Semi-automated Modular Programming Construction for Physiological Modeling

by Gary M. Raymond and James B. Bassingthwaighte

Abstract

A major obstacle in writing complex programs for modeling physiological processes is the large amount of time it takes to code in a computer programming language the myriad processes taking place simultaneously in cells, tissues, and organs. Additional time is involved in debugging, testing, and validating the code before it can be used for the planning and analysis of experiments. Most instances of “modular” systems are incompletely described, are primarily used only by their creators despite claims of “ease of useability” and “human readable/writeable code”, and none have enjoyed wide scale adoption by the physiological modeling community. New paradigms for the creation of complex models are required. A modest attempt at creating a technology for shortening the time between thought and computational results is presented. Our experience is that we have shortened the time it takes to create complex models from weeks and months to hours and days.

Introduction:

There have been many attempts to provide modular programming systems for physiological applications (   ). All of them take slightly different approaches and are better or worse for some particular applications. None as yet have met with universal adoption by the physiological modeling community. We propose yet another system which we describe as semi-automated modular programming construction. It is not inherently novel, but it is easy to learn and use.

The Modular Program Constructor (MPC) is build upon the Mathematical Modeling Language (MML) of JSim (). MPC has three components. The first component is MML with its capabilities for declaring parameters and variables and defining algebraic, ordinary differential equations, and partial differential equation with their associated constraints, initial conditions and boundary conditions. The second component is a set of code libraries which contain the essential generic code modules (mathematical equations) for simple processes, e.g., flow in a capillary, diffusion in a distributed region, enzyme conversion of a molecule into another molecule, transport across a membrane, etc. The code libraries also contain modules for parameter and variable declarations, model “boilerplate”, entire models, etc. An example of a code library is illustrated in the Appendix. The short code library was generated by MPC and is used to make the first two example models.The third component is a set of directives which direct the selection of processes, gathering the code from existing models and code libraries, renaming parameters and variables to reflect the new purpose for which it will function, and automatically combining the mathematical structures into new structures.

Three simple examples of using MPC to generate models of increasing complexity are given. Example 1 has two species in a flowing region exchange with a stagnant region which exchanges with a second stagnant region exchanging with a third stagnant region where the irreversible conversion of species A to species B occurs. Example 2 relabels the two species as adenosine and inosine, changes the the regions designated 1, 2, and 3 to plasma, interstitial fluid, and cell respectively, adds a competitive transporter on the cell membrane and a reversible enzyme conversion process for adenosine and inosine in the cell. Example 3 takes the model generated in example 2 and makes it into a heterogeneous flow whole organ model with ten flow paths equally spaced in the logarithm of the relative flow range with weights selected from a probability density distribution. Statistics on mass balance and transit time are also added to the third example. The amount of code written for the MPC and generated by MPC is given in Table 1, along with the number of submodels used.

Model Name	Lines of input in .mpc*	Lines produced .mod	# of submodels used
A2B	27	83	9
Ado2Ino	48	174	18**
MultiFlowAdo2Ino	28	1358	183***

Table 1: Statistics on number of lines inputted to MPC, number of lines of code produced for JSim, and number of submodels included.

* Does not include blank lines. Continued //%GET and //%REPLACE directives counted as single lines. Does not include comments.

** Includes the 9 submodels in A2B.mod.

*** Includes ten times the number of submodels in Ado2Ino.mod.

Methods:

The Modular Program Constructor (MPC) has been designed to work with JSim's Mathematical Modeling Language (MML). MPC is a pre-compiler written in Java. It reads an input file, FileName.mpc, and generates an output file, FileName.mod, a JSim model file. The input file combines MML with “directives” embedded as comments and is able to utilize code from other JSim model files and constructed code libraries. The directives all begin with “`//%`” so they are comments. Directives control the identification, fetching, relabeling of variables and parameters, and assembling and recombining code into new equations. MPC currently has ten directives:

```
(1)     //%COM (comment directive not copied to final model file)
(2)     //%START codeBlockName
(3)     //%END codeBlockName
(4)     //%GET Model.mod codeBlockName (“oldName1=newName1”, ...)
(5)     //%REPLACE %replacer%=(“replacement1”, ... “replacementN”)
(6)     //%       ... )  continuation used only with GET and REPLACE
(7)     //%ENDREPLACE
(8)     //%COLLECT(“variableName”)
(9)     //%INSERTSTART codeBlockName
(10)    //%INSERTEND codeBlockName
```

COM directive

The COM directive indicates a comment in the input FileName.mpc file which is not copied to the output FileName.mod file. An example is given:

```
//%COM This comment is in Comment.mpc but WILL NOT appear in Comment.mod.
```

START and END directives

The START and END directives encapsulate a block of code. Code can consist of "boiler plate" for models, variable declarations, equations, comments or any combination of these elements. The encapsulating directives are inserted in existing models where they function as comments. The syntax of these directives is

```
//%START codeBlockName
. . .
//%END codeBlockName
```

Examples of encapsulating directives are shown in FlowDiffusion.mod, the blood tissue exchange model for a single capillary:

```
import nsrunit; unit conversion on;
math FlowDiffusion { // Partial Differential Equation Model for capillary

//%START pdeDomains
realDomain  t s; t.min=0; t.max=30; t.delta = 0.1;
real L = 0.1 cm;
real Ndivx = 31;
realDomain x cm ; x.min=0; x.max=L; x.ct = Ndivx;
//%END pdeDomains

// Parameters and Variables
real F = 1 ml/(min);    // Flow
real V = 0.05 ml;       // Volume
real D = 1e-6 cm^2/s;   // Diffusion coefficient
real C0 = 0 mM;         // Initial Concentration
extern real Cin(t) mM;  // Input conc from func generator
real C(t,x) mM;         // Concentration
real Cout(t) mM;        // Outflow conc
when(t=t.min) C=C0;     // Set Init condition

//%START flowBC
when  (x=x.min) (-F*L/V)*(C-Cin)+D*C:x = 0;
when  (x=x.max) { C:x = 0; Cout = C;}
//%END flowBC

//%START flowDiffCalc
C:t = D*C:x:x
      -(F*L/V)*C:x;
//%END flowDiffCalc
}
```

Continuation directives

The continuation directive is only used with the GET and REPLACE directives. At the beginning of MPC all continued statements are compressed to single lines. GET and REPLACE directives are therefore always treated as single lines. This will become important when discussing the REPLACE directive. The syntax of the continuation directive is

`//% ... )` where ". . ." stands for part of the continued directive.

GET directives

The GET directive identifies a file containing desired code, what code block to copy, and the changing of old names (names of parameters and variables in the extracted code) to new names. The syntax of the GET directive is

```
//%GET FileName.extension codeBlockName ("oldName1=newName1", ...)
```

The codeBlockName in the GET directory must find a matching codeBlockName in the file named FileName.extension with the START and END directives. The GET directive can use the continuation directive as illustrated with the following example. The following code in `Adenosine.mpc`

```
//%GET FlowDiffusion.mod flowDiffCalc ("C=Adenosine","F=Flow","L=Lcap",
//% "V=Vcap","x=xcap","D=Dcap")
```

produces the following output in file Adenosine.mod:

```
Adenosine:t = Dcap*Adenosine:xcap:xcap
    -(Flow*Lcap/Vcap)*Adenosine:xcap;
// This MML file generated from Adenosine.mpc using MPC.
```

INSERTSTART and INSERTEND directives

The INSERTSTART and INSERTEND directives insert START and END directives in the output file identifying pieces of code for later use. The syntax of these directives is

```
//%INSERTSTART codeBlockName
//%INSERTEND codeBlockName
```

A primary use of the insert directives is to generate a code library where many pieces of code can be stored for ease of reference and use. An example of a short code library input and output file is shown in the Appendix.

REPLACE and ENDREPLACE directives

The REPLACE directive and it's companion directive, ENDREPLACE, govern replacing parameter and variable names in other statements and directives. It duplicates lines of code and other directives. The REPLACE directive consists of two parts, a “replacer” surrounded by “%” signs on the left of an “=” sign, and a list of “replacements” on the right hand side. The syntax of the REPLACE directive has multiple forms which are explained below. The following example will create the flow calculations for two PDE concentration variables

```
//%REPLACE %Species%=("ATP","ADP")
real %Species%(t,x) mM; // Concentration of %Species% in capillary
//%GET FlowPDE.mod flowPDECalc ("C=%Species%","F=Flow","L=Lcap",
//% "V=Vcap","x=xcap","D=D%Species%")
```

```
//%ENDREPLACE
```

produces

```
real ATP(t,x) mM;       // Concentration of ATP in capillary
real ADP(t,x) mM;       // Concentration of ADP in capillary
ATP:t = DATP*ATP:xcap:xcap
    -(Flow*Lcap/Vcap)*ATP:xcap;
ADP:t = DADP*ADP:xcap:xcap
    -(Flow*Lcap/Vcap)*ADP:xcap;
```

Multiple replacers on same level

There can be multiple replacements on the same replacement level surrounded by an outer set of parentheses. For example, `%Species%` and `%name%` are replacers on the same level. Each has the same number of replacements which is required.

```
//%REPLACE (%Species%=("Ado","Ino") ,
            %name% = (“adenosine”,”inosine”) )
real %Species%(t,x) mM; // Concentration of %name% in capillary
//%GET FlowPDE !ja.mod flowPDECalc ("C=%Species%","F=Flow","L=Lcap",
//% "V=Vcap","x=xcap","D=D%Species%")
//%ENDREPLACE
```

```
produces
```

```
real Ado(t,x) mM;       // Concentration of adenosine in capillary
real Ino(t,x) mM;       // Concentration of inosine in capillary
Ado:t = DAdo*Ado:xcap:xcap
    -(Flow*Lcap/Vcap)*Ado:xcap;
Ino:t = DIno*Ino:xcap:xcap
    -(Flow*Lcap/Vcap)*Ino:xcap;

// This MML file generated from AdoIno.mpc using MPC.
```

Replacements with numeric ranges

Replacements can have the form `"text#N#M"` where text is optional, N and M are integers and N is less than or equal to M. This type of replace is expanded into M-N+1 replacements beginning with `"textN"` and ending with `"“textM"`. The following example demonstrates this:

```
//%REPLACE (%Species%=("A#1#2"), %N%=("#1#2"),
//%        %vol%=("0.05","0.15"), %place%=("plasma",
//%        "endothelial cell") )
real %Species%(t,x) = mM;       // Concentration of %Species% in V%N%
(%place%)
real V%N% = %vol% ml/g; // V%N% is volume of %place%
//%GET Exchange.mod exchangeCalc( "C1=A1","C2=A2","PS=PSg")
//%ENDREPLACE
```

```
produces
```

```
real A1(t,x) = mM;      // Concentration of A1 in V1 (plasma)
real A2(t,x) = mM;      // Concentration of A2 in V2 (endothelial cell)
real V1 = 0.05 ml/g;    // V1 is volume of plasma
real V2 = 0.15 ml/g;    // V2 is volume of endothelial cell
A1:t    = PSg/V1*(A2-A1);
A2:t    = PSg/V2*(A1-A2);
// This MML file generated from A.mpc using MPC.
```

Embedded REPLACE directives

REPLACE directives can be embedded. The innermost directives are executed first. The REPLACE directives are executed from left to right.

COLLECT directive

The COLLECT directive assembles equations with matching left hand sides. The syntax is

```
//%COLLECT("VariableName").
```

Most often the variable name is `VariableName:t`, the time derivative of a variable in MML

Processing of directives and removal of Duplicated lines of code

The order of processing of the code is as follows:
While there are remaining GET and REPLACE directives, (1) compress the GET and REPLACE directives into single lines of code; perform all REPLACE directives in the following order (2) from most embedded to least embedded and (3) from left to right for replacers and replacements; (4)perform all GET directives from first to last; (5) if the GET directives have returned additional GET or REPLACE directives repeat steps (1) through (4) until no GET and REPLACE directives remain.

Remove all duplicate lines of code with the exception of lines which contain only "/*", "//", and "*/". It may be necessary to modify some entries in the Code Library, for example,

```
A = if(C>D) C else
    0.0;
B = if (D>C) D else
    0.0;
```

produces

```
A = if(C>D) C else
    0.0;
B = if (D>C) D else
```

Note the missing line of code. However by shifting the fourth statement to the right,

```
A = if(C>D) C else
    0.0;
B = if (D>C) D else
```

```
       0.0;
```

the correct code is produced:

```
A = if(C>D) C else
    0.0;
B = if (D>C) D else
      0.0;
```

Finally, all of the COLLECT directives are performed. When a COLLECT directive is executed, the assembled differential equation is placed where the first instance of the variable being collected is encountered as the left hand side of a calculation. Therefore, it is imperative that the user declare all variables in the collected equation before they are used. We encourage the practice of declaring all variables and parameters at the front end of an MML model.

Results:

We demonstrate the utility of this method with the three examples mentioned above in the introduction.

Example 1: Using all the directives

The first example uses all of the directives to generate a model for three-regions (flowing plasma, stagnant interstitial fluid and cell; exchange between plasma and interstitial fluid region and also between the interstitial fluid region and the cell) for two species with conversion of the first species to the second species in the cell. The regions will be labeled “1”, “2”, and “3”, and the species “A” and “B”. Sample output is displayed in Figure 1 when the inflowing concentration for species A is given by the Longtail function provided by the function generator and there is no inflowing concentration for species B.

The input file to MPC: A2B.mpc

```
/* SHORT DESCRIPTION: A two species (A and B) model in
   3 regions (1=flowing, 2 and 3 stagnant) with A->B
   in region 3 */
import nsrunit; unit conversion on;
math A2B {
//%REPLACE %CL% =("../SHORTCODELIB/ShortCodeLibrary.mod")
//%REPLACE (%N%=("#1#3"), %vol%=("0.05","0.15","0.60") )
//%REPLACE (%P%=("12","23"), %s1%=("#1#2"),
//%         %s2%=("#2#3"), %R%=("3","5") )
//%REPLACE %AB%=("A","B")

// INDEPENDENT VARIABLES
//%GET %CL% pdeDomains()

//%INSERTSTART a2bParmsVars
// PARAMETERS
real Flow = 1 ml/(g*min);     // Flow rate
real PS%AB%%P% = %R% ml/(g*min);    // Exchg rate
real Ga2b = 10 ml/(g*min);    // Conversion rate
real V%N% = %vol% ml/g;       // Volume of V%N%
real D%AB%%N% = 1e-6 cm^2/sec;      // Diffusion coeff
extern real %AB%in(t) mM;     // Inflowing concentration
// DEPENDENT VARIABLES
real %AB%out(t) mM;           // Outflowing concentration
real %AB%%N%(t,x) mM;         // Concentration
// INITIAL CONDITIONS
when(t=t.min)  %AB%%N%=0;
// BOUNDARY CONDITIONS
//%GET %CL% flowBC ("C=%AB%1","V=V1","F=Flow","D=D%AB%1",
//%                     "Cin=%AB%in","Cout=%AB%out")
//%GET %CL% noFlowBC ("C=%AB%%s2%","D=%AB%%s2%")
//%INSERTEND a2bParmsVars

//%INSERTSTART a2bCalc
// PDE CALCULATIONS
//%GET %CL% flowDiffCalc  ("C=%AB%1","V=V1","F=Flow","D=D%AB%1")
//%GET %CL% diffusionCalc ("C=%AB%%s2%","D=D%AB%%s2%");
//%GET %CL% exchangeCalc ("C1=%AB%%s1%","V1=V%s1%","PS=PS%AB%%P%",
//%                       "C2=%AB%%s2%","V2=V%s2%")
//%GET %CL% reactionCalc  ("A=A3","B=B3","V=V3","G=Ga2b")
//%COLLECT("%AB%%N%:t")
//%INSERTEND a2bCalc

//%ENDREPLACE
//%ENDREPLACE
//%ENDREPLACE
//%ENDREPLACE
}
```

The output file from MPC: A2B.mod

```
/* SHORT DESCRIPTION: A two species (A and B) model in
   3 regions (1=flowing, 2 and 3 stagnant) with A->B
   in region 3 */
import nsrunit; unit conversion on;
math A2B {

// INDEPENDENT VARIABLES
realDomain  t s; t.min=0; t.max=30; t.delta = 0.1;
real L = 0.1 cm;
real Ndivx = 31;
realDomain x cm ; x.min=0; x.max=L; x.ct = Ndivx;
//%START a2bParmsVars
// PARAMETERS
real Flow = 1 ml/(g*min);     // Flow rate
real PSA12 = 3 ml/(g*min);    // Exchg rate
real PSA23 = 5 ml/(g*min);    // Exchg rate
real PSB12 = 3 ml/(g*min);    // Exchg rate
real PSB23 = 5 ml/(g*min);    // Exchg rate
real Ga2b = 10 ml/(g*min);    // Conversion rate
real V1 = 0.05 ml/g;          // Volume of V1
real V2 = 0.15 ml/g;          // Volume of V2
real V3 = 0.60 ml/g;          // Volume of V3
real DA1 = 1e-6 cm^2/sec;     // Diffusion coeff
real DA2 = 1e-6 cm^2/sec;     // Diffusion coeff
real DA3 = 1e-6 cm^2/sec;     // Diffusion coeff
real DB1 = 1e-6 cm^2/sec;     // Diffusion coeff
real DB2 = 1e-6 cm^2/sec;     // Diffusion coeff
real DB3 = 1e-6 cm^2/sec;     // Diffusion coeff
extern real Ain(t) mM;  // Inflowing concentration
extern real Bin(t) mM;  // Inflowing concentration
// DEPENDENT VARIABLES
real Aout(t) mM;        // Outflowing concentration
real Bout(t) mM;        // Outflowing concentration
real A1(t,x) mM;        // Concentration
real A2(t,x) mM;        // Concentration
real A3(t,x) mM;        // Concentration
real B1(t,x) mM;        // Concentration
real B2(t,x) mM;        // Concentration
real B3(t,x) mM;        // Concentration
// INITIAL CONDITIONS
when(t=t.min)  A1=0;
when(t=t.min)  A2=0;
when(t=t.min)  A3=0;
when(t=t.min)  B1=0;
when(t=t.min)  B2=0;
when(t=t.min)  B3=0;
// BOUNDARY CONDITIONS
when  (x=x.min) (-Flow*L/V1)*(A1-Ain)+DA1*A1:x = 0;
when  (x=x.max) { A1:x = 0; Aout = A1;}
when  (x=x.min) (-Flow*L/V1)*(B1-Bin)+DB1*B1:x = 0;
when  (x=x.max) { B1:x = 0; Bout = B1;}
when(x=x.min)  A2:x=0;
when(x=x.max)  A2:x=0;
when(x=x.min)  A3:x=0;
when(x=x.max)  A3:x=0;
when(x=x.min)  B2:x=0;
when(x=x.max)  B2:x=0;
when(x=x.min)  B3:x=0;
when(x=x.max)  B3:x=0;
//%END a2bParmsVars
//%START a2bCalc
// PDE CALCULATIONS
A1:t = -(Flow*L/V1)*A1:x
     + DA1*A1:x:x
     +PSA12/V1*(A2-A1);
B1:t = -(Flow*L/V1)*B1:x
     + DB1*B1:x:x
     +PSB12/V1*(B2-B1);
A2:t = DA2*A2:x:x
     +PSA12/V2*(A1-A2)
     +PSA23/V3*(A3-A2);
A3:t = DA3*A3:x:x
     +PSA23/V3*(A2-A3)
     -Ga2b/V3*A3;
B2:t = DB2*B2:x:x
     +PSB12/V2*(B1-B2)
     +PSB23/V3*(B3-B2);
B3:t = DB3*B3:x:x
     +PSB23/V3*(B2-B3)
     +Ga2b/V3*A3;
//%END a2bCalc
}
```

```
// This MML file generated from ../PDF_EXAMPLES/A2B.mpc using MPC.
```

Jsim produced Figure 1 when running the model for 60 seconds with Ain, the inflowing concentration of species A, equal the Longtail function from the function generator and Bin, the inflowing concentration of B equal zero.

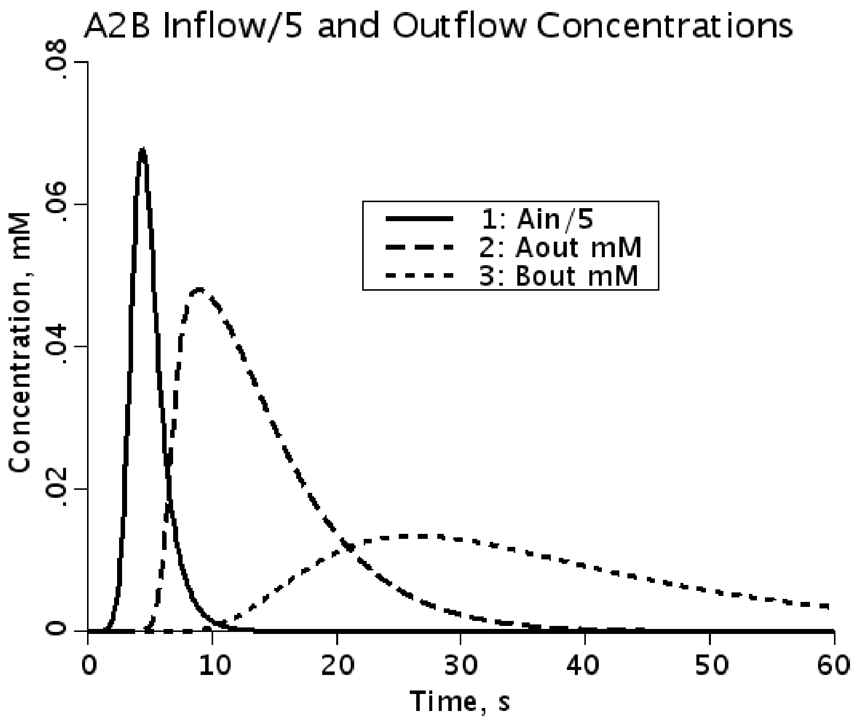

Figure 1: The inflowing concentration of A divided by 5 (solid), the capillary outflow of A (dashed) and B (dotted) are plotted as functions of time.

Example 2: A more complex model

The regional labels, “1”, “2”, and “3”, are replaced by slightly more descriptive labels, “p” for plasma, “i “ for interstitial fluid, and “c” for cell. Labels “A” and “B” are changed to “Ado” for adenosine and “Ino” for inosine. In addition to the passive exchange between the interstitial fluid region and the cell, a competitive transporter on the cell membrane for adenosine and inosine will be added. The competitive transporter is composed of four copies of a on-off membrane binding model and three copies of a conformational change model. In addition to the irreversible conversion of adenosine to inosine in the cell, a reversible enzyme conversion process is added. This model will use code from the previous model.

The input file to MPC: Ado2Ino.mpc:

```
/* SHORT DESCRIPTION: A modified version of A2B with
   relabeling the two species for adenosine (Ado)
   and inosine (Ino) in three regions (plasma (p),
   interstitial fluid region (i) and parenchymal cell
   (c) ) with addition of competitive transporter for
   Ado and Ino on cell membrane and enzyme conversion
   of Ado to Ino.
*/
import nsrunit; unit conversion on;
math Ado2Ino {
//%INSERTSTART ado2inoModel
//%REPLACE  %CL% =("../SHORTCODELIB/ShortCodeLibrary.mod")
//%REPLACE %PL%=("../PDF_EXAMPLES/")
//%GET %CL% pdeDomains()

//%GET %PL%A2B.mod a2bParmsVars("A1=Adop","A2=Adoi","A3=Adoc",
//%                    "Aout=Adoout","Ain=Adoin",
//%                    "B1=Inop","B2=Inoi","B3=Inoc","Bout=Inoout","Bin=Inoin",
//%                    "V1=Vp",  "V2=Vi",  "V3=Vc",
//%                    "DA1=DpAdo","DA2=DiAdo","DA3=DcAdo",
//%                    "DB1=DpIno","DB2=DiIno","DB3=DcIno",
//%                    "PSA12=PSgAdo","PSB12=PSgIno",
//%                    "PSA23=PSpcAdo","PSB23=PSpcIno",
//%                    "Ga2b=Gado2ino")

//%REPLACE (%ic%=("i","c"), %ci%=("c","i") )
//%REPLACE (%E_Ec% =("Enz","ECmplx"),%eic%=("Etot","ZEROM") )
//%REPLACE %AdoIno%=("Ado","Ino")
// ADDITIONAL PARAMETERS
private real ZEROT = 0 mmol/cm^2;// Removes Initial Conditions from Parameter List
private real ZEROM = 0 mM;    // Removes Initial Conditions from Parameter List
//     ENZYME CONVERSION PARAMETERS
real Etot   = 0.001 mM,       // Enzyme Concentration in cell
     kf1    = 10 mM^(-1)*s^(-1),    // rate const ado+enz->ecmplx
     kb1    = 10 s^(-1),            // rate const ecmplx->ado+enz
     kf2    = 10 sec^(-1),          // rate const ecmplx->ino+enz
     kb2    = 0 mM^(-1)*s^(-1);     // rate const ino+enz->ecmplx
//     COMPETITIVE TRANSPORTER PARAMETERS
real Ttot   = 7e-6 mmol/cm^2; // Transporter density on membrane
real Kd%AdoIno%%ic%     = 1 mM;           // Equilib Dissoc const
real kon%AdoIno%%ic%    = 10 mM^(-1)*s^(-1);    // Bind rate
real kof%AdoIno%%ic%    = Kd%AdoIno%%ic%*kon%AdoIno%%ic%;        // Dissoc rate
real S            = 1 cm^2/g;       // Surface area
real SoV%ic%      = S/V%ic%;        // Surface to volume ratio
real kT%ic%2%ci%  = 100 sec^(-1);   // Flip rate
real kT%AdoIno%%ic%2%ci%       = 100 sec^(-1);// Flip rate ;
// DEPENDENT VARIABLES
real %E_Ec%(t,x) mM;          // Concentration of %E_Ec% in Vc
real T%AdoIno%%ic%(t,x) mmol/cm^2;       // Transporter complex
real T%ic%(t,x) mmol/cm^2;          // Free transporter
// INITIAL CONDITIONS
when(t=t.min) %E_Ec%    = %eic%;
when(t=t.min) T%AdoIno%%ic%   = ZEROT;
when(t=t.min) T%ic%     = Ttot/2;
// BOUNDARY CONDITIONS
//%GET %CL% noFlowBC ("C=%E_Ec%")
//%GET %CL% noFlowBC ("C=T%AdoIno%%ic%")
//%GET %CL% noFlowBC ("C=T%ic%")
// PDE CALCULATIONS
//%GET %PL%A2B.mod a2bCalc("A1=Adop","A2=Adoi","A3=Adoc","Aout=Adoout","Ain=Adoin",
//%                    "B1=Inop","B2=Inoi","B3=Inoc","Bout=InoOut","Bin=Inoin",
//%                    "V1=Vp",  "V2=Vi",  "V3=Vc",
//%                    "DA1=DpAdo","DA2=DiAdo","DA3=DcAdo",
//%                    "DB1=DpIno","DB2=DiIno","DB3=DcIno",
//%                    "PSA12=PSgAdo","PSB12=PSgIno",
//%                    "PSA23=PSpcAdo","PSB23=PSpcIno",
//%                    "Ga2b=Gado2ino")
//%GET %CL% onOffMembraneCalc("M=%AdoIno%%ic%","B=T%ic%","MB=T%AdoIno%%ic%",
//%    "kon=kon%AdoIno%%ic%","kof=kof%AdoIno%%ic%","SoV=SoV%ic%")
//%GET %CL% flipa2bCalc("a=T%AdoIno%i","b=T%AdoIno%c",
//%                     "ka2b=kT%AdoIno%i2c","kb2a=kT%AdoIno%c2i")
//%GET %CL% flipa2bCalc("a=Ti","b=Tc",
//%                     "ka2b=kTi2c","kb2a=kTc2i")
//%GET %CL% enzymeCalc("A=Adoc","B=Inoc","Enzyme=Enz",
//%                    "Complex=ECmplx")
//%COLLECT("%AdoIno%%ci%:t")
//%COLLECT("%E_Ec%:t")
//%COLLECT("T%AdoIno%%ic%:t")
//%COLLECT("T%ic%:t")
//%INSERTEND ado2inoModel
//%ENDREPLACE
//%ENDREPLACE
//%ENDREPLACE
//%ENDREPLACE
//%ENDREPLACE
}
```

The output file from MPC: Ado2Ino.mod

```
/* SHORT DESCRIPTION: A modified version of A2B with
   relabeling the two species for adenosine (Ado)
   and inosine (Ino) in three regions (plasma (p),
   interstitial fluid region (i) and parenchymal cell
   (c) ) with addition of competitive transporter for
   Ado and Ino on cell membrane and enzyme conversion
   of Ado to Ino.
*/
import nsrunit; unit conversion on;
math Ado2Ino {
//%START ado2inoModel
// INDEPENDENT VARIABLES
realDomain  t s; t.min=0; t.max=30; t.delta = 0.1;
real L = 0.1 cm;
real Ndivx = 31;
realDomain x cm ; x.min=0; x.max=L; x.ct = Ndivx;

// PARAMETERS
real Flow = 1 ml/(g*min);     // Flow rate
real PSgAdo = 3 ml/(g*min);   // Exchg rate
real PSpcAdo = 5 ml/(g*min);  // Exchg rate
real PSgIno = 3 ml/(g*min);   // Exchg rate
real PSpcIno = 5 ml/(g*min);  // Exchg rate
real Gado2ino = 10 ml/(g*min);      // Conversion rate
real Vp = 0.05 ml/g;          // Volume of Vp
real Vi = 0.15 ml/g;          // Volume of Vi
real Vc = 0.60 ml/g;          // Volume of Vc
real DpAdo = 1e-6 cm^2/sec;   // Diffusion coeff
real DiAdo = 1e-6 cm^2/sec;   // Diffusion coeff
real DcAdo = 1e-6 cm^2/sec;   // Diffusion coeff
real DpIno = 1e-6 cm^2/sec;   // Diffusion coeff
real DiIno = 1e-6 cm^2/sec;   // Diffusion coeff
real DcIno = 1e-6 cm^2/sec;   // Diffusion coeff
extern real Adoin(t) mM;      // Inflowing concentration
extern real Inoin(t) mM;      // Inflowing concentration
// DEPENDENT VARIABLES
real Adoout(t) mM;            // Outflowing concentration
real Inoout(t) mM;            // Outflowing concentration
real Adop(t,x) mM;            // Concentration
real Adoi(t,x) mM;            // Concentration
real Adoc(t,x) mM;            // Concentration
real Inop(t,x) mM;            // Concentration
real Inoi(t,x) mM;            // Concentration
real Inoc(t,x) mM;            // Concentration
// INITIAL CONDITIONS
when(t=t.min)  Adop=0;
when(t=t.min)  Adoi=0;
when(t=t.min)  Adoc=0;
when(t=t.min)  Inop=0;
when(t=t.min)  Inoi=0;
when(t=t.min)  Inoc=0;
// BOUNDARY CONDITIONS
when  (x=x.min) (-Flow*L/Vp)*(Adop-Adoin)+DpAdo*Adop:x = 0;
when  (x=x.max) { Adop:x = 0; Adoout = Adop;}
when  (x=x.min) (-Flow*L/Vp)*(Inop-Inoin)+DpIno*Inop:x = 0;
when  (x=x.max) { Inop:x = 0; Inoout = Inop;}
when(x=x.min)  Adoi:x=0;
when(x=x.max)  Adoi:x=0;
when(x=x.min)  Adoc:x=0;
when(x=x.max)  Adoc:x=0;
when(x=x.min)  Inoi:x=0;
when(x=x.max)  Inoi:x=0;
when(x=x.min)  Inoc:x=0;
when(x=x.max)  Inoc:x=0;
// ADDITIONAL PARAMETERS
private real ZEROT = 0 mmol/cm^2;// Removes Initial Conditions from Parameter List
private real ZEROM = 0 mM;    // Removes Initial Conditions from Parameter List
//     ENZYME CONVERSION PARAMETERS
real Etot   = 0.001 mM,       // Enzyme Concentration in cell
     kf1    = 10 mM^(-1)*s^(-1),    // rate const ado+enz->ecmplx
     kb1    = 10 s^(-1),            // rate const ecmplx->ado+enz
     kf2    = 10 sec^(-1),          // rate const ecmplx->ino+enz
     kb2    = 0 mM^(-1)*s^(-1);     // rate const ino+enz->ecmplx
//     COMPETITIVE TRANSPORTER PARAMETERS
real Ttot   = 7e-6 mmol/cm^2; // Transporter density on membrane
real KdAdoi = 1 mM;           // Equilib Dissoc const
real KdAdoc = 1 mM;           // Equilib Dissoc const
real KdInoi = 1 mM;           // Equilib Dissoc const
real KdInoc = 1 mM;           // Equilib Dissoc const
real konAdoi      = 10 mM^(-1)*s^(-1);    // Bind rate
real konAdoc      = 10 mM^(-1)*s^(-1);    // Bind rate
real konInoi      = 10 mM^(-1)*s^(-1);    // Bind rate
real konInoc      = 10 mM^(-1)*s^(-1);    // Bind rate
real kofAdoi      = KdAdoi*konAdoi;       // Dissoc rate
real kofAdoc      = KdAdoc*konAdoc;       // Dissoc rate
real kofInoi      = KdInoi*konInoi;       // Dissoc rate
real kofInoc      = KdInoc*konInoc;       // Dissoc rate
real S            = 1 cm^2/g;       // Surface area
real SoVi   = S/Vi;           // Surface to volume ratio
real SoVc   = S/Vc;           // Surface to volume ratio
real kTi2c  = 100 sec^(-1);   // Flip rate
real kTc2i  = 100 sec^(-1);   // Flip rate
real kTAdoi2c      = 100 sec^(-1);// Flip rate ;
real kTAdoc2i      = 100 sec^(-1);// Flip rate ;
real kTInoi2c      = 100 sec^(-1);// Flip rate ;
real kTInoc2i      = 100 sec^(-1);// Flip rate ;
real Enz(t,x) mM;       // Concentration of Enz in Vc
real ECmplx(t,x) mM;          // Concentration of ECmplx in Vc
real TAdoi(t,x) mmol/cm^2;          // Transporter complex
real TAdoc(t,x) mmol/cm^2;          // Transporter complex
real TInoi(t,x) mmol/cm^2;          // Transporter complex
real TInoc(t,x) mmol/cm^2;          // Transporter complex
real Ti(t,x) mmol/cm^2;       // Free transporter
real Tc(t,x) mmol/cm^2;       // Free transporter
when(t=t.min) Enz = Etot;
when(t=t.min) ECmplx    = ZEROM;
when(t=t.min) TAdoi     = ZEROT;
when(t=t.min) TAdoc     = ZEROT;
when(t=t.min) TInoi     = ZEROT;
when(t=t.min) TInoc     = ZEROT;
when(t=t.min) Ti  = Ttot/2;
when(t=t.min) Tc  = Ttot/2;
when(x=x.min)  Enz:x=0;
when(x=x.max)  Enz:x=0;
when(x=x.min)  ECmplx:x=0;
when(x=x.max)  ECmplx:x=0;
when(x=x.min)  TAdoi:x=0;
when(x=x.max)  TAdoi:x=0;
when(x=x.min)  TAdoc:x=0;
when(x=x.max)  TAdoc:x=0;
when(x=x.min)  TInoi:x=0;
when(x=x.max)  TInoi:x=0;
when(x=x.min)  TInoc:x=0;
when(x=x.max)  TInoc:x=0;
when(x=x.min)  Ti:x=0;
when(x=x.max)  Ti:x=0;
when(x=x.min)  Tc:x=0;
when(x=x.max)  Tc:x=0;
// PDE CALCULATIONS
Adop:t = -(Flow*L/Vp)*Adop:x
     + DpAdo*Adop:x:x
     +PSgAdo/Vp*(Adoi-Adop);
Inop:t = -(Flow*L/Vp)*Inop:x
     + DpIno*Inop:x:x
     +PSgIno/Vp*(Inoi-Inop);
Adoi:t = DiAdo*Adoi:x:x
       +PSgAdo/Vi*(Adop-Adoi)
       +PSpcAdo/Vc*(Adoc-Adoi)
       +(-konAdoi*Adoi*Ti + kofAdoi*TAdoi)*SoVi;
Adoc:t = DcAdo*Adoc:x:x
       +PSpcAdo/Vc*(Adoi-Adoc)
       -Gado2ino/Vc*Adoc
       +(-konAdoc*Adoc*Tc + kofAdoc*TAdoc)*SoVc
       -kf1*Adoc*Enz + kb1*ECmplx;
Inoi:t = DiIno*Inoi:x:x
```

```
       +PSgIno/Vi*(Inop-Inoi)
       +PSpcIno/Vc*(Inoc-Inoi)
       +(-konInoi*Inoi*Ti + kofInoi*TInoi)*SoVi;
Inoc:t = DcIno*Inoc:x:x
       +PSpcIno/Vc*(Inoi-Inoc)
       +Gado2ino/Vc*Adoc
       +(-konInoc*Inoc*Tc + kofInoc*TInoc)*SoVc
       +kf2*ECmplx - kb2*Inoc*Enz;
Ti:t = -konAdoi*Adoi*Ti + kofAdoi*TAdoi
     -konInoi*Inoi*Ti + kofInoi*TInoi
     - kTi2c*Ti + kTc2i*Tc;
TAdoi:t = konAdoi*Adoi*Ti - kofAdoi*TAdoi
        - kTAdoi2c*TAdoi + kTAdoc2i*TAdoc;
Tc:t = -konAdoc*Adoc*Tc + kofAdoc*TAdoc
     -konInoc*Inoc*Tc + kofInoc*TInoc
     +kTi2c*Ti - kTc2i*Tc;
TAdoc:t = konAdoc*Adoc*Tc - kofAdoc*TAdoc
        +kTAdoi2c*TAdoi - kTAdoc2i*TAdoc;
TInoi:t = konInoi*Inoi*Ti - kofInoi*TInoi
        - kTInoi2c*TInoi + kTInoc2i*TInoc;
TInoc:t = konInoc*Inoc*Tc - kofInoc*TInoc
        +kTInoi2c*TInoi - kTInoc2i*TInoc;
Enz:t = -(kf1*Adoc + kb2*Inoc)*Enz
      + (kb1 + kf2)*ECmplx;
ECmplx:t = (kf1*Adoc + kb2*Inoc)*Enz
         - (kb1 + kf2)*ECmplx;
//%END ado2inoModel
}
// This MML file generated from ../PDF_EXAMPLES/Ado2Ino.mpc using MPC.
```

Jsim produced Figure 2 while running the model for 30 seconds with Adoin(t) equal to the Longtail function, Inoin(t)= 0, Ttot=7e-4 mmol/cm^2, Gado2ino=0, PspcAdo=0, and PspcIno=0.

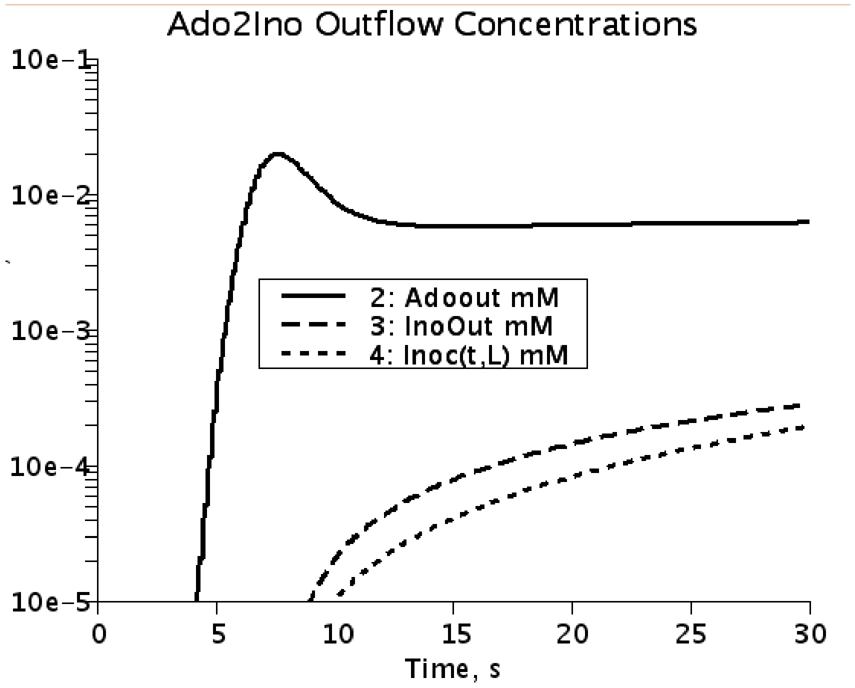

Figure 2: The outflow concentrations of adenosine(curve 2) and inosine (curve 3) are plotted with the concentration of inosine at the right end of the cell (curve 4). Note that even though inosine is being produced in the cell, it has a lower concentration than inosine in the outflow. This is caused by the inflow of adenosine moving the competitive transporters to the cell side of the membrane.

Example 3: A heterogeneous multi-flow model

A heterogeneous multi-flow model representing an organ is constructed using ten copies of Ado2Ino.mod with the multi-flow algorithm. A modification is made to Ado2Ino.mod, removing the flow declaration because the flow declarations will be generated by the extended multi-flow code. Variable names that will be subscripted, e.g., Adop (adenosine in the plasma becomes Adop1, Adop2, ... Adop10). Additionally statistics on the summed outflow concentrations (area of curves, transit time, etc.) have been added. The output file for example 3 is not shown because of its length.

Input file for MPC: MultiUserAdo2Ino.mpc

```
//%GET Transit.mod curveStatJava()
import nsrunit; unit conversion on;
math multiFlowAdo2Ino {
//%GET extendedMultiFlow.mod multiFlowCalc1("PATHS=10")
//%REPLACE %n%=("#1#10")
real UserF%n%     = 0;
real UserWt%n%    = 0;
userF =
        if( abs(NP-%n%)<0.1) UserF%n% else
        0;
userWt =
         if( abs(NP-%n%)<0.1) UserWt%n% else
         0;
//%GET extendedMultiFlow.mod multiFlowCalc2()
//%GET CodeLibrary.mod pdeDomains()
real Fmean = 1 ml/(g*min);
real Fi(NP) ml/(g*min);
Fi=Fmean*f;
real Fi%n% ml/(g*min);
Fi%n%=Fi(%n%);
//%GET NoFlowAdo2Ino.mod ado2inoModel("Flow=Fi%n%",
//% "Adoout=Adoout%n%", "Inoout=Inoout%n%",
//% "Adop=Adop%n%", "Adoi=Adoi%n%", "Adoc=Adoc%n%",
//% "Inop=Inop%n%", "Inoi=Inoi%n%", "Inoc=Inoc%n%",
//% "TAdoi=TAdoi%n%", "TAdoc=TAdoc%n%",
//% "TInoi=TInoi%n%", "TInoc=TInoc%n%",
//% "Ti=Ti%n%", "Tc=Tc%n%",
//% "Enz=Enz%n%", "ECmplx=ECmplx%n%")

//%REPLACE %k%=("Ado","Ino")
//%COLLECT("%k%p%n%:t")
// COLLECT OUTFLOWS SUMMED BY WEIGHTS
real %k%outTot(t) mM;
real %k%outw%n%(t) mM;
%k%outw%n% = %k%out%n%*wts(%n%);
%k%outTot = %k%outw%n%;
//%COLLECT("%k%outTot")
//%GET Transit.mod transitCalc("Cin@t=%k%in@t","Cout@t=%k%outTot@t",
//%                           "ai=ai%k%","ti=ti%k%", "Rdi=RDi%k%",
//%                           "ao=ao%k%","tao=to%k%","RDo=RDo%k%",
//%                           "tsys=tsys%k%","RDsys=RDsys%k%")
//%ENDREPLACE
//%ENDREPLACE
}
```

Figure 3 is combined from two separate figures produced by JSim running MultiFlowAdo2Ino.mod. It can be seen that the multi-flow model gives decreased maximum peak height and greater dispersion than the single flow path model.

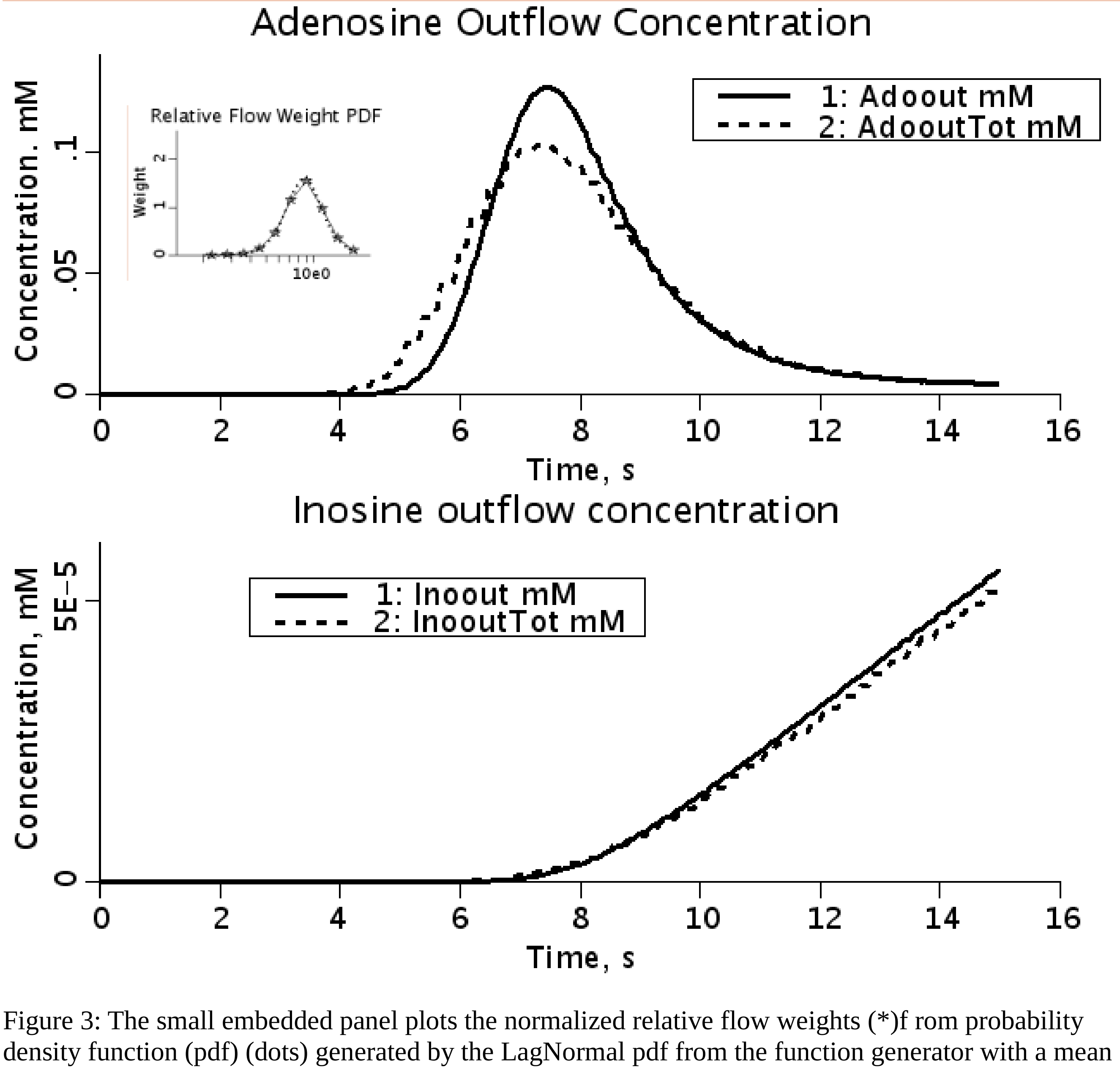

Figure 3: The small embedded panel plots the normalized relative flow weights (*)f rom probability density function (pdf) (dots) generated by the LagNormal pdf from the function generator with a mean of 1 and relative dispersion of 0.4. The pdf covers relative flows from 0.2 to 2.0. For the ten path model, the relative flows have been chosen to be equally spaced in the logarithm of relative flow. The

two larger plots show the difference between the Ado2Ino model outflow concentrations (solid lines) with the summed outflow concentrations from the MultiFlowAdo2Ino model (dashed lines).

Summary:

A limited set of directives allows us to build complex models using small models for simple processes. Using the MPC, we have generated a full organ model with heterogeneity of flow and competitive transporters on the cellular membrane surfaces for multiple species. MPC automatically combines both ordinary and partial differential equations under the control of ten easily understood directives to form new models. The amount of actual code a user needs to write is reduced, especially for more complicated models. The Java code for MPC, the examples presented here, and the JSim project files, along with instructions are available at html://www.physiome.org/?????

APPENDIX: ShortCodeLibrary input and output files

ShortCodeLibrary.mpc :

```
math ShortCodeLibrary {
real version = 1.0;
/*--------------- BEGIN CONSTRUCT MODULAR PROGRAM CONSTRUCTOR LIBRARY
//%REPLACE %CL%=("../SHORTCODELIB/")
//----------------------------- PDE DOMAINS
//%INSERTSTART           pdeDomains
// INDEPENDENT VARIABLES
//%GET %CL%FlowDiffusion.mod pdeDomains()
//%INSERTEND             pdeDomains
//----------------------------- BOUNDARY CONDITIONS
//%INSERTSTART           flowBC
//%GET %CL%FlowDiffusion.mod flowBC()
//%INSERTEND             flowBC

//%INSERTSTART       noFlowBC
//%GET %CL%Diffusion.mod noFlowBC()
//%INSERTEND         noFlowBC
//----------------------------- FLOW DIFF CALCULATION
//%INSERTSTART           flowDiffCalc
//%GET %CL%FlowDiffusion.mod flowDiffCalc()
//%INSERTEND             flowDiffCalc
//----------------------------- DIFFUSION CALCULATION
//%INSERTSTART       diffusionCalc
//%GET %CL%Diffusion.mod diffusionCalc()
//%INSERTEND         diffusionCalc
//----------------------------- EXCHANGE CACULATIONS
//%INSERTSTART      exchangeCalc
//%GET %CL%Exchange.mod exchangeCalc()
//%INSERTEND        exchangeCalc
//----------------------------- CONSUME CALCULATION
//%INSERTSTART      consumeCalc
//%GET %CL%Consume.mod  consumeCalc()
//%INSERTEND        consumeCalc
//----------------------------- REACTION A->B
//%INSERTSTART      reactionCalc
//%GET %CL%Reaction.mod reactionCalc()
//%INSERTEND        reactionCalc
//------------------------------- ON OFF MEMBRANE BINDING SITE
//%INSERTSTART           onOffMembraneCalc
//%GET %CL%OnOffMembrane.mod onOffMembraneCalc()
//%INSERTEND             onOffMembraneCalc
//------------------------------- CONFORMATIONAL CHANGE (FLIP)
//%INSERTSTART                  flipa2bCalc
//%GET %CL%ConformationalChange.mod flipa2bCalc()
//%INSERTEND                    flipa2bCalc
//------------------------------- ENZYME CONVERSION
//%INSERTSTART    enzymeCalc
//%GET %CL%Enzyme.mod enzymeCalc()
//%INSERTEND      enzymeCalc
//%ENDREPLACE
/*--------------- END CONSTRUCT MODULAR PROGRAM CONSTRUCTOR LIBRARY
}
```

ShortCodeLibrary.mod :

```
math ShortCodeLibrary {
real version = 1.0;
/*--------------- BEGIN CONSTRUCT MODULAR PROGRAM CONSTRUCTOR LIBRARY
//------------------------------- PDE DOMAINS
//%START           pdeDomains
// INDEPENDENT VARIABLES
realDomain  t s; t.min=0; t.max=30; t.delta = 0.1;
real L = 0.1 cm;
real Ndivx = 31;
realDomain x cm ; x.min=0; x.max=L; x.ct = Ndivx;
//%END             pdeDomains
//------------------------------- BOUNDARY CONDITIONS
//%START           flowBC
when  (x=x.min) (-F*L/V)*(C-Cin)+D*C:x = 0;
when  (x=x.max) { C:x = 0; Cout = C;}
//%END             flowBC

//%START       noFlowBC
when(x=x.min)  C:x=0;
when(x=x.max)  C:x=0;
//%END         noFlowBC
//------------------------------- FLOW DIFF CALCULATION
//%START           flowDiffCalc
C:t =  -(F*L/V)*C:x
       + D*C:x:x ;
//%END             flowDiffCalc
//------------------------------- DIFFUSION CALCULATION
//%START       diffusionCalc
C:t = D*C:x:x ;
//%END         diffusionCalc
//------------------------------- EXCHANGE CACULATIONS
//%START      exchangeCalc
C1:t  = PS/V1*(C2-C1);
C2:t  = PS/V2*(C1-C2);
//%END        exchangeCalc
//------------------------------- CONSUME CALCULATION
//%START      consumeCalc
C:t = -(G/V)*C;
//%END        consumeCalc
//------------------------------ REACTION A->B
//%START      reactionCalc
A:t = -G/V*A;
B:t = G/V*A;
//%END        reactionCalc
//------------------------------ ON OFF MEMBRANE BINDING SITE
//%START           onOffMembraneCalc
M:t = (-kon*M*B + kof*MB)*SoV;
B:t =  -kon*M*B + kof*MB ;
MB:t =  kon*M*B - kof*MB;
//%END             onOffMembraneCalc
//------------------------------ CONFORMATIONAL CHANGE (FLIP)
//%START                  flipa2bCalc
a:t = - ka2b*a + kb2a*b;
b:t =   ka2b*a - kb2a*b;
//%END                    flipa2bCalc
//------------------------------ ENZYME CONVERSION
//%START    enzymeCalc
A:t = -kf1*A*Enzyme + kb1*Complex;
B:t =  kf2*Complex - kb2*B*Enzyme;
Enzyme:t = -(kf1*A + kb2*B)*Enzyme
           + (kb1 + kf2)*Complex;
Complex:t = (kf1*A + kb2*B)*Enzyme
           - (kb1 + kf2)*Complex;
//%END      enzymeCalc
/*--------------- END CONSTRUCT MODULAR PROGRAM CONSTRUCTOR LIBRARY
}
// This MML file generated from ../SHORTCODELIB/ShortCodeLibrary.mpc using MPC.
```

References: