DIVERSIFIED CHEMICAL PRODUCTS
Specialty Products Division
Cambridge, MA 02139

MEMO #5

TO: U.R. Engineer

FROM: A.I. Jockey (Technical Services Group)

DATE: Sep 13, 2007

SUBJECT: Instructions for using the JACOBIAN batch reactor model on Windows

---

This memo explains how to access JACOBIAN, a Dynamic Simulation and Optimization Software developed by Numerica Technology LLC (spin-off from MIT). JACOBIAN will be used to simulate the distillation tasks and the first reaction task during the design of the Lucretex monomer process. You will be using JACOBIAN's dynamic simulation features. Dynamic simulation is the use of mathematical models to predict the time dependent or dynamic behavior of a chemical process (as opposed to steady-state simulation which is only concerned with predicting the steady-state of a process).

Accessing JACOBIAN on Windows

The latest distribution of JACOBIAN (version 4.0) is available only for Windows. You will be able to run JACOBIAN either in the PC cluster in the DeGregory room (RM 66-064, basement) or your own computers (Laptop or Desktop). The following instructions will explain how to access JACOBIAN in the DeGregory room and how to obtain JACOBIAN for your personal use.

Accessing Jacobian in the PC cluster in the DeGregory room

The PC cluster in the DeGregory room is a Windows Athena cluster. This basically means that you log on with your Athena account information and you work in the Windows environment. Your information is stored in a remote server and no matter which computer of the cluster you are using you will have access to the files in you account. You have access to the drive *H:* in these machines. The *C:* drive is locked and you don't have access to it. Don't store files in your Desktop, this makes your computer slower. Instead create folders in *My Documents* folder.

Accessing JACOBIAN in your own Laptop or Desktop

To get JACOBIAN in your computer you need to follow these steps:

- Go to *http://www.numericatech.com/register_academic.htm*
- Fill your personal information and include the following:
  - Advisor = Paul Barton
  - Research Area = 10.490

  - Company = MIT
  - Software = Jacobian

- you will receive an email from Numerica Technology with a license file (*jac.lic*) and information how to download the software.
- download the software and run the installer. Accept the default options. This will install JACOBIAN in the folder *C:\Program Files\Jacobian*,
- at the end copy *jac.lic* and paste it in the folder *C:\Program Files\Jacobian\System*.
- Now JACOBIAN is installed in your computer.

Creating a project and getting the example files

Figure 1 shows the layout of the JACOBIAN IDE, which has five areas: a Menu bar across the top, a Toolbar below the Menu, a Project Explorer panel on the left, a Messages panel at the bottom, and an empty space taking up the rest of the window which will serve as the Main Workspace.

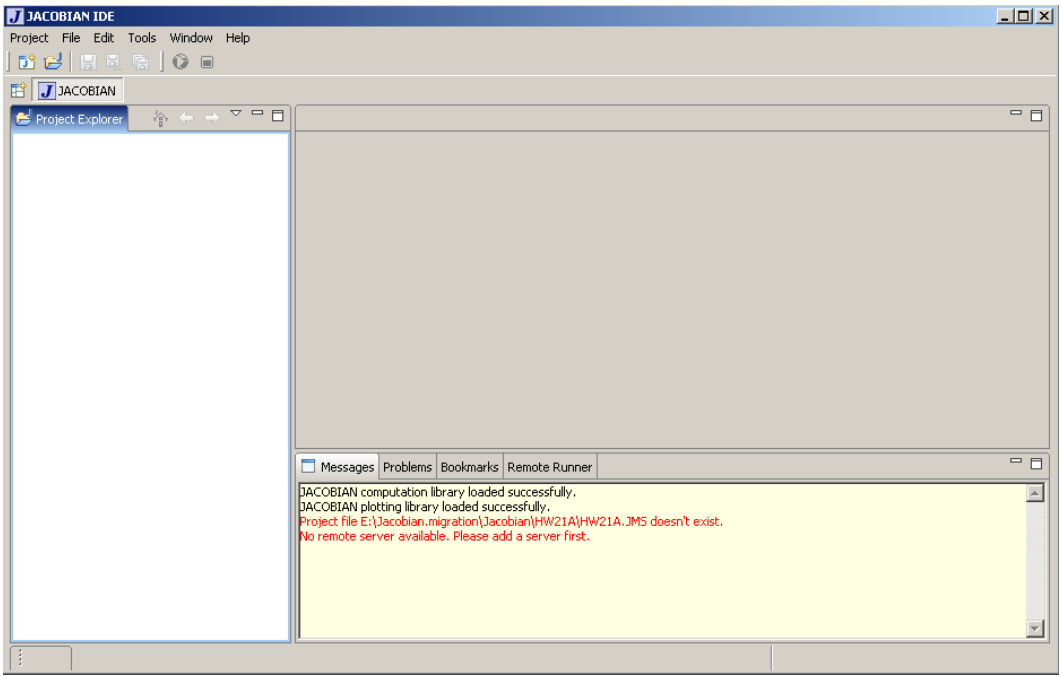

Figure 1: JACOBIAN Graphical User Interface

JACOBIAN helps to organize your problem (input files and results) by creating separate projects. This time we are going to create a new project for the examples discussed in this tutorial. This can be done by selecting *New* from the Project menu (or clicking the *New Project* button). In the pop-up window, enter the name the location for the new project. For example, we can name this project as *ICEExamples* and select the *H:\My Documents\Jacobian* folder as the location for

this project (Figure 2). Once this is done, the Project Explorer tree will be updated with the new project named *ICEExamples* and empty *Input* and *Output* nodes.

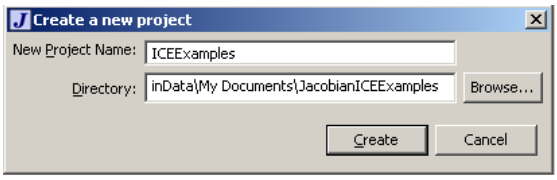

Figure 2: Pop-up window for creating a new JACOBIAN project

Download the example files from the 10.490 Stellar website. They are located in the *Examples* subsection in the *Materials* section. Download all of them into a folder in your drive.

You can create JACOBIAN input files from scratch or you can import input files into a project. In this case we are going to import files that are already written. This can be done by selecting the input node under the project name in the Project Explorer tree, right-clicking to display the pop-up menu, selecting Import and then selecting JACOBIAN Input File from the submenu (Figure 3). In the following pop-up window, the users can browse over the folders to select the existing JACOBIAN input file. Import all the downloaded example files.

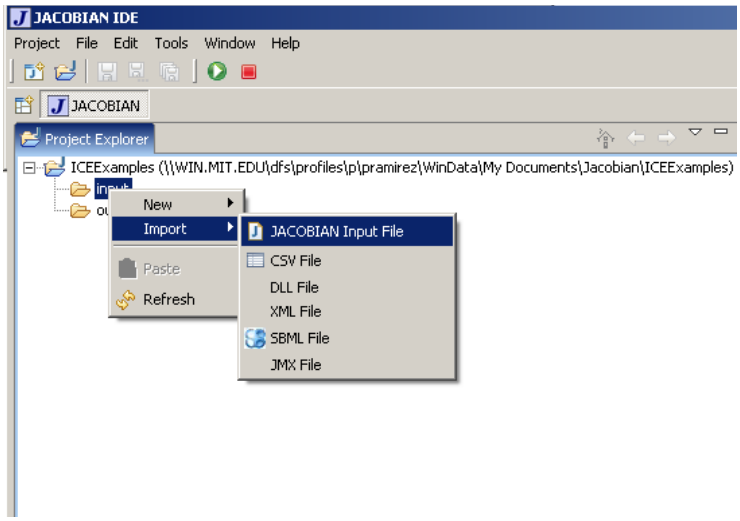

Figure 3: Pop-up menu for addding/importing JACOBIAN input file

Tutorial Example: Simple Mass Action Kinetics

This section guides you through a simple tutorial introduction to the JACOBIAN input language, and how to run an JACOBIAN simulation. We consider the following system of sequential irreversible elementary reactions:

$$A \rightarrow B$$

$$B \rightarrow C$$

We will assume that the reactions are conducted under isothermal conditions, and that the mixture is sufficiently dilute that volume changes on reaction are negligible. A mass balance under these assumptions yields the equations:

$$\frac{dC_A}{dt} = -r_1 \quad (1)$$

$$\frac{dC_B}{dt} = r_1 - r_2 \quad (2)$$

$$\frac{dC_C}{dt} = r_2 \quad (3)$$

where the reaction rates r_1 and r_2 are given by the equations:

$$r_1 = k_1 C_A \quad (4)$$

$$r_2 = k_2 C_B \quad (5)$$

This complete system of equations forms a simple mathematical model for our reaction system. The goal of this exercise is translate this mathematical model into a format that JACOBIAN can recognize, and then use JACOBIAN to predict the variation of the concentrations with time as the reaction progresses.

The engineer describes a mathematical model of a chemical process to JACOBIAN by preparing an ASCII file in the JACOBIAN input language. This is a high level computer programming language specifically designed to express mathematical models of chemical processes. The input language is basically used to describe the equations and variables in a mathematical model to JACOBIAN. This capability to add new models as equations makes JACOBIAN a very flexible and powerful modeling system. Simulators like JACOBIAN are known as equation-oriented or simultaneous process simulators.

The JACOBIAN input file for our mass action kinetics example is shown in figures 4 and 5. It is already in your JACOBIAN Input Files and it is called *Series_Reactions.JAC*.

To open the file in the editor, select the file *Series_Reactions.JAC* in your input folder and right click on it. Then click *Open*. Let's go through what the input means (see also figures 4 and 5).

JACOBIAN is a case insensitive language: it does not recognize the difference between upper case and lower case characters. In the example input file, the convention has been adopted that keywords of the JACOBIAN language are typed in upper case and identifiers introduced by the user are typed in mixed case. Identifiers introduced by the user may be any sequence of the letter characters *A-Z*, the number characters *0-9*, and the underscore character _. The sequence must always start with a letter character, and can have a maximum of eighty characters. Note that any text following a # symbol on a line of input is treated as comment and ignored by JACOBIAN.

Variables in an JACOBIAN model represent physical quantities (such as the temperature or pressure of the contents of a vessel). Equations (such as a mass balance) relate these variables to form a model of a physical system. All variables in an JACOBIAN model must have a *type*. The type of a variable states certain properties held in common by all variables of that type. A variable type declaration states the following:

- an identifier for the type (for reference in subsequent input)
- a default initial guess for the value of variables of this type (for use in iterative calculations that require an initial guess)
- a lower bound on the value of variables of this type
- an upper bound on the value of variables of this type

```
DECLARE

  TYPE
    # identifier          # default  # lower # upper
    Molar_Concentration = 1.0    :  −1E−12 : 1E5   UNIT = "mol/m3"
    Reaction_Rate       =  0.0   :  −1E−4  : 1E9   UNIT = "mol/(m3 s)"

END # don't forget to mark the end of the DECLARE block

MODEL Series_Reactions # a MODEL must have an identifier for future reference

PARAMETER
  Rate_Constant1      AS REAL # which denotes the type
  Rate_Constant2      AS REAL

VARIABLE
  Concentration_A, Concentration_B  AS Molar_Concentration # variable type
  Concentration_C                   AS Molar_Concentration
  Rate_1, Rate_2                    AS Reaction_Rate

EQUATION

  $Concentration_A = −1*Rate_1 ;

  $Concentration_B = 1*Rate_1 − 1*Rate_2 ;

  $Concentration_C = 1*Rate_2 ;

  Rate_1 = Rate_Constant1*Concentration_A ;

  Rate_2 = Rate_Constant2*Concentration_B ;

END # don't forget to mark the end of the MODEL block!!
```

Figure 4: JACOBIAN MODEL and DECLARE Blocks for Mass Actions Kinetics Example

```
SIMULATION Series_Reactions_Simulation # identifier for future reference

UNIT
  Reactor      AS Series_Reactions

SET
  Reactor.Rate_Constant1 := 0.3 ;
  WITHIN Reactor DO
    Rate_Constant2 := 0.5 ;
  END # within structure                                                    10

PRESET
  WITHIN Reactor DO
    # identifier        #  guess  # lower # upper
    Concentration_A :=  2.0    :   0.0   : 5.0 ;
    Concentration_B :=  0.0 ;
    Concentration_C :=  0.0    :         : 5.0 ;
  END

INITIAL                                                                     20
  WITHIN Reactor DO
    Concentration_A = 2 ;
    Concentration_B = 0.0 ;
    Concentration_C = 0 ;
  END

SCHEDULE
  CONTINUE FOR 25.0

END # and, don't forget to end the SIMULATION block                         30
```

Figure 5: JACOBIAN SIMULATION Block for Mass Action Kinetics Example

- an optional declaration of the units of measurement for variables of this type (currently only used for display purposes)

Any calculation with an JACOBIAN model will ensure that the values of variables of a particular type stay within these bounds. For example, absolute temperature is a positive quantity. If the value really is outside these bounds, the calculation will fail with an appropriate error message.

JACOBIAN input is divided into a series of *blocks*. A DECLARE block is used to declare variable types. Any number of DECLARE blocks may appear in an input file, but a variable type must be declared before it is used. In our example, we need types for molar concentration quantities and reaction rate quantities. The DECLARE block in figure 4 states that variables of type Molar_Concentration will have a default initial value of 1.0, a lower bound of -1×10^{-12} (concentrations are always positive quantities), an upper bound of 1×10^{5}, and have units of moles per cubic meter. The better the initial guess, and the tighter the bounds you can provide, the better the numerical methods will perform. Note that specifying the bounds too tightly can lead to disaster because the value of a variable will vary with time over a range during a dynamic simulation, and this range must lie within the bounds specified.

MODEL blocks are used to state the variables and equations that make up the mathematical model of a unit operation in a chemical process. Note that a MODEL will typically be an *under determined* system of equations: there will be less equations than variables. This leaves a subset of variables that must be specified by the user or determined by other models so that the number of unknown variables remaining equals the number of equations. When the number of equations equals the number of unknown variables, the model can be solved (the model is now *fully determined*). The variables specified to make the model fully determined in this manner are sometimes called the *degrees of freedom* of the simulation. A MODEL block for our simple example is shown in figure 4.

A MODEL may have a series of time invariant parameters, introduced in the PARAMETER section of a MODEL. This makes a model more general and enables it to be used again in many applications. For example, a MODEL of a cylindrical vessel that is parameterized by its cross sectional area may be used to represent any cylindrical vessel provided a value for this area is specified. In figure 4 we want to introduce two parameters to represent the rate constants of the two reactions of type REAL. Parameters may also be of type INTEGER and type LOGICAL (i.e., values of TRUE or FALSE). Note that any *attribute* of a model (e.g., a parameter or a variable) must be declared before it is referred to.

Next, we must introduce the variables that represent the various time dependent quantities in the model. In this case, we want to introduce a variable of type molar concentration for each of the chemical species involved in the reaction: A, B, and C; and a variable for the rate of each reaction. If a set of variables all have the same type, they can just be listed before the type. This is done in the VARIABLE section of figure 4.

The heart of an JACOBIAN MODEL are the equations that relate the variables. The EQUATION section of figure 4 shows the equations (1)–(5) transformed into a form that JACOBIAN can understand. An equation is two real expressions linked by the equality operator = and terminated by the ; character. Equations may involve constants (e.g., 3.142), parameters, and variables as operands, and the operators +, −, ∗, / and ^ (exponentiation – i.e., raising to a power). There are also the built-in transcendental functions shown in table 1. You can find more functions in the Jacobian Syntax Manual in the *Materials* section in the 10.490 Stellar site. In figure 4 note that the symbol $ is used to denote the time differential operator:

$$\$\text{Concentration\_A} \text{ is equivalent to } \frac{dC_A}{dt}$$

Hence the left hand sides of the first three equations in figure 4 are the accumulation terms in the mass balances.

Identifier	Function
ABS	The absolute value (magnitude) of the argument
SIGN	The sign of the argument
SQRT	The square root of the argument
SIN	The sine of an argument in radians
COS	The cosine of an argument in radians
TAN	The tangent of an argument in radians
ASIN	The arcsine in radians of the argument
ACOS	The arccosine in radians of the argument
ATAN	The arctangent in radians of the argument
SINH	The hyperbolic sine of the argument
COSH	The hyperbolic cosines of the argument
TANH	The hyperbolic tangent of the argument
EXP	The exponential of the argument
LOG	The natural logarithm of the argument
LOG10	The logarithm to base 10 of the argument
INT	Truncate real argument towards negative infinity

Table 1: Table of Built-in Vector Functions

Our model is therefore a system of ordinary differential equations (ODEs) coupled with algebraic equations (AEs). These are usually known as differential-algebraic equations (DAEs).

It is important to note the difference between the equality operator = and the assignment operator := in the JACOBIAN language. The equality operator is used to denote a general mathematical relationship between two expressions (i.e., an equation), whereas the assignment operator is used to denote the assignment of the value of an expression to a variable; the variable assigned a value appearing on the left hand side of the assignment operator. Examples of the use of the assignment operator appear in figure 5.

A SIMULATION block is used to specify a particular simulation with a MODEL previously declared. Obviously, many different simulations may be performed with a single model, each simulation being a different scenario in which the physical system represented by the model is studied. An JACOBIAN dynamic simulation calls for the numerical solution of an initial value problem (IVP) in the DAEs making up the model. In order to fully specify a simulation, we must make the model fully determined by specifying the degrees of freedom, and define an initial condition for the IVP.

Figure 5 shows the SIMULATION block for our example. First, we must state which MODEL(s) we are going to use for the simulation. The UNIT section creates active *instances* of the models listed during execution of the simulation.

All the time invariant parameters must be assigned values before the simulation is well-posed. This is shown in the SET section of figure 5. Note that the particular variables of a model instance are referenced by what is called a *pathname mechanism*. In its most simple form this is:

```
Model_Instance_Identifier.Variable_Identifier
```

In other words, the model instance identifier, a . character, and then the variable identifier. In more sophisticated models (e.g., in the ICE reactor example) there may be a whole list of model instance

identifiers separated by . characters prefixing the variable identifier. This occurs when models are nested inside each other to manage the complexity of developing a large process model.

In the first line of the SET section in figure 5, the parameter is referred to by its complete pathname. In the third through fifth lines, a WITHIN structure is used to define Reactor as the *scope* within which any pathname is interpreted. This shorthand avoids the need to keep prefixing variable identifiers with model instance identifiers. WITHIN structures may be nested. If an identifier is not found within the current scope, the enclosing scopes are searched sequentially. If the identifier is still not found, the compiler will issue an error message.

In many situations it is desirable to override the default initial guesses (defined in the variable type) for specific variables. For example, we may have more specific information on the value taken by a variable. The better the initial guess, the higher the chance that JACOBIAN will converge! Initial guesses and bounds for variables may be changed in the PRESET section, as shown in figure 5.

The initial conditions of a simulation are defined in an INITIAL section, as shown in figure 5. The example input states that the initial concentration of species A is 2 [mol/m^3], and the concentration of the other species is zero (i.e., they are not present initially). In general, initial conditions may be expressed as completely general equations that are added to the MODEL equations in order to calculate consistent initial values for all the variables in the model. Note that the WITHIN structure may be used with equations as well as assignments.

Finally, we need to state how long the simulation should run for. This is done with a SCHEDULE section. In general, the SCHEDULE section may be used to declare very complex sequences of operations to be performed during the simulation (this is discussed below). This feature makes JACOBIAN unique amongst process simulators.

An JACOBIAN input file may contain any number of DECLARE, MODEL and SIMULATION blocks as required in any order, provided that a block is declared before its identifier is referenced in another block. Your input may also be in several different files, but again, a file containing the relevant declaration must be loaded and compiled before files referring to the block in question can be loaded successfully.

Now that we can understand what the input file means, let's execute a simulation. First you need to load the file *Series_Reactions*, to do that right click on the file name and then click on *Load* (Figure 6). Loading a file for translation will make JACOBIAN translate the input file and perform certain checks on the correctness of the input. If there are any syntax errors, etc. in the input file, JACOBIAN will inform you that the translation failed in the dialog box at the bottom of the GUI. JACOBIAN also will show where are the errors in the open file (see Figure 7). All errors must be corrected before JACOBIAN will allow the input file to be executed. When the file is translated successfully, the simulations names will turn green. Note, remember to unload the file if you want to load a new file. If the two files share variable types, JACOBIAN will not load the new file.

At this point you should have a correct input file loaded into the JACOBIAN environment. The loaded files are displayed on the lefthand side of the GUI. To execute Series_Reactions_Simulation, right-click on it and then click on *Execute* (Figure 8). JACOBIAN will proceed to set up the problem and integrate the differential equations numerically. For this problem, the simulation will be executed virtually instantaneously. During execution, text output concerning the numerical solution and possible input errors is displayed in the dialog box at the bottom of the GUI. JACOBIAN will tell you whether the simulation was successful, or terminated prematurely. The simulation should be successful! You are now ready to display the results generated by JACOBIAN, as described in the next section of this memo.

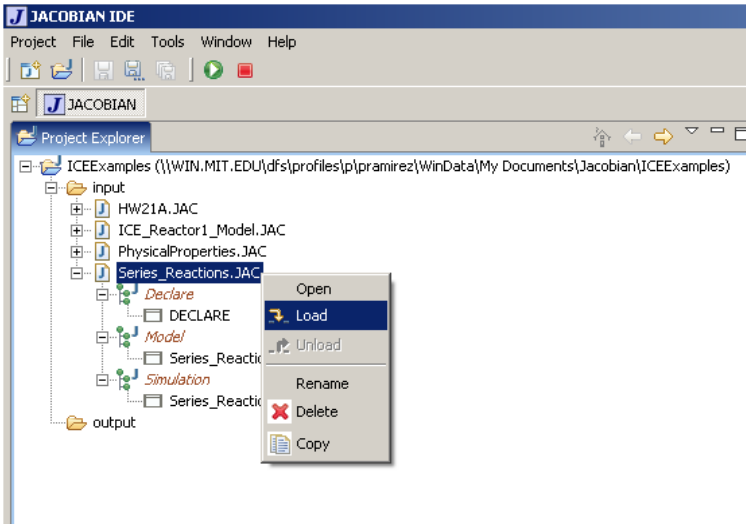

Figure 6: Pop-up menu for loading JACOBIAN input file

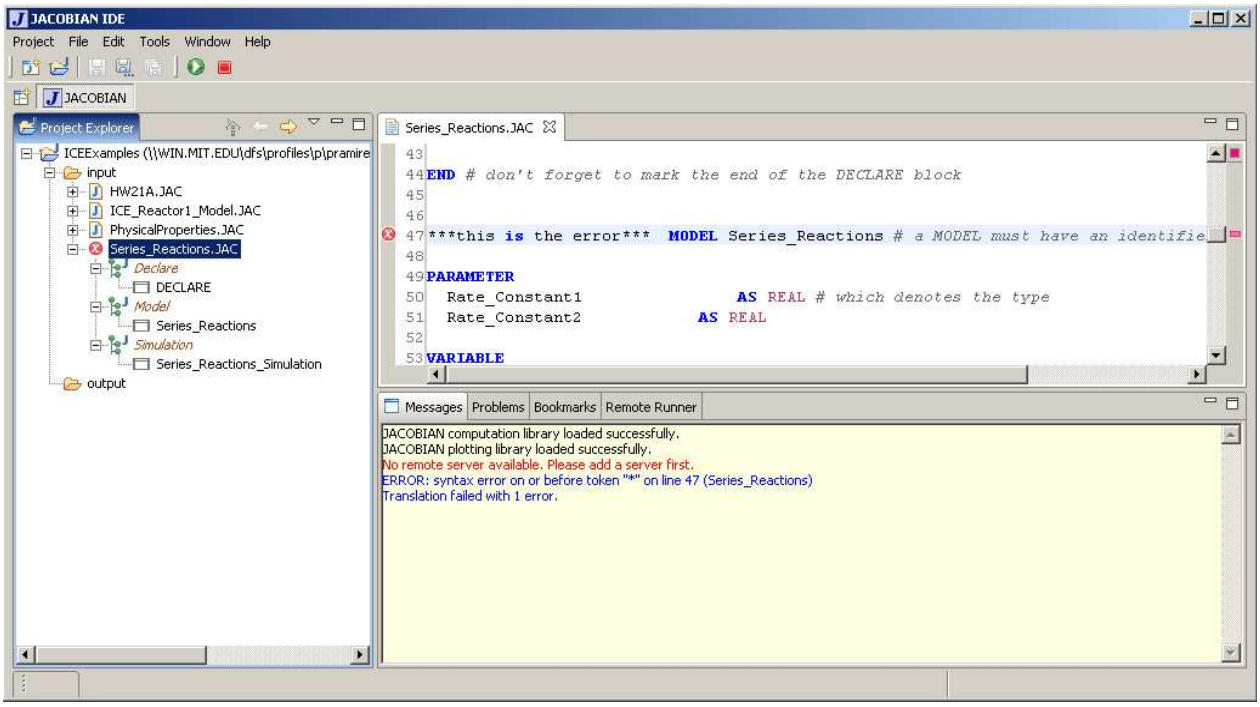

Figure 7: JACOBIAN Error window

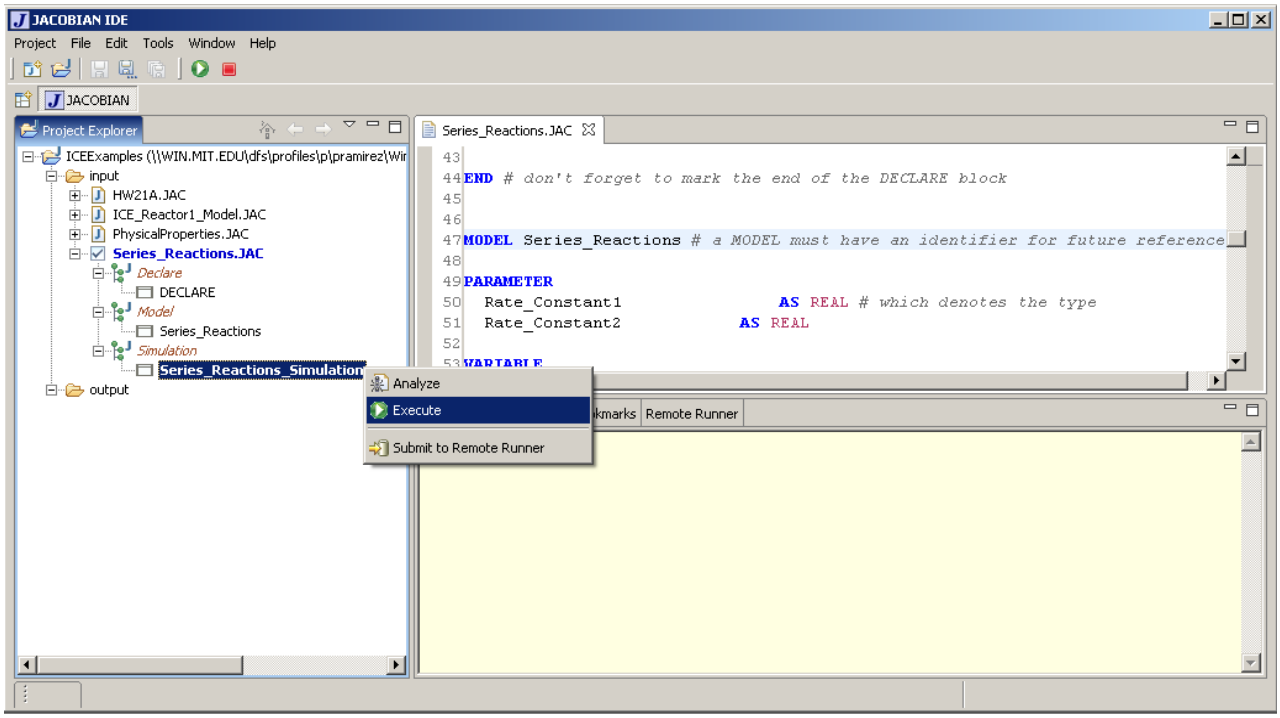

Figure 8: Pop-up menu for executing simulation

Plotting JACOBIAN Results

Plotting environment

The best way to view the results of a dynamic simulation is to generate graphs of the variation of the variables with time over the simulation. After a successful simulation, the output node in the Project Explorer tree will be updated with a highlighted node with the name of the computation just executed (Figure 9). Simulation results for this computation node can be viewed in the Main Workspace by:

- Double clicking the computation name,
- or right-clicking the computation name and select open from the pop-u menu (Figure 9).

The output tab which shows up in the Main Workspace has three sub-tabs: the Summary tab, Saved Plot tab, and Visualization tab. Figure 10 shows the content of the Summary tab, which consists of four sections: Annotation, Computation Report, Computation Input, and Executation Summary. Each of these sections can be collapsed or expanded by clicking the triangle on the left of the section name. The Annotation section can be used to comment the current computation; the Computation Report section displays the computation report from JACOBIAN; the Computation Input tab shows the simulation block extracted for this computation; and the Executation Summary section contains a copy of the output printed by the JACOBIAN during the executation of this computation.

Simulation results can be visualized by clicking the Visualization sub-tab of the Output tab for current computation. This shows the JACOBIAN plotting environment. Variables to view

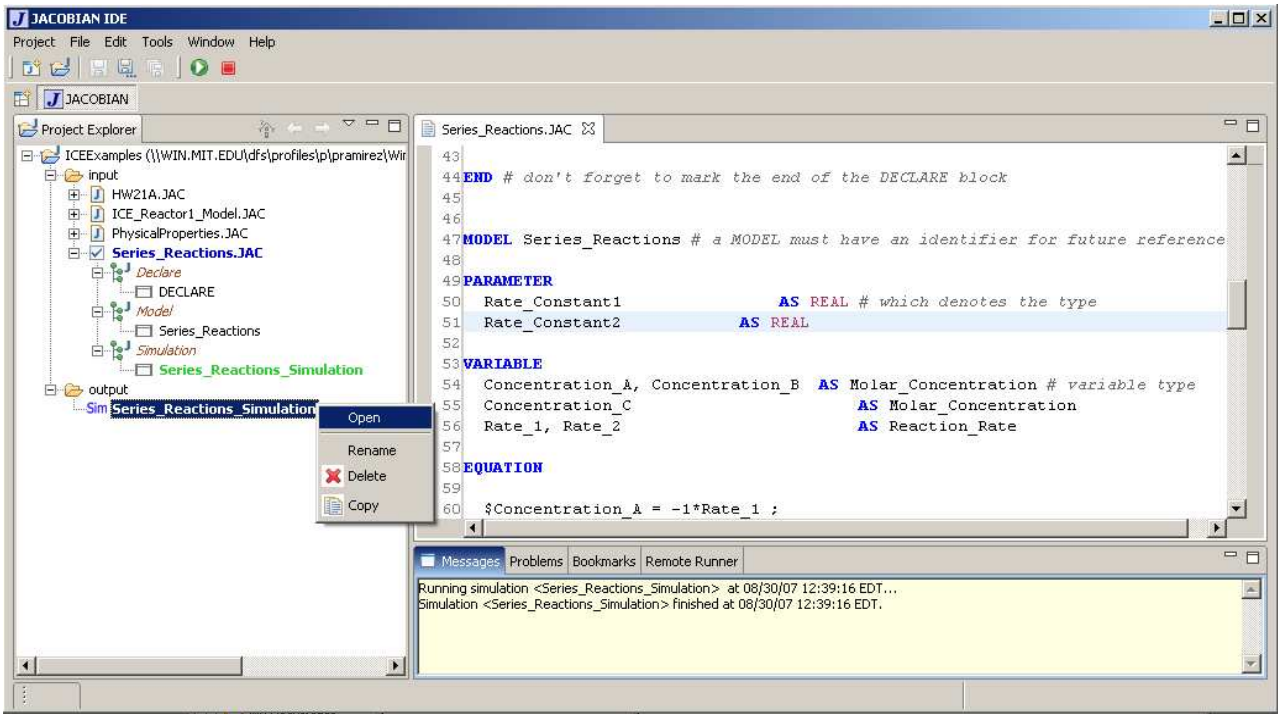

Figure 9: Pop-up menu for opening computation output.

are selected from the tree in the left panel of the Plotting Environment. The structure of this variable selection tree is the same as the hierarchical structure of the input file. To select variables for viewing, simply double click to select, or right click and select *Select* from the pop-up menu. Multiple variables can be selected simultaneously while holding Control key, or Shift key to select contiguous ranges of variables.

Once you have selected the variables, the trajectories for these selected variables will be displayed in the plot panel (Figure 11). The variable table which locates under the tree will also be populated with the selected variables. There are two checkbox columns for this variable table with the header of the column to be *Y1* and *Y2*, respectively. They correspond to the left and right y-axis which you can use for the plotting purpose. When the selected variables are of very different scales, it will be very helpful to use one y-axis for the large-scale variables and use another y-axis for the small-scale variables.

By default, when the variables are first selected, the checkbox for the *Y1* column will be checked, which means that the primary y-axis (i.e., the left y-axis) are used for all the variables. You can simply select the corresponding checkbox for the *Y2* column for those variables onto which you want to use the secondary y-axis (the right y-axis). In Figure 12, the variable of *RATE_1* is selected to use the secondary y-axis in the plot panel. As you can see the legends for these two y-axis variables are displayed in separate legend boxes to help you to identify the variables. A number of plot customizations are available by right clicking on the plotting area (Figure 11). For those variables which are already shown in the plot panel, if you want to remove some of them from the plot, you can simply un-check the corresponding selected checkbox. They can be put back into the plot if you re-check their corresponding checkbox. The values for the variables displayed in the plot panel can be viewed by clicking the *Table* tab (Figure 13).

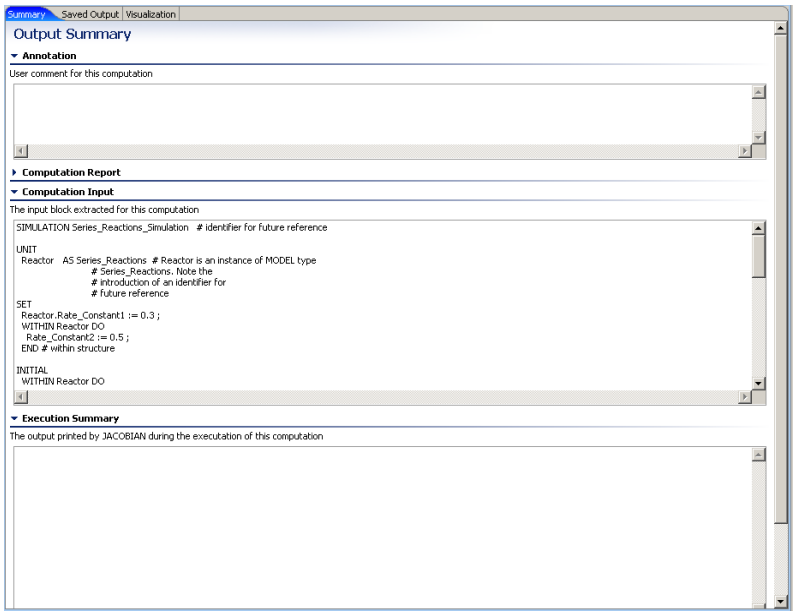

Figure 10: The Summary tab for the opened computation output.

For the variables displayed in the plot panel, they can be saved by clicking the *Save* button under the variable table. This will bring up a popup window (Figure 14) which asks for the name of the file you want to use for the saved plot. You can also write annotation for the saved plot.

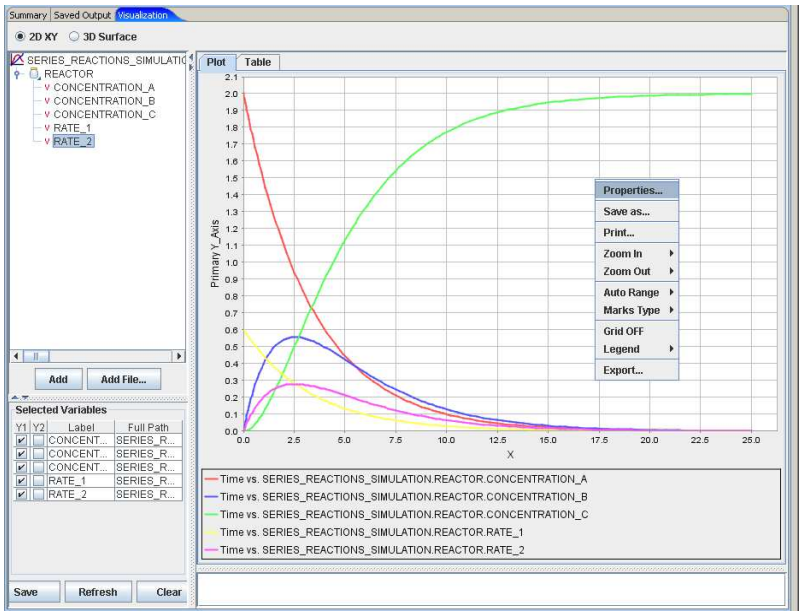

Figure 11: The plot showing the trajectories for the selected variables. Only the primary y-axis is used in this plot.

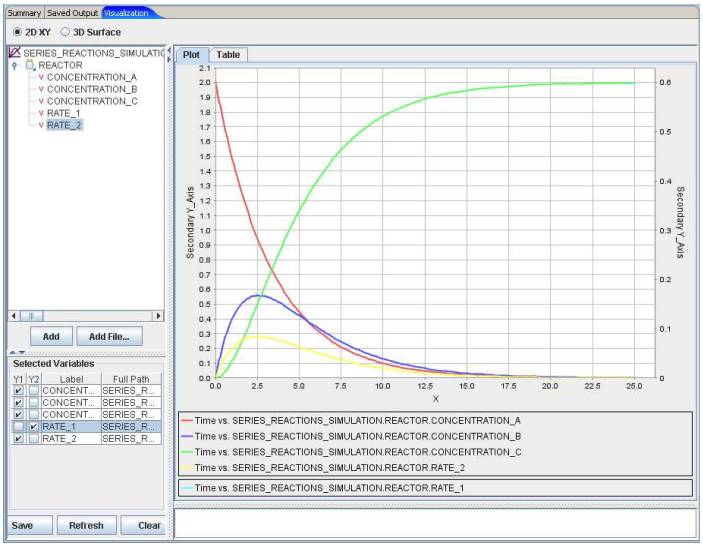

Figure 12: Plot showing the trajectories for the selected variables. Both the primary y-axis and secondary y-axis are used in the plot.

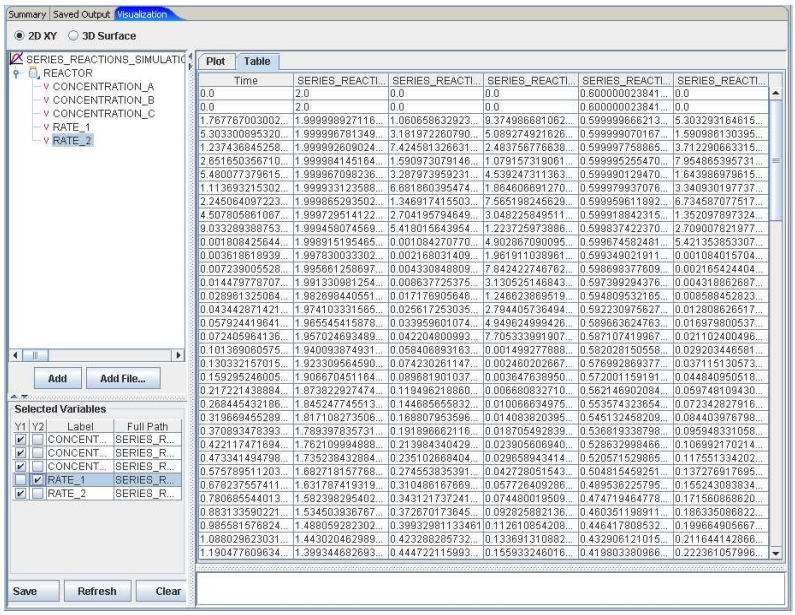

Figure 13: Table showing the values for the selected variables

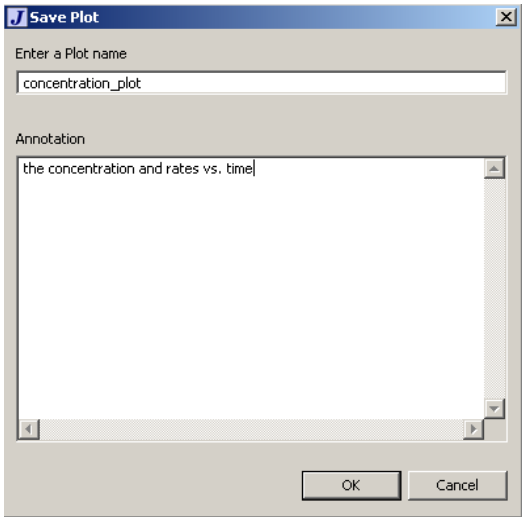

Figure 14: The pop-up window for saving the plot

Tutorial Example: First Reactor in ICE Process

If necessary, unload the input file for the mass action kinetics example. ICE_Reactor1_Model.JAC in the input directory contains the MODEL of the first reactor in the ICE process and the associated feed tanks and valves. Note that these MODELs will never change throughout this project. Hence you should not edit this file! However, it should always be included in files with SIMULATION blocks that represent particular operating policies applied to this physical system.

The physical property models are in the file PhysicalProperties.JAC in the input directory and you will always need to include PhysicalProperties.JAC in the models for the reaction and distillation tasks of the ICE process. At your leisure, take a look at the contents of this file, but you don't really need to understand it to proceed with the tutorial.

To simulate a particular operating policy for the first reactor, you must write a SIMULATION block describing your operating policy. We have prepared one for you already. Load HW21A.JAC. This file contains the initial conditions, values for the inputs to the system (degrees of freedom), and a schedule that describes any changes to these inputs as well as the conditions that define the end of the simulation. This SIMULATION block represents an operating policy that is uncannily similar to that of the first part of the homework problem. We strongly suggest that you copy this file and edit it in order to experiment with different operating policies for the first reactor.

To perform the simulation, we need to execute the SIMULATION block that was declared in the file HW21A.JAC. The simulation takes less than 10 seconds on any of the computers in the PC ChemE cluster. JACOBIAN will report on how far it has advanced the simulation time. Once the simulation is complete, you can start to answer the homework problem by using the plot environment. Don't forget to copy this file before editing it for the other parts of the homework.

Getting Help

The most current syntax manual for JACOBIAN is posted on the 10.490 Stellar website. This contains a detailed description of the complete syntax.

Using existing MODELs

There are many ways of using an existing MODEL. One way is to use INHERITS which copies all the contents of the MODEL in the new MODEL (see Figure 15). Another way is to use a UNIT that is a copy of the MODEL (see Figure 16). Note the different path of the variables in the two options.

```
MODEL Plant1 INHERITS Series_Reactions #INHERITS makes a copy of the previous model
VARIABLE
  Extra_rate AS Reaction_Rate
EQUATION
  Extra_rate=Rate_1+Rate_2;
END
SIMULATION ex1
UNIT
  Plant        AS Plant1
SET                                                                    10
  Plant.Rate_Constant1 := 0.3 ;
  Plant.Rate_Constant2 := 0.5 ;
INITIAL
  WITHIN Plant DO
    Concentration_A = 2 ;
    Concentration_B = 0.0 ;
    Concentration_C = 0 ;
  END
SCHEDULE
   SEQUENCE                                                            20
    CONTINUE FOR 25.0
    DISPLAY Plant.Extra_rate
   END
END
```

Figure 15: Using INHERITS

```
MODEL Plant2
UNIT
   Reac AS Series_Reactions #Reac is an object in Plant2
VARIABLE
  Extra_rate AS Reaction_Rate
EQUATION
  Extra_rate=Reac.Rate_1+Reac.Rate_2;
END

SIMULATION ex2 # identifier for future reference                       10
UNIT
  Plant        AS Plant2
SET
  Plant.Reac.Rate_Constant1 := 0.3 ;
  Plant.Reac.Rate_Constant2 := 0.5 ;
INITIAL
  WITHIN Plant.Reac DO
    Concentration_A = 2 ;
    Concentration_B = 0.0 ;
    Concentration_C = 0 ;                                              20
  END
SCHEDULE
   SEQUENCE
    CONTINUE FOR 25.0
    DISPLAY Plant.Extra_rate
   END
END
```

Figure 16: Using UNIT

Usefull features

Getting numerical values of variables without plotting

The easiest way of getting the value of a few variables at a particular point in the simulation is to use a DISPLAY followed by the path to a variable, as shown for the Simple Mass Action Kinetics.

```
...
...
SCHEDULE
  SEQUENCE
   CONTINUE FOR 25.0
   DISPLAY Reactor.Concentration_A
   END
  END
```

If you want to access or process all the values of all variables at all simulation times you can incorporate them in a spreadsheet, such as Microsoft Excel. In the OPTIONS section of the SIMULATION block turn the option of CSVOUTPUT (Comma Separated Value) to be TRUE. JACOBIAN will write a file called *SIMULATION-NAME.csv* in the directory */PROJECT-NAME/output*. For the Simple Mass Action Kinetics:

```
...
...
SIMULATION Series_Reactions_Simulation # identifier for future reference
OPTIONS
  CSVOUTPUT:=TRUE;
...
...
```

JACOBIAN will write a file called *SERIES_REACTIONS_SIMULATION.csv*. By the extension Excel will recognize the format of the file and create a spreadsheet where each column corresponds to one variable and each row to a time value.

Printing the JACOBIAN input files

Common errors and problems in simulations

All examples in this section are based on the Simple Mass Action Kinetics example.

How to avoid errors in the first place

Both inexperienced and experienced users run into problems when using software for complex models and most of the time it is their own fault, rather than a software bug.

Some things to keep in mind for your simulations:

1. If possible, gradually write models or change simulations. After each change run your simulation, and check the results.

2. Take your time when writing the equations (first think, then type).

3. Try to be as explicit as possible (e.g 50 kmol/hr should be written as 50*1000/3600 and not as 1/0.072 or 13.88889). Put lots of comments explaining to yourself and others changes to the input file.

4. When defining types, be careful with the bounds and default values. Having tight bounds and good defaults helps the solver to converge, but if the bounds are too tight the equations will have no solution.

5. When good guesses for the variable values are available, use them in the PRESET section.

6. Remember to take breaks and get enough sleep ☺.

Variable type already used, model identifier previously used

If you are trying to upload two files which have conflicting models or variables identifiers,JACOBIAN will complain in the error log (Figure 17). You have to unload the previous file (*right-click on File → Unload*).

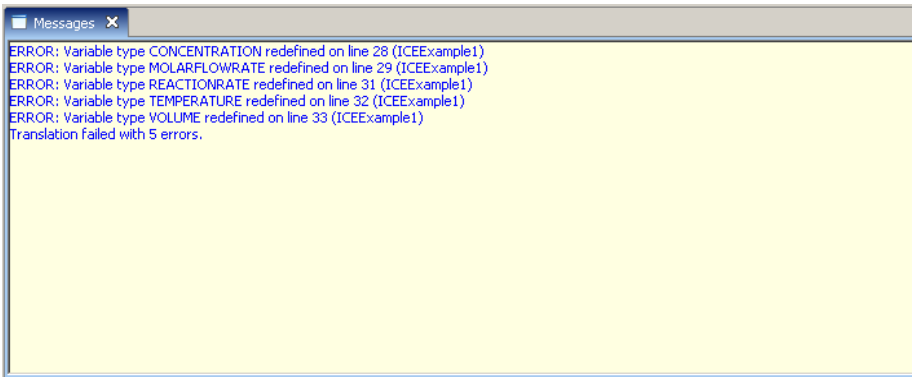

Figure 17: Types already used.

Could not find a value for ...

If you forget to set a value for a parameter, JACOBIAN will translate the file without complaints, but when you try to execute it, you will get an error message in the error log. You have to set a value for each parameter (even if the parameter is not used). Figure 18 shows an example of a simulation with a missing parameter value.

```
SIMULATION no_set # forgot to set a parameter
UNIT
  Reactor      AS Series_Reactions
SET
#  Reactor.Rate_Constant1 := 0.3 ; #did not set this
  Reactor.Rate_Constant2 := 0.5 ;

INITIAL
  WITHIN Reactor DO                                          10
    Concentration_A = 2 ;
    Concentration_B = 0.0 ;
    Concentration_C = 0 ;
  END
SCHEDULE
    CONTINUE FOR 25.0
END
```

Figure 18: Forgetting to assign a parameter value

Wrong number of initial conditions, There is 1 overdetermined block, There is 1 underdetermined block

In complex models it is very difficult to have a well defined system of equations. Because of a typo or a conceptual error you may have wrong number (Figure 19) or wrong combination (Figure 20) of equations or initial conditions. JACOBIAN will complain about that in the error log.

```
SIMULATION Wrong_init_num # wrong number of initial conditions
UNIT
  Reactor      AS Series_Reactions
SET
  Reactor.Rate_Constant1 := 0.3 ;
  Reactor.Rate_Constant2 := 0.5 ;

INITIAL
  WITHIN Reactor DO
    Concentration_A = 2 ;                                              10
#    Concentration_B = 0.0 ;#did not provide initial condition
    Concentration_C = 0 ;
  END
SCHEDULE
    CONTINUE FOR 25.0
END
```

Figure 19: Wrong number of initial conditions

```
SIMULATION Wrong_init_comb # wrong combination of initial conditions
UNIT
  Reactor      AS Series_Reactions
SET
  Reactor.Rate_Constant1 := 0.3 ;
  Reactor.Rate_Constant2 := 0.5 ;

INITIAL
  WITHIN Reactor DO
    Concentration_A = 2 ;                                              10
    $Concentration_A=0.6; #thought I should calculate the derivative at TIME=0
#    Concentration_B = 0.0 ;#did not provide initial condition
    Concentration_C = 0 ;
  END
SCHEDULE
    CONTINUE FOR 25.0
END
```

Figure 20: Wrong combination of initial conditions

Debugging

JACOBIAN has an analysis tool that will help you find which equations are underdetermined or overdetermined. Right click on the Simulation name and click on *Analyze*. A new window will pop up (Figure 21). JACOBIAN divides the analysis task in three steps and each time tries to break down the equation system in as many blocks as possible. You can use the analysis tool even without understanding the underlying mathematics. By clicking on the different analysis options (*DAE, High Index, Initialization*) you can study the different analysis. If you click on one of the

Permuted options then you can get information about the blocks of equations and where are the errors. To do this go to the *Information* window and expand the blocks saying *Overdetermined* or *Undetermined*. You will find two trees for the corresponding variables and equations. By expanding these trees and double clicking their components you can see the equations and the variables in the *Variables and Equations* window. Usually the blocks are not too big and it is relatively easy to find out where the mistake is. An ovedetermined block is a system of equations with more equations than variables; remove one or more equations. An underdetermined block is a system of equations with more variables than equations; add more equations.

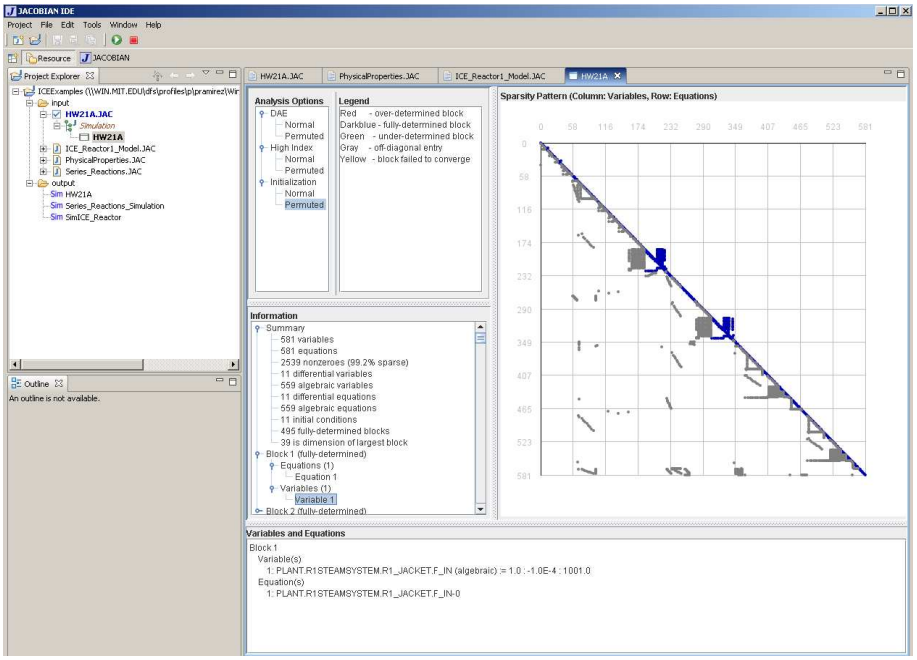

Figure 21: Using the analysis tool.

Consistent initialization failed or Integration failed

Common reasons

1. Typo or error in EQUATIONS, SET, INPUT. Example 50 kmol/s instead of 50 kg/hr
2. Big changes in the course of the SIMULATION in the RESETS and REINITIAL sections. An example is a step-change in pressure, instead of a ramp-change or feeding a large amount of a reagent instantaneously. Note that changes like that are often physically impossible or very dangerous.
3. Too tight bounds in type definition.
4. Bad initial guesses.

Figure 22 shows an example for a simulation where the consistent initialization will fail, because of a typo that makes a variable value out of bounds.

```
SIMULATION ConsIn # the consistent initialization fails
UNIT
  Reactor      AS Series_Reactions
SET
  Reactor.Rate_Constant1 := 0.3 ;
  Reactor.Rate_Constant2 := 0.5 ;

INITIAL
  WITHIN Reactor DO
    Concentration_A = 2e6 ; #2e6 instead of 2                                  10
    Concentration_B = 0.0 ;
    Concentration_C = 0 ;
  END
SCHEDULE
    CONTINUE FOR 25.0
END # and, don't forget to end the SIMULATION block
```

Figure 22: Simulation example where the consistent initialization will fail

Debugging

Take a break and think about the problem and your input file. Use the analysis tool 21. In the OPTIONS section of the SIMULATION block you can turn the option of INIT_PRINT_LEVEL, REINIT_PRINT_LEVEL or DYNAMIC_PRINT_LEVEL to be a number greater than zero. The larger the number, the more information you get. Depending on where the error is generated, you can select the appropriate option and get more information. For the Simple Mass Action Kinetics:

```
...
...
SIMULATION Series_Reactions_Simulation # identifier for future reference
OPTIONS
  INIT_PRINT_LEVEL:=100;
  REINIT_PRINT_LEVEL:=100;
  DYNAMIC_PRINT_LEVEL:=100;
...
...
```

Problems with JACOBIAN or the plotting environment

Freezing of JACOBIAN

JACOBIAN may seem to freeze when running a big simulation on a slow machine. You will have to be patient and wait for the simulation to end.

Segmentation fault (crash) of JACOBIAN

Reasons

1. Pressing buttons or key combinations too quickly
2. Bug in program, or underlying packages.

What to do

1. Don't press buttons or key combinations quickly.
2. Often save your files if you make many changes without executing (each time you press the *Update* button the file is saved anyway).
3. If the segmentation fault is reproducible, contact the TAs with the exact sequence of steps and the files. If there is a bug in the program, the developers might be able to identify and fix the problem.

JACOBIAN Language Reference

This section provides more detailed information on the JACOBIAN input language so that you can change input files for the homework problem.

SIMULATION Blocks – Modeling Operating Policies

The conditions characterizing a particular dynamic simulation in JACOBIAN are given in the specification of a SIMULATION block. The block refers to the mathematical model that will be used, and adds the information necessary to use the model for a simulation. This information includes the following:

- initial conditions for the simulation.
- a specification of the input variables (of degrees of freedom) for the simulation.
- a schedule that describes any changes to the inputs at later times in the simulation. This is how control actions imposed upon the system are modeled.

The description of a SIMULATION block is divided up into the following sections:

- OPTION: this section enables you to set a number of parameters related to numerical options, convergence criteria, output, etc. For details see the syntax manual. You should not need to change these options.
- UNIT: this section declares the MODEL blocks that will be used during the simulation.

- SET: this sections assigns values to any model parameters not already assigned values in the MODEL blocks. JACOBIAN will not execute a simulation until all the time invariant parameters are assigned values.
- INPUT: this section specifies inputs or degrees of freedom for the simulation. The inputs may be assigned constants or functions of time. The degrees of freedom must be satisfied for JACOBIAN to execute a simulation.
- PRESET: this optional section allows the specification of initial guesses for some or all of the simulation variables that override the defaults associated with a variable's type.
- INITIAL: this section is used to specify the initial condition of the system for the simulation. JACOBIAN will not execute a simulation unless the correct number of initial conditions are specified.
- SCHEDULE: this section specifies the stopping criteria as well as any changes to the inputs to the system during the course of the simulation.

You will need to modify the INPUT, INITIAL, and SCHEDULE sections in order to run the distillation and reaction tasks through more sophisticated operating policies.

The INPUT Section

The INPUT section sets the values of a subset of the variables in the process model. For instance, the vapor rate, reflux ratio and column pressure of the distillation tasks are specified as inputs. Note that each of the variables that is specified is referred to by the full pathname of models that denotes its location. The values assigned in the INPUT section may be changed as part of the SCHEDULE; later on in the simulation, these can be changed by using a RESET task that will assign a new value to the variable. Note that the assignment operator (:=) is used rather than the equality operator (=) for all specifications in the INPUT section. The values assigned to a variable in the INPUT section may be expressed as a constant, or a function of time. The keyword TIME is used to denote the time since the start of the simulation (at the start, TIME = 0)

The INITIAL Section

The INITIAL section defines the initial conditions for the simulation. The mathematical model of the reactor is a set of differential-algebraic equations, and an initial value problem in these equations is not fully defined without a set of initial conditions. In the SIMULATION block given to you the initial number of moles in the still pot and the accumulators are required as initial conditions. These initial conditions fully define the initial state of the column flowsheet; the model equations define all other quantities in terms of this subset. Note that the equality operator (=) is used rather than the assignment operator (:=), since JACOBIAN treats initial conditions as additional equations that are required to fully specify the system of differential-algebraic equations in order to determine consistent initial conditions.

The SCHEDULE Section

The SCHEDULE section allows the simulation to represent sophisticated operating policies. The SCHEDULE describes the sequence of external actions that are imposed on the model, and varying the sequence of external actions imposed upon the reactor allows us to simulate the reactor with different operating policies. Basically, these features of JACOBIAN allow you to experiment with and design an optimal operating policy for the reactor without experimenting with the real plant.

Clearly, electronic experiments with JACOBIAN are safer, cheaper, and produce no waste when compared with pilot plant experiments.

The distillation and reaction simulations you may want to perform will make use of several features of the SCHEDULE language. These features will be briefly described below, and some of their potential uses will be given. A SCHEDULE block is written in a similar fashion to a computer programming language. The individual statements in a SCHEDULE are known as *tasks*.

SEQUENCE <list_of_tasks> END The SEQUENCE task encloses a list of other tasks, and specifies that the next task in the list will be started only after the preceding task has been completed. It is used to describe sequences of primitive operations.

CONTINUE FOR <real_expression> The CONTINUE FOR task directs the computer to continue the simulation for the length of time specified by the real expression. The real expression is evaluated at the simulation time at which the task is executed.

CONTINUE UNTIL <logical_proposition> The CONTINUE UNTIL task is used to direct the computer to advance the simulation until the state of the system satisfies the condition defined by the logical proposition. Note that if this condition is never satisfied, the simulation will run forever (or until some limit is reached, such as a tank becoming empty, at which point the simulation fails). The atomic propositions of a logical proposition are *relational expressions* of the following form:

<real_expression> <relational_operator> <real_expression>

where <relational_operator> is one out of the set of operators { <,>,<=,>=,<>,= }. <> means not equal. One has to be careful with the use of <> and = because the simulations are done with finite precision arithmetic on a computer. An example of a relational expression is:

```
CONTINUE UNTIL Reactor.Temp >= 373.15
```

A logical proposition may involve one or more relational expressions linked by the logical operators AND, OR and NOT. For example:

```
CONTINUE UNTIL (Temp > Bub_Temp) OR (Temp < Dew_Temp)
```

The brackets are not strictly necessary, but they show the logic more clearly. A common problem is not to use the full pathname to the variables in a CONTINUE UNTIL task. For example, you must code:

```
CONTINUE UNTIL Plant.ProcessSystem.Reactor1.No_Mols(7) < 1E-6
```

You must always use the full pathname in CONTINUE UNTIL tasks (this is a feature (!?) of JACOBIAN). There is also a hybrid form of the CONTINUE UNTIL and CONTINUE FOR tasks:

CONTINUE FOR <real_expression> <logical_op> UNTIL <logical_proposition>

where <logical_op> is AND or OR. The OR will continue for a fixed period, or until the logical condition is satisfied, whichever occurs first. The AND form will continue until both criteria are satisfied. The OR form is particularly useful if you are not sure if a certain logical proposition will ever be satisfied and want the simulation to time out as a precaution.

RESET <variable_assignments> END The RESET task is used to reset the value of variables that have been originally specified in the INPUT section at some later point in time during the simulation. For example, a RESET task may be used to change the flowrate through a valve by manipulating the variable that represents its stem position. The following would change the stem position of Valve_1 after 3600 seconds:

```
SEQUENCE
  CONTINUE FOR 3600
  RESET
    WITHIN Plant.Valve_1 DO
      Stem_Position := 1 ;
    END # Within
  END # Reset
END # Sequence
```

This has the effect of opening the valve if the original value for this variable was zero (corresponding to a closed valve). The assignments inside a RESET task are expressed in an identical fashion to those in the INPUT section.

REINITIAL <variable_list> WITH <equation_list> END The REINITIAL task can be used to define discontinuities in a set of the model variables. For example, a REINITIAL task may be used to indicate the instantaneous addition of a particular reactant to the vessel:

```
CONTINUE FOR 3600
REINITIAL
   Plant.Reactor1.No_Mols(8)
WITH
   Plant.Reactor1.No_Mols(8) = OLD(Plant.Reactor1.no_Mols(8)) + 2 ;
END # Reinitial
```

This task states that variable Plant.Reactor1.No_Mols(8) is discontinuous, and that its new value is calculated by the equation that follows (i.e., the number of moles of species no. 8 jumps by 2 moles on execution of the REINITIAL task). Note the use of the built-in OLD function to refer to the value of a variable immediately before the discontinuity. In general, the discontinuous variables listed must be differential state variables (i.e., their time derivatives appear explicitly in the process model. For example, mole numbers of each species in the reactor). The number of equations must exactly equal the number of discontinuous variables.

MODEL Blocks – Modeling Physical Behavior

This information is here for reference purposes in order to help you understand the input.

Arrays

All the attributes of a MODEL block may be declared as a regular structure, or *array*, of a base type. Attribute arrays may have an arbitrary number of *dimensions*.[1] The total number of scalar quantities, or *elements*, represented by an attribute array is determined from the product of the number of elements in each dimension of that array. The number of elements in each dimension is declared in

[1]It is important here to distinguish between the dimensions of an array or regular structure, and the fundamental physical dimensions of a quantity, such as mass or length.

terms of a scalar integer expression involving integer constants and/or any previously declared integer parameters of the MODEL block in question (e.g. `Flow_In AS ARRAY(3,NoStream+1) OF REAL`).

References to array attributes may be made in several different fashions. For example, a reference to an entire array is made through use of the attribute identifier alone, and a reference to an individual element of an array is made by an explicit index to the element in question. This index is determined from a list of scalar integer expressions enclosed by brackets following the attribute identifier (e.g. `Flow_In(2,NoStream-1)`). Each expression in this list represents an index into one dimension of the array. Individual elements of a dimension are indexed from one to the number of elements in that dimension.

A reference to a subset of the elements in one or more dimensions of an array is termed a reference to a *slice* of that array. The elements that are included within a slice is again determined by a list of references into each dimension of the array in question enclosed by brackets. A subset of the elements in a particular dimension is denoted by two scalar integer expressions separated by a colon, representing the lower and upper bounds of the reference into that dimension respectively (e.g. `Flow_In(2:3,1:NoStream)`). The value of the upper bound must be greater than or equal to that of the lower bound, and both values must lie with the lower and upper indices of the dimension itself. A reference to an entire dimension is made by leaving a blank, so a list of blanks enclosed in brackets and separated by commas is identical to the use of an attribute identifier alone. A reference to an individual element is again made by a single scalar integer expression (e.g. `Flow_In(2:3,1)`).

Arrays of equation attributes are not declared explicitly, but are implied by their declaration in terms of expressions involving references to arrays or slices of variable and/or parameter attributes. The dimensionality[2] of a unary expression is the same as that of its operand. For binary expressions, three cases are distinguished:

- if both operands are scalar, then the expression is scalar.
- if only one operand is scalar, then the expression adopts the dimensionality of the other operand.[3]
- If neither operand is scalar, then both operands must be of the same dimensionality, which is also adopted by the expression itself.[4]

The dimensionality of the equation itself is obtained by applying the rules for binary expressions to the equality operator =.

Stream Attributes

Streams attributes are subsets, not necessarily disjoint, of the variables describing the time dependent behavior of a system. They represent a system's interface with its environment, and are useful in specifying the complex connection mechanisms that exist between different components of a physical system.

The STREAM section is used to declare stream attributes, which must be declared as instances of already declared *stream types*. This declaration also includes a specification of the subset of variable attributes that is to be included in the stream. The number and types of the variable

[2]The dimensionality of any attribute is defined as the number of dimensions and the number of elements in each dimension.

[3]Each element of this expression is obtained by the binary operation between the scalar operand and the corresponding element of the other operand.

[4]Each element of the resulting expression is obtained by the binary operation between the corresponding elements of the two operands.

attributes in a stream must normally match directly those in the stream type declaration. An example of a STREAM section is shown in figure 23.

```
STREAM
  Inlet  : Flow_In,  Temp_In,  Press_In,  Enth_In    AS MainStream
  Outlet : Flow_Out, Temp_Out, Press_Out, Enth_Out   AS MainStream
```

Figure 23: Example STREAM section

It should be noted that no assumptions concerning the *dimensionality* of the variable attributes included in a stream are made in a stream type declaration. Therefore, a slice or an entire array of variable attributes may appear in any field of a stream attribute, provided the base type of the array matches the variable type of the corresponding field in the stream type. For instance, the following is a valid stream declaration:

```
STREAM
  Inlet : Flow_In(1:NoComp-1),Temp_In,Press_In,Enth_In  AS MainStream
```

Stream attributes may themselves be declared as arrays of the basic stream types. For instance, a mixer involving several inlet streams could have a corresponding stream declaration of the form:

```
STREAM
  Inlet : Flow_In, Press_In    AS ARRAY (NoStream) OF MainStream
```

Each variable attribute in a k-dimensional stream must have at least k dimensions, and each of its *first* k dimensions must have exactly the same number of elements as the corresponding dimension of the stream. For instance, a possible declaration of the variables in the above example would be:

```
VARIABLE
  Flow_In                    AS ARRAY (NoStream,NoComp) OF Flowrate
  Press_In                   AS ARRAY (NoStream)        OF Pressure
```

This rule allows a natural identification of the variable attributes to be associated with each element of the stream array.

Functions

Expressions may include built-in functions as operands. A function performs a mathematical operation on its arguments that would be difficult or even impossible to declare using the standard language operators. At present, there are two categories of built-in function:

- *Vector functions* take a single argument and return a set of values with dimensionality equal to that of the argument.
- *Scalar functions* take an arbitrary number of arguments of arbitrary dimensionality and return a scalar value.

Identifier	Function
SIGMA	The sum of the arguments
PRODUCT	The product of the arguments
MIN	The smallest argument
MAX	The largest argument

Table 2: Table of Built-in Scalar Functions

All function arguments may themselves be expressions of the appropriate type. Table 1 contains a summary of the vector functions currently included in the language definition and table 2 contains a summary of scalar functions.

If any of the arguments of a scalar function are references to an array or a slice, the operation is applied to the entire array or slice. For example, if an array is passed as an argument to the function SIGMA, a scalar value equal to the sum of all the elements of that array will be returned (e.g. `Total_Flow_Out = SIGMA(Flow_Out);`). All function identifiers may be used in the declaration of model attributes, thereby locally overriding the built-in function definitions.