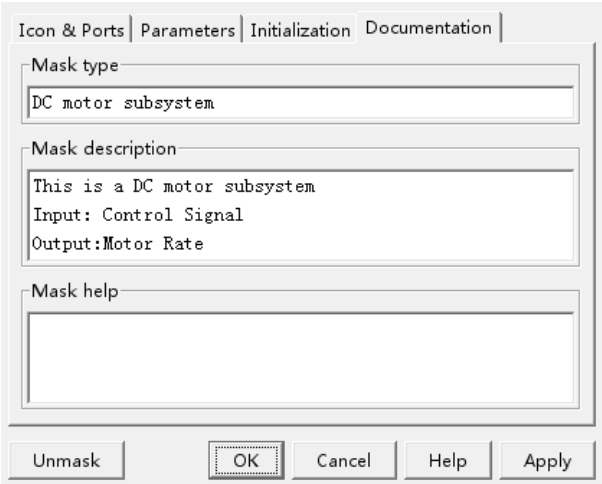


图 2.2.39 Documentation 页面

2.2.4 添加模块到库浏览器及知识产权保护

在设计大型模型时，通常会遇到多次重复使用一个子系统模块或团队的其他成员需要使用你所搭建的子系统的情况，用户可以建立自定义的模块库并添加到模块浏览器中，将这些子系统归类整理，分门别类地管理，便于后期维护与利用。

1. 建立自定义模块库并添加到库浏览器

- 建立自定义模块库

在 Simulink 库浏览器窗口，选择菜单项 **File**→**New**→**Library**，打开库编辑窗口。打开电机模型，将封装后的子系统拖入库编辑窗口，并保存为 `DC_motor_subsystem_lib`。如图 2.2.40 所示：

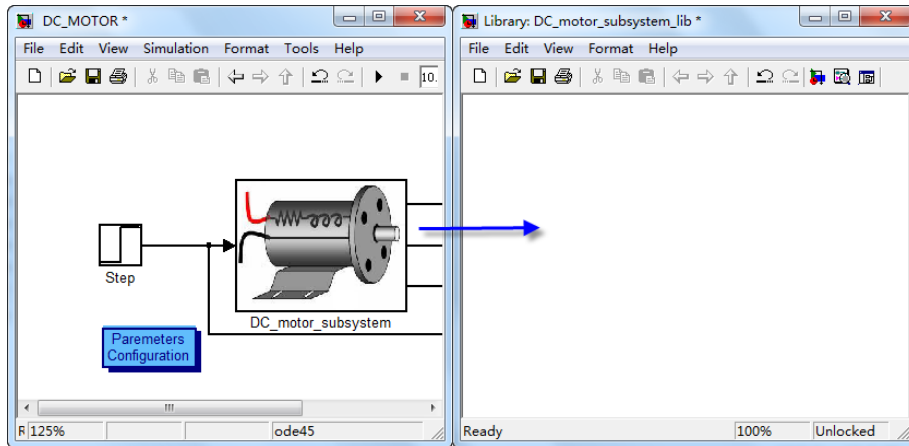


图 2.2.40 建立自定义模块库

- 将自定义模块库添加到库浏览器中

将自定义模块库添加到库浏览器中，可以在以后的工作中更方便快捷的调用。**MATLAB** 中已经为用户提供了添加自定义模块库的相关函数，用户可以编写如下 **M** 函数代码：

```
function blkStruct = slblocks
blkStruct.Name = ['DC_motor_subsystem_lib'];
blkStruct.OpenFcn = 'DC_motor_subsystem_lib';
Browser(1).Library = 'simulink'; Browser(1).Name = 'Simulink';
Browser(1).IsFlat = 0;
Browser(2).Library = 'DC_motor_subsystem_lib';
Browser(2).Name = 'DC_motor_subsystem_lib';
Browser(2).IsFlat = 0;
```

将其保存为 `slblock.m` 并保存到 `DC_motor_subsystem_lib.mdl` 所在的目录下，如图 2.2.41 所示。

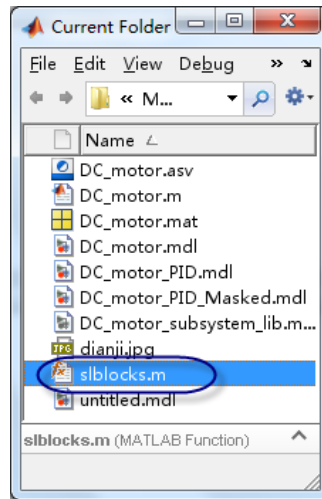


图 2.2.41 M 文件存储路径

打开模块库浏览器，自定义的模块库已经成功添加。该模块可以直接使用，如图 2.2.42 所示。

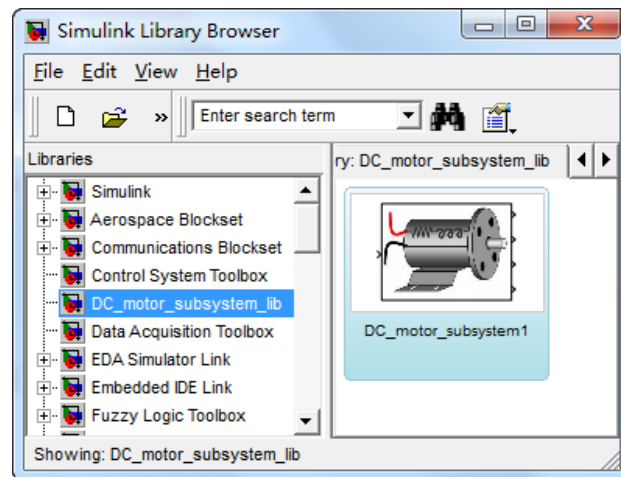


图 2.2.42 添加到模块库浏览器

2. 知识产权保护

有时用户需要把自定义的模块库提供给别人使用，但是却不想让别人看到或修改子系统的底层模块结构。通过以下设置可以在不影响使用的条件下禁止他人访问底层模块，有效保护了用户的个人资料和知识产权。

- 在将封装后的子系统拖入库编辑窗口之前右键点击子系统，选择 `subsystem parameters`，如图 2.2.43 所示。

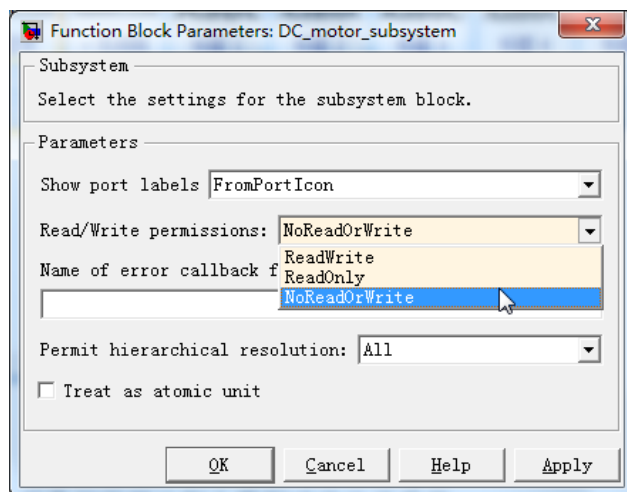


图 2.2.43 Block Parameters 页面

- 在 Read/Write Permissions 下拉菜单中选择 NoReadOrWrite，点击 OK 确认
- 将子系统模块拖入库编辑窗口

在库编辑窗口中右键单击模块，可以发现这时 Look Under Mask 选项变成了灰色，禁止访问底层模块。如图 2.2.44 所示：

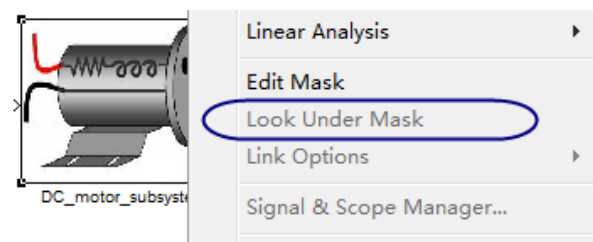


图 2.2.44 保护后的模块

2.2.5 数据格式与输入输出

Sources 或 Sinks 子库中的现成信号源或输出模块可能不能满足实际应用，因此用户可以选择需要选用 From Workspace 及 To Workspace 模块，自行定义模型的输入及输出。

用户可以通过上述模块导入 MATLAB 工作空间的数据，也可以把输出信号保存到工作空间。这项功能让用户能够使用由标准或自定义的 MATLAB 函数产生的数据作为输入信号和图形，使仿真更具多样性。

1. 把数据导出至工作空间

1.新建一个演示模型，向模型中添加“sink”库中的“To Workspace”模块，该模块位于如图 2.2.45 所示的位置。

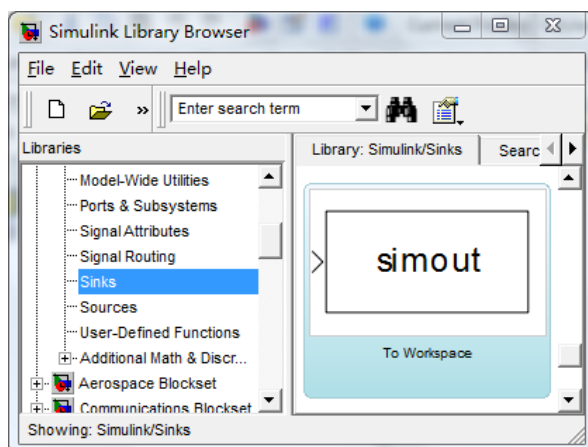


图. 2.2.45 simout 模块

打开 To Workspace 模块的参数设置页面，其中 Data 栏显示的“simout”即变量名，用户可根据需要自行更改。

为便于“From Workspace”访问，可以将“To Workspace”模块的 save format 设置为“Array”，如图 2.2.46 所示。

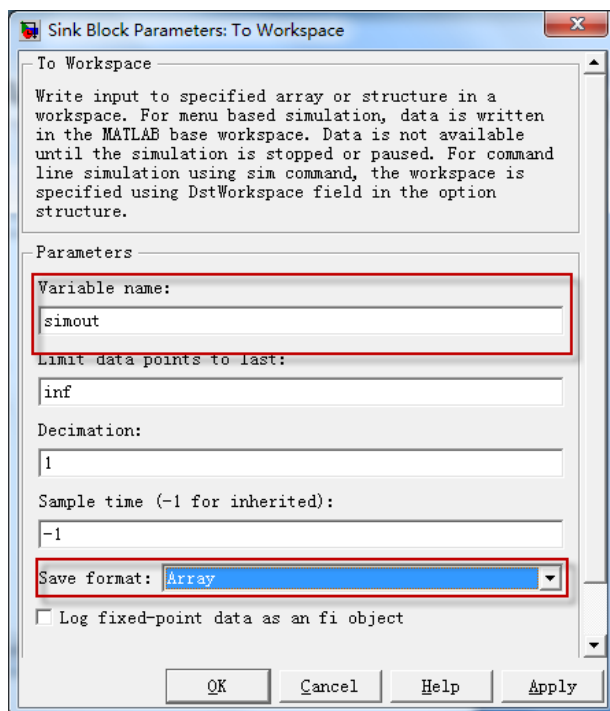


图 2.2.46 设置 simout 模块

2. 向模型中添加“source”库中的“signal builder”模块，如图 2.2.47 所示。

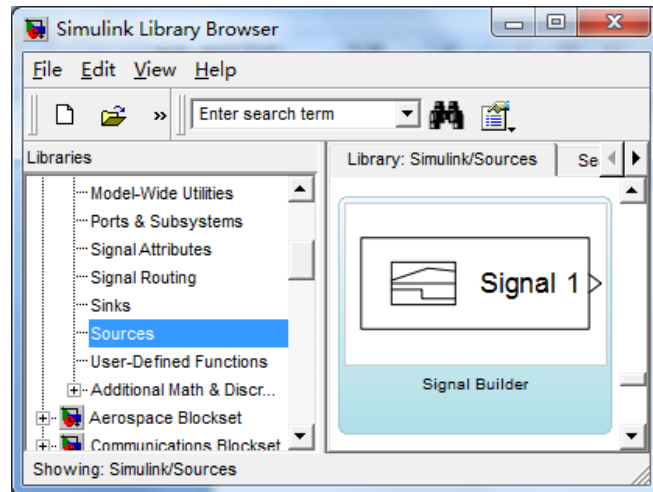


图 2.2.47 signal builder 模块

打开该模块的设置页面，用户可以方便的通过其 GUI 界面画出任意需要的信号波形。这里画出一个从第 3 秒起持续 4 秒的矩形脉冲，如图 2.2.48 所示。关于该模块的更详细用法不是本书重点，如有兴趣可参看帮助文档。

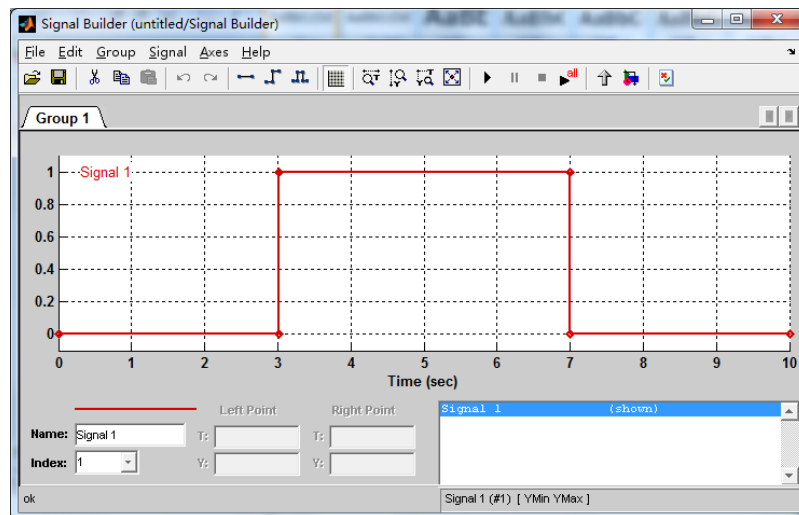


图 2.2.48 信号波形

3.向模型中添加“gain”模块和“scope”模块，并按图 2.2.49 连接模块：

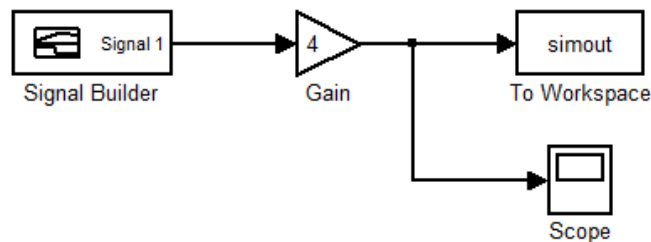


图 2.2.49 向工作空间传递数据

点击工具栏上的“▶”按钮运行仿真，MATLAB 的工作空间中会显示变量如图 2.2.50 所示。

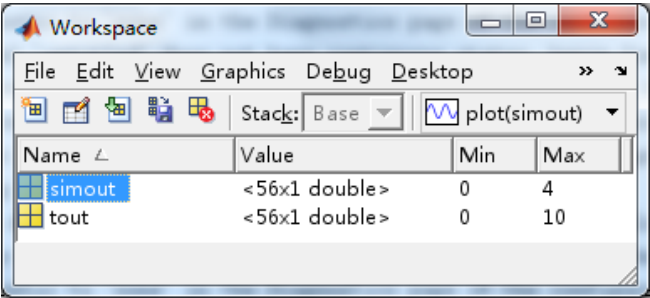


图 2.2.50 工作空间的数据

示波器中显示波形如图 2.2.51 所示：

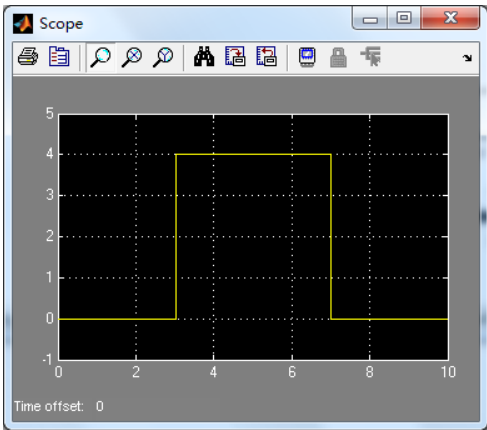


图 2.2.51 仿真波形

Simulink 仿真中的数据已经导入了工作空间，用户可以根据需要对这些数据做进一步处理。

2. 从工作空间导入数据

1.向模型中添加“source”库中的“From Workspace”模块代替“signal builder”，即可用于从工作空间导入数据，该模块位于如图 2.2.52 所示的位置。

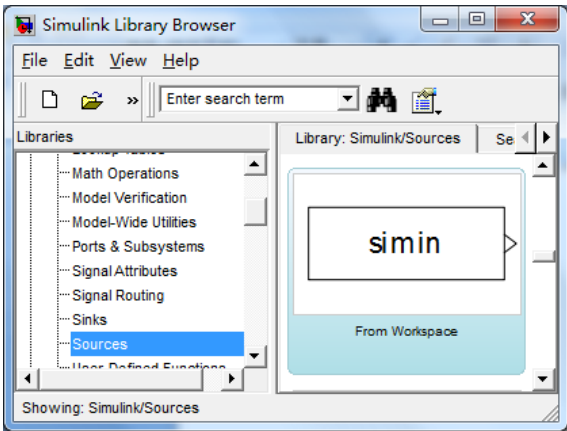


图 2.2.52 simin 模块

打开 From Workspace 模块的参数设置页面，其中 Data 栏显示的”simin”即变量名，用户

可根据需要自行更改。如图 2.2.53 所示：

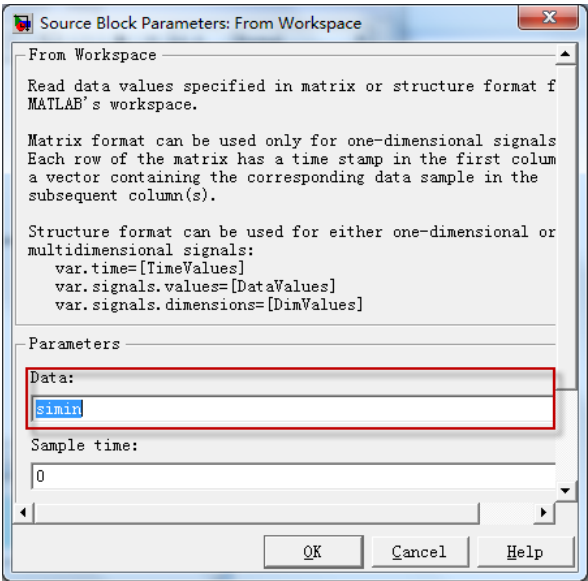


图 2.2.53 simin 设置

2. 按图 2.2.54 连接模块：

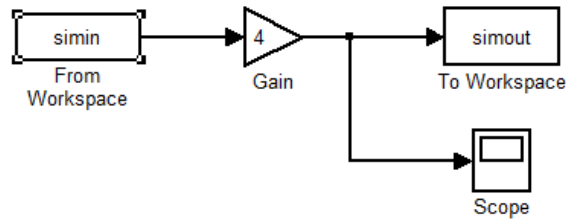


图 2.2.54 从工作空间读数据

“From Workspace” 模块调用工作空间的数据时，是遵循严格的格式的，一般是矩阵形式，而且输入矩阵的列数必须比输入信号多一列，Simulink 会自动把首列数据当做时间向量。根据此规则，需要在工作空间内生成一个待输入信号：

```
>> simin=[tout simout];生成 2 列的矩阵%
```

用户可以打开 workspace 中的变量 simin 查看其数据存储结构，如图 2.2.55 所示。

simin <56x2 double>

	1	2
1	0	0
2	0.2000	0
3	0.4000	0
4	0.6000	0
5	0.8000	0
6	1	0
7	1.2000	0
8	1.4000	0
9	1.6000	0
10	1.8000	0
11	2.0000	0
12	2.2000	0
13	2.4000	0
14	2.6000	0
15	2.8000	0
16	3.0000	0
17	3.0200	0
18	3.0200	4

图 2.2.55 数据存储结构

3. 实现外部信号对电机的控制

回到电机模型，我们用一个外部信号替代阶跃信号控制电机，用户可以通过这个外部信号实现对电机转速的控制。

向模型中添加“From Workspace”模块替代“Step”模块，并重命名为 Vin_Control Signal。如图 2.2.56 所示：

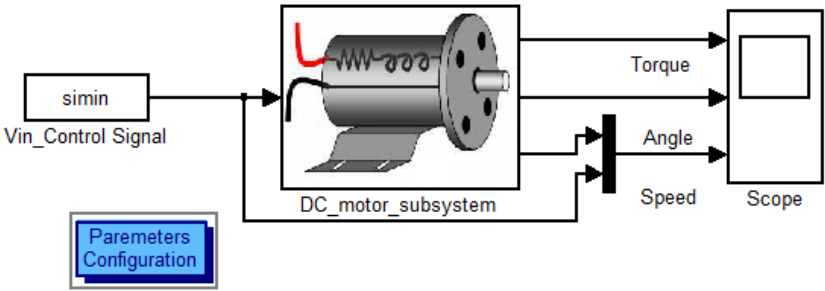


图 2.2.56 添加 simin 模块

这里的 From Workspace 模块仍然使用上文中生成的变量 simin 作为控制信号，Data 栏设置变量名为 simin”即变量名，如图 2.2.57 所示。

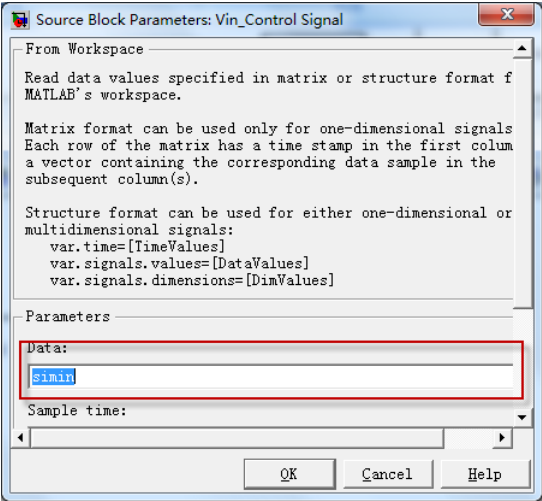


图 2.2.57 设置 simin 模块

运行仿真后得到的结果如图 2.2.58 所示：

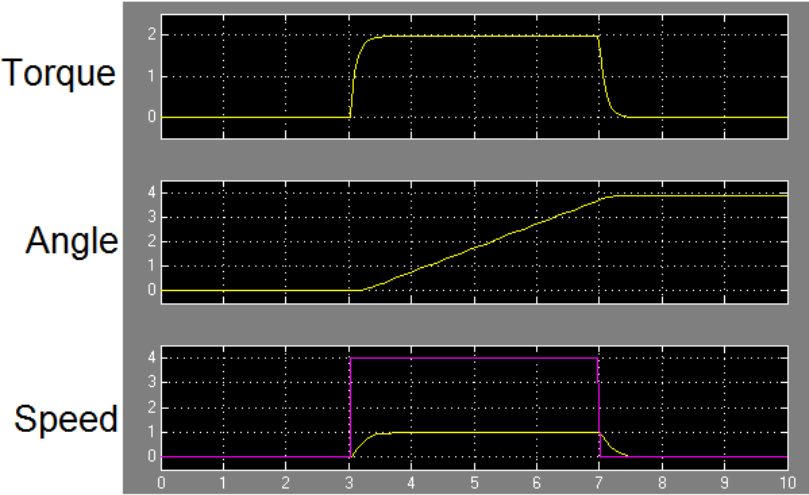


图 2.2.58 仿真结果

由仿真结果可知，系统响应曲线变化的总体趋势是跟随控制信号相吻合的，说明外部信号对电机转速的控制是有一定效果的。但系统响应速度还不理想，且其响应幅度远远没有达到控制信号的要求，需要通过进一步矫正才能符合需要。

2.2.6 PID 控制

在上文的仿真结果中可以看到：输出电机转速 $\omega(t)$ 只是大体上跟随控制信号的变化趋势，而并没有严格的达到控制信号所要求的转速大小。这是由于模型还缺少一个反馈控制模块，PID 模块能很好的完成这个工作。

PID（比例积分微分）英文全称为 ProportionIntegration Differentiation，它以结构简单、稳定性好、工作可靠、调整方便等特点成为工业控制的主要技术之一。

PID 控制器由比例部分，积分部分和微分部分构成（如下图所示）。控制器的输入端是被控系统的控制信号与被控量间的误差，误差信号分别通过比例、积分、微分环节，并与相应的系数相乘，输出的 PID 控制器信号即为这三个部分输出信号之和，最终 PID 控制器输出信号与原始控制信号做差后控制被控量，最终形成完整的反馈回路。因此，要确定一个 PID 控制器只需要确定其比例、积分、微分的系数： K_p 、 K_i 、 K_d ，如图 2.2.59 所示。

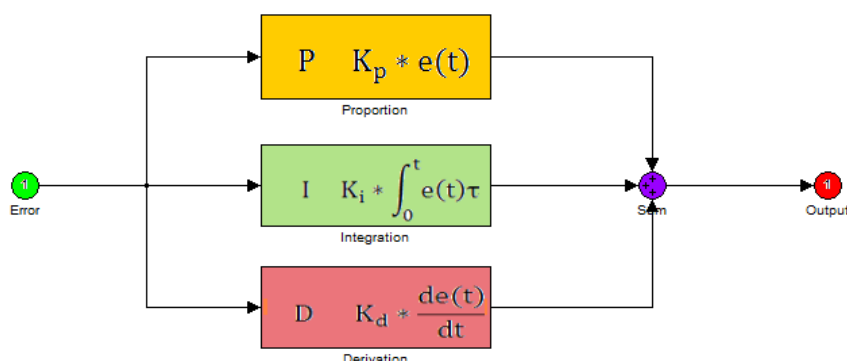


图 2.2.59 PID 控制器结构

PID 控制器的参数设定是控制系设计的核心内容，但是传统的设计方法存在不少缺陷，在实际应用中会遇到很多问题：

1. 传统的设计方法中，参数 K_p 、 K_i 、 K_d 的设定主要依赖以往的工程经验，对控制器的参数进行手动调节，如果以前的项目中有过类似的控制器，就可以直接在控制系统硬件上进行手动微调，这样工作量很小。但如果被控对象是一个不稳定系统，手动调节就会变得比较复杂，因为很可能由于不好的参数设定和控制算法，使系统硬件面临烧毁的危险。

2. 积分饱和现象会使控制品质变差。所谓积分饱和是指：对于具有积分运算的控制器，只要系统控制信号与被控量间存在偏差，PID 控制器的输出就会不停的变化来矫正偏差，但是当系统出现某种问题导致偏差一时无法消除时，控制器还试图矫正这个偏差，这样经过一段时间后会造造成控制器进入深度饱和状态。进入积分饱和的控制器只有等被控量偏差反向后才能逐渐消除，重新恢复控制作用。因此，在涉及控制器时，要考虑到抗积分饱和的问题。

3. 微分近似。由于纯微分作用在实际中是不可能实现的，设计控制器时就要考虑如何对微分进行准确的近似。

4.在设计控制器时,经常会遇到无法确定需要使用哪种结构的问题,如:P、PI、PD、PID;需要确定使用一维还是二维的算法;需要确定是否需要输出饱和和设置;需要考虑抗积分饱和特性;特别是系统含有两个以上控制器时,还要考虑到算法切换时的瞬态稳定性。

5.如果要把PID控制器算法生成代码应用到实际系统中去,还需要对算法和参数做离散化,定点化处理。

使用PID模块,可以很好的解决上述问题。下面通过为上文提到的直流电机模型添加PID控制模块介绍其使用方法。

打开电机模型,装载变量到工作空间,添加“math operations”库中的“sum”模块到模型中,如图2.2.60所示。

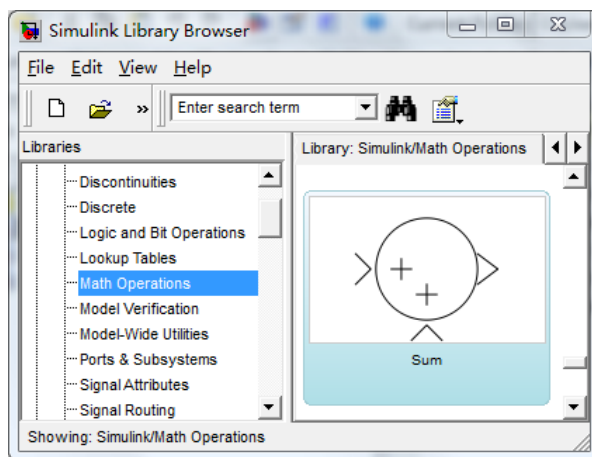


图 2.2.60 sum 模块

双击模块打开参数设置对话框,将其符号设置为“|+-”,如图2.2.61所示。

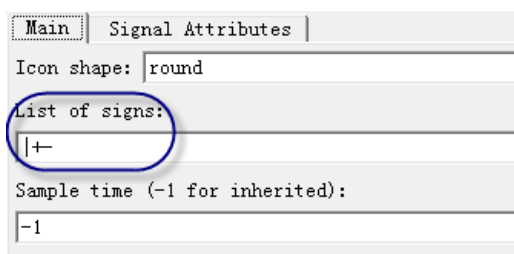


图 2.2.61 sum 设置

添加“continuous”库中的“PID”模块到模型中,该模块位于如图2.2.62所示的库中:

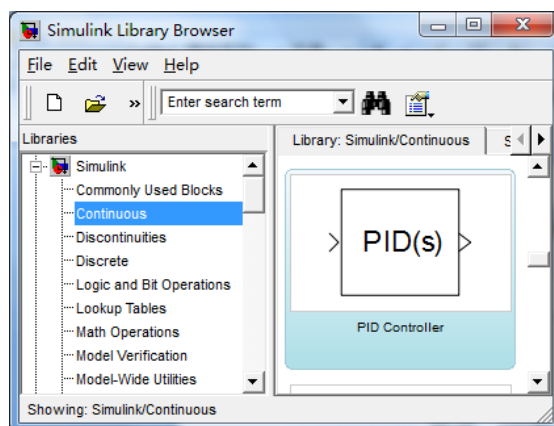


图 2.2.62 PID 模块

连接模型如图 2.2.63 所示：

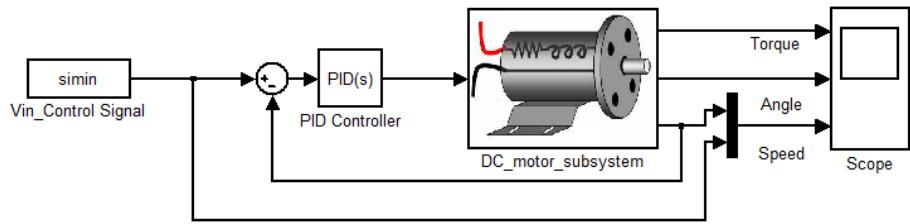


图 2.2.63 添加 PID，sum 模块

1. 用 PID Tuner 设计 PID 控制器

双击打开 PID 模块的设置页面，如图 2.2.64 所示：

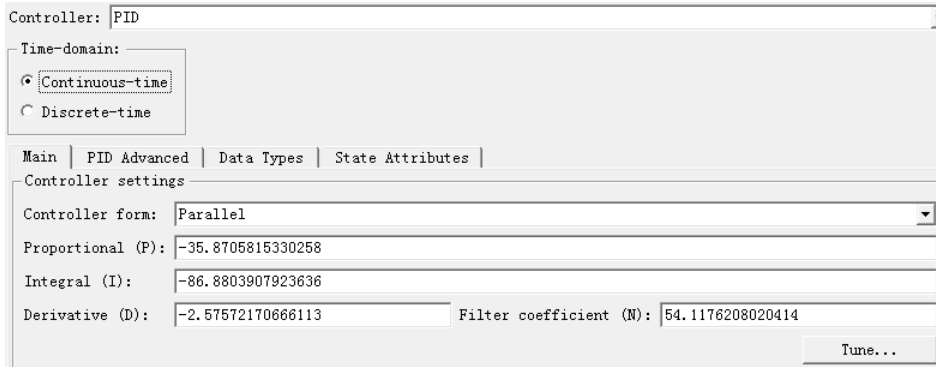


图 2.2.64 PID 模块设置页面

- Controller 通过 Controller 下拉菜单中的 P、I、PI、PD、PID 选项轻松的切换控制器的结构模式。
- Time-domain 选项可以快速的在连续域和离散域间切换。
- Controller Setting 选项可以为控制器选择工作状态“并行”或“理想型”具体区别可参考 help 文件。
- PID Advanced 页面中可以对控制器做一些高级设置，比如输出饱和和限制，抗积分饱和和特性，追踪模式等。如图 2.2.65 所示：

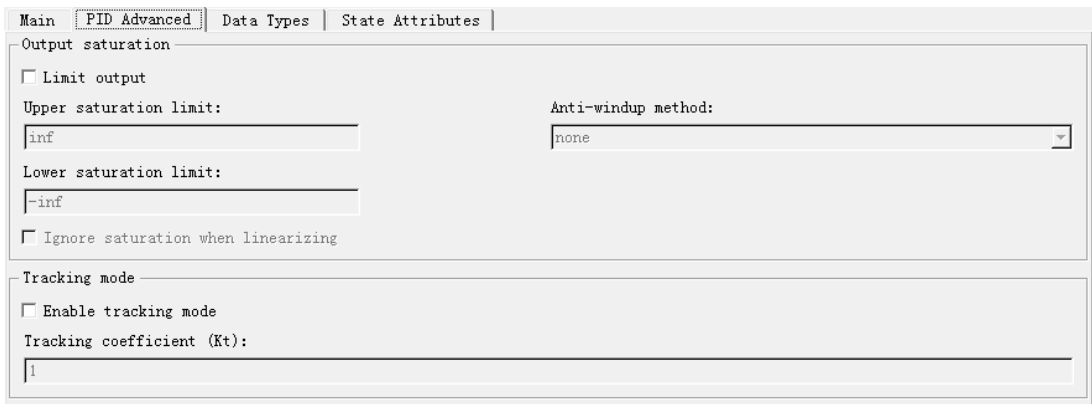


图 2.2.65 PID Advanced 页面

- Limit output 勾选复选框使能输出饱和和限制
- Anti-windup method 选择抗积分饱和模式

- **Enable tracking mode** 勾选复选框使能追踪模式，使算法切换时更加平滑。

下面是最重要的一步：设置 P、I、D 的系数。传统的手动调节法不仅费时费力，还有可能对系统造成破坏，并且你永远不知道自己设计出的参数是否为最优。

而基于规则的调节法无法应用于开环的不稳定系统，也不能用于高阶系统和有延迟的系统，且对用户的控制理论背景要求很高，不易掌握。

Simulink 提供了一个全新的 GUI 调节算法可以方便的完成这些复杂的工作，自动调节控制器参数以达到所期望的性能指标，通过简单的滚动条操作完成微调的功能。

点击参数设置页面上 **Controller settings** 中的“Tune”按钮打开 GUI 界面(点击 show parameters 可以完整的显示参数)。如图 2.2.66 所示：

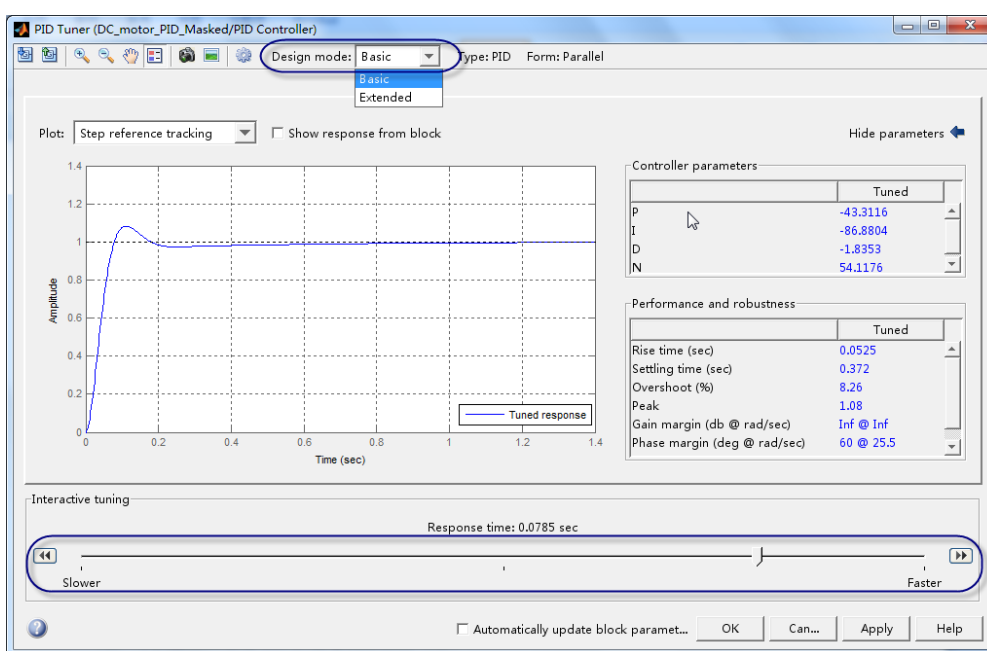


图 2.2.66 PID 自动调节界面

PID Tuner 会自动在系统缺省的工作点处对模型进行线性化处理，设计出控制器的参数。用户可以直观的通过 GUI 界面看到系统的响应。

界面的下方有一个滚动条工具，通过拖动滚动条可以调节系统的响应时间，这里我们将响应时间调节到 0.0785s。通过 GUI 界面可以看到系统响应速度明显提高，并出现了一个小的过冲。

如果在界面上方的 **Design mode** 下拉菜单中选择“extended”，会出现额外的两个滚动条“带宽”和“相位裕量”，通过拖动它们可以改变系统响应的快速性和平稳性。如图 2.2.67 所示：

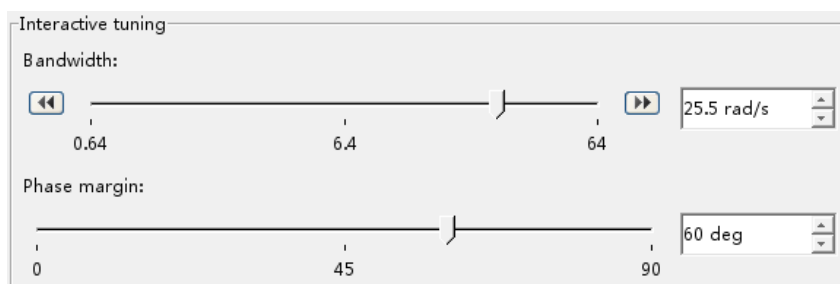


图 2.2.67 调节滑块

在显示阶跃响应曲线的区域内单击鼠标右键，根据需要选择 **characteristics** 中的一项或

几项，会在响应曲线上添加相应的蓝点来表示这些特征点：

- Peak Response 峰值
- Setting Time 稳定时间
- Rise Time 响应时间
- Steady State 稳定状态

左键点击这些蓝点会显示其详细信息，如图 2.2.68 所示：

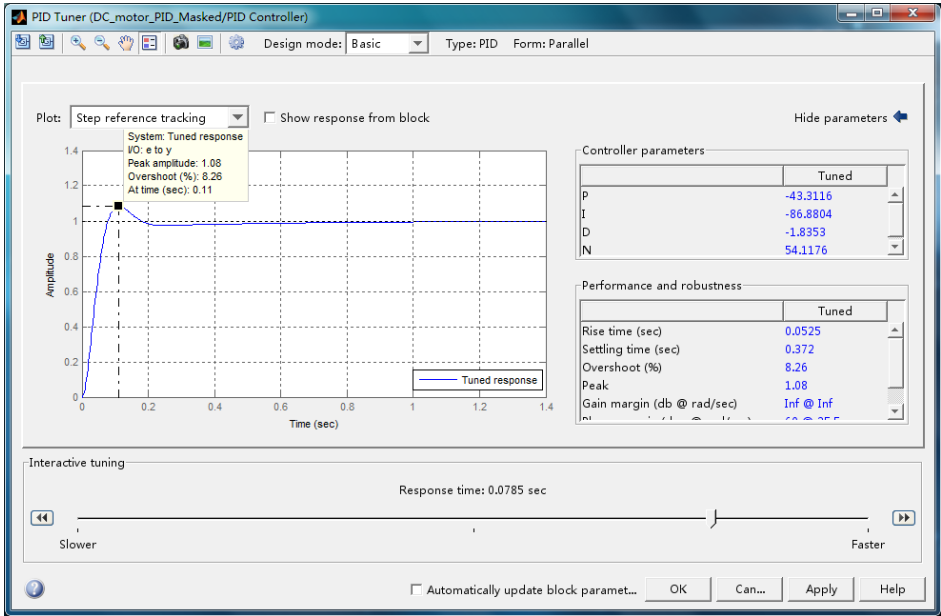


图 2.2.68 定位特征点

当用户得到满意的响应曲线后，点击 Apply 按钮，PID Tuner 自动设计的参数就已经写入了参数设置框中。如图 2.2.69 所示：

The figure shows the 'Controller settings' window. It contains the following fields:

- Controller form: Parallel (dropdown menu)
- Proportional (P): 7.61876506172232
- Integral (I): 32.0768093880999
- Derivative (D): 0.0885540903015234
- Filter coefficient (N): 23.3510675715948

At the bottom right is a 'Tune...' button.

图 2.2.69 PID 参数

运行加入 PID 控制器后的直流电机模型。可以看到，响应速度得到了很大提高，被控量转速 $\omega(t)$ 基本上已经和控制信号 V_{in} 基本吻合了。如图 2.2.70 所示：

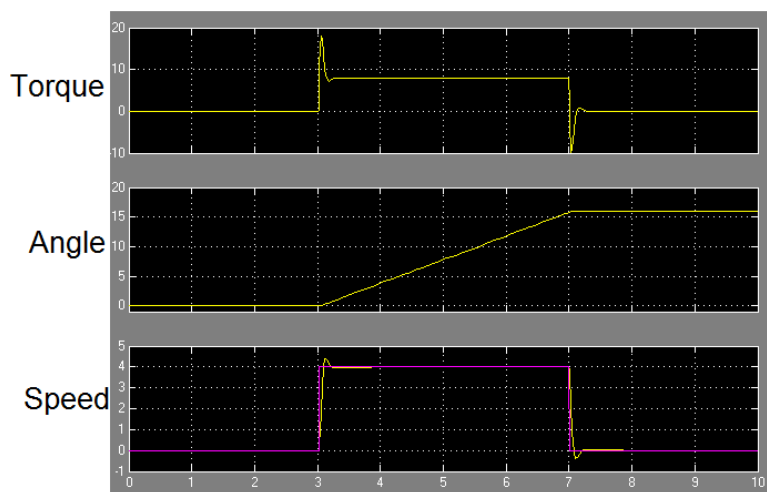


图 2.2.70 PID 调节后的仿真结果

2. 用波特图法设计 PID 控制器

Simulink 的 Control Design 工具提供了更多设计 PID 控制器的方法。这里以波特图设计法为例介绍如何用 Control Design 工具设计 PID 控制器。

打开添加过 PID 模块的电机模型，选择 Tools→Control Design→Compensator Design 打开 Control and Estimation Tools Manager。选择 Simulink Compensator Design Task 节点，点击 Select Block，确定将要调节的模块。如图 2.2.71 所示：

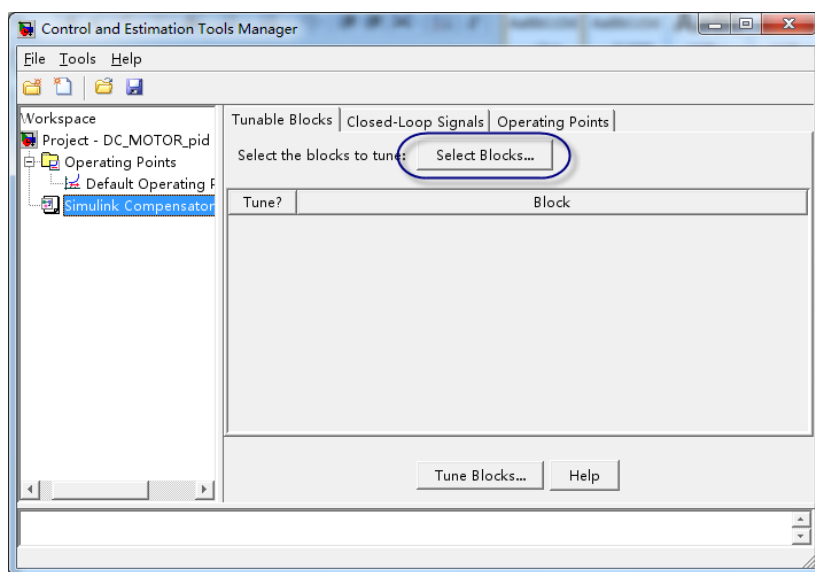


图 2.2.71 控制与估计管理工具界面

在弹出窗口中勾选 Tune? 复选框，点击确定，如图 2.2.72 所示：

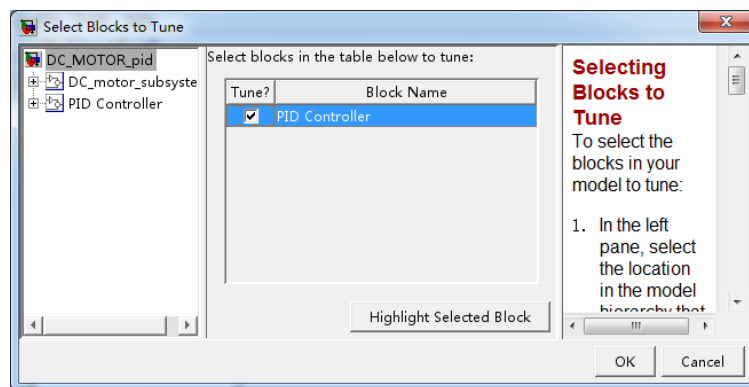


图 2.2.72 选择待调节的目标模块

设计 PID 控制器前，要先为系统指定需要调节的闭环系统，标识其输入输出信号。在 Vin_Control Signal 模块后的信号线处单击右键，选择 Linearization points → Input Point；在 DC_motor_subsystem 模块后的 Speed 信号线处单击右键，选择 Linearization points → Output Point。这将在信号线处添加 “ \downarrow ” 和 “ \uparrow ”，它们分别标识该闭环的输入和输出，如图 2.2.73 所示：

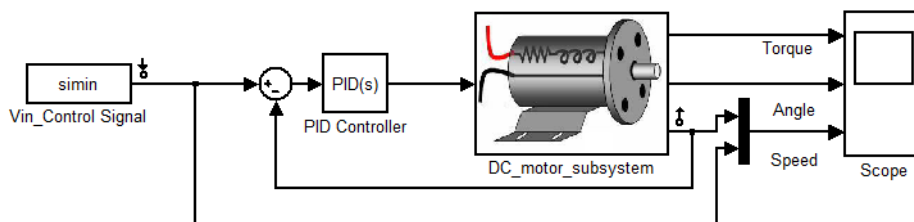


图 2.2.73 标记系统输入输出

在 Control and Estimation Tools Manager 中点击 Tune Blocks 打开 Design Configuration Wizard，直接点击 Next，开始设置。

Wizard 的第一步是选择调节控制器所用到的设计图形，这里使用其默认配置并点击 Next；第二步选择分析响应曲线所用到的图形的类型，在 Analysis Plots 区域选择 Step 作为图形类型，在 Content in Plots 区域，勾选 1，用来画出从 Vin_Control Signal 到 DC_motor_subsystem 的闭环阶跃响应曲线。如图 2.2.74 所示：

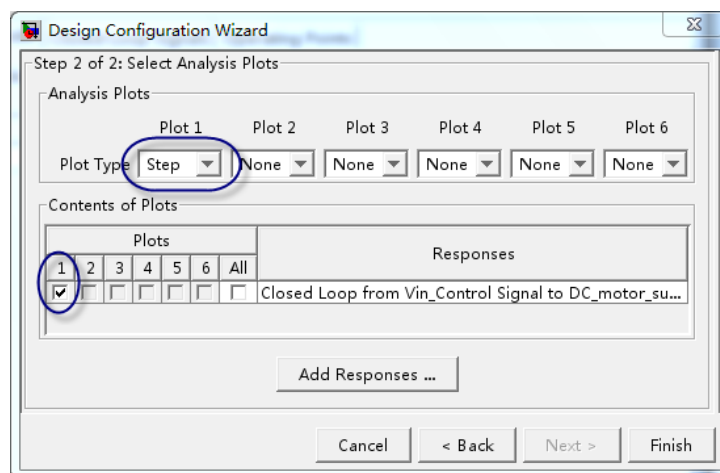


图 2.2.74 设置 step 类型

点击 Finish 退出 Wizard，并画出阶跃响应曲线，如图 2.2.75 所示。

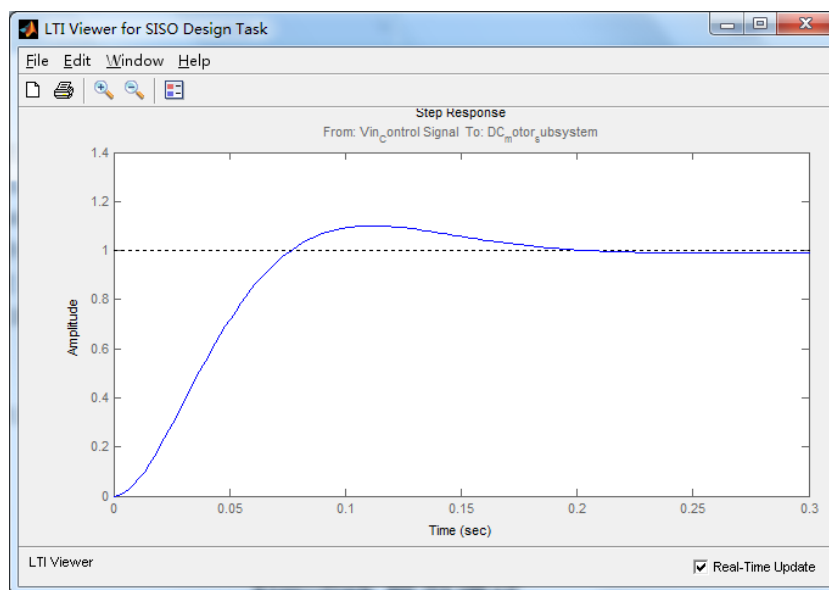


图 2.2.75 阶跃响应

这时还会弹出一个空白的 SISO Design for SISO Design Task 窗口，将在后面用到，不要关闭。

在 Control and Estimation Tools Manager 的 SISO Design Task 节点中打开 Graphical Tuning 页面，将 Plot Type 中 Plot1 的类型设置为 Open-Loop Bode。如图 2.2.76 所示：

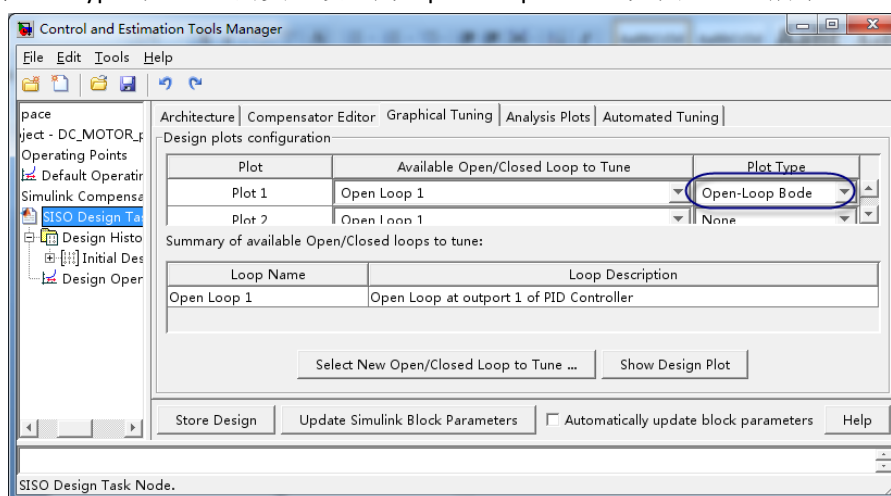


图 2.2.76 设置为开环波特图

完成上述操作后点击 Show Design Plot 会在 SISO Design for SISO Design Task 窗口绘制出开环波特图。如图 2.2.77 所示：

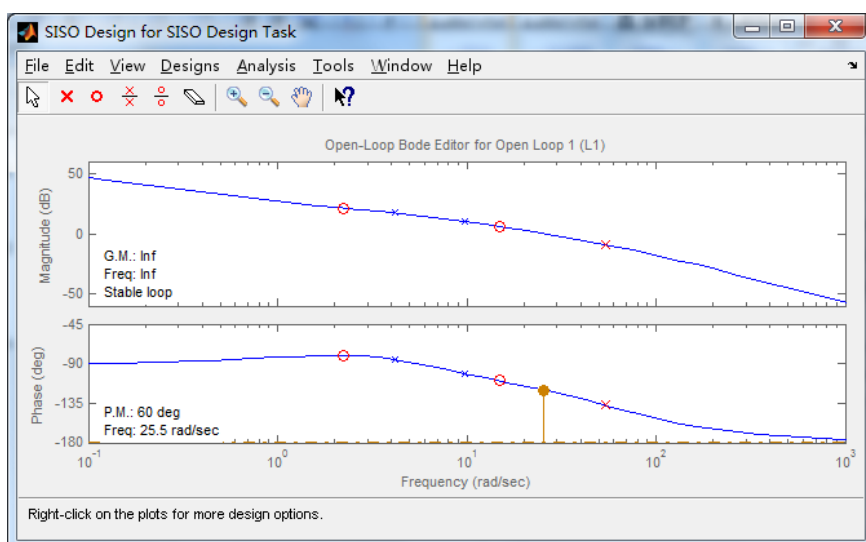


图 2.2.77 调节界面

图中的红色的圆圈和叉均可用鼠标拖动，曲线形状也会随之变化，同时，LTI Viewer for SISO Design Task 窗口中的阶跃响应曲线也会同步更新。例如，增大波特图中的增益，阶跃响应时间会缩短。

使用与 PID Tuner 设计 PID 控制器法相同的设置方式，同样可以显示调节得到的阶跃响应曲线的特征点数据。如图 2.2.78 所示：

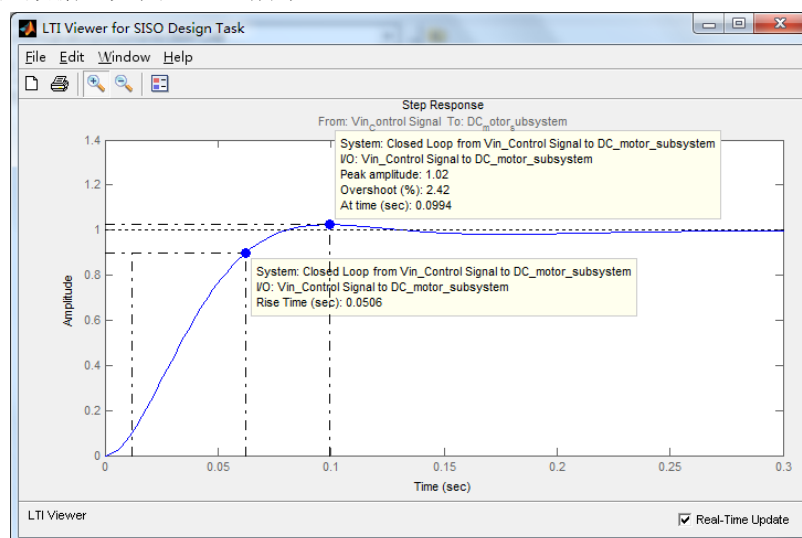


图 2.2.78 定位特征点

当用户确定阶跃响应曲线符合要求时，可在 Control and Estimation Tools Manager 的 SISO Design Task 节点中点击 Update Simulink Block Parameters。这样，设计的 PID 控制器参数即被写入到了 PID 模块中，设计过程完成。

运行模型，观察电机转速响应曲线，如图 2.2.79 所示：

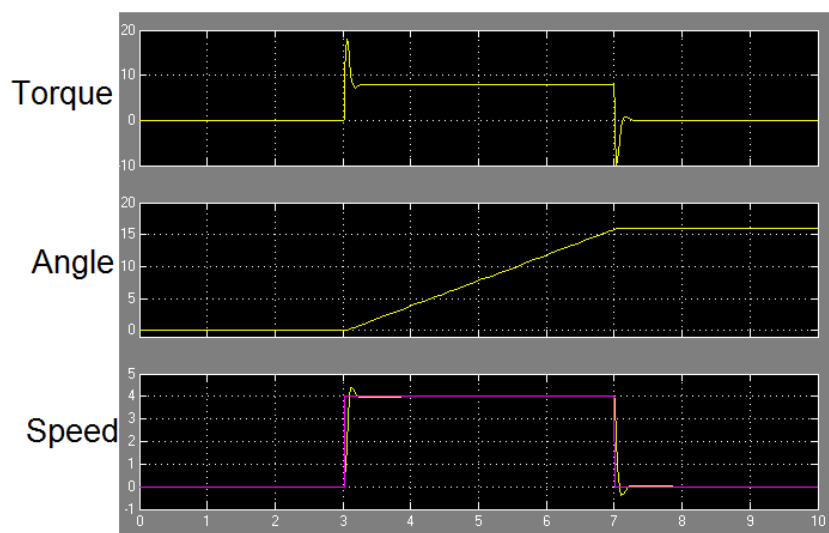


图 2.2.79 调节后的仿真结果

3. 离散域 PID 控制器

要把控制器算法应用到实际的控制器中，还需要将其转化到离散域，并且要考虑到实际处理器的位数和算法定标是否吻合。如果离散化与定点化做的不好，很可能会导致一个在连续域性能不错的控制器转换到离散域、定点化后完全不能达到设计指标。

双击用上文方法设计好的 PID 模块，打开其设置页面，在 Time Domain 中选择 Discrete-Time，采样时间设为 0.1 秒，点击运行模型，可以看到系统已经变得不稳定了，如图 2.2.80，2.2.81 所示。

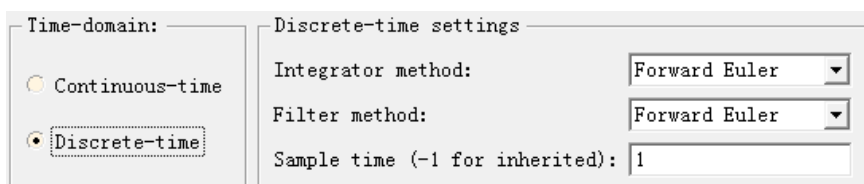


图 2.2.80 设置为离散 PID

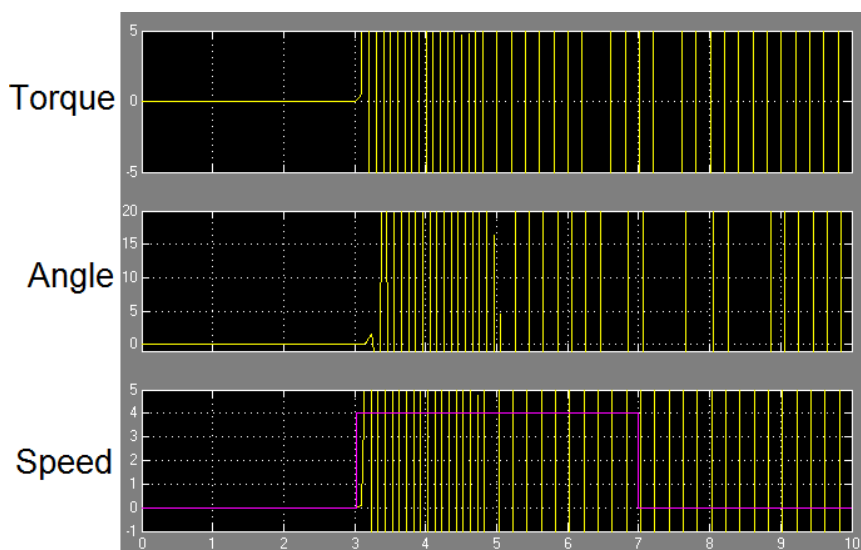


图 2.2.81 仿真结果

这是由于我们在用 PID Tuner 设计控制器时，为控制器设置的带宽为 25.5rad/s（如图....所示），将其除以 2π 转化为 4.11Hz，而采样带宽为 10Hz，不到系统带宽的 3 倍。在实际工程应用中，采样带宽应该为系统带宽的 5~10 倍，可根据硬件系统的采样率来设置。这里假设为 0.01 秒，点击 Tune 按钮打开 PID Tuner。按照本节“1. 用 PID Tuner 设计 PID 控制器”中介绍的方法调节出与图 2.2.66 相似的响应曲线，选择 Extended 模式，可以发现这时的系统带宽为 2.73Hz，采样带宽是系统带宽的 30 多倍，点击确认，更新 PID 参数。这时如果手动调节到更大的系统带宽，则又会导致系统稳定性变差。如图 2.2.82 所示：

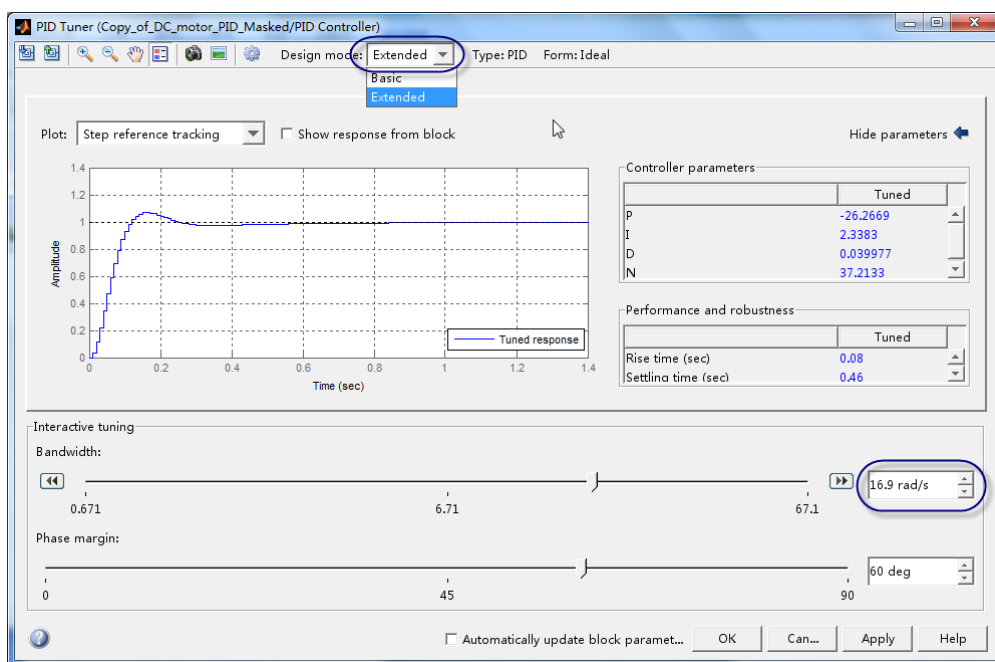


图 2.2.82 PID 调节界面

点击 OK 确认，更新 PID 参数，运行模型。这时系统再次变得稳定了，如图 2.2.83 所示：

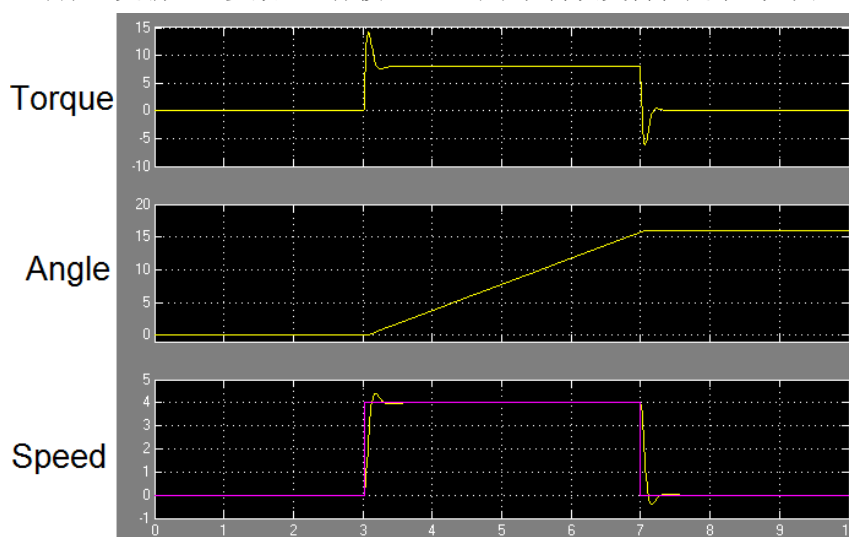


图 2.2.83 离散 PID 控制后的仿真结果


2.3 Simulink 模型调试

Simulink 为用户提供了功能强大的模型调试工具：**Simulink Debugger**。它是一个交互式的调试 Simulink 模型的工具。该工具可以设置断点，控制仿真的执行，显示模型的运行信息。通过使用调试器，用户可以一步一步的仿真模型，可以在任意步骤处单步运行，发现模型中可能存在的问题，或对其进行修改。

Simulink Debugger 有图形界面(GUI)和命令行界面两种运行方式：命令行界面模式可以实现调试器的所有功能，但需要用户事先了解调试命令；图形用户界面模式使用方便简洁，可以实现调试器的绝大部分功能，这里重点介绍图形用户界面模式。

2.3.1 图形界面调试

1. 启动调试器

图形界面调试器可以通过模型编辑窗口中的 **Tools→Simulink Debugger** 命令启动，也可以通过单击其工具栏上的图标 “” 启动。调试器界面如图 2.3.1 所示：

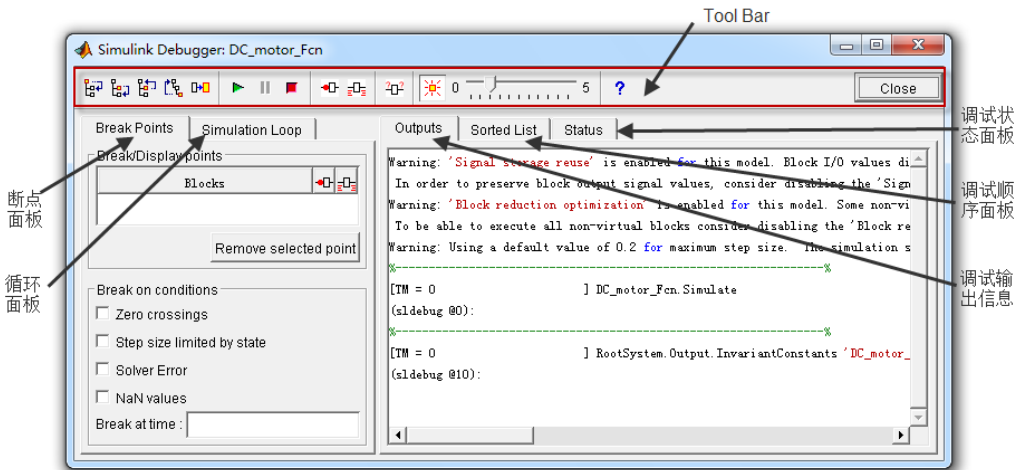


图 2.3.1 调试界面

2. 图形界面调试器介绍

1. 工具栏

工具栏位于调试器窗口的最上方，从左至右依次为以下指令：“进入当前方法”、“跳过当前方法”、“跳出当前方法”、“在下一个方法开始时返回第一个方法”、“开始/继续运行仿真”、“暂停调试”、“终止调试”、“在选中的模块前设置断点”、“执行时显示选中模块的 I/O 信息”、“显示选中模块的当前 I/O 信息”、“开启/关闭动画”、“动画延时”、“帮助信息”和“关闭调试器”，如图 2.3.2 所示。

这里提到的“方法”是指 Simulink 在仿真过程中逐时间步长对模型求解时所用到的运算方法。其实每一个模块都是由若干个这样的“方法”构成的，运行模型也就等价于对各

个模块中的“方法”按一定的逻辑顺序逐时间步长运行。



图 2.3.2 调试工具栏

2. 断点页面

调试器提供了两种断点设置方法：一是普通断点，另一种是条件断点。当模型运行到用户设置的普通断点时，会无条件停止；而遇到条件断点时，则会判断是否达到指定的条件（过 0，除 0，限步长等），是则停止，否则继续运行。用户可以在断点页面中选择断点产生的条件。

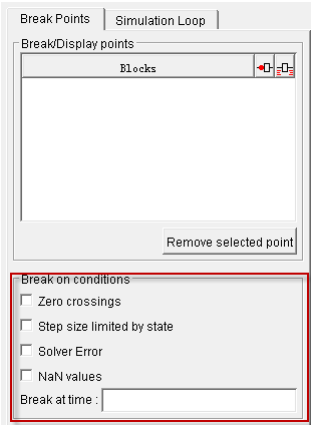


图 2.3.3 断点设置页面

条件断点：如图 2.3.3 所示。

- (1) **Zero crossing**: 模型在遇到过零点检测时产生断点。
- (2) **Step size limited state**: 在状态手动条件约束是产生断点。
- (3) **Solver error**: 求解错误时产生断点。
- (4) **NaN value**: 仿真过程中遇到无限大，或溢出时产生断点。
- (5) **Break at time**: 指定产生断点的具体时刻。

在断点产生条件选项框上面，调试器提供了一个断点信息显示框，它能显示当前系统中已设置断点的模块，并且能设置是否显示该断点的输入/输出。通过 **Remove Selected Point** 按钮可以取消相应的断点。

断点是非常有用的功能。特别是当用户知道模型中的某处可能存在问题时，通过设置适当的断点，可以迅速发现异常的模块和数据。断点设置的方法将在后面提到。

3. 仿真循环页面

仿真循环页面显示“方法”的三列相关信息：**Method**、**Breakpoints**、**ID**。如图 2.3.4 所示：

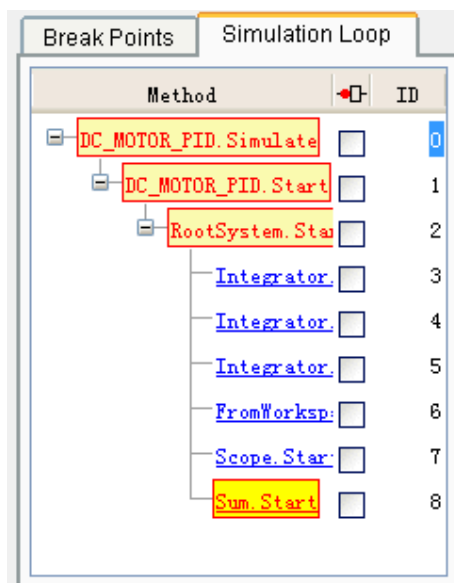


图 2.3.4 仿真循环页面

(1) **Method**: 方法名列以树形结构列出了从仿真开始时刻至此的所有“方法”。树形图的节点表示“方法”对其他“方法”的调用, 展开一个节点即可观察上一级“方法”调用的次级“方法”。单击包含于模块中的“方法”(蓝色)会使模型中相应的模块高亮显示。

(2) **Breakpoints**: 用户可以通过断点列的复选框设置断点。(在动画模式下次功能无效)

(3) **ID**: ID 列按顺序显示方法名列中所列出的所有“方法”的 ID 号码。

4. 信息显示页面

信息显示页面共有三个子标签: **Outputs**, **Sorted Lists** 和 **Status**。如图 2.3.5 所示:

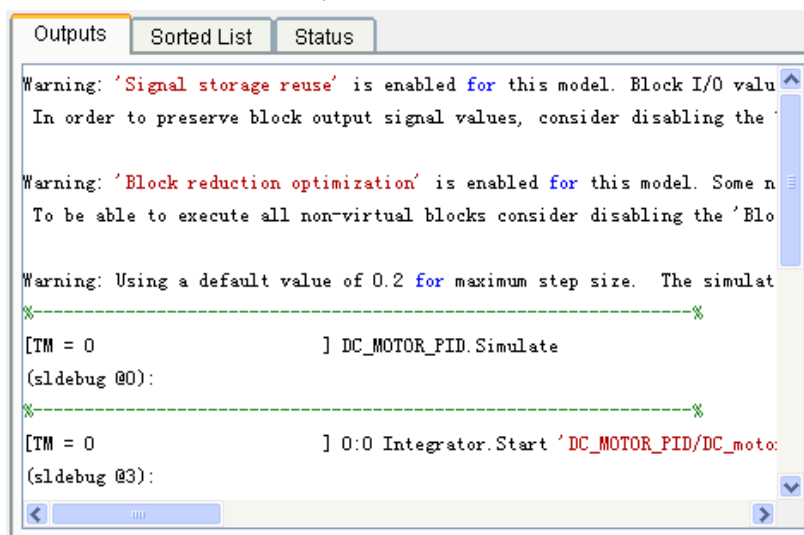


图 2.3.5 信息显示页面

Outputs: 输出调试信息和结果, 如调试命令、调试模块状态和该模块的输入/输出。

Sorted Lists: 按顺序显示调试过程中的非虚模块。

Status: 显示各种调试设置的值和其他一些状态信息, 如仿真时间, 调试命令, 及调试断点等。

2.3.2 命令行调试

通过命令行模式调试可以调用调试器的所有功能，这对高级用户来说非常必要，也很方便。

在 matlab 中的命令窗口中输入 `sim` 命令和 `sldebug` 命令都可以启动调试器：

```
>>sim('模型名称',[0,10],simset('debug','on'))
```

或者


```
>>sldebug '模型名称'
```

可以看到模块已经被打开，并在命令窗口中显示了如下信息：


```
%-----%  
[TM = 0 ] 模型名称.Simulate
```


此时调试器已经开始运行，用户可以继续输入其他调试指令、`matlab` 指令或者输入 `stop` 指令停止仿真调试。更多的调试命令可通过输入 `help` 命令获取。

2.3.3 运行调试器

在 GUI 模式下，单击【开始/继续】按钮“”开始进行调试。Simulink 可以从一个时间节点执行到下一个时间节点；从一个端点执行到下一个端点；或从一个模块执行到另一个模块，即单步调试。

1. 逐模块调试

这里仍然使用上文提到的直流电机模型。打开模型，在 matlab 工作空间中定义参数，点击工具条中的“”按钮开始调试，模型上方会出现一个调试方法指示框“@DC_MOTOR_PID.Simulate”，提示用户模型已进入调试状态。

单击工具条中的“”按钮执行单步运行，会看到有一个箭头由指示框指向模块，这个模块表示下一步即将执行的模块。以后每执行一次，指示框和箭头都会产生相应的变化。如图 2.3.6 所示：

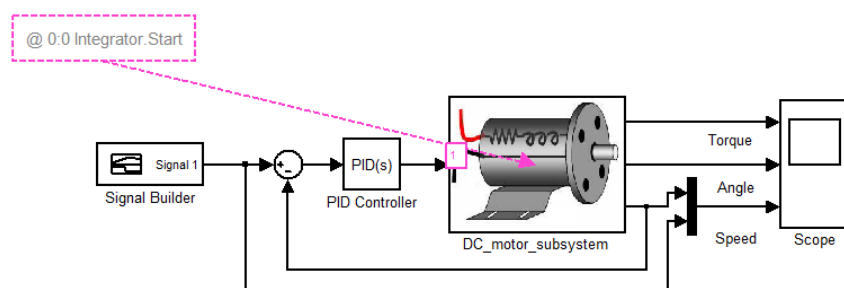


图 2.3.6 模型显示调试信息

在进行单步调试的同时，仿真循环页面中会以高亮显示的下一步即将执行的模块。如他 2.3.7 所示：

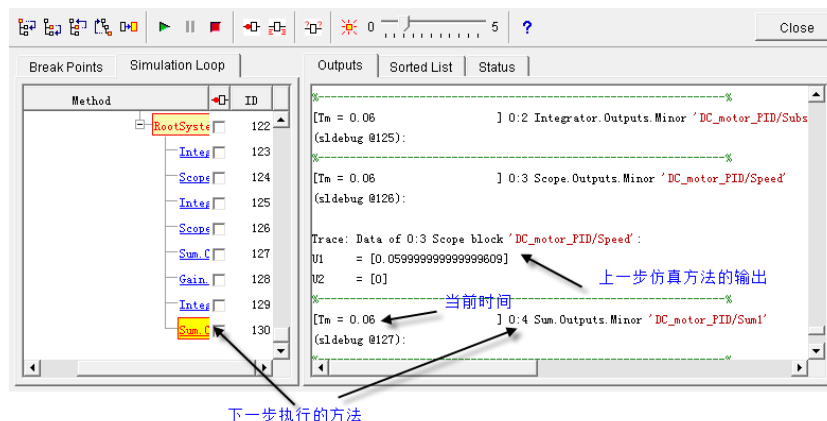




图 2.3.7

如果采用 Matlab 命令行方式调试，则在使用 `sldebug' DC_MOTOR_PID'` 命令后输入 `step` 命令即可，命令窗口输出如下信息，`TM=0` 表示当前仿真时刻为 0；`sldebug @0` 表示即将被执行的模块为当前系统的第一个模块。

```
%-----%
[TM = 0                                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> step
%-----%
[TM = 0                                ] DC_MOTOR_PID.Start
(sldebug @1): >> step
%-----%
[TM = 0                                ] RootSystem.Start 'DC_MOTOR_PID'
(sldebug @2): >> step
%-----%
[TM  =  0                                ] 0:0  Integrator.Start
'DC_MOTOR_PID/DC_motor_subsystem/Integrator'
(sldebug @3): >> step
Data of 0:0 Integrator block 'DC_MOTOR_PID/DC_motor_subsystem/Integrator':
CSTATE = [0]
%-----%
[TM  =  0                                ] 0:1  Integrator.Start
'DC_MOTOR_PID/DC_motor_subsystem/Integrator2'
(sldebug @4): >>
```

2. 逐时间步长单步调试

逐时间步长调试是按时间进程进行的，相对于逐模块调试，其调试效率较高，这对于复杂或模块数量较多的模型相当有效。

在 GUI 方式中，按下调试器上的绿色开始按钮“”，启动调试，在未设置断点情况下，每按下“”按钮一次，即运行一次仿真步长，调试输出信息如图 2.3.8 所示：

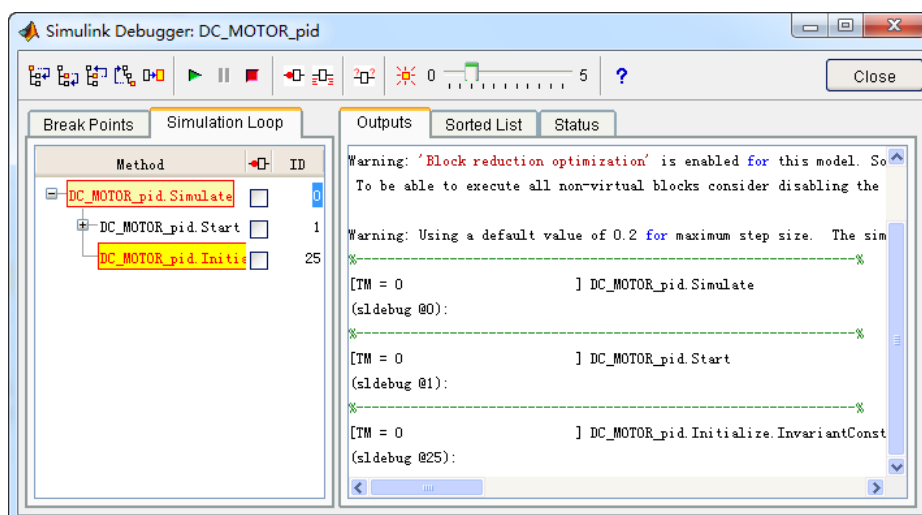




图 2.3.8 逐时间步长单步调试

对应命令行方式中的 `next` 指令，利用 `next` 命令可以执行当前时间步长中的所有模块(通常都大于一个)，直接跳到下一个仿真时间步长。

```
%-----%
[TM = 0                                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> next
%-----%
[TM = 0                                ] DC_MOTOR_PID.Start
(sldebug @1): >> next
%-----%
[TM = 0                                ] DC_MOTOR_PID.Initialize.InvariantConstants
(sldebug @25): >> next
%-----%
[TM = 0                                ] DC_MOTOR_PID.Enable.InvariantConstants
(sldebug @27): >>
.....
```

3. 自动调试

在自动调试模式下，系统在当前仿真步长内，自动逐方法调试，每个方法间由一定的停顿，同时，用指针指向下一步运行的方法。

按下工具条中的 “” 按钮开始调试，点击 “” 启动自动模式，通过滚动条可以调整停顿时间。运行结果如图 2.3.9 所示：

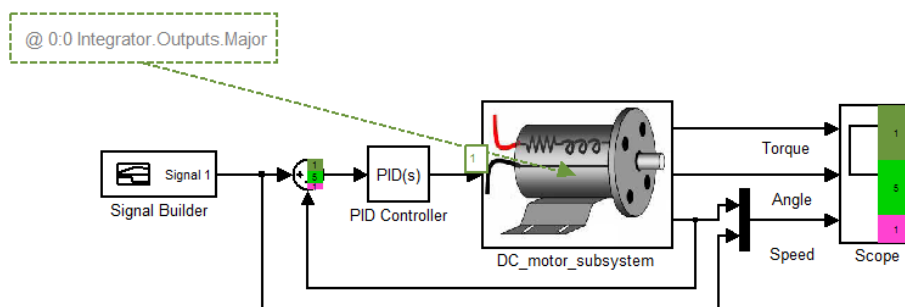


图 2.3.9 自动调试

调试指针指向的模块，会出现颜色不同、带有数字的小方块，各种颜色代表了不同的方法，数字表示该方法被调用的次数。

注意到 **subsystem** 和 **PID Controller** 模块并没有出现带颜色的方块，这是因为系统将其判定为虚模块，而模块指针只做用于非虚模块。打开 **subsystem**，可以看到其中的非虚模块已经被模块指针标注了调用次数，如图 2.3.10 所示。PID Controller 的情况类似，如图 2.3.11 所示。可通过右击模块，在弹出菜单中选择 **Look Under Mask** 打开其底层非虚模块。

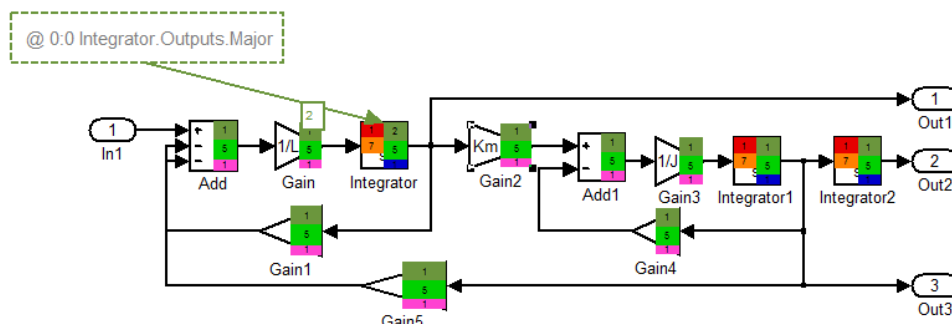


图 2.3.10 电机子系统的调试信息

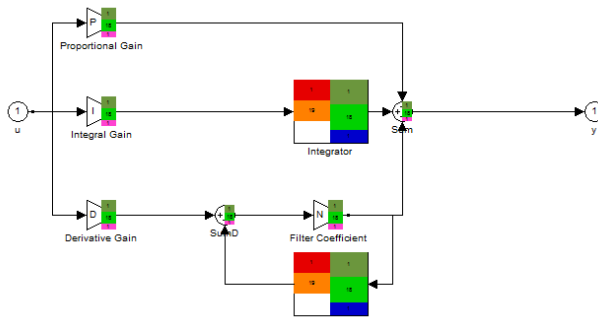



图 2.3.11 PID 系统的调试信息

4. 运行至断点

当确知模型中的某处可能存在问题时，断点是非常有用的，通过设置适当的断点可以快速确定异常的位置，或跳过异常执行后续仿真。

在积分器模块处设置断点后，点击工具条中的 “” 输出调试信息如图 2.3.12 所示：

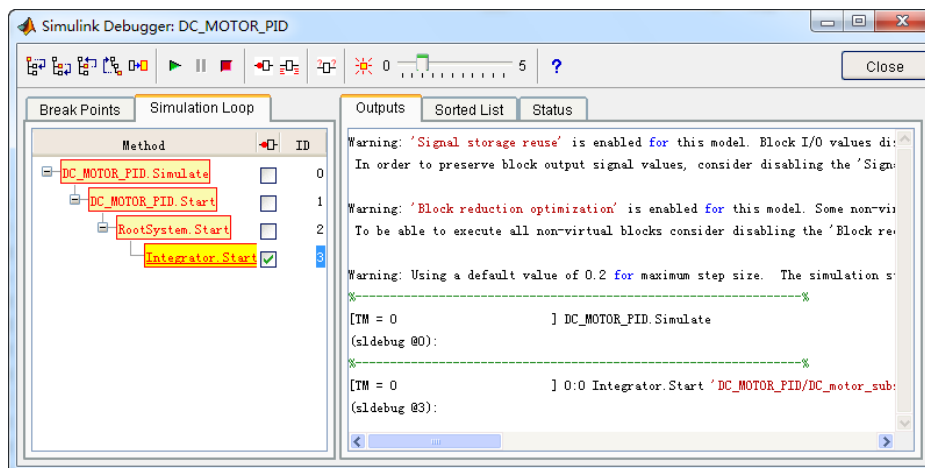


图 2.3.12 运行至断点

可见调试进行到第三个方法时遇到了断点，并暂停了调试。在命令行模式中，可以用 `continue` 命令从当前断点运行到下一个断点或仿真的终点。断点设置方法将在后面提到。

2.3.4 断点设置

借助断点可以快速诊断系统，找到仿真过程中存在的错误。Simulink 调试器允许用户针对不同情况设置不同类型的断点：无条件断点与有条件断点。


- 无条件断点：无论任何条件，仿真到达预设断点处即中断运行。
- 有条件断点：若满足断点条件，仿真到达预设断点处即中断运行。

1. 无条件断点

无条件断点可通过以下三种方法设置：

- GUI 调试器工具栏快捷按钮
- GUI 调试器的 Simulation Loop 页面
- 命令窗口

1. 通过 GUI 调试器工具栏快捷按钮设置断点

在模型窗口选中需要设置断点模块(例如选择 `Integrator 2`)，然后单击图形调试器工具栏中的 Breakpoint 按钮 “” 设置断点。断点页面即显示当前断点。如图 2.3.13 所示：

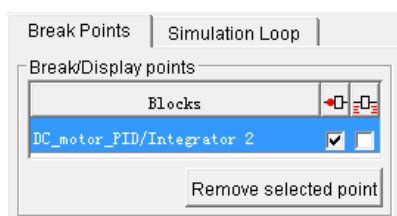


图 2.3.13 断点设置

用户可以通过模块后面的断点复选框取消断点，若要删除模块列表中的断点，可以点击 `Remove selected point` 按钮。

2. 通过 GUI 调试器的 Simulation Loop 页面设置断点

此设置方法需要先运行调试器，当调试器仿真若干个模块后，`Simulation Loop` 页面中经历的模块才能显示出来，对于复杂系统，需要执行得更多步才能显示出期望的模块。如图 2.3.14 所示：

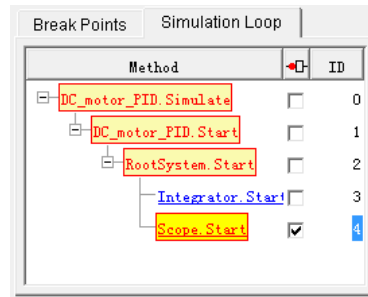


图 2.3.14 运行至断点

3. 通过命令窗口设置断点

命令窗口设置断点的方法只适用于命令行调试模式下,在该模式下,命令 **break** 和 **bafter** 分别用来在模块的前端和后端设置断点, **clear** 命令则用来清除断点, 命令行调试窗口的输出信息如下:

```
%-----%
[TM = 0                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> step
%-----%
[TM = 0                ] DC_MOTOR_PID.Start
(sldebug @1): >> break
Installed break point:0 before m:1

(sldebug @1): >> bafter
Installed break point:1 after m:1

(sldebug @1): >> clear
Break point '0' has been removed.
Break point '1' has been removed.

(sldebug @1): >>
```

2. 条件断点

无条件断点的特点是: 每当系统运行到无条件断点处时都会暂停。而条件断点处是否发生中断还不确定, 取决于具体的运行情况是否符合条件。如图 2.3.15 所示:

在断点设置页面中的下部复选框中可以选择中段条件, 每项的含义已在上文做过介绍:

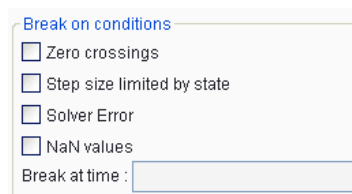


图 2.3.15 条件断点页面

其对应的命令行指令为:

- **zcbreak** 仿真发生过零时设置断点
- **xbreak** 仿真步长超过模型步长限制是设置断点
- **ebreak** 求解出错处设置断点
- **nanbreak** 数值溢出或无穷大时设置断点

1. Zcbreak

当需要对模型的过零情况进行中断时，勾选 **Break on conditions** 中的 **Zero crossings** 复选框或在命令窗口中使用 **zcbreak** 命令。暂停后显示该中断在模型中的 ID、类型、名称、时间，以及在过零时时上升还是下降。

在命令行调试模式下，使用 **zcbreak** 命令设置过零断点，**continue** 命令开始运行仿真，在下一断点或当前仿真步长末尾前停止运行，过零中断的输出信息如下：

```
%-----%
[TM = 0                                ] DC_MOTOR_pid.Simulate
(sldebug @0):
1 Zero crossing detected at the following location
   0  0:3:0  FromWorkspace 'DC_MOTOR_pid/Signal Builder/FromWs'
ZeroCrossing Events detected. Interrupting model execution
.....
```

2. Xbreak

该命令主要是针对变步长求解器，勾选 **Break on conditions** 中的 **Step size limited by state** 复选框，或在命令窗口中使用 **Xbreak** 命令。当求解器所采用的步长超过模型的步长限制时，仿真就会自动中断，这种中断方式主要用于调试那些有可能需要过大仿真步长的模型。

```
.....
[TM = 0                                ] DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> xbreak
Break on failed integration step          : enabled

(sldebug @30): >>
```

3. Ebreak

勾选 **Break on conditions** 中的 **Solver Error** 复选框，或在命令窗口中输入 **ebreak** 命令，系统将在检测到一个可恢复的错误时产生中断。如果用户未设置此条件断点，系统将会自动修复错误，但是不会发出提示信息。

```
.....
%-----%
[TM = 0                                ] DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> ebreak
Break on solver error                    : enabled

(sldebug @30): >>
```

4. NaNbreak

勾选 Break on conditions 中的 NaN values 复选框，或在命令行调试模式输入 nanbreak 命令，系统将在求解器计算出一个无穷大数值或溢出时产生中断。设置这类中断，可以精确地找到系统中的计算错误。



```
.....
%-----%[TM = 0 ]
DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> nanbreak
Break on non-finite (NaN,Inf) values : enabled
(sldebug @30): >>
```

5. Tbreak


在 Break on conditions 的 Break at time 文本框输入时间，或使用 tbreak 命令设置时间断点，系统将在指定的时刻的下一个步长处中断并停留在模型的 Outputs.Major 方法。

2.3.5 显示模型和仿真信息

1. 显示模块的输入输出信息

通过调试器工具栏上的按钮 “” 和按钮 “” 显示模块的输入输出。用于命令行模式的相应指令由 probe、disp、trace。调试命令 trace gcb 与 probe gcb 分别对应于上述两个按钮，但 probe 和 disp 命令并没有与之相对应，下面解释其不同之处。

1. 设置模块的输入输出显示点


在 GUI 界面下，选中模型中某一模块后，单击 “” 按钮，断点/显示点列表将列出当前设置的显示点。在命令行模式下，该按钮对应的命令是 trace gcb 或 trace s:b。前者为当前模块设置显示点，后者可直接针对某个模块进行设置，其中 s 表示系统顺序，b 表示模块顺序。例如，在模型中选中 sum 模块，然后再命令行模式下调试，使用 next 命令执行到 TM=3 之后（即阶跃信号开始跳变），执行 probe 命令，则可显示此时的 sum 模块输入/输出。


```
.....
%-----%
[TM = 3.020454598515591 ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe
Entering block probe mode. Click on any block to see its data.
Type any command to leave probe mode.
Probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1      = [0.98177007864064314]
U2      = [0.018229921359356905]
Y1      = [0.98177007864064314]
.....
```

若要观察同一时刻其他模块的输入/输出，可在模型中选中相应的模块，命令行中会立即显示其当前输入/输出。例如选中 PID 模块，命令行输出如下信息：

```
.....
%------%
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe
Entering block probe mode. Click on any block to see its data.
Type any command to leave probe mode.
Probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1      = [0.98177007864064314]
U2      = [0.018229921359356905]
Y1      = [0.98177007864064314]
Probe: Data of SubSystem block (virtual) 'DC_MOTOR_PID/PID Controller':
U1      = [0.98177007864064314]
Y1      = [8.6319002707974715]
.....
```

2. 显示选中模块的输入输出

在 GUI 界面下，选中模型中某一模块后，单击 “” 按钮会在 Output 页面中显示当前状态下选中模块的其 I/O 信息，在命令行模式下，probe s:b 命令与之对应。

选中 sum 模块并按下 “” 后可得到如图 2.3.16 所示的输出结果：

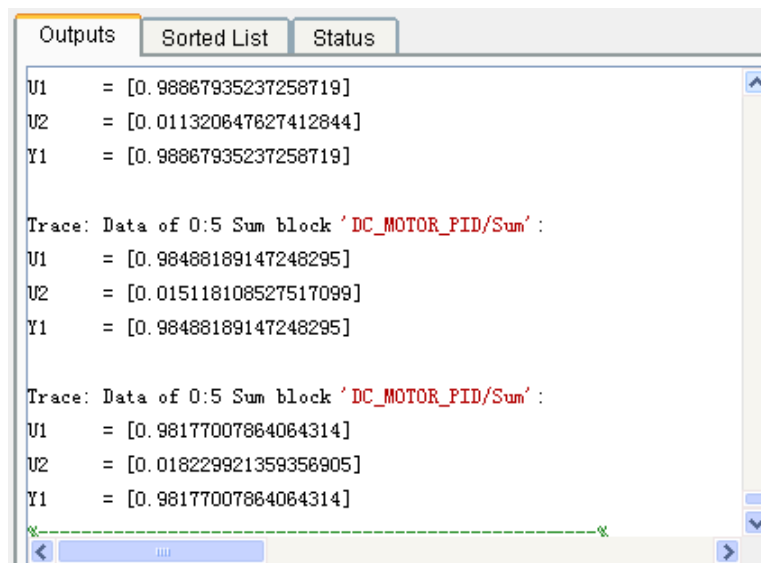


图 2.3.16 outputs 页面

在命令行模式下，选中分别选中 PID 模块和 sum 模块并执行 probe gcb 指令，命令窗口中结果如下：

```
.....
%------%
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe gcb
probe: Data of SubSystem block (virtual) 'DC_MOTOR_PID/PID Controller':
U1      = [0.98177007864064314]
```

```

Y1      = [8.6319002707974715]

(sldebug @41): >> probe gcb
probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1      = [0.98177007864064314]
U2      = [0.018229921359356905]
Y1      = [0.98177007864064314]

(sldebug @41): >>
.....

```

使用 **probe** 指令可以使调试器进入 **probe** 状态，这时用户选中模型的任意一个非虚模块时，命令窗口中就会显示其 I/O 信息，这比上面的方法更加简单。再次输入 **probe** 指令可退出该模式。

3. 显示断点处的模块输入输出信息

在命令行模式下，调试命令 **disp gcb / disp s:b** 用于设置当前或指定的模块在调试中断时显示其输入输出，命令 **undisp gcb / undisp s:b** 可移去当前或指定的显示点(其中 **s** 表示系统顺序，**b** 表示模块顺序)。

该指令只能在命令行模式下运行，在 GUI 界面中无法实现。

在设置中断点后，每单步调试一个模块，命令行窗口都会显示一次该模块的输入输出。例如，输入指令 **disp 0:0**，将会显示模块 **DC_MOTOR_PID/DC_motor_subsystem/Integrator** 的输入/输出信息：

```

.....
%-----%
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> disp 0:0
Installed data display of 0:0 Integrator block 'DC_MOTOR_PID/DC_motor_subsystem/Integrator'.

(sldebug @41): >> next

Disp:  Data of 0:0 Integrator block 'DC_MOTOR_PID/DC_motor_subsystem/Integrator':
U1      = [17.263800541594943]
Y1      = [0.033780793929875291]
CSTATE = [0.37426778201746674]
%-----%
[TM = 3.020454598515591      ] DC_MOTOR_PID.Update
(sldebug @65): >>
.....

```

注意，0:0 为 **DC_MOTOR_PID/DC_motor_subsystem/Integrator** 模块的 ID 序号，每个模块的 ID 序号都可以在 **Sorted List** 子页面中找到，下面将具体介绍该页面。

2. 显示模型信息

1. 显示模块执行顺序

GUI 界面的 Sorted List 子页面显示了模型中的每个模块和非虚子系统。该页面列出了各模块的执行顺序及模块 ID 号。如图 2.3.17 所示：

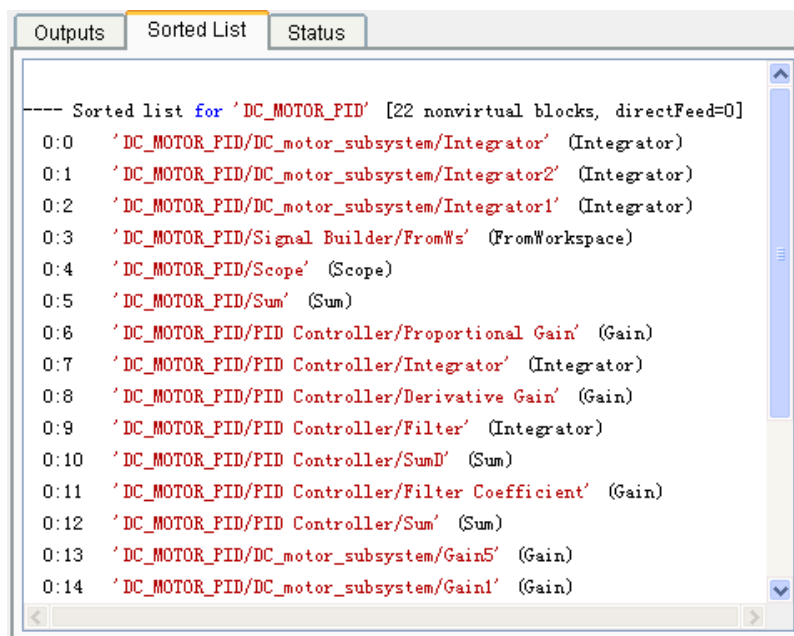


图 2.3.17 执行顺序页面

在命令行模式下可以用 slist 命令来显示模型的列表顺序：

```
(sldebug @0): >> sldebug 'DC_motor_PID'
(sldebug @0): >>slist
(sldebug @0): >> slist

---- Sorted list for 'DC_MOTOR_PID' [22 nonvirtual blocks, directFeed=0]
0:0    'DC_MOTOR_PID/DC_motor_subsystem/Integrator' (Integrator)
0:1    'DC_MOTOR_PID/DC_motor_subsystem/Integrator2' (Integrator)
0:2    'DC_MOTOR_PID/DC_motor_subsystem/Integrator1' (Integrator)
0:3    'DC_MOTOR_PID/Signal Builder/FromWs' (FromWorkspace)
0:4    'DC_MOTOR_PID/Scope' (Scope)
0:5    'DC_MOTOR_PID/Sum' (Sum)
0:6    'DC_MOTOR_PID/PID Controller/Proportional Gain' (Gain)
0:7    'DC_MOTOR_PID/PID Controller/Integrator' (Integrator)
0:8    'DC_MOTOR_PID/PID Controller/Derivative Gain' (Gain)
0:9    'DC_MOTOR_PID/PID Controller/Filter' (Integrator)
0:10   'DC_MOTOR_PID/PID Controller/SumD' (Sum)
0:11   'DC_MOTOR_PID/PID Controller/Filter Coefficient' (Gain)
0:12   'DC_MOTOR_PID/PID Controller/Sum' (Sum)
0:13   'DC_MOTOR_PID/DC_motor_subsystem/Gain5' (Gain)
0:14   'DC_MOTOR_PID/DC_motor_subsystem/Gain1' (Gain)
```

```

0:15 'DC_MOTOR_PID/DC_motor_subsystem/Add' (Sum)
0:16 'DC_MOTOR_PID/DC_motor_subsystem/Gain2' (Gain)
0:17 'DC_MOTOR_PID/DC_motor_subsystem/Gain4' (Gain)
0:18 'DC_MOTOR_PID/DC_motor_subsystem/Add1' (Sum)
0:19 'DC_MOTOR_PID/DC_motor_subsystem/Gain' (Gain)
0:20 'DC_MOTOR_PID/DC_motor_subsystem/Gain3' (Gain)
0:21 'DC_MOTOR_PID/PID Controller/Integral Gain' (Gain)

```

2. 显示调试器状态

在 GUI 界面下，Status 子页面中列出了调试器各种选项的设置情况，例如条件断点。如图 2.3.18 所示：

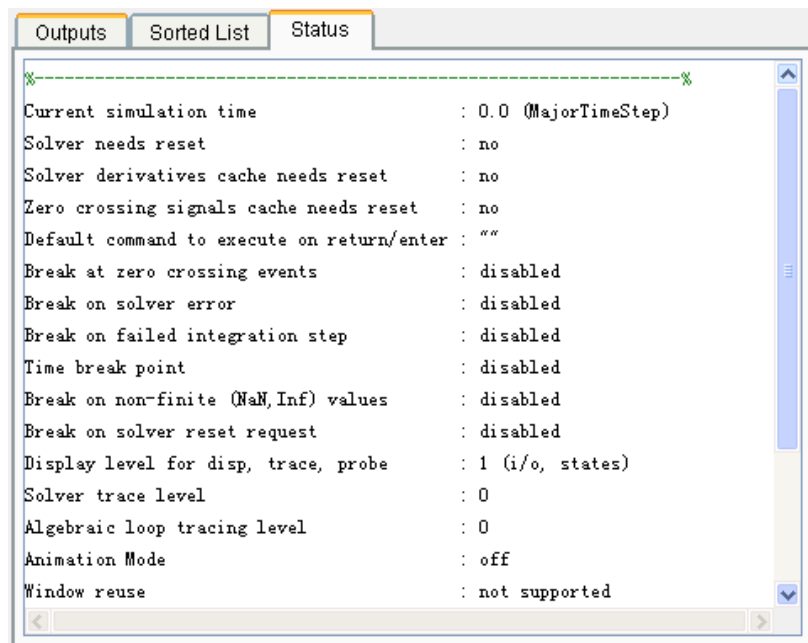


图 2.3.18 调试状态页面

在命令行模式下，可以使用 `status` 指令来显示调试器的设置情况：

```

%-----%
[TM = 0 ] DC_MOTOR_PID.Simulate
(sldebug @0): >> status
%-----%
Current simulation time : 0.0 (MajorTimeStep)
Solver needs reset : no
Solver derivatives cache needs reset : no
Zero crossing signals cache needs reset : no
Default command to execute on return/enter : ""
Break at zero crossing events : disabled
Break on solver error : disabled
Break on failed integration step : disabled
Time break point : disabled
Break on non-finite (NaN, Inf) values : disabled
Break on solver reset request : disabled

```

Display level for disp, trace, probe	: 1 (i/o, states)
Solver trace level	: 0
Algebraic loop tracing level	: 0
Animation Mode	: off
Window reuse	: not supported
Execution Mode	: Normal
Display level for etrace	: 0 (disabled)
Break points	: none installed
Display points	: none installed
Trace points	: none installed

3. 显示潜在过零点模块

在仿真过程中 **zclist** 指令能够列出模型中所有可能会产生非采样过零点的模块

```
%-----%
[TM = 0                               ] DC_MOTOR_PID.Simulate
(sldebug @0): >> zclist
0  0:3:0    R  FromWorkspace 'DC_MOTOR_PID/Signal Builder/FromWs'
```