Scicos extension for matrix data type

Alan Layec

***Abstract*— This paper presents recent advances concerning the handling of matrix typed data in the current Scilab/Scicos environment (Scilab 4.1.2). New functionnalities of editor, compiler, simulator, code generation and also some Scicos blocks are detailled. Examples of a Kalman filter(matrix operations) and a quasic-chaotic digital recursive filter(integer handling) are given to provide an overview of possibilities of Scicos concerning that extension.**

I. Introduction

Scicos is a toolbox included in Scientific software package Scilab. It is dedicated to the modeling and the simulation of hybrid dynamical systems, where components and subsystems can be both described with discrete and continuous time. Developed since fifteen years, that modeler/simulator is now composed of a consequent number of functionnalities which have been progressively grafted during its evolution. To only quote the most used of them, we can cite the possibilites of :

- building systems with a block-diagram graphical editor,
- use of the modelica modeling language,
- simulation of explicit/implicit continuous-time systems by the use of ODE/DAE solvers,
- zero crossing detection,
- use of discrete events to modeling multirate system,
- code generation.

Since its original version, Scicos was given with no particular tools (i.e. blocks) for matrix operations and natively doesn't handle integer numbers.

Matrix operations are intensively used in modeling and simulation of dynamical systems. In a time domain simulator, the most popular use of matrix operations is probably the multiplication encountered in the computation of CLSS (Continuous Linear State-space System) where parameters are defined with A,B,C,D matrices. But they also find many others applications such the Kalman filter where the covariance matrix of prediction error must be integrated in time domain. Integer numbers are not really used in general systems of nonlinear differential equation (for i.e. in mecanical, electrical or thermohydraulic systems), because continuous-time variables are almost always considered as real numbers evolving during time. But they are in fact essential in systems evolving in finite state machines. Indeed digital subsystems such digital signal processor, ASIC (Application Specific Integrated Circuit) and other FPGA (Field Programmables Gates Array) chips are today very used in embedded control systems and also in the wide field of radar and communication systems through SiP (System in Package) and SoC (System on Chip). Although that digital subsystems are most of them deterministic systems, they present hardware nonlinearity because of the nature of the quantified discrete-time variables and parameters. That nonlinearities are sources of noise and sources of instabilities. For example, we can cite the case school of the nonlinearity of an Analog to Digital Converter which reports quantification noise to its output or also the well know modulo behavior of adders or multipliers encountered in Arithmetic Logic Unit that can involve quasi-chaotic oscillation in discrete recursive systems.

To provide to the users of Scicos the possibilities to design realistic models which take into account digital nature of signals used in digital chips and to perform powerful simulations with matrix operations, we have strongly improve all parts of Scicos. This have been done to allow the handling of typed matrix regular input/output port, states and parameters. The section II gives a hierarchical overview of Scicos by focussing on what was the limitations before introduction of matrix data type. Section III discuss memory management and signals classification allowed by matrix data type. Then the section IV reports and details new functionnalities provided by the current version of Scicos. Section V deals with examples of application through the linear continuous-time Kalman filter and an example of a quasi-chaotic IIR (Infinite Impulse Response) filter designed in a DSP with use of actual code generation. Finally section VI presents some current and future work concerning matrix data type and code generation and concludes that paper.

II. Scicos overview

A. Hierarchical view of Scicos

In a hierarchical and simple point of view, Scicos is composed of two parts as shown in the figure 1.

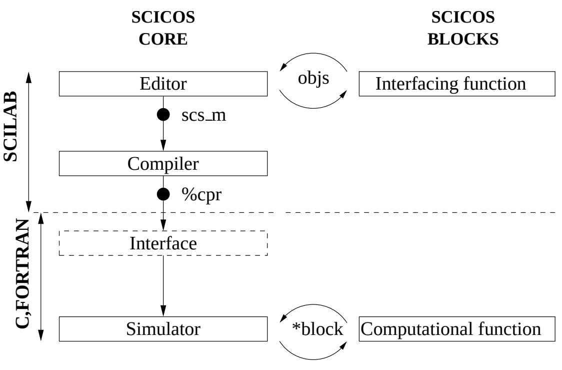

Fig. 1. Hierarchical view of Scicos

The left part, called here ”Scicos core”, is the part that encloses the main functionnalities of Scicos that allows the modeling and the simulation and the second named ”Scicos blocks" can be understand as the user part of Scicos and will be explored in details in the next two subsections.

Scicos core is composed of three main levels :

- an editor which allows to do one diagram in a graphical environnement that is composed of entities called blocks. That blocks are together connected by links with different colors and interact with the editor by the interfacing functions.
- a compiler : in a first pass it realizes a flat model of the diagram and in a second time it computes the scheduling tables of models to be simulated, extracts all informations contained in the diagram that must be used for the simulation and also realizes others operations such the automatic adjustment of the regular ports size.
- a simulator which use the informations provided by the compiler to realize the scheduling call of computational functions of models and ensures different tasks such the computation of discrete and contiuous states, the zero crossing detection, update of output date...

The graphical editor handles one Scilab list called "scs_m" that encloses informations concerning the different objects that are present in the diagram (blocks, links and text). For blocks, this informations are various and stored in sublist of scs_m called scs_m.objs. That mainly concerns graphical informations and model properties. Graphical informations are used to draw the blocks, to know and update the position of blocks, and model properties are to inform the number and the sizes of regular input/output port, the values of parameters, the initial conditions of states and etc... Another main Scilab list intensively used in the core of Scicos is the %cpr list that encloses informations issued from the compiler. That informations are the only necessary ones needed for the simulation. Moreover it exchanges in input/output a part of its content with the simulator notally concerning discrete and continuous states and also the output register of blocks. Note that the simulator which is written in C language only work with low level arrays (arrays issued from C or fortran functions). So the simulator can't directly work with the Scilab list %cpr which is an object of higher level. To do the traduction of %cpr it is then needed to call a Scilab interfacing function (called "scicosim" for the case of Scicos simulator) that will do the extraction from the %cpr elements to provide understandable working array to the simulator. That interfacing function is used any time the user runs a simulation.

B. Overview of a Scicos Basic Block

1) Block viewed in the graphical editor: In the graphical editor, blocks are handled by the core of Scicos via one specific function written in Scilab language named interfacing function. This one will ensures the coherence and the update of the data contained in the sublist scs_m.objs. This is done interactively with the user for example when he moves the block or when he opens the dialog box to edit parameters. Let us recall that the blocks are together connected with links of differents natures : red links are to model discrete activation dates (events) and black links are to drive data flow between blocks. The size of regular ports is parametrized via the interfacing function by the field model.in and model.out. In the original versions of Scicos for each input/output ports, only one dimension was used to set the size of the port as shown in the figure 2.

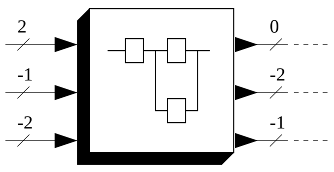

Fig. 2. Size of regular ports viewed in the editor

That size could also be negative or equal to zero. In fact that means that in the editor, the user can define sizes of port which are undetermined for the negative case or sizes which are equal to the sum of all other sizes for the case equal to zero. In Scicos when negative or zero sizes are used, this is the compiler which will adjust the sizes by the use of fixed-point algorithm : all links are scanned the ones after the others and that several times while each sizes becomes positive. In a first pass, when a negative size is met in input, the compiler look the size of the source port linked to the target port and automatically affect that size to size of the target port if and only if it is positive. In a second pass, when a regular ouput port size is negative then the compiler look which size in input have the same negative value and then affects the size that it have found in the first pass. If all sizes with negatives values are solved then the compiler treats the zero sizes and affects this ones by summing all positive size in an unilateral manner in input or in output (where the zero size is encountered). If the algorithm doesn't solve that problem, then the compiler asks to the user to explicitly gives the sizes that it can't found.

2) Block viewed by the simulator: When the size are fully informed by positive values, the compiler build the output state registers in the sublist %cpr.state.outtb. All output states are initially set to zero. When the Scilab interfacing function (scicosim) is called, it exchanges and does the traduction of the data contained in that output registers for the simulator. During the simulation when blocks are called, the simulator provides a pointer to that data in arguments of the computational function by way of a C structure called "*block". That structure is composed of various fields and notally with fields of integers/reals parameters values and size, discrete/continuous register state values and sizes, the number of event input,... and of course the size (field insz[]/outsz[]), the number (nin/nout) and the values (inptr[][]/outptr[][]) of regular ports. A block viewed by the simulator (computational function) can be summarized by the figure 3.

In the computational function, the regular ports were detailled as arrays of double (real number) with only one dimensions. We can note that the regular input port doesn't really exists. They must be understand as arrays pointing on output registers of preceding blocks. These arrays can also

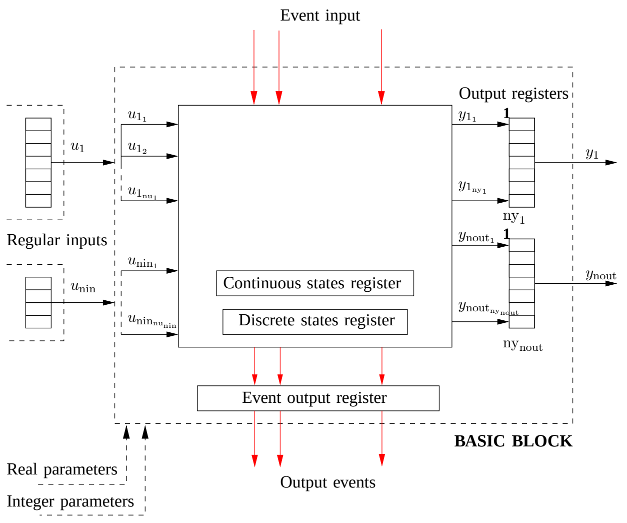

Fig. 3. Basic block viewed by the simulator

be viewed as vector because of the only one dimension. Each input/output have a first index that informs the port number and a subindex for the element number in the vector. As each element is a double, each vector take $n \times 8$ bytes in memory, with n the size (or the dimension) of the considered port.

III. Signals modeling problematics

That implementation of regular input/output ports allowed in fact various modelings of signals encountered in many fields of applications. For example in control, we typically use formalism of vector of continuous-time state variables to do time domain integration of nonlinear equation systems and in digital signal processing, systems are described with discrete linear equation systems. For both examples, vectors of real number are able to store the data used in that systems. Moreover we can understand that a matrix of complex numbers of size $[n,m]$ can be concatenated in a single vector of real number of size nxmx2, where first indices going from 1 to mxn can be used to store real part and indexes from mxn+1 for imaginary part and a vector of integer numbers can be stored in a vector of real numbers.

But this implicit encapsulation really increases the modeling work when functions of systems intensively use matrix operation and when the systems are mixed-signals systems. Indeed in a mixed-signal system, digital subsystems can use quantified discrete signals (evolving in $\mathbb{N}$) and nonlinear handle continuous-time variable which are real or complex signals (evolving in $\mathbb{R}$ or in $\mathbb{C}$). As Scicos only saw vector of double, the users didn't have feedback of informations in the level of editor concerning type of data driven by regular port and were obliged to write somme additional lines of code in computational functions, mainly composed of cast operator and indices computation, to correctly handled the data driven between blocks.

A. Advantages of typed data

Data type for regular input/ouput ports brings many befenits for the modeling and simulation of hybrid dynamical systems.

First the typing offers an optimal memory management. Indeed it is always constraining to use oversized blocks of memory because memory is primary ressource of computation units. This is notally the case of use of real numbers (double) to represent integer number (int) because we use eight times more memory for a simulation than it really needs. Moreover in some monte carlo simulation of digital communication systems, large vector of data composed of thousands elements are used. For example if we consider one vector of 8-bit coded thousand values, we need 8 Kb in the case where port handle double typed data and only 1 Kb if it use good data type (char).

In a second point of view, the typing allows easiest ways to interface computational libraries[BLAS/LAPACK] in its own models. Indeed the previous cited *block structure matches very well to call by reference external functions concerning computation of states and output registers. In preceding version of Scicos, before calling a function which used integer array in arguments, we were obliged to convert double arrays in integer and even to sometimed do the back convertion from integer to double array after the call of external function. This reveals an overcost of work for people who realize the computational function. If parameters, states and input/output registers directly pointing on good data type then the work of writing code is more easy because arrays naturaly take their place in argument of C or Fortran external function.

Finally if states, parameters and input/output registers are detailled in integer arrays, implementation and code generation for embedded systems with computational units working with integer arithmetic can be now fully considered as a functionnality of Scicos.

B. Advantages of a second dimension

Use of a second dimension also improve the work of modeling and simulation.

First it does an explicit distinction between vectors and matrices. Because of the lack of this dimension, Scicos blocks was only detailled for vector-based operations and most of matrix-based operations that we find in Scilab wasn't avialable.

Secondly, the explicit distinction between column-vector, row-vector and matrix allows one essential aspect for the modeling of mixed-signal system, the possibility to do a classification of signals issued from different sort of subsystems.

- In the control domain and more especially in Scicos, the signals are used with a particular formalism. It combines for each sample one value but also one date when the sample is produced. That signals are called "time-based signals". The concept of vector is then understand as state vector of a nonlinear equation system explicitly dependant of time. A vector of size m correspond then to m state variables.
- In the digital processing domain, signals are frequently issued from subsystems where they are sampled with constant step. The value of the date is not take into

account and a simple index is used to locate the sample in a normalized time scale. That signals are called ”sample-based” signals. Vector are then understand of frame of sample containing n indices. In that case we speak of ”frame-based” signals.

The modeling and the simulation of mixed-signal systems use both control domain and digital signal processing. Before the introduction of typed-data in Scicos, it was difficult to confront one environment which worked by default with a formalism issued from the control domain with all computational technics encountered in the digital signal processing. Today users of Scicos can choose and mix different sort of signals in a same diagram.

IV. Current implementation of matrix typed data

To do the typing and the addition of a second dimension for states, parameters and input/output registers, many improvments have been performed in the core of Scicos :

- The compiler have been updated concerning data type but also modified to automatically add the second dimension to ensure the compatibility with one-dimension blocks. The algorithm which adjusts the negative and zero size have been extented to also support the second dimension of regular input/output ports. At the level of the compiled structure (%cpr list), output register of blocks (outtb) is no more a vector of double[] but a list of Scilab objects. A discrete state (oz) and a parameter (opar) based on matrix typed data and have been added.
- The interfacing function (scicosim) have been entirely rewritten in C language because it was written in Fortran that didn’t allow to pass arrays of pointers to the simulator needed to use arbitrary data type.
- The simulator have been locally modified. The fixed point algorithm for output blocks (flag 6) used in the initialization phase have been rewritten to support matrix. Synchronous blocks of Scicos (if-then-else, event select) have also been treated concerning that extension. They can now use typed inputs to work.

At the editor level, a Scicos basic block can be then now viewed as shown in the figure 4.

Interfacing function of block can now handle eight different types of data which are : real numbers, complex numbers, signed integer numbers and unsigned integer number (8-bits, 16-bits and 32-bits for both). Morevover the management of negative and zero size by the compiler ensures coherence for some matrix operations. For example, one transpose matrix block will possess a regular input port with size [-1,-2] and an ouput port with size [-2,-1]. One matrix multiplication

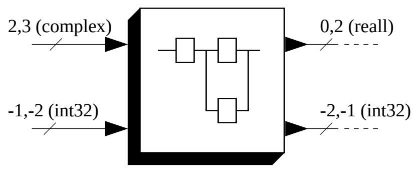

Fig. 4. Regular ports viewed in the editor with two dimensions and type

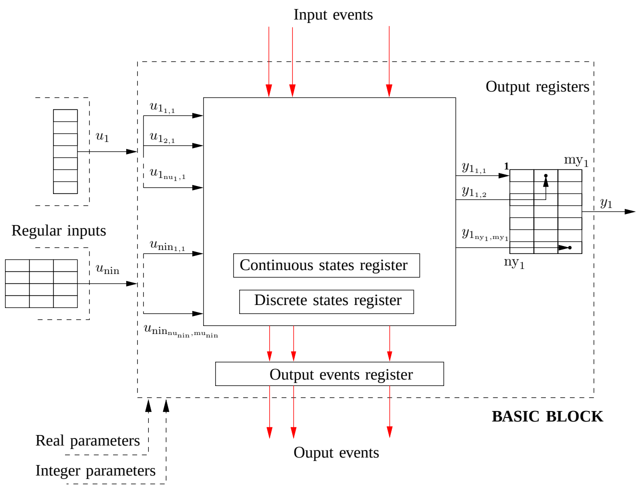

Fig. 5. Basic block viewed by the simulator with new registers

block has two inputs with respective sizes [-1,-2], [-2,-3] and therefore an output with size [-1,-3]. In interfacing functions, the original field model.in and model.out remains unchanged but new fields appears in the sublist model :

- model.in2/model.out2 to specify the second dimension of input/output matrix,
- model.intyp/model.outyp for the type number of input/output port (values between 1 and 8),
- model.odstate to store values of a matrix typed data of discrete states,
- model.opar to store values of a matrix typed data of parameters.

Improved registers provided by a computational function of a basic block are shown in the figure 4. To preserve compatibility with blocks that handle only one dimension for regular input/output port, the second dimensions and the type number have been contatenated in the fields insz[]/outsz[]. For example if a block have two inputs respectively defined with a matrix of real numbers of size [2,3] and a matrix of int32 numbers of size [1,5], the insz[] field is coded such as:

- first dimensions are stored in insz[0:nin-1] with insz[0]=2, insz[1]=1,
- second dimensions are stored in insz[nin-1:2*nin-1] with insz[2]=3, insz[3]=5,
- data-type number are stored in insz[2*nin:3*nin-1] with insz[4]=10(real), insz[5]=84(int32).

The table I gives the correspondence of the data type numbers for the interfacing function (model.intyp/outyp) and for the computational function.

Finally in the computational function, the input/output (inptr[][]/outptr[][]) of the *block structure are now C **void type.

Scilab	
Number	Type
1	real matrix
2	complex matrix
3	int32 matrix
4	int16 matrix
5	int8 matrix
6	uint32 matrix
7	uint16 matrix
8	uint8 matrix

C	
Number	Type
10	double
11	double
84	long int
82	short
81	char
841	unsigned long int
812	unsigned short
812	unsigned char

TABLE I

DATA TYPE NUMBER FOR THE EDITOR LEVEL AND FOR THE C LEVEL

To simplify the code writing of computational function, we also provide a set of C macros to handle the different informations (size/type/pointers/values) usable by the matrix data-type implementation. For example, a C computational function skeleton (type 4) for a block with one 8-bits integer matrix output port (char) and with a second complex number matrix output port will be :

```
#include "Scicos_block4.h"
void mycomputfunc(Scicos_block *block,int flag)
{
  /*varaible declaration*/
  char *y;
  double *y2_r;
  double *y2_i;
  int i,j,k;
  int nout;
  int ny[2],my[2];

  /*get number of output ports*/
  nout=block->nout;

  if (flag==1) {
   for (i=0;i<nout;i++) {
     /*get size of output ports*/
      ny[i]=GetOutPortRows(block,i+1);
      my[i]=GetOutPortCols(block,i+1);
   }

   /*get pointer of the first ouput port*/
   y=Getint8OutPortPtrs(block,1);

   /*get pointers of the second ouput port*/
   y2_r=GetRealOutPortPtrs(block,2); /*real part*/
   y2_i=GetImagOutPortPtrs(block,2); /*imag part*/
 ...
}
```

V. EXAMPLE OF APPLICATIONS

Block diagram of a Kalman Filter

One could find many examples that would use the improvments explained in the previous sections. The first example that we have choose here is a realization of a Kalman filter to demonstrate the modeling possibilities of Scicos concerning matrix operation.

Consider the following linear continuous-time system

$$\left\{\begin{array}{rcl} \dot{x} & = & Ax + Bu, \\ y & = & Cx, \end{array}\right.$$

where x is a state variable vector, assimilable to a matrix of size [n,1], u inputs vector assimilable to a matrix of size [p,1], y the vector of outputs assimilable to matrix of size [q,1], A a square matrix of constants of size [n,n], B a matrix

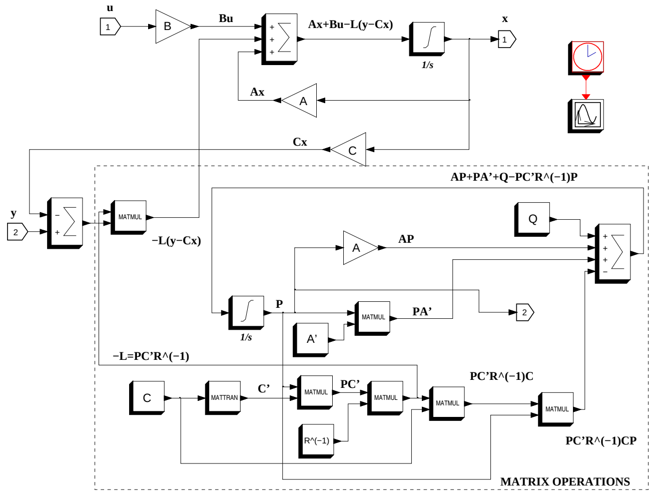

Fig. 6. Block diagram of the Kalman filter

of constants of size [n,p] and C a matrix of constants of size [q,n]. For that system, the associated Kalman filter is written

$$\left\{\begin{array}{rcl} \dot{\hat{x}} & = & A\hat{x} + B\tilde{u} - L\,(t)\,(\tilde{y} - C\hat{x})\,, \\ L & = & -PC^{'}R^{-1}, \\ \dot{P} & = & AP + PA^{'} + Q - PC^{'}R^{-1}CP. \end{array}\right.$$

with L a matrix of state variables of size [n,q], P a square matrix of states variables of size [n,n] and R a square matrix of constants of size [q,q]. Note that the resolution of that filter needs various matrix operations such the addition, the time-domain integration, the transposition, the multiplication and the invertion. All of these operations have been then implemented as basic blocks that users can find in the standard palettes of Scicos. The resulting modeling by the use of block diagram of the previous filter is given in the figure 6 and simulation results concerning dynamical computation of elements of the P matrix are shown in the figure 7 for sizes $n = 3$, $p = 1$, $q = 2$ with $A = [-0.3, 3, 1; 0, 0, 2; 0, 0, 0]$, $B = [1; 2; 3]$, $C = [1, 1, 2; 0, 2, 3]$, $x(0) = [-2; 1; 2]$, $P(0) = \text{zeros}(3, 3)$, $R = [1, 0; 0, 1]$ and $Q = [10, 0, 0; 0, 10, 0; 0, 0, 10]$.

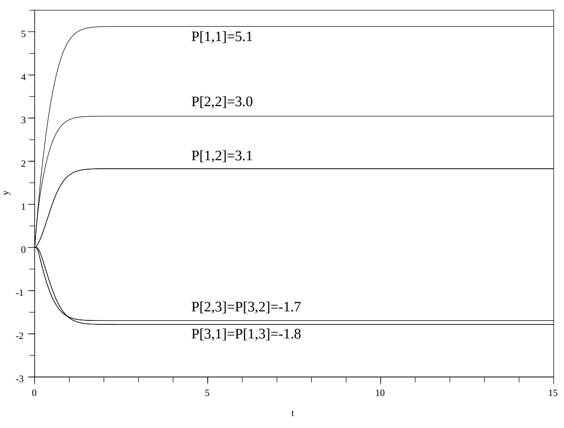

Fig. 7. $P(t)$ matrix values of the simulated Kalman filter