

# **FUNDER**

FUNCTIONAL DERIVATIVES FOR PREDICTING METABOLIC RESPONSES

SUPPLEMENTARY INFORMATION

Author: Antonio S. Torralba  
Version: 01.01.01

## INTRODUCTION

*Funder* is an application that illustrates the calculation of functional derivatives in metabolic networks. Functional derivatives are closely related to the susceptibilities of a nonlinear system and can be used to predict its response to external perturbations. One input flux is considered to be the excitation of the system, whereas the response is any given velocity or output flux. Although it could be defined to be a concentration, this possibility has not been implemented in *Funder*. Estimations of the response can be given for any variation (perturbation) of the input flux with respect to a reference steady state. It is not required that such a variation be small, although the accuracy of the approximation will depend on the order of the approximation, i.e. on how many functional derivatives of increasing orders are used.

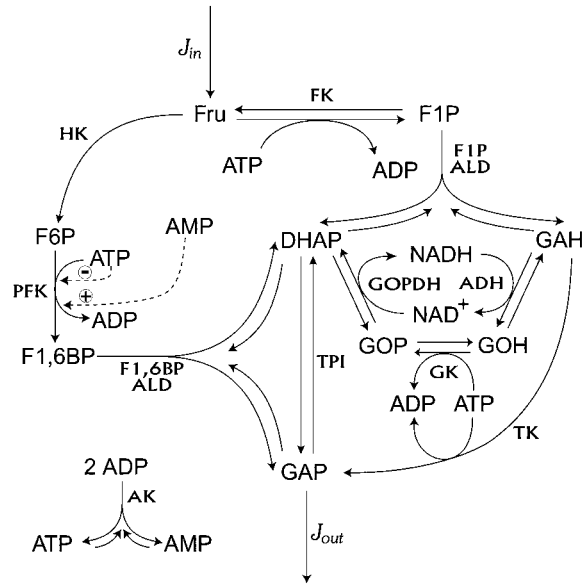
*Funder* was implemented in C++. The class *Susceptibility* was defined to calculate and efficiently store the susceptibilities associated to an object of the abstract class *System*. *System* provides response data to *Susceptibility* as required. In order to minimize the number of queries to *System*, *Susceptibility* stores the results in a multiple linked list (classes *IntegList* and *IntegListNode*). The user must derive a new class from *System* that contains the actual description of the system in terms of differential equations and appropriate parameters. However, the class *System* provides a suitable interface with *Susceptibility* and methods for the numerical integration of the equations. In principle, *System* could be substituted by a database of experimental data, as long as the queries of *Susceptibility* can be answered (see description below). Two supporting classes are provided, *Signal* and *Response*, that help manipulate signals and responses of several orders. Methods include convolution, deconvolution, addition and subtraction of signals and convolution of signals and susceptibilities, among others.

A part of fructose catabolism was implemented in *Funder* to illustrate the use of *Susceptibility*, *System*, *Signal* and *Response*, and the way a user can derive a concrete system from the abstract class *System*. This document is devoted to the description of these classes and the parameters and dynamics of the example system. Instructions for compilation and descriptions of the source and output files are also given.

## DESCRIPTION OF THE EXAMPLE SYSTEM

A section of fructose catabolism was chosen as an example of a metabolic network (Fig. 1). A dynamic model was built using deterministic ordinary differential equations (see below), composed of 11 enzymes and 14 metabolites (Tables 1 and 2). This network is a branched system. The branch involving fructose 1-phosphate occurs in liver, whereas in muscle fructose is converted to fructose 6-phosphate [Voet, D. and Voet J.G., *Biochemistry*, 1990, John Wiley & Sons, Inc., New York.]. Transport processes

between tissues were not included, because they are immaterial to the purpose of illustrating the calculation of functional derivatives.



**Figure 1.** Reaction scheme of a portion of fructose catabolism and the adenylate kinase reaction.  
For abbreviations, see Tables 1 and 2.

**Table 1.** Abbreviations of metabolites

METABOLITE	ABBREVIATION
Fructose	Fru
Fructose 1-Phosphate	F1P
Fructose 6-Phosphate	F6P
Fructose 1,6-Bisphosphate	F1,6BP
Dihydroxyacetone Phosphate	DHAP
Glyceraldehyde	GAH
Glycerol	GOH
Glycerol 3-Phosphate	GOP
Glyceraldehyde 3-Phosphate	GAP
Adenosine Triphosphate	ATP
Adenosine Diphosphate	ADP
Adenosine Monophosphate	AMP
Nicotinamide Adenine Dinucleotide (Ox.)	NAD
Nicotinamide Adenine Dinucleotide (Red.)	NADH

The only input to the system is a flux of fructose ( $J_{in}$ ). Transformation of this metabolite into the intermediates of the network leads to glyceraldehyde 3-phosphate, which is irreversibly extracted with first-order kinetics ( $J_{out}$ ). Due to the breakage of the hexoses

into trioses, the overall stoichiometry of the network is 1:2. As a consequence, the area under the first-order susceptibility of the system is 2. (After executing **funder**, the file **sums.dat** is generated that contains the integrals of the susceptibilities. A value of 1.885604 is obtained only because the system reaches a steady state in a very slow way and hence the susceptibilities cannot be calculated completely.)

**Table 2.** Abbreviations of enzymes

ENZYME	ABBREVIATION	E.C. NUMBER	MECHANISM
Hexokinase	HK	2.7.1.1	Irreversible Random Bi-Bi
Phosphofructokinase	PFK	2.7.1.11	Hill
F1,6BP Aldolase	F1,6BP ALD	4.1.2.13	Reversible Ordered Uni-Bi
F1P Aldolase	F1P ALD	4.1.2.13	Reversible Ordered Uni-Bi
Fructokinase	FK	2.7.1.4	Reversible Ordered Bi-Bi
Alcohol Dehydrogenase	ADH	1.1.1.1	Reversible Ordered Bi-Bi
Glycerokinase	GK	2.7.1.30	Reversible Ordered Bi-Bi
GOP Dehydrogenase	GOPDH	1.1.99.5	Reversible Ordered Bi-Bi
Triokinase	TK	2.7.1.28	Irreversible Ordered Bi-Bi
Triose Phosphate Isomerase	TPI	5.3.1.1	Equilibrium
Adenylate Kinase	AK	2.7.4.3	Equilibrium

The reactions catalyzed by triose phosphate isomerase and adenylate kinase are considered to be in equilibrium, with the following equilibrium constants:

$$K_{eq}^{TPI} = \frac{[GAP]}{[DHAP]} \quad (1)$$

$$K_{eq}^{AK} = \frac{[ATP][AMP]}{[ADP]} \quad (2)$$

Two conservation relationships hold:

$$[NADH] + [NAD] = NAD_T \quad (3)$$

$$[ATP] + [ADP] + [AMP] = AP_T \quad (4)$$

The rate equations used in the model are described in Table 3 and the values of the parameters, including equilibrium constants and others, are listed in Tables 4-9. The equilibrium constant of triose phosphate isomerase, Eq. (1), can be used to eliminate one variable, by defining the pool  $[GAP] + [DHAP]$ . In addition, the concentrations of adenylates can be calculated, for a given ratio  $r = [ATP]/[ADP]$ , from the conservation Eq. (4) and the equilibrium constant of adenylate kinase, Eq. (3). These simplifications are listed below, along with a list of the differential equations of the model.

**Table 3.** Rate equations used in the model

MECHANISM	RATE EQUATION	MEANING
Hill	$v = \frac{V^+ [Sus]^n}{K_M + K_M R \frac{[Inh]^n}{[Act]^n} + [Sus]^n}$	$V^+$ Maximum Velocity
		$K_M$ Michaelis Constant
		$R$ R-T Equilibrium Constant
		$n$ Hill Coefficient
		$Sus$ Substrate
		$Inh$ Inhibitor
		$Act$ Activator
Reversible Ordered Uni-Bi	$v = V^+ \frac{\frac{[Sus]}{K_{Sus}} \left(1 - \frac{\Gamma}{K_{eq}}\right)}{\left(1 + \frac{[Sus]}{K_{Sus}} + \frac{[P_1]}{K_{P_1}} + \frac{[P_2]}{K_{P_2}} + \frac{[Sus][P_2]}{K_{Sus}K_{P_2}^i} + \frac{[P_1][P_2]}{K_{P_1}K_{P_2}}\right)}$	$V^+$ Maximum Velocity
		$K_{Sus}$ Subs. Michaelis Constant
		$\Gamma$ Mass Action Ratio
		$K_{eq}$ Equilibrium Constant
		$K_{P_1}$ Prod. 1 Michaelis Constant
		$K_{P_2}$ Prod. 2 Michaelis Constant
		$K_{P_2}^i$ Prod. 2 Inhibition Constant
		$Sus$ Substrate
		$P_1$ Product 1
		$P_2$ Product 2
Irreversible Random Bi-Bi	$v = V^+ \left(1 + \frac{K_{S_1}^d K_{S_2}}{K_{S_2}^d [S_1]} + \frac{K_{S_2}}{[S_2]} + \frac{K_{S_1}^d K_{S_2}}{[S_1][S_2]}\right)^{-1}$	$V^+$ Maximum Velocity
		$K_{S_1}^d$ Sus. 1 Dissociation Constant
		$K_{S_2}^d$ Sus. 2 Dissociation Constant
		$K_{S_2}$ Sus. 2 Michaelis Constant
		$S_1$ Substrate 1
		$S_2$ Substrate 2
Irreversible Ordered Bi-Bi	$v = V^+ \left(1 + \frac{K_{S_1}}{[S_1]} + \frac{K_{S_2}}{[S_2]} + \frac{K_{S_1}^i K_{S_2}}{[S_1][S_2]}\right)^{-1}$	$V^+$ Maximum Velocity
		$K_{S_1}$ Sus. 1 Michaelis Constant
		$K_{S_2}$ Sus. 2 Michaelis Constant
		$K_{S_1}^i$ Sus. 1 Inhibition Constant
		$S_1$ Substrate 1
		$S_2$ Substrate 2
Reversible Ordered Bi-Bi	$v = \frac{\left( V^+ \frac{[S_1][S_2]}{K_{S_1}^i K_{S_2}} - V^- \frac{[P_1][P_2]}{K_{P_1} K_{P_2}^i} \right)}{\left( 1 + \frac{[S_1]}{K_{S_1}^i} + \frac{K_{S_1}[S_2]}{K_{S_1}^i K_{S_2}} + \frac{K_{P_2}[P_1]}{K_{P_1} K_{P_2}^i} + \frac{[P_2]}{K_{P_2}^i} + \frac{[S_1][S_2]}{K_{S_1}^i K_{S_2}} + \frac{K_{P_2}[S_1][P_1]}{K_{S_1}^i K_{P_1} K_{P_2}^i} + \frac{K_{S_1}[S_2][P_2]}{K_{S_1}^i K_{S_2} K_{P_2}^i} + \frac{[P_1][P_2]}{K_{P_1} K_{P_2}^i} + \frac{[S_1][S_2][P_1]}{K_{S_1}^i K_{S_2} K_{P_1}^i} + \frac{[S_2][P_1][P_2]}{K_{S_1}^i K_{S_2} K_{P_2}^i} \right)}$	$V^+$ Maximum Direct Velocity
		$V^-$ Maximum Reverse Velocity
		$K_{S_1}$ Sus. 1 Michaelis Constant
		$K_{S_2}$ Sus. 2 Michaelis Constant
		$K_{P_1}$ Prod. 1 Michaelis Constant
		$K_{P_2}$ Prod.2 Michaelis Constant
		$K_{S_1}^i$ Sus. 1 Inhibition Constant
		$K_{P_1}^i$ Prod.1 Inhibition Constant
		$K_{P_2}^i$ Prod. 2 Inhibition Constant
		$S_1$ Substrate 1
		$S_2$ Substrate 2
		$P_1$ Product 1
		$P_2$ Product 2

## POOL OF TRIOSSES

$$\frac{d[\text{DHAP}]}{dt} = v_{F1,6BPALD} + v_{F1PALD} + v_{GDPDH} - v_{TPI} \quad (5)$$

$$\frac{d[\text{GAP}]}{dt} = v_{F1,6BPALD} + v_{TK} + v_{TPI} - J_{out} \quad (6)$$

$$\frac{d([\text{DHAP}] + [\text{GAP}])}{dt} = 2v_{F1,6BPALD} + v_{F1PALD} + v_{GDPDH} + v_{TK} - J_{out} \quad (7)$$

## CONCENTRATIONS OF THE ADENYLATE SYSTEM

$$r = \frac{[\text{ATP}]}{[\text{ADP}]} \quad (8)$$

$$[\text{ATP}] = \frac{r^2 AP_T}{K_{eq}^{AK} + r + r^2} \quad (9)$$

$$[\text{ADP}] = \frac{r AP_T}{K_{eq}^{AK} + r + r^2} \quad (10)$$

$$[\text{AMP}] = \frac{K_{eq}^{AK} AP_T}{K_{eq}^{AK} + r + r^2} \quad (11)$$

## DIFFERENTIAL EQUATIONS

$$\frac{d[\text{Fru}]}{dt} = J_{in} - v_{HK} - v_{FK} \quad (12.1)$$

$$\frac{d[\text{F6P}]}{dt} = v_{HK} - v_{PFK} \quad (12.2)$$

$$\frac{d[\text{F1,6BP}]}{dt} = v_{PFK} - v_{F1,6BPALD} \quad (12.3)$$

$$\frac{d[\text{F1P}]}{dt} = v_{FK} - v_{F1PALD} \quad (12.4)$$

$$\frac{d[\text{DHAP}]}{dt} = \frac{1}{1 + K_{eq}^{TPI}} (2v_{F1,6BPALD} + v_{F1PALD} + v_{GDPDH} + v_{TK} - J_{out}) \quad (12.5)$$

$$\frac{d[\text{GAH}]}{dt} = v_{F1PALD} - v_{ADH} - v_{TK} \quad (12.6)$$

$$\frac{d[\text{GOH}]}{dt} = v_{ADH} - v_{GK} \quad (12.7)$$

$$\frac{d[\text{GOP}]}{dt} = v_{GK} - v_{GDPDH} \quad (12.8)$$

$$\frac{d[\text{GAP}]}{dt} = \frac{K_{eq}^{TPI}}{1 + K_{eq}^{TPI}} (2v_{F1,6BPALD} + v_{F1PALD} + v_{GDPDH} + v_{TK} - J_{out}) \quad (12.9)$$

$$\frac{d[\text{NADH}]}{dt} = -\frac{d[\text{NAD}]}{dt} = v_{GDPDH} - v_{ADH} \quad (12.10)$$

**Table 4.** Parameters of hexokinase

SYMBOL	HK
$V^+$	1.0
$K_{S_1}^d$	1.5
$K_{S_2}^d$	2.0
$K_{S_2}$	0.8
$S_1$	Fru
$S_2$	ATP

**Table 5.** Parameters of aldolases

SYMBOL	F1,6BP ALD	F1PALD
$V^+$	5.0	10.0
$K_{Sus}$	0.7	0.1
$K_{eq}$	0.3	10.0
$K_{P_1}$	3.0	10.0
$K_{P_2}$	1.0	10.0
$K_{P_2}^i$	7.5	1.0
$Sus$	F1,6BP	F1P
$P_1$	DHAP	DHAP
$P_2$	GAP	GAH

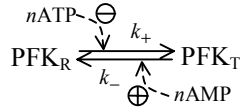
**Table 6.** Parameters of phosphofructokinase

SYMBOL	PFK
$V^+$	10.0
$K_M$	0.1
$R^a$	0.008
$n$	4.0
$Sus$	F6P
$Inh$	ATP
$Act$	AMP

**Table 7.** Parameters of triokinase

SYMBOL	TK
$V^+$	5.0
$K_{S_1}$	0.1
$K_{S_2}$	1.0
$K_{S_1}^i$	1.0
$S_1$	GAH
$S_2$	ATP

<sup>a</sup> Equilibrium constant  $R = k_+/k_-$  of



where R=Relaxed form and T=Tight form ( $k_+ = 0.2, k_- = 25$ ).

**Table 8.** Parameters of enzymes with reversible ordered bi-bi mechanism

SYMBOL	FK	ADH	GK	GOPDH
$V^+$	10.0	1.0	3.0	5.0
$V^-$	0.1	2.0	1.0	1.0
$K_{S_1}$	1.0	1.0	0.1	0.5
$K_{S_2}$	1.0	0.5	0.7	0.25
$K_{P_1}$	10.0	1.0	0.5	1.0
$K_{P_2}$	10.0	0.75	1.0	1.0
$K_{S_1}^i$	0.5	1.0	1.0	1.0
$K_{P_1}^i$	10.0	10.0	5.0	1.0
$K_{P_2}^i$	100.0	5.0	15.0	1.0
$S_1$	ATP	GAH	GOH	GOP
$S_2$	Fru	NADH	ATP	NAD
$P_1$	F1P	GOH	GOP	DHAP
$P_2$	ADP	NAD	ADP	NADH

**Table 9.** Other parameters

SYMBOL	VALUE	MEANING
$K_{eq}^{TPI}$	0.045	TPI Equilibrium Constant
$K_{eq}^{AK}$	0.45	AK Equilibrium Constant
$r$	1.0	Ratio $[ATP]/[ADP]$
$AP_T$	120.0	Total Adenylate Concentration
$NAD_T$	1.0	Total Nicotine Adenine Dinucleotide Concentration

## COMPILATION AND FILES

*PROJECT NAME:* Funder

*VERSION:* 01.01.01

### COMPILATION INSTRUCTIONS

- Decompress **funder01.01.01.tar.gz** (see below).
- In Unix systems, build the project with the provided Makefile. There are two alternative compilation modes. The default mode generates the executable file **funder**. This file calculates the susceptibilities of the example system up to order 3 and uses them to produce several estimations of responses for different inputs to the system (see below for specifications of the model). It also writes the susceptibilities to file in several formats. An alternative mode generates the executable file **funderalt**, which reads the susceptibilities of the example from the files **suscept.dat** and **errors.dat**. This alternative is recommended once the susceptibilities have been calculated, since such a process can take a couple of hours (about 1h in a PC AMD K7 at 750 MHz). Both compilation modes produce the library **funderlib.a** in an intermediate step. This should be used to compile user-defined systems. However, if your data are noisy, you may want to use the single-file source code **funderns.cpp** (see below).

```

Default compilation:      make [example]
Alternative compilation:  make alt_example
Library compilation only: make library
Clean objects with:      make clean

```

- In Non-Unix systems or if experiencing problems with *make*, the file **funder.cpp** can be used to compile **funder** and **funderalt**. This is the single-file version of the source code. It compiles without errors with GNU *g++* (versions 2.8.1 and egcs-



2.91.57) and *Visual C++ 5.0*. If your data are noisy, you probably want to use **funderns.cpp**, which implements Savitzky-Golay filters for smoothing and differentiation.

Compile funder with: `g++ funder.cpp -o funder -lm`  
Compile funderalt with: `g++ funder.cpp -o funder -lm -D FD_FILE`  
Compile funderns with: `g++ funderns.cpp -o funderns -lm`

## DESCRIPTION OF FILES

### **funder01.01.01.tar.gz**

Source code and Makefile.

Contains: Makefile, funder.h, example.h, funder.cpp, fdsusder.cpp, fdsusio.cpp, fdsusaux.cpp, fdlist.cpp, fdsignal.cpp, fdrespon.cpp, fdsystem.cpp, exsystem.cpp, exmain.cpp, funderns.cpp

Decompression: `gzip -d funder01.01.01.tar.gz`  
`tar xf funder01.01.01.tar`

### **Makefile**

To be used by make (unix systems) for compiling and linking the example executable file and the library funderlib.a. For different modes of compilation, see Compilation Instructions above.

### **funder.cpp**

Single-file version of the source code. Includes the files funder.h, example.h, fdsusder.cpp, fdsusio.cpp, fdsusaux.cpp, fdlist.cpp, fdsignal.cpp, fdrespon.cpp, fdsystem.cpp, exsystem.cpp and exmain.cpp. For compilation with g++, see above.

### **funderns.cpp**

Single-file version of the source code. It implements a Savitzky-Golay filter for computing derivatives, instead of a Ridders-Neville algorithm. Hence it is noise-resistant. NOTE: Errors are not calculated. Therefore several flags, like FD\_ERRORS and FD\_WITH\_ERRORS are ignored. The method *Susceptibility::ReadBinaryErrors* is not available in this version.

**funder.h**

Header file. Constant definitions and class prototypes of Susceptibility, IntegList, IntegListNode, System, Signal and Response. It also declares the structures ILN (linked-list node description), DER (derivative parameters), FUNDER (functional derivative parameters), APPDER (approximate functional derivative parameters).

**example.h**

Header file. Constant definitions and class prototype of UserSystem, the user-defined system that describes the example model.

**fdsusder.cpp**

C++ Source file. Susceptibility class: Methods for derivative calculations.

Methods: Susceptibility constructors and destructors, FunctionalDerivative, IterTimes, TensorIndexes, Derivative, CalculateDiff, ApproxDer, IterateDer.

**fdsusio.cpp**

C++ Source file. Susceptibility class: Methods for file input/output.

Methods: Susceptibility constructor from file, WriteToBinaryFile, ReadBinaryErrors, RawToTextFile, FirstToTextFile, SecondToTextFile, ThirdToTextFile.

**fdsusaux.cpp**

C++ Source file. Susceptibility class: Methods for auxiliary functions.

Methods: AllocateSusceptibilities, FreeSusceptibilities, CombinatorialFactor, Combinations, PermutaRepe, Facts, MaxTime, Displacement, TestSums, Sum, GetOrder, GetSize, GetTimeInc, operator\*.

**fdlist.cpp**

C++ Source file. IntegListNode and IntegList classes: Methods for the linked list and its nodes.

Methods: IntegListNode: Construction and destruction, SearchLevel, Insert.  
IntegList: Construction and destruction, GetData, Exists.

**fdsignal.cpp**

C++ Source file. Signal class: Methods for signal manipulation

Methods: Construction and destruction, operator+, operator-, operator\*, operator/ (for signals and constants), operator=, Derivative, Integral, Primitive, WriteToTextFile, ReadFromTextFile, GetSize, GetTimeInc, GetPtrToData

### **fdrespon.cpp**

C++ Source file. Response class: Response of all orders of a system.

Methods: Construction and destruction, operator=, operator[], WriteToTextFile, GetOrder, GetSize, GetTimeInc.

### **fdsystem.cpp**

C++ Source file. Abstract System class: Basic functional definition and operation.

Methods: Construction and destruction, Functional, IntegrateOneStep, InitSS.

### **exsystem.cpp**

C++ Source file. Example of user-defined system class derived from System.

Methods: Construction from System and destruction, EvaluateSystem (virtual function of System; must be provided), VelRevOrdUniBi, VelIrrRanBiBi, VelRevOrdBiBi, VelIrrOrdBiBi, VelPFK, IntegrateInputVariation.

### **exmain.cpp**

C++ Source file. Main function.

### **funder**

Executable file generated by `make`. It calculates the susceptibilities of the example system up to third order and uses them to illustrate the functional-Taylor-series approximation to the response by convoluting several signals (input flux variations) with the susceptibilities. The exact integration of the equations is also calculated.

Generated files: first.dat, second.dat, third1.dat, raw.dat, sums.dat, suscept.dat, errors.dat, in05min.dat, out05min.dat, con05min.dat, in2plus.dat, out2plus.dat, con2plus.dat, in14940.dat, out14940.dat, con14940.dat, in19930.dat, out19930.dat, con19930.dat, in24920.dat, out24920.dat, con24920.dat.

### **funderalt**

Executable file generated by `make alt_example`. It reads the susceptibilities of the example system and their errors from **suscept.dat** and **errors.dat**, respectively, and uses them to illustrate the functional-Taylor-series approximation to the response by convoluting several signals (input flux variations) with the susceptibilities. The exact integration of the equations is also calculated. This executable should be use if the files **suscept.dat** and **errors.dat** are available, since the calculation of the susceptibilities can take several hours.

Generated files: in05min.dat, out05min.dat, con05min.dat, in2plus.dat, out2plus.dat, con2plus.dat, in14940.dat, out14940.dat,

con14940.dat, in19930.dat, out19930.dat, con19930.dat,  
in24920.dat, out24920.dat, con24920.dat.

### **funderns**

Executable file generated from **funderns.cpp**. It calculates the susceptibilities of the example system up to third order by means of a Savitzky-Golay algorithm and uses them to illustrate the functional-Taylor-series approximation to the response by convoluting several signals (input flux variations) with the susceptibilities. The exact integration of the equations is also calculated.

Generated files: first.nsd, second.nsd, third1.nsd, raw.nsd, sums.nsd, suscept.nsd, in05min.nsd, out05min.nsd, con05min.nsd, in2plus.nsd, out2plus.nsd, con2plus.nsd, in14940.nsd, out14940.nsd, con14940.nsd, in19930.nsd, out19930.nsd, con19930.nsd, in24920.nsd, out24920.nsd, con24920.nsd.

### **funderlib.a**

Library containing the object files of the classes Susceptibility, IntegList, IntegListNode, Signal, Response and System. Users should compile their own source files with this library as follows:

```
g++ <user.cpp files> funderlib.a -o <user_exe_file> -lm
```

Object files: fdsusder.o, fdsusio.o, fdsusaux.o, fdlist.o, fdsignal.o, fdrespon.o, fdsystem.o

### **suscept.dat**

File of susceptibilities of the example system, in binary format. Generated by **funder**.

### **errors.dat**

File of errors of the susceptibilities of the example system, in binary format. Generated by **funder**.

### **first.dat**

First-order susceptibility of the example system, in text format. The space-separated columns of the file are:

*real\_time susceptibility\_value [error\_value o/\*]*

The last two columns are written only if the method *Susceptibility :: FirstToTextFile* is called with the flag `FD_WITH_ERRORS`. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. Generated by **funder**.

### second.dat

Second-order susceptibility of the example system, in text format. The space-separated columns of the file are:

*real\_time second\_perturbation\_time susceptibility\_value [error\_value o/\*]*

The last two columns are written only if the method *Susceptibility :: SecondToTextFile* is called with the flag `FD_WITH_ERRORS`. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. If the flag `FD_GNUPLOT` is on, method *Susceptibility :: SecondToTextFile* writes a carriage return after every real time block. This is required for plotting in the parametric mode of the GNU plot utility gnuplot. Generated by **funder**.

### third1.dat

Section of the third-order susceptibility of the example system for the second perturbation time fixed at initial time, in text format. The space-separated columns of the file are:

*[second\_perturbation\_time] real\_time third\_perturbation\_time susceptibility\_value [error\_value o/\*]*

The fixed second perturbation time is written (in the first column) only if the method *Susceptibility :: ThirdToTextFile* is called with the flag `FD_SECOND_TIME`. The last two columns are written only if the flag `FD_WITH_ERRORS` is on. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. If the flag `FD_GNUPLOT` is on, method *Susceptibility :: ThirdToTextFile* writes a carriage return after every real time block. This is required for plotting in the parametric mode of the GNU plot utility gnuplot. Columns 2:3:4 or 1:2:3 should be used if the flag `FD_SECOND_TIME` was on or off, respectively. Generated by **funder**.

### raw.dat

Raw data, that is, the susceptibilities as they are stored in the memory of the computer, but in text format. This file should be used primarily to check the correctness of the calculations. If the method *Susceptibility :: RawToTextFile* is called with the flag `FD_WITH_ERRORS`, two additional columns are written with the errors and the accuracy, 'o' (accurate) or '\*' (inaccurate):

*susceptibility\_order susceptibility\_value [error\_value o/\*]*

Generated by **funder**.

### **sums.dat**

The integrals of the susceptibilities of the example system. Generated with the method *Susceptibility :: TestSums*. Generated by **funder**.

### **in05min.dat/out05min.dat/con05min.dat**

Input (in), exact response (out) and approximate response by convolution with the susceptibilities (con) of the example system with input flux variation  $\Delta J_{in} = -0.5$ . The first column of all files is real time and the second one is the value of the signal. In the case of the 'con' file, the second column is the sum of responses of all orders, which should be compared to 'out'. The following columns are one-order responses, starting with the first-order one. Generated by **funder** and **funderalt**.

### **in2plus.dat/out2plus.dat/con2plus.dat**

Input (in), exact response (out) and approximate response by convolution with the susceptibilities (con) of the example system with input flux variation  $\Delta J_{in} = 2$ . The first column of all files is real time and the second one is the value of the signal. In the case of the 'con' file, the second column is the sum of responses of all orders, which should be compared to 'out'. The following columns are one-order responses, starting with the first-order one. Generated by **funder** and **funderalt**.

### **in14940.dat/out14940.dat/con14940.dat**

Input (in), exact response (out) and approximate response by convolution with the susceptibilities (con) of the example system with input flux variation  $\Delta J_{in} = 1 + 0.49 \cos(40\omega)$ . The first column of all files is real time and the second one is the value of the signal. In the case of the 'con' file, the second column is the sum of responses of all orders, which should be compared to 'out'. The following columns are one-order responses, starting with the first-order one. Generated by **funder** and **funderalt**.

### **in19930.dat/out19930.dat/con19930.dat**

Input (in), exact response (out) and approximate response by convolution with the susceptibilities (con) of the example system with input flux variation  $\Delta J_{in} = 1 + 0.99 \cos(30\omega)$ . The first column of all files is real time and the second one is the value of the signal. In the case of the 'con' file, the second column is the sum of responses of all orders, which should be compared to 'out'. The following columns are one-order responses, starting with the first-order one. Generated by **funder** and **funderalt**.

### **in24920.dat/out24920.dat/con24920.dat**

Input (in), exact response (out) and approximate response by convolution with the susceptibilities (con) of the example system with input flux variation  $\Delta J_{in} = 2 + 0.49 \cos(20\omega)$ . The first column of all files is real time and the second one is the value of the signal. In the case of the 'con' file, the second column is the sum of responses of all orders, which should be compared to 'out'. The following columns are one-order responses, starting with the first-order one. Generated by **funder** and **funderalt**.

### **\*.nsd**

The same as the files with .dat extension, except that noisy data are used (additive white Gaussian noise, standard deviation equal to 5% the value of the reference state). Generated by **funderns**.

## **PUBLIC METHODS OF *Susceptibility***

`Susceptibility();`

Default constructor. It is not allowed and will terminate the program.

`Susceptibility(int order, int size, double scale, System *name, char flags);`

#### **Parameters**

<i>order</i>	Order of the approximation	
<i>size</i>	Number of discrete-time points	
<i>scale</i>	Time increment between discrete-time points	
<i>name</i>	Name of the system to be used for derivative calculations	
<i>flags</i>	FD_VERBOSE	Give extra information while calculating
	FD_NO_VERBOSE	Do not give extra information

Allocates memory for *order* susceptibilities and their errors and calculates functional derivatives of the system response. The response must be defined internally in the system. The FD\_VERBOSE flag causes the object to display information while calculating the susceptibilities, including the order of the susceptibility being calculated, the number of perturbations used at any given time, the perturbation times and any inaccuracies of the calculation. An asterisk indicates the latter (\*), which is followed by the derivative orders, the time point at which the error occurred and the accuracy achieved.

`Susceptibility(char *name);`

#### **Parameters**

<i>name</i>	Name of the binary file to be read
-------------	------------------------------------

Allocates memory for the number of susceptibilities specified in the binary file *name* and reads the data from the file. It also allocates memory for the errors, but

does not read them. Susceptibilities need not be calculated if this constructor is used. If the file is not found, cannot be opened or its format is incorrect, an error is prompted and the program is terminated.

`~Susceptibility();`

Deallocates the susceptibilities and their errors. It need not be called explicitly.

`void WriteToBinaryFile(char *name, char flag);`

**Parameters**

<i>name</i>	Name of the binary file to be written	
<i>flag</i>	FD_SUSCEPTIBILITY	Write susceptibilities
	FD_ERRORS	Write errors

Writes a binary file with the susceptibilities of all orders. It also stores information on their size and scale, and on the order of the approximation. Only one flag is accepted. If ambiguous, the errors will be written.

`void ReadBinaryErrors(char *name);`

**Parameters**

<i>name</i>	Name of the binary file to be read
-------------	------------------------------------

Reads a binary file and sets the errors of the susceptibilities. The latter must have been read from another file by using `Susceptibility(char *name)`. If the file is not found, or if its format is not correct, an error is prompted and the program terminates.

`void RawToTextFile(char *name, char flag);`

**Parameters**

<i>name</i>	Name of the text file to be written	
<i>flag</i>	FD_WITH_ERRORS	Include the errors in the text file
	FD_WITHOUT_ERRORS	Do not include the errors

Writes the susceptibilities to a text file exactly in the same order as they are stored in the computer memory. The structure of the file is:

*susceptibility\_order susceptibility\_value [error\_value o/\*]*

This file should be used primarily to check the correctness of the calculations. If this method is called with the flag `FD_WITH_ERRORS`, the last two columns are written, with the errors and the accuracy, 'o' (accurate) or '\*' (inaccurate). If the file cannot be created, the program terminates.



```
void FirstToTextFile(char *name, char flag);
```

#### Parameters

<i>name</i>	Name of the text file to be written	
<i>flag</i>	FD_WITH_ERRORS	Include the errors in the text file
	FD_WITHOUT_ERRORS	Do not include the errors

Writes the first susceptibility to a text file:

*real\_time susceptibility\_value [error\_value o/\*]*

The last two columns are written only if the method is called with the flag FD\_WITH\_ERRORS. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. If the file cannot be created, the program terminates.

```
void SecondToTextFile(char *name, char flags);
```

#### Parameters

<i>name</i>	Name of the text file to be written	
<i>flags</i>	FD_WITH_ERRORS	Include the errors in the text file
	FD_WITHOUT_ERRORS	Do not include the errors
	FD_GNUPLOT	Use gnuplot format

Writes the second susceptibility (if any) to a text file:

*real\_time second\_perturbation\_time susceptibility\_value [error\_value o/\*]*

The last two columns are written only if the method is called with the flag FD\_WITH\_ERRORS. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. If the flag FD\_GNUPLOT is on, a carriage return is written between real time blocks. This is required for plotting with the GNU application gnuplot. The plotting mode must be parametric. For example:

```
set parametric
splot 'file.dat' u 1:2:3
```

If the file cannot be created or there is no second susceptibility, the program terminates.

```
void ThirdToTextFile(char *name, int 2nd_time, char flags);
```

#### Parameters

<i>name</i>	Name of the text file to be written	
<i>2nd_time</i>	Fixed time of the second perturbation	
<i>flags</i>	FD_WITH_ERRORS	Include the errors in the text file
	FD_WITHOUT_ERRORS	Do not include the errors
	FD_GNUPLOT	Use gnuplot format
	FD_SECOND_TIME	Write the 2 <sup>nd</sup> perturbation time

Writes a section of the third susceptibility (if any) to a text file:

```
[second_perturbation_time] real_time third_perturbation_time susceptibility_value [error_value o/*]
```

The first column is the fixed second perturbation time only if the flag `FD_SECOND_TIME` is on. The last two columns are written only if the method is called with the flag `FD_WITH_ERRORS`. The last column is 'o', if the specified accuracy was achieved, or '\*', if not. If the flag `FD_GNUPLOT` is on, a carriage return is written between real time blocks. This is required for plotting in parametric mode with the GNU application `gnuplot`. For example:

```
set parametric
splot 'file.dat' u 1:2:3 or
splot 'file.dat' u 2:3:4 (if FD_SECOND_TIME is on)
```

If the file cannot be created or there is no third susceptibility, the program terminates.

```
void TestSums(char *name);
```

#### Parameters

*name*            A file name to write the areas under the susceptibilities

Calculates the areas under the susceptibilities and writes them to `stdout` and to the specified text file.

```
Response operator*(Signal &);
```

Calculates multiple convolutions of a signal (the input flux variation) with the susceptibilities of a system. The result is returned as a `Response` object, which contains all the orders of the response and their total sum. This sum is the approximation to the (variation of the) response of the system. The operation itself is commutative, but the method is written so that the signal must be left-multiplied. This method overloads '\*'. It needs not be called explicitly. For example:

```
int tau=50;
double dt=0.1;

Signal sig(tau, dt);           // Create a step function
                                // of 50 points and time
                                // increment of 0.1
Susceptibility sus("suscept.dat"); // Read the susceptibilities
                                // of a system from a file
Response res;                  // Create a response object

// NOTE: sig*sus is not defined
res = sus*sig;                 // Calculate the response
```

```
// The method
// Susceptibility::operator*
// is implicitly called
```

If the signal and the susceptibilities are not in the same scale or the number of discrete-time points is different, this method prompts an error and the program terminates.

```
int GetOrder();
```

Retrieves the order of the approximation, i.e. the number of susceptibilities in the object.

```
int GetSize();
```

Retrieves the number of discrete-time points of the approximation.

```
double GetTimeInc();
```

Retrieves the time increment used for discretizing the signals.

## **PUBLIC METHODS OF *Signal***

```
Signal();
```

Creates a step function of 100 discrete-time points and time increment equal to 0.1

```
Signal(int size, double scale);
```

**Parameters**

*size*            Number of discrete-time points

*scale*          Time increment between discrete-time points

Creates a step function with the specified parameters.

```
Signal(double *data, int size, double scale);
```

**Parameters**

*data*           Pointer to a vector containing the discrete time course of the signal

*size*           Number of discrete-time points

*scale*          Time increment between discrete-time points

Creates a signal with the specified parameters and from a vector of data.

```
Signal(const Signal &);
```

Copy constructor. Implicitly called when initializing in declarations.

```
~Signal();
```

Destruction of the signal deallocates the memory reserved for the data.

`Signal operator+(const Signal &);`

Addition of signals means addition point by point. If the signals are not in the same scale or the number of discrete-time points is different, this method prompts an error and the program terminates.

`Signal operator-(const Signal &);`

Subtraction of signals means subtraction point by point. If the signals are not in the same scale or the number of discrete-time points is different, this method prompts an error and the program terminates.

`Signal operator*(const Signal &);`

Product of signals means convolution of signals. This is a commutative operation. If the signals are not in the same scale or the number of discrete-time points is different, this method prompts an error and the program terminates.

`Signal operator/(const Signal &);`

Division of signals means deconvolution. The first point of the second signal must be different from zero. Otherwise, the program terminates. If the signals are not in the same scale or the number of discrete-time points is different, this method prompts an error and the program terminates.

`Signal operator+(const double);`

Addition of a constant means addition of the constant to every point.

`Signal operator-(const double);`

Subtraction of a constant means subtraction of the constant from every point.

`Signal operator*(const double);`

Product by a constant means product of every point by the constant.

`Signal operator/(const double);`

Division by a constant means division of every point by the constant. If the constant is zero, the program terminates.

`Signal & operator=(const Signal &);`

Assignment of signals.

`Signal Derivative();`

Derivative of the signal, assuming that the value of the signal at the origin is zero. It returns the derivative obtained after the operation.

`double Integral();`

Returns the area under the signal.

`Signal Primitive();`

Returns the primitive of the signal that crosses the origin.

`void WriteToTextFile(const char *name);`

Parameters

*name*      A text file name to write the signal

Writes the signal to a text file, as follows:

*real\_time value\_of\_the\_signal*

If the file cannot be created, the program terminates.

`void ReadFromTextFile(const char *name);`

Parameters

*name*      A text file from which the signal has to be read

Reads a signal from a text file of the structure created by `Signal :: WriteToTextFile`. If the file cannot be created or if it is too short, the program terminates.

`int GetSize();`

Retrieves the number of discrete-time points of the signal.

`double GetTimeInc();`

Retrieves the time increment between two consecutive points of the signal.

`double *GetPtrToData();`

Retrieves a pointer to the data.

## **PUBLIC METHODS OF *Response***

`Response();`

Creates an order-one response of 100 points and time increment equal to 1.

`Response(double **data, int order, int size, double scale);`

Parameters

*data*      Pointer to a matrix containing the time course of the responses

*order*      Order of the response

*size*      Number of discrete-time points

*scale*      Time increment between discrete-time points

Allocates responses up to *order*, plus another vector for the sum of the responses of all orders and copies the data from the input matrix.

```
Response(const Response &);
```

Copy constructor. Implicitly called when initializing in declarations.

```
~Response();
```

Destruction of the object deallocates the memory reserved for the data.

```
Response & operator=(const Response &);
```

Assignment of responses.

```
Signal operator[](const int index);
```

Parameters

*index*      Order of the response to be returned. Zero is the sum of all orders

Retrieves the response of a given order, as indicated by the index. For example:

```
Signal sig;           // Creates a Signal object
Response res;         // Creates a default response

sig=res[0];           // Sum of the responses of all orders
sig=res[1];           // Response of first order
```

If the index is negative or exceeds the order, the program terminates.

```
void WriteToTextFile(const char *name);
```

Parameters

*name*      A text file name to write the response

Writes the response of all orders to a text file of the form:

*real\_time sum\_of\_responses first\_order\_response [...]*

If the file cannot be created, the program terminates.

```
int GetOrder();
```

Retrieves the order of the response, i.e. the number of responses of different orders that form the object.

```
int GetSize();
```

Retrieves the number of discrete-time points of the responses.

```
double GetTimeInc();
```

Retrieves the time increment used for discretizing the signals.

## PUBLIC METHODS OF *System*

*System* is an abstract class. Therefore, it cannot be instantiated. Users must derive a class from this one providing at least the method `void EvaluateSystem()`. See below for a description of the function of this method and for a list of the protected data members that can be exploited for its development.

```
System();
```

Default construction is not allowed and terminates the program.

```
System(double step, int species, int rates, int input_species, int output_rate);
```

### Parameters

<i>step</i>	Time increment used for integration
<i>species</i>	Number of species or concentrations
<i>rates</i>	Number of velocities, including output fluxes
<i>input_species</i>	Number of the species that receives the input flux
<i>output_rate</i>	Number of the rate to be considered the response of the system

Allocates memory for the concentrations, their time derivatives, the rates, the input fluxes (in principle, as many as concentrations) and the steady state concentrations. The integration *step* must be shorter than the time increment used in *Susceptibility*, *Signal* and *Response*. Otherwise, the first time the *Functional* is used the program terminates. The details of the system must be provided by the user in a class derived from this one.

```
virtual ~System();
```

Deallocates the memory of all the concentrations, rates, fluxes, et cetera. The user-derived system need not worry about this.

```
void Functional (const DER &derivative_parameters);
```

### Parameters

<i>derivative_parameters</i>	A structure of type DER. Its members are:	
	<i>int tau</i>	Number of discrete-time points
	<i>int pert</i>	Number of perturbations
	<i>int *perttimes</i>	Vector of perturbation times
	<i>double dt</i>	Time increment (func. derivatives)
	<i>double *x</i>	Perturbation magnitudes
	<i>double *integ</i>	Integration result to be returned

Integrates the differential equations provided in a user-derived system. The time increment *dt* must be equal or longer than the increment used for integration. The functional is evaluated for *pert* instantaneous variations of the concentration at *perttimes* times. The values of the variations are given by *x*, which is the point where an evaluation is needed for calculating an approximate derivative. The

result is returned in the vector `integ`. This method uses `EvaluateSystem` and `IntegrateOneStep`. A simple Euler algorithm is provided for the latter, but the former is expected to be completely defined by the user.

## PROTECTED METHODS OF *System*

```
void IntegrateOneStep();
```

`System` provides this elementary integration engine that uses Euler's algorithm. It uses the values of the protected data members `numc`, `c`, `dc` and `dtint` (see below). The user could override this method, which is needed by `Functional` and `InitSS`.

```
void InitSS();
```

This method calculates and stores in `ss` (see protected data members below) the steady state concentrations of the system. It also sets the input and output reference fluxes, `inR` and `outR`. The input fluxes and other parameters must be specified in the user-defined constructor of the derived class. This method should *always* be called by the user's constructor. The steady state is defined as the set of concentrations that produce concentration time derivatives below the constant `CONV` for all species. This constant is set to `1e-6` in **funder.h**. The user can override it by defining it before including `funder.h` (or at the beginning of **funder.cpp**, if the single-file version of the source code is used).

```
virtual void EvaluateSystem()=0;
```

This is an abstract method, for which no definition is given in *System*. Hence, the class *System* is an abstract one and cannot be instantiated (`=0` prevents the user from doing so). The user must provide this method in any derived class. It is supposed to calculate the velocities,  $v$ , and the derivatives of the concentrations,  $dc$ , from the current values of the concentrations,  $c$  (see below for protected data members). The differential equations of the system are likely to be implemented in this method.

## PROTECTED DATA MEMBERS OF *System*

The class *System* contains the following protected data members, which can be accessed from any derived class:

<code>double t</code>	Real time. For defining time-dependent properties
<code>double dtint</code>	Integration time increment or step
<code>double *c</code>	Vector of concentrations
<code>double *dc</code>	Vectors of derivatives of the concentrations



<code>double *v</code>	Vector of velocities, including output fluxes
<code>double *f</code>	Vector of input fluxes
<code>double *ss</code>	Vector of steady state concentrations
<code>double inR</code>	Reference input flux
<code>double outR</code>	Reference output flux
<code>int numc</code>	Number of concentrations (and possibly input fluxes)
<code>int numv</code>	Number of velocities, including output fluxes
<code>int pertc</code>	Index of the species to be perturbed (input species)
<code>int resv</code>	Index of the rate to be considered the response

## CREATING A USER-DEFINED SYSTEM BY DERIVING A CLASS FROM THE ABSTRACT CLASS *System*

Deriving a class from the abstract class *System* involves three mandatory steps, which are described below.

1. Declare a new class from *System*. Grant public inheritance access.

```
class UserSystem : public System
{
    ...
};
```

2. Write a constructor for the derived class that constructs an object of the class *System* by means of an initialization list. Make sure that the constructor calls `InitSS` at the end of the body. The constructor may be used to initialize the parameters of the system.

```
UserSystem :: UserSystem() :
System(step, species, rates, input_species, output_rate)
{
    ...
    InitSS();
}
```

3. Provide the method `EvaluateSystem`. This method should evaluate the velocities and the derivatives of the concentrations (the differential equations of the system).

```
void UserSystem :: EvaluateSystem(void)
{
    // Evaluation of the velocities
    v[0] = ...;
    ...
    v[numv-1] = ...;
```

```
// Differential equations
dc[0]      = ...;
...
dc[numc-1] = ...;
}
```

In addition, the user may want to define additional data types and methods that support the constructor and `EvaluateSystem`, and that provide additional functionality. For example, methods may be defined that evaluate a rate equation from a set of parameters and concentrations.