International Journal of Control

ISSN: 0020-7179 (Print) 1366-5820 (Online) Journal homepage: www.tandfonline.com/journals/tcon20

Symbolic analysis and design of control systems using *Mathematica*

M. D. Lutovac & D. V. Tošić

To cite this article: M. D. Lutovac & D. V. Tošić (2006) Symbolic analysis and design of control systems using ***Mathematica*** , International Journal of Control, 79:11, 1368-1381, DOI: 10.1080/00207170600725552

To link to this article: https://doi.org/10.1080/00207170600725552

International Journal of Control
Vol. 79, No. 11, November 2006, 1368–1381

Taylor & Francis
Taylor & Francis Group

Symbolic analysis and design of control systems using *Mathematica*

M. D. LUTOVAC* and D. V. TOŠIĆ

School of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandra 73, 11000 Belgrade, Serbia

(*Received 29 June 2005; in final form 12 March 2006*)

Contemporary computer tools can generate a tremendous amount of numerical data so the user might easily lose insight into the phenomenon being investigated. Those who use powerful computer algebra systems must thoroughly understand the assumptions that underlie the software. In this paper, the role and importance of symbolic computation in control engineering and signal processing is exemplified. Real-life application examples are presented in which systems are symbolically solved and simulated with *Mathematica*. We introduce an original approach to algorithm development, system design and symbolic processing that successfully overcomes some problems encountered in the traditional approach. Benefits of symbolic methods and the role of computer algebra systems are highlighted from the viewpoint of both academia and industry.

1. Introduction

The strong trend to use more sophisticated and complex algorithms combines expertise in system design, signal processing and software engineering. The classic computer tools are essentially based on numeric computation and they might be inefficient when used to satisfy the needs of a new generation efficient system because the design space is virtually unbounded. The design of modern control systems based on only extensive numeric simulations is an obsolete approach, that is, symbolic techniques should be considered to complement traditional numeric algorithms. The current symbolic computation environments are extremely powerful in doing symbolic and mixed symbolic-numeric mathematics for technical computing. They can be used for the efficient and effective search for optimal solutions.

A common approach taken in modern software is to use a graphic interface, where clicking a mouse button while the cursor is positioned over an active area of the screen causes something to happen. Programs can provide knowledge about the design of algorithms and employ that knowledge in its symbolic manipulation. The power of an environment that is interactive can help in systematically exploring possible algorithm designs.

The design process is the activity of taking algorithms expressed in a high-level algorithmic form and turning them into working systems, either hardware or software. In other words, the design process turns algorithms into implementations. Programs can generate computer source codes from an algebraic expression or a graphical representation that can then be compiled onto a target architecture.

Computers have become recognized as symbol processors (Oppenheim and Nawab 1992). Correspondingly, a program can be viewed as a set of instructions for manipulating symbols. Numbers are only one of a range of symbols that the computer can handle. A symbol processor with the appropriate programs is usable on a much wider range of tasks, such as intelligence amplification or augmenting our ability to think. Thus, the symbol processor has affected a change in how programming is viewed. Programming has become a task of knowledge accumulation that tells the computer what to know, when to use information, and how to apply the knowledge in solving problems. Viewing programs as knowledge repositories leads

*Corresponding author. Email: lutovac@etf.bg.ac.yu

International Journal of Control
ISSN 0020–7179 print/ISSN 1366–5820 online © 2006 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/00207170600725552

naturally to the need for programs to be written to communicate, not simply to compute.

The use of computer algebra systems (CAS) and symbolic processing can help us to gain insight into how a complex system works, which is preferred to experimenting with numeric simulations. The current leader in implementing CAS (*Mathematica* by Wolfram Research) is extremely powerful in doing symbolic mathematics for technical computing. In this paper, the enormous potential of symbolic computation tools and their applications is discussed.

The paper focuses on the application of CAS in controling engineering and demonstrates the use of symbolic computation in the time and frequency domains (Lutovac *et al*. 2001, Lutovac and Tošić 2003). The computations are performed with *Mathematica* 5 (Wolfram 2003) and the application packages *Control System Professional* (Bakshee 2003, Palancz *et al*. 2005) and *SchematicSolver* (Lutovac and Tošić 2004).

2. Schematic representation of systems

Traditional mathematical formulas that describe a system can give relations and results but, typically, cannot capture the calculation processes that lead to them. Instead, system models are often best represented directly as visualized algorithms by means of pictorial block-diagram representations.

SchematicSolver is an application package that provides drawing block-diagrams that represent various systems, such as control systems. The graphical representation of a system is essential for supporting a designer's view of implementation, which often comes in the form of block-diagrams. It provides an easy-to-use graphical user interface (GUI) for building models as block-diagrams, using mouse operations for performing drawing tasks.

2.1 *Interactive drawing*

SchematicSolver provides an easy point-and-click interface for performing the most common drawing tasks. When drawing a new element, the corresponding specification is automatically added into the schematic specification. The schematic specification contains all the details necessary for drawing, solving, simulating and implementing the system.

Consider, as an example, a space shuttle which has a system that incorporates feedback to control the pitch of a vehicle. Measurements are made by the vehicle's inertial unit, gyros and accelerometers. A simplified model of the shuttle pitch controller is shown in figure 1 (Nise 2000).

SchematicSolver describes a system as a symbolic object in the form of a list of elements. Here is a portion of the list called `shuttlePitchController`

```
ShuttlePitchController = {
  {"Input", {1, 7}, X1, "Commanded pitch",
    TextOffset → {0, −1}},
  {"Adder", {{0, 6},{1, 0},{2, 6},{1, 7}},
    {0, −1, 2, 1}},
  {"Amplifier", {{2, 6},{4, 6}}, K1},
  {"Block", {{10, 6},{12, 6}}, G1,
    "Controller"},
  {"Output", {14, 6}, Y4, "Pitch"},
  ...
  {"Line", {{14, 6},{13, 4}}}
};
```

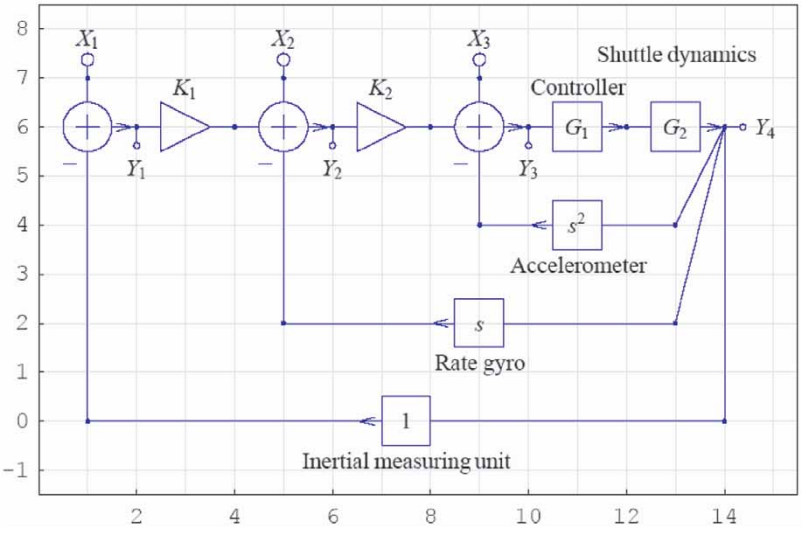

Figure 1. Simplified model of the pitch controller for the space shuttle.

In two consecutive *Mathematica* cells, two different representations exist: (1) the graphical representation of the block-diagram and (2) the textual description of the block-diagram. Interactive drawing means that (1) by adding a new graphical element, one can automatically change the textual description or (2) by editing the textual specification, a new graphical diagram is drawn.

The `shuttlePitchController` list specifies what elements are in the system and how they are interconnected. A list that describes a system is referred to as the schematic specification. Each element in the system is also described as a list that states what the element is, to which other elements it is connected and what its value is. A list that describes an element is referred to as the element specification. For example, the controller block is represented as the `Block` element {"`Block`", `{{10,6}, {12,6}}, G1,` "`Controller`"} where `{{10,6}, {12,6}}` are the element coordinates, `G1` is the symbolic element value, and "`Controller`" is the label associated with the element. It is not necessary to manually insert all elements. A large schematic can be made from replicas of other schematics.

2.2 *Automated drawing of systems*

SchematicSolver provides functions that create schematics important for practice. New models can be constructed from automatically generated schematics. For example, given the system order `systemOrder=3`, one can automatically generate (1) system parameters `pNum`, `pDen`, and (2) a schematic of a recursive discrete system `discreteSchematic`

```
pNum=UnitSymbolicSequence[systemOrder,
  a,0]
pDen=UnitSymbolicSequence[systemOrder
+1,b,0]
{schematicSpec,inCoords,outCoords} =
  TransposedDirectForm2IIRFilter
    Schematic[{pDen,pNum}];
```

Input and output elements are added to complete the schematic

```
discreteSchematic=Join[schematicSpec,
  {{"Input",inCoords[[1]],X},{"Output",
    outCoords[[1]],Y}}];
```

The schematic can be drawn using `ShowSchematic[discreteSchematic]` as shown in figure 2.

The graphical representation of a system is not simply a static picture, but it is a symbolic representation of the system. Once a schematic of a system is generated, it can be used to build a schematic of another system. For example, the element names can be replaced: the `Delay` element is replaced by the Integrator element and the `Multiplier` element is replaced by the `Amplifier` element, as shown in figure 3; a continuous-time system is created from the schematic of a discrete system. This unique feature is possible because the schematic element is also a symbolic object and the symbol processor can be used to change the overall functional description of a system. In addition, the symbolic value of a multiplier can be substituted by a numeric value. Moreover, the multiplier element can be replaced by an arbitrary block element

```
continuousSchematic=discreteSchematic/.
  {"Delay"→"Integrator",
  "Multiplier"→"Amplifier"};
  %//ShowSchematic
```

Here is another example: for known symbolic parameters c0, . . . , c7, an automatically generated schematic of the Hilbert transformer is shown in figure 4.

Only one numeric value, the system order, is specified and the symbolic system parameters and the schematic specification are automatically generated.

3. Continuous-time systems

Application packages can be used to work as an integrated environment. For example, *SchematicSolver* can be used for (a) drawing a schematic of a control system and then (b) computing the system transfer function directly from the schematic. *Control System*

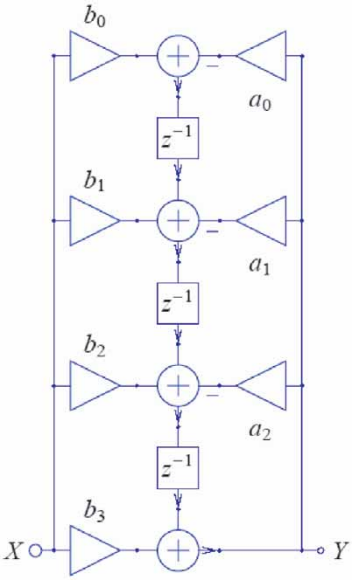

Figure 2. Schematic of the third-order recursive discrete system.

Professional can be used for post-processing the results (c) to find the state-space realization, (d) simplify and find a more convenient presentation of a continuous-time system, (e) represent the state-space realization of an MIMO system in the traditional typeset form, (f) display state-space objects as the familiar state-space equations and (g) compute the state response or output signal of a continuous system.

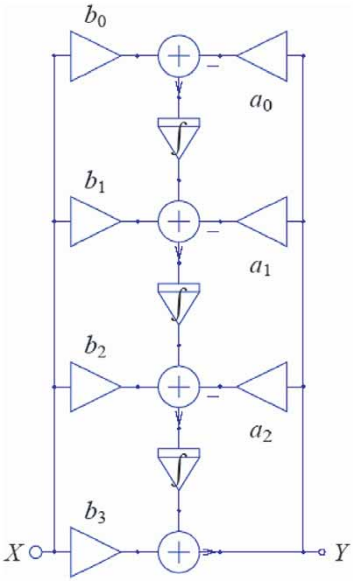

Figure 3. Schematic of the third-order continuous system.

3.1 *Solving continuous-time systems*

The following example describes a typical use of CAS in control system analysis. Suppose that the published block-diagram of a system and the corresponding transfer function should be proved. Manual derivation can be tedious and error prone even for a motivated researcher. Alternatively, *SchematicSolver* can be used to draw the system schematic, such as that shown in figure 1, and to find the transfer function. The transfer function matrix of this three-input four-output MIMO system is shown in figure 5.

3.2 *Time-domain analysis of continuous-time systems*

The step response of a system might be found symbolically, as well. For example, the block-diagram of the CD-media controller FAN8024D/BD, figure 6, which is a 4CH motor drive IC suitable for CD-media applications (include CD-ROM, CD-RW, DVDP, and DVD-ROM), has two current feedback control channels for the focus and tracking actuator (Application Note 2001).

The computed transfer function (denoted by `TF`) is presented in the traditional mathematical form

$$\frac{2(CRs+1)A_P}{CL_aR_{\text{in}}s^2+CR_aR_{\text{in}}s+2CRA_PA_SR_Ss+2A_PA_SR_S}.$$

Assuming the sensing resistor of 0.5 Ω, the load impedance of Maker1 40× focus actuator 18.5 Ω and 228.5 μH at 1 kHz, the required system bandwidth of 60 kHz, ($Rs=0.5, A_P=2, As=2, C=76.5\times10^{-12}$,

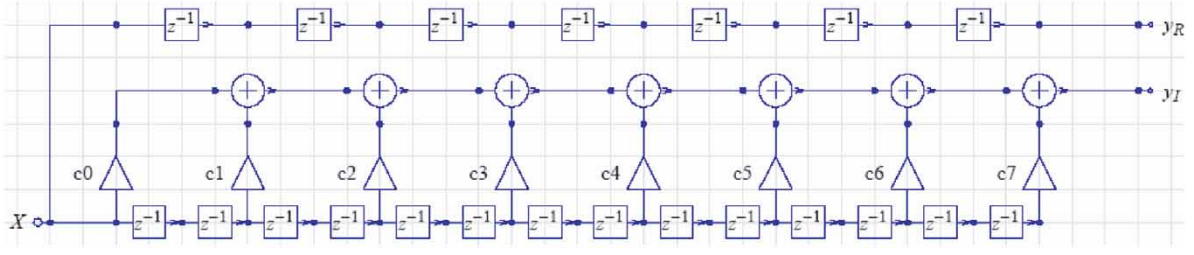

Figure 4. Schematic of the Hilbert transformer.

$$\begin{pmatrix} \frac{G_1G_2s^2+G_1G_2K_2s+1}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & -\frac{G_1G_2K_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & -\frac{G_1G_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} \\ \frac{G_1G_2K_1s^2+K_1}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{G_1G_2s^2+1}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{-sG_1G_2-G_1K_1G_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} \\ \frac{K_1K_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{K_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{1}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} \\ \frac{G_1G_2K_1K_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{G_1G_2K_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} & \frac{G_1G_2}{G_1G_2s^2+G_1G_2K_2s+G_1G_2K_1K_2+1} \end{pmatrix}$$

Figure 5. Transfer function matrix of the MIMO system shown in figure 1.

$L_a = 228.5 \times 10^{-6}$, $R = 161500$, $R_a = 18.5$, $R_{\text{in}} = 7500$), the step response can be computed by

```
InverseLaplaceTransform[TF/s, s,t]
```

which yields

$$4(0.25 - 0.249976\mathrm{e}^{-376979.t} - 0.0000239502\mathrm{e}^{-80934.4t})$$

3.3 *Simplifying and comparing realizations*

Consider a single-input two-output system generated from the state-space model (Bakshee 2003), as shown in figure 7(a), and its transfer function $\{\{1/s\}, \{1/(s+\alpha)\}\}$. For the same transfer function matrix, a simpler realization can be achieved as shown in figure 7(b). There is no realization with the number of integrators less than two, but the number of adders and amplifiers has been reduced.

The state-space representation of the system from figure 7(a) can be derived and represented in the traditional typeset form using the `StateSpace` function (Bakshee 2003)

$$\left(\begin{array}{cc|c} 0 & 1 & 0 \\ 0 & -\alpha & 1 \\ \hline \alpha & 1 & 0 \\ 0 & 1 & 0 \end{array}\right)^{S}.$$

Obviously, the state-space representation of the system from figure 7(b) is different in spite of the same transfer functions. The state-space representation that is obtained directly from the transfer function matrix corresponds to the controllable companion form, sometimes referred to as the controllable canonical form. This example shows the importance of the pictorial representation of the system, that is, a traditional formula does not illustrate the process described by the block-diagram. State-space equations cannot be uniquely determined from the transfer functions; they are uniquely defined only if the schematic of the system is known.

3.4 *Design of a system from the step response*

Linear causal systems can be designed in a straightforward manner for a known step response. Assume that a closed-form expression of the step response is known

$$h(t) = \frac{1}{10} - \frac{1}{5}e^{-10t} + \frac{1}{10}e^{-5t}(3\cos(t5\sqrt{3}) - \sqrt{3}\sin(t5\sqrt{3})), \quad t > 0.$$

The corresponding transfer function can be symbolically computed from

```
H = s*LaplaceTransform[h, t, s]
```

which yields

$$H = \frac{s^2 + 100}{s^3 + 20s^2 + 200s + 1000}.$$

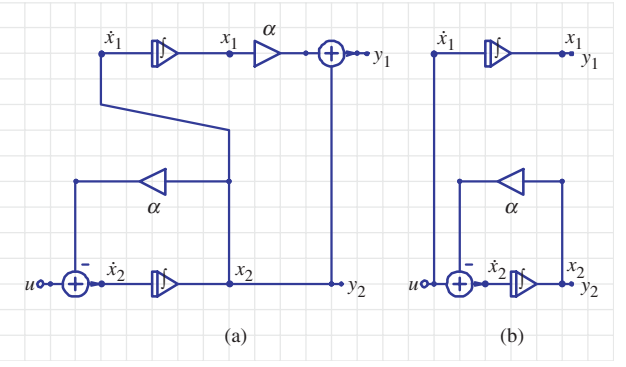

Figure 7. Schematics of two systems with the same transfer functions.

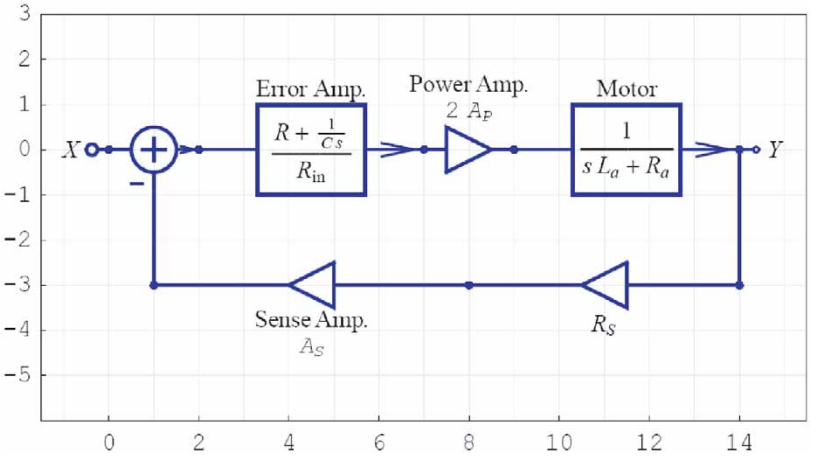

Figure 6. Schematic of the CD-media controller.

The corresponding system block-diagram might be as shown in figure 8.

There is a simple procedure to prove that the system represented by the above schematic has the given step response: (1) find the transfer function directly from the schematic; and (2) compute the inverse Laplace transform.

4. Discrete-time systems

SchematicSolver can be used to (a) draw the schematic of a discrete MIMO system, (b) compute the system transfer function directly from the schematic (c) find the response for given input sequences or (d) derive some properties of the system.

4.1 *Solving discrete-time systems*

Consider a high-speed filter that can be used in multirate systems for interpolation and decimation with a factor of two (Milic and Lutovac 2002). The automatically generated schematic of the filter is composed of all-pass filters and extra multipliers (Milic and Lutovac 2001), figure 9.

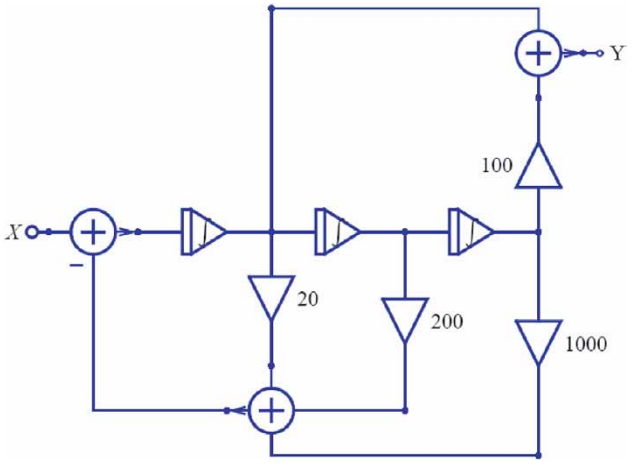

Figure 8. Block-diagram of the system designed from the step response.

```
{s, inXY, outXY} =
HighSpeedIIR3FIRHalfbandFilterSchematic
  [{b, k0, k1, k2, k3}, {0, 0}];
```

`DiscreteSystemTransferFunction` computes the transfer functions `H1=Y1/X`:

$$\frac{1}{z^5(b+z^2)^5} \times (b^5k0k3 - b^4k0k1k3z + b^3k0k2z^2 + 5b^4k0k3z^2 + b^3k0k1k2k3z^2 - b^2k0k1k2z^3 - 4b^3k0k1k3z^3 - b^5k0k1k3z^3 - b^2k0k2k3z^3 + bk0k1z^4 + 3b^2k0k2z^4 + 2b^4k0k2z^4 + 10b^3k0k3z^4 - 3b^2k0k1k2k3z^4 - 2b^4k0k1k2k3z^4 + k0z^5 - 2bk0k1k2z^5 - 3b^3k0k1k2z^5 - 6b^2k0k1k3z^5 - 4b^4k0k1k3z^5 - 2bk0k2k3z^5 - 3b^3k0k2k3z^5 + k0k1z^6 + 4b^2k0k1z^6 + 3bk0k2z^6 + 6b^3k0k2z^6 + b^5k0k2z^6 + 10b^2k0k3z^6 - 3bk0k1k2k3z^6 - 6b^3k0k1k2k3z^6 - b^5k0k1k2k3z^6 + 5bk0z^7 - k0k1k2z^7 - 6b^2k0k1k2z^7 - 3b^4k0k1k2z^7 - 4bk0k1k3z^7 - 6b^3k0k1k3z^7 - k0k2k3z^7 - 6b^2k0k2k3z^7 - 3b^4k0k2k3z^7 + 4bk0k1z^8 + 6b^3k0k1z^8 + k0k2z^8 + 6b^2k0k2z^8 + 3b^4k0k2z^8 + 5bk0k3z^8 - k0k1k2k3z^8 - 6b^2k0k1k2k3z^8$$

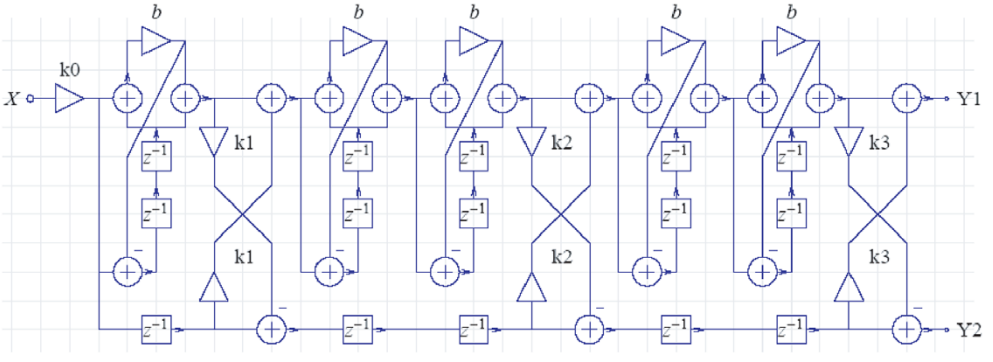

Figure 9. Power complementary high-speed filter.

$$
\begin{aligned}
&- 3b^4k0k1k2k3z^8 + 10b^2k0z^9 - 3bk0k1k2z^9 \\
&- 6b^3k0k1k2z^9 - b^5k0k1k2z^9 - k0k1k3z^9 \\
&- 4b^2k0k1k3z^9 - 3bk0k2k3z^9 - 6b^3k0k2k3z^9 \\
&- b^5k0k2k3z^9 + 6b^2k0k1z^{10} + 4b^4k0k1z^{10} \\
&+ 2bk0k2z^{10} + 3b^3k0k2z^{10} + k0k3z^{10} - 2bk0k1k2k3z^{10} \\
&- 3b^3k0k1k2k3z^{10} + 10b^3k0z^{11} - 3b^2k0k1k2z^{11} \\
&- 2b^4k0k1k2z^{11} - bk0k1k3z^{11} - 3b^2k0k2k3z^{11} \\
&- 2b^4k0k2k3z^{11} - 4b^3k0k1z^{12} + b^5k0k1z^{12} + b^2k0k2z^{12} \\
&- b^2k0k1k2k3z^{12} + 5b^4k0z^{13} - b^3k0k1k2z^{13} \\
&- b^3k0k2k3z^{13} + b^4k0k1z^{14} + b^5k0z^{15}).
\end{aligned}
$$

Needless to say, derivation of this transfer function by hand is very time consuming and difficult (if possible at all) for more complex high-speed filters.

4.2 *Proving the property of the system*

The filter of figure 9 is a power complementary filter, that is,

$$H_1(z)H_1\left(\frac{1}{z}\right) + H_2(z)H_2\left(\frac{1}{z}\right) = 1.$$

It cannot be simply proved without a powerful CAS. The *Mathematica* built-in function FullSimplify shows that the sum of the squares of the magnitudes is constant

```
tfSum2 = H1*(H1/. z → 1/z) + H2*(H2/. z →
  1/z)//FullSimplify
```

which yields

$$2k0^2(1 + k1^2)(1 + k2^2)(1 + k3^2).$$

Note that `tfSum2` does not depend on `z` or `b`.

The parameter `k0` can be symbolically computed for a given value of `tfSum2` (one in this case)

```
k0Rule = Solve[tfSum2 == 1, k0]
```

that is

$$k0 \to \frac{1}{\sqrt{2}\sqrt{1 + k1^2}\sqrt{1 + k2^2}\sqrt{1 + k3^2}}.$$

One can proceed without spending time on a proof of the power complementary property.

4.3 *Original design procedure*

The design of a system starts with a set of requirements and it usually consists of (1) approximation, (2) realization, (3) study of imperfections and (4) implementation (Lutovac *et al.* 2001). The production cost may depend on the type of components, packaging, methods of manufacturing, and testing. If the requirements are not met, or the cost is very high, the realization step or the approximation step must be redone, and then, the development cost can become very large.

CAS can provide a new original design procedure. Starting with the preferred efficient implementation, the functional description (say the transfer function) can be expressed in terms of symbolic parameters. Thus, approximation is no more the first step, but it becomes the last step in the design of the system. For example, consider a system shown in figure 9. The symbolic design parameters are `b`, `k0`, `k1`, `k2`, and `k3`. Assume the requirements which imply that `H1` should be zero for `z = −1`

```
num3 = Numerator[H1//Together]/. z → −1//
  Factor
(1 + b)^5 k0 (1 − k1 − k2 − k3 − k1 k2 − k1 k3 − k2
k3 + k1 k2 k3) == 0.
```

The coefficient `k0` has been already computed in the previous section. The poles of a stable discrete system should be within the unit circle, that is, $-1 < b < 1$. Finally, `k2` can be expressed in terms of `k1` and `k3`

```
Solve[num3 == 0, k2]//Flatten//FullSimplify
```

$$\left\{k2 \to \frac{-1 + k1 + k3 + k1k3}{-1 + k1(-1 + k3) - k3}\right\}.$$

The two expressions for `k0` and `k2` are derived in terms of `k1` and `k3`, directly from the schematic, using the two properties of the high-speed filter (Lutovac and Tošić 2005b). No assumption on the filter approximation function has been made. Furthermore, the closed-form expressions show that `k0` and `k2`, and the corresponding properties, do not depend on the parameter `b`. This means that the coefficient `b` can be independently optimized to satisfy other requirements. Other requirements can be met by a simpler two-variable numeric optimization (`k1` and `k3`), instead of a much more complex optimization of the five design parameters. This way, CAS helps to come up with an efficient implementation and balance time spent for optimizing the design cost against completing the project quickly.

4.4 *Time-domain analysis of discrete systems*

In symbolic processing, the signal can be represented as a formula, rather than as a sequence of numbers. Thus, the value of a signal might be known in terms of a formula. In a similar manner, signal-processing operators, which correspond to the building blocks of systems, are maintained in symbolic form. This enables

CAS to simplify, rearrange and rewrite symbolic expressions until they take a desired form. When one of the operators is applied to a function, no evaluation takes place. The resulting function is stored in the symbolic form until it becomes convenient to compute it explicitly.

Consider a discrete-time recursive system whose transfer function is known, say the numerator is $(1+2z^{-1}+z^{-2})$ and the denominator is $(1+0.5z^{-2})$. First, the schematic of the system is created by the command

```
{schematic, {inpCoord}, {outCoord}} =
TransposedDirectForm2IIRFilterSchematic
  [{{1, 2, 1}, {0, 1/2}}];
```

Next, the input element and the output element are added:

```
system =
Join[schematic, {{"Input", inpCoord, X},
{"Output", outCoord, Y}}];
```

The block-diagram of the system is shown in figure 10.

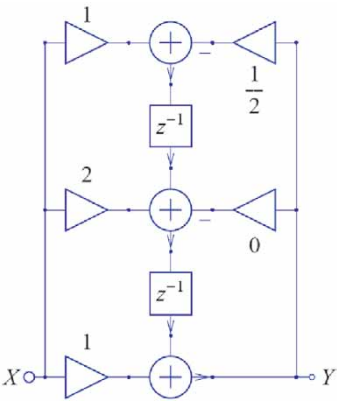

Figure 10. Block-diagram of the recursive system that has been automatically generated for given parameters.

Transfer function can be computed directly from the schematic system as follows:

```
{tfMatrix, systemInp, systemOut} =
   DiscreteSystemTransferFunction[system];
tf = tfMatrix[[1, 1]];
```

Suppose that the input signal can be represented by a formula

```
sineSignal = Sin[n/5];
```

Traditionally, the output signal can be found by (1) multiplying the z-transform of the input signal by the transfer function and (2) taking the inverse z-transform of the product

```
sineTransform = ZTransform[sineSignal, n, z];
response = InverseZTransform
  [sineTransform*tf, z, n];
```

which yields

$$\frac{8\,\mathrm{Cos}[1/10]^2(2^{-n/2}\mathrm{Cos}[n\pi/2]\mathrm{Sin}[1/5]-2\mathrm{Sin}[(1-n)/5]+\mathrm{Sin}[(1+n)/5])+2^{1/2-n/2}(\mathrm{Sin}[1/5]-4\mathrm{Sin}[2/5])\mathrm{Sin}[n\pi/2]}{5+4\mathrm{Cos}[2/5]}.$$

4.5 *Symbolic processing*

In many cases, the traditional approach to computing the response of a discrete system, for known input signal, cannot be accomplished successively. Even a very powerful CAS can fail to find the z-transform or the inverse z-transform. Symbolic processing can be used instead to overcome the problem.

For the schematic of figure 10, symbolic processing can be carried out as follows: (1) automatically generate a code that implements the system, (2) compute the input sequence whose elements can be symbols, numbers, or formulas, and (3) process the input sequence with the code

```
DiscreteSystemImplementation[system, "imp"];
inSeq = UnitSineSequence[8, 1/(10 π), 0];
{outSeq,finals} =
  DiscreteSystemImplementation
    Processing[inSeq, {0, 0}, {}, imp];
```

For example, the seventh element of the output sequence is not a number, it is an expression

$$\frac{1}{4}\left(2\mathrm{Sin}\left[\frac{1}{5}\right]-\mathrm{Sin}\left[\frac{2}{5}\right]-4\mathrm{Sin}\left[\frac{3}{5}\right]+2\mathrm{Sin}\left[\frac{4}{5}\right]+8\mathrm{Sin}[1]\right)+\mathrm{Sin}\left[\frac{6}{5}\right].$$

The expression derived by using the z-transform may look different, such as

$$\frac{8\mathrm{Cos}[1/10]^2(-1/8\mathrm{Sin}[1/5]+2\mathrm{Sin}[1]+\mathrm{Sin}[7/5])}{5+4\mathrm{Cos}[2/5]}.$$

Simplifying the difference of these two expressions shows that the output sequences in both cases are the same. Symbolic processing of a symbolic sequence, by a system that can be described with symbolic parameters,

always returns a symbolic output sequence, even in the cases when CAS fails to compute the z-transform or the inverse z-transform. In addition, at any step of the processing, any symbol can be replaced by a number.

4.6 *Comparing multirate realizations*

If transfer functions of two different linear single-rate discrete systems are the same, one can choose the system that is more suitable for practical implementations. Classic signal processing assumes a single-rate system in which the sampling rates are the same at all nodes of the system. Multirate systems work with two or more sampling rates and the comparison of different structures becomes more difficult.

Suppose that two interpolation systems, called classic and efficient, have been reported in the literature (Milic and Lutovac 2002). The down-sampling and up-sampling operations can change the spectrum of the signal and the traditional z-transform analysis is not directly applicable. The block-diagrams of the two interpolation systems can be automatically generated as shown in figures 11 and 12.

For both schematics, symbolic processing can be automated as follows: (1) generate a code that implements the system, (2) generate the input sequence whose elements can be symbols, and (3) process the input sequence with the generated codes. The output sequences, outClassicSeq and outSeq, are sequences whose elements are some combination of symbols from the input sequence and system parameters. For example, the 100th element of the output sequence is

```
(c59 x41 + c54 x46 + c49 x51 + c44 x56 + c39
x61 + c34 x66 + c29 x71 + c24 x76 + c19 x81 +
c14 x86 + c9 x91 + c4 x96)
```

Comparing the output sequences of the two systems with

```
SameQ[outClassicSeq, outSeq]
```

returns the answer `True`. This means that the two symbolic sequences are the same for arbitrary values of

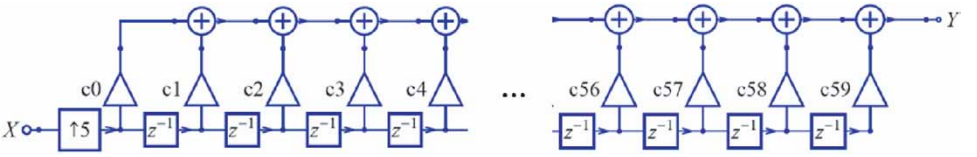

Figure 11. Classic interpolation system.

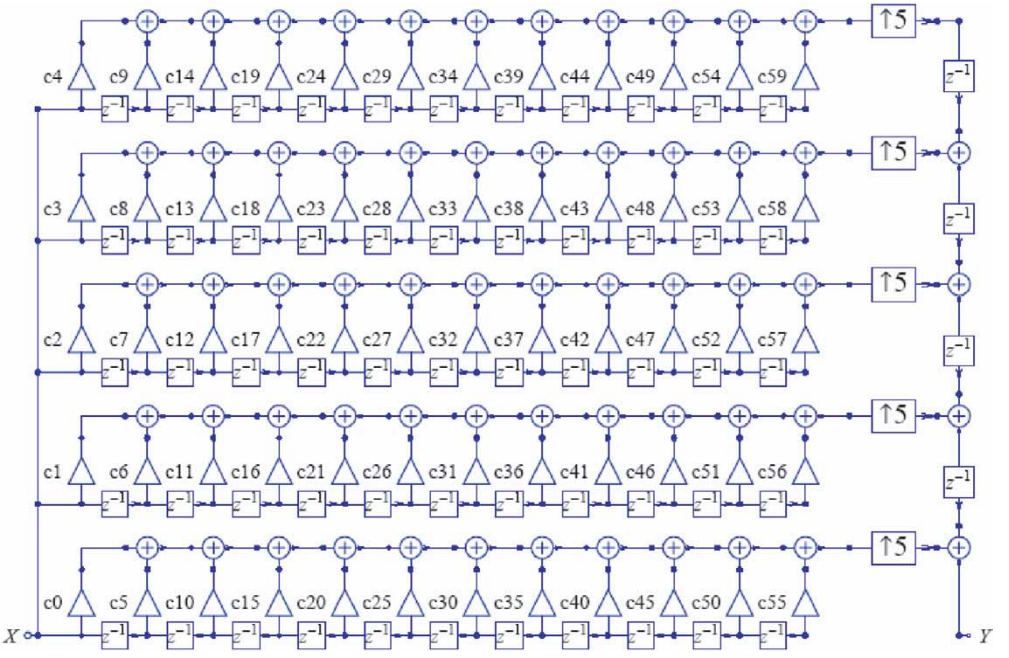

Figure 12. Efficient interpolation system.

the input sequence and the system parameters. This proves that the systems have the same functionality.

However, the efficient system has five times fewer computations than the classic interpolation system.

4.7 *Finite wordlength effects*

Quantized filter coefficients make the transfer function different from the ideal one (with infinite-precision coefficients). Those errors can be efficiently predicted by the coefficient sensitivity analysis that is based on symbolic derivation and substitution operations (Lutovac *et al*. 2001). On the contrary, the computation of the variance of the uncorrelated noise due to rounding the output of the multiplier element is very non-linear and the numeric analysis was extensively used in practice. CAS is capable to successively solve this non-linear problem in a straightforward manner. *SchematicSolver* can be used to build a symbolic object of the system as schematic specification and to identify the symbolic system parameters. After that, *Mathematica* can be used to analyze the symbolic object, to derive the partial transfer function that corresponds to the identified system parameter, to express the noise variance in terms of the pole magnitude and the pole argument, to generate a noise transfer function, and finally to derive the variance of uncorrelated noise due to rounding, see VQNR function in Lutovac *et al*. (2001, p. 362).

4.8 *Performance optimization using symbolic algorithm*

The traditional numeric algorithms may fail to find the optimal solution such as the special case of elliptic rational functions—Minimum-Q elliptic function (Lutovac and Tosic 2005a). Minimum-Q elliptic became a standard function in manufacturing integrated filters (FilterCAD 2002). Using symbolic optimization, a very efficient digital signal processing (DSP) system based on programmable logic devices and very large-scale integrated circuits was implemented (Lutovac and Lutovac 2002). An efficient DSP system is a system that can be implemented by employing a small number of adders and binary shifters, only.

The design of a system starts with requirements and the approximation in which all design parameters are kept as symbols. The symbolic optimization is performed in the realization step when the most critical parameters are symbolically optimized for the simplest or the most robust implementation. Finally, the less sensitive parameters are optimized using the numeric optimization algorithms. This way, the closed-form relations can be used to reduce the number of design parameters for numeric optimization and simplify the numeric optimization algorithm which has been reported in (Lutovac and Lutovac 2002).

5. Non-linear systems

Many control systems are non-linear and CAS might find the response of a system in terms of the input signal and system parameters. The preferred solution is to find the closed-form expression of the output signal for a known stimulus given by a closed-form expression.

Consider a part of an adaptive system based on the least mean squares (LMS) algorithm, shown in figure 13. Can the output signal be found as a closed-form expression in terms of the sample index?

Directly from the schematic (named system), a code (called implement) that implements the system is automatically generated

```
DiscreteSystemImplementation[system,
  "implement"];
```

For known input values (say, `x1 = 1` and `x2 = 10*x1`) and an unknown previous state `d1` in the memory element, the two successive output values, `y2` and `y3`, are

```
{{y2}, {d2}} = implement[{1, 10}, {d1}, {}];
{{y3}, {d3}} = implement[{1, 10}, {d2}, {}];
```

Suppose that `y2` is the output value for the sample index `(n − 1)` and `y3` is the output value for the index `n`. Both values are functions of the unknown state `d1`. The function Reduce tries to eliminate the initial state `d1` and tries to find the relation between the two output samples

```
eqns = Reduce[{y[n − 1] == y2, y[n] == y3},
{d1}];
```

The following recurrence equation is obtained:

```
reducedEqn = (15 y[−1 + n] == −10 + 16 y[n]);
```

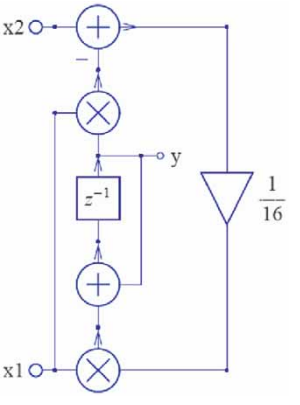

Figure 13. Non-linear system based on the LMS algorithm.

The closed-form solution can be found by

```
sol = RSolve[{reducedEqn, y[0] == 0}, y[n], n];
```

which yields

$$\left\{\left\{y[n] \to 10\left(1-\left(\frac{15}{16}\right)^{n}\right)\right\}\right\}.$$

The number of samples after which the output sequence reaches the value b, say 90% of the amplified input, is obtained by

```
Solve[y[n] == 10*b, n];
```

yielding

$$n \to \frac{\text{Log}[1-b]}{\text{Log}[16/15]}.$$

For `b = 0.9`, the number of samples is 36. Has it any application for a practical system?

Random binary signals are very suitable for system identification. For an input sequence of randomly generated values 1 or −1, the output of the system is shown in figure 14.

Obviously, the number of samples, after which output reaches 90% of the amplification `x2/x1 = 10`, is the same as that obtained by symbolic solving the system. Therefore, the closed-form response to the deterministic signal (step signal) helps to find the time required to identify the unknown system when the input signal has white-noise-like properties.

6. Algorithm development

Consider an algorithm for the implementation of an efficient method for approximating the reciprocal using a modified Newton-Raphson iteration. The algorithm can be implemented in any hardware that executes the add, subtract, and multiply operations and that uses memory elements. Can the iteration error be found as a closed-form expression in terms of the number of iterations?

The algorithm can be visualized by a block-diagram as shown in figure 15.

Directly from the schematic (named `systemNR`), a code (called implementNR) that implements the algorithm can be automatically generated:

```
DiscreteSystemImplementation[systemNR,
"implementNR"];
```

Two successive output values, `y2` and `y3`, for a given input value `x`, an initial guess `b ≈ 1/x`, and an unknown previous state `d1` in the memory element, are

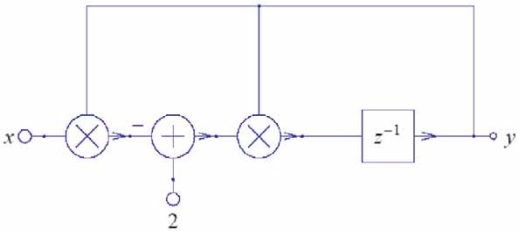

Figure 15. Schematic of the algorithm for computing $y = 1/x$ based on the modified Newton-Raphson iteration.

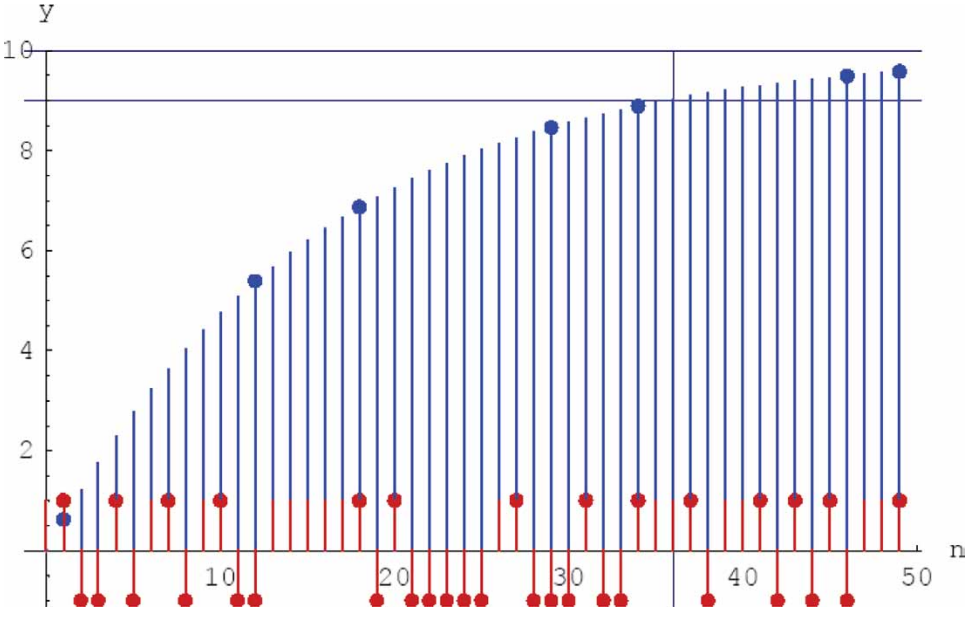

Figure 14. Input signal of randomly generated values 1 or −1 and output signal of the non-linear LMS system.

```
{{y2}, {d2}} = implementNR[{x, 2}, {d1}, {}];
{{y3}, {d3}} = implementNR[{x, 2}, {d2}, {}];
```

Suppose that `y2` is the output value for the iteration `(n − 1)` and `y3` is the output value for the next iteration `n`. Both values depend on the unknown state `d1`. The function `Reduce` tries to eliminate the state `d1` and find the relation between the two successive output values:

```
eqns = Reduce[{y[n − 1] == y2, y[n] == y3}, {d1}];
```

The corresponding recurrence equation is

```
reducedEqn = (x*y[n − 1]^2 == 2*y[n − 1]
  − y[n]);
```

For the given initial state `y[0] = b`, the closed-form solution is computed by

```
sol = RSolve[{reducedEqn, y[0] == b}, y[n], n];
```

In this example, a solution exists

$$y[n] \to -\frac{-1 + (1 - bx)^{2^n}}{x}.$$

The derived formula approximates reciprocal in terms of the given number `x`, the initial guess `b`, and the number of iterations `n`. The error function

```
e[n]:= y[n] − 1/x;
```

simplifies to

$$-\frac{(1 - bx)^{2^n}}{x}.$$

The algorithm was originally developed for x from the range $0.5 \le x < 1$. For a given number represented by 16 bits, the initial guess can be optimized under the condition that the error should be less then 2^{-16} after 10 iterations:

```
FindRoot[e[n] + 1/2^16 == 0, {b,2}];
```

It yields `b = 1.98923`.

For the same initial guess, the error is smaller than 2^{-16} for x over the range $0.01 \le x < 1$. The above symbolic analysis extends the range of x over which the algorithm successfully computes the reciprocal.

7. Algebraic loop

Symbolic analysis of systems is inherently immune to the problem imposed by algebraic, or implicit, loops occurring when two or more blocks with direct feedthrough of their inputs form a feedback loop (Tosic *et al.* 1996). On the other hand, in numerical simulations, i.e. in the MATLAB/Simulink interpreter, algebraic loops considerably reduce the speed of a simulation and may be unsolvable.

Consider a simple feedback system and the corresponding model in MATLAB/Simulink (MATLAB 2005) as shown in figure 16. After running the simulation, an error message appears: "... Stopping simulation. There may be a singularity in the solution ...".

The same system can be symbolically solved with *Mathematica* and *SchematicSolver* as shown in figure 17. CAS successfully and accurately computes the required response. It is important to point out that the schematic specification (denoted as schematic in this example) is a symbolic object that can be used for both visualization and solving.

8. MATLAB-based approach

MATLAB is supplemented by the Symbolic Math Toolboxes which incorporate symbolic computation into the numeric environment (MATLAB 2005). The computational engine underlying the toolboxes is the kernel of Maple. Several application toolboxes utilize the MATLAB symbolic features.

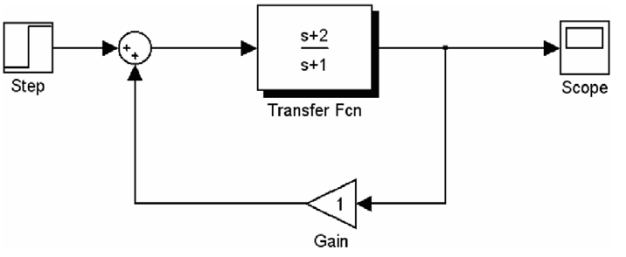

Figure 16. Simulink model of a system with an algebraic loop.

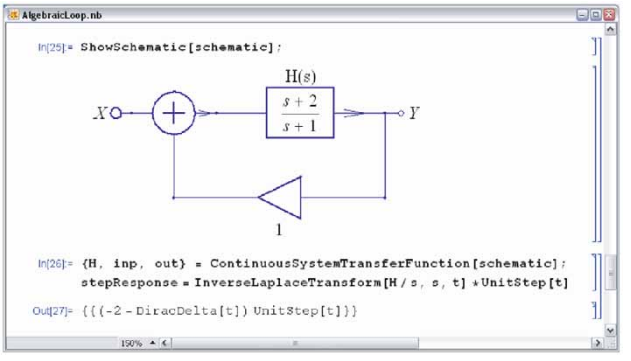

Figure 17. *SchematicSolver* model of a system with an algebraic loop.

PARADISE is a toolbox for the design and analysis of robust control systems in which parametric models can be specified using Simulink (PARADISE 2004). A closed loop symbolic representation of the control system is generated automatically using symbolic computation. The parameter space approach and various derivatives can be applied to this parametric model. Graphical user interface facilitates design and analysis. Some of the blocks supported by PARADISE are Transfer Function, State Space, Zero Pole, Integrator, Gain, Multiplexer, Demultiplexer, Sum, Selector, Source, Sink, PID Controller, Matrix Gain.

DFSYM is another toolbox based on standard MATLAB commands and the Symbolic Math Toolboxes (Lutovac and Tošić 2002). It provides symbolic computation of transfer function of linear time-invariant discrete systems. The transfer function is derived directly from the schematic created by DrawFilt (Lutovac and Tošić 2001). Each schematic is a standalone, executable, and editable MATLAB function. DFSYM finds transfer function with arbitrary real or complex coefficients. It can be used to evaluate, validate, or verify discrete systems. DFSYM handles important errors that may occur when one draws a schematic, such as unconnected (floating) component nodes or overlapping interconnection components.

It is reported in literature (Mitra 2006) that some limitations of CAS exist: ''Since the symbolic matrix analysis using the Symbolic Analysis Toolbox of MATLAB is limited to lower order matrices, operations on the transfer matrix of a structure with a large number of components very quickly becomes impractical''.

9. Conclusion

Contemporary trends to use very sophisticated algorithms combine expertise in many areas, such as dynamic system design, control engineering, and signal processing. This trend caused programming to become a task of knowledge accumulation and an efficient human-machine interface.

This paper presented a vision of the role of symbolic computations in control engineering and signal processing. It provided illustrative application examples as appropriate to linear systems, non-linear systems, algorithm development, modelling, and simulation.

System models were highlighted as visualized algorithms by means of block-diagram representations—schematics. The schematic was established as a symbolic object that contained all details for drawing, symbolic solving, simulating, and implementing the system. It was not seen as a static picture. It was shown how computer algebra systems (CAS) analysed the schematic as the symbolic object and identified symbolic system parameters to be used, for example, as arguments of the implementation function. The knowledge embedded in the schematic object was used according to the required task, such as, to generate the implementation code or to derive the transfer function.

A typical use of CAS in control system analysis was illustrated by solving a continuous-time linear MIMO system; the transfer function matrix was determined directly from the schematic. It was illustrated how to find the step response in the traditional mathematical form. An example linear causal system was designed in a straightforward manner from an analytically given step response by using symbolic computation of the Laplace transform.

Transfer function matrix of a complex MIMO discrete system was derived in terms of system parameters kept as symbols. The important power complimentary property was proved, for a class of high-speed filters, for arbitrary symbolic system parameters. The proof involved manipulation of complex expressions that was practically impossible to perform by hand.

It was demonstrated how a discrete signal could be represented as a formula, rather than as a sequence of numbers. Next, it was shown that symbolic processing of a symbolic sequence, by a system that could be described with symbolic parameters, returned a symbolic output sequence. It was presented how symbolic processing overcame the problem when the traditional approach, based on the z-transform, failed.

Multirate interpolation systems were symbolically treated in an automated way. It was shown how to generate the schematic, implementation code, and process a symbolic sequence. Moreover, it was illustrated how very complex multirate structures could be compared for functionality.

For a class of nonlinear systems, it was demonstrated how to use CAS to obtain a closed-form expression of the output signal for a known symbolic stimulus.

An algorithm was considered for implementation of a method for approximating the reciprocal by the Newton-Raphson iteration. It was shown how to optimize symbolically the algorithm parameter.

Original approach to algorithm development, that successfully overcame problems encountered in the traditional approach, was introduced: (a) approximating the reciprocal by the Newton-Raphson iteration which extended the algorithm domain, (b) design of a system which required fewer variables for numerical optimization, and (c) symbolic processing with a multirate system in the case when the z-transform could not be used.

It was presented how programs could provide knowledge about the design and employ that knowledge in the symbolic manipulation: (a) automated generation of

schematic objects and the corresponding implementation codes; and (b) computation of the variance of the uncorrelated noise due to rounding the output of the multiplier element. It was exemplified how CAS was usable on a much wider range of tasks, such as intelligence amplification: (a) derivation of the transfer function, system properties and time response, and (b) symbolic optimization.

Superiority of symbolic computation against numerical computation was shown: (a) by the example system with an algebraic loop, CAS yielded the exact solution while the traditional numeric approach failed, (b) by the closed-form solution of a nonlinear LMS subsystem, and (c) by deriving the analytic expression for the error of the Newton-Raphson iteration. Mixed symbolic-numeric techniques were used for the efficient and effective search for optimal solutions: (a) high-speed digital filter and (b) programmable logic DSP system based on minimum-Q elliptic function.

It was pointed out that those who use powerful computer algebra systems had to thoroughly understand the assumptions that underlie the software and the limits of CAS: (a) analytic computation of the z-transform might fail, (b) the closed-form response of a non-linear system might not be found, and (c) the symbolic matrix analysis with a large number of symbols might become impractical.

Benefits of symbolic methods and the role of CAS were highlighted from the viewpoint of both academia (derivation of time and frequency response, proving system properties) and industry (LMS algorithm, noise analysis, verification of realizations, design alternatives in multirate systems, CD-media controller, and pitch controller).

Acknowledgement

The authors are grateful to the anonymous reviewers for their thorough review and beneficial suggestions. We thank Ministry of Science and Environment Protection of the Republic of Serbia for partial support of our research on this topic (Project TR–6105B).

References

Application Note 4109, ''A guide to the design of current feedback control'', Fairchild Semiconductor Corporation, Available online at: www.fairchildsemi.com/an/AN/AN-4109.pdf (accessed 22 March 2006).

I. Bakshee, *Control System Professional*, Champaign: Wolfram Research, 2003.

FilterCAD (FCAD) 3.0, Milpitas, CA: Linear Technology Corporation. Available online at: www.linear.com/designtools/filtercad.jsp (accessed September 2002).

B. Lutovac and M.D. Lutovac, ''Design and VHDL description of multiplierless half-band IIR filter'', *Int. Journal of Electronics and Communications, AEÜ*, 56, pp. 348–350, 2002.

M. Lutovac and D. Tošić, DRAWFILT - Drawing filter realizations in MATLAB, PC Programs for Engineers, in *IEEE Circuits and Devices Magazine*, L. Huelsman, Ed. 17, January 2001, pp. 3–4.

M.D. Lutovac and D.V. Tošić, Symbolic computation of digital transfer function using MATLAB, in *Int. Conf. Microelectronics MIEL*, Serbia: Niš, May 2002, pp. 651–654.

M. Lutovac and D. Tošić, ''Symbolic signal processing and system analysis'', *Facta Universitatis Electronics and Energetics*, 16, pp. 423–431, 2003.

M.D. Lutovac and D.V. Tošić, ''*SchematicSolver* Version 2'', Available online at www.schematicsolver.com (accessed 12 April 2004).

M. Lutovac and D. Tosic, ''Elliptic Rational Functions'', *The Mathematica Journal, Wolfram Media*, 9, pp. 598–608, 2005a.

M.D. Lutovac and D.V. Tošić, High-Speed Filter Design using *Mathematica*, in *Int. Conf. EUROCON*, 2005b, pp. 1626–1629.

M.D. Lutovac, D.V. Tošić and B.L. Evans, *Filter Design for Signal Processing Using MATLAB and Mathematica*, Upper Saddle River: Prentice Hall, 2001.

MATLAB Version 7, MathWorks, Inc., Natick, MA, 2005.

Lj.D. Milic and M.D. Lutovac, ''High speed IIR filters for QMF banks'', in *Int. Conf. Serbia: NIS TELSIKS*, 2001, pp. 171–174.

Lj. Milić and M. Lutovac, ''Efficient multirate filtering'', in *Multirate Systems: Design and Applications*, G. Dolecek, Ed. Hershey: Idea Group Publishers, 2002, pp. 105–142.

S. Mitra, *Digital Signal processing, A Computer Based Approach*, New York: McGraw-Hill, 2006.

N.S. Nise, *Control Systems Engineering*, New York: John Wiley and Sons, 2000.

A.V. Oppenheim and S.H. Nawab, *Symbolic and Knowledge-based Signal Processing*, Englewood Cliffs: Prentice Hall, 1992.

B. Palancz, Z. Benyo and L. Kovacs, ''Control System Professional Suite'', *IEEE Control Systems Magazine*, 25, pp. 67–75, 2005.

PARADISE, Parametric Robustness Analysis and Design Interactive Software Environment, DLR, Institut für Robotik und Mechatronik. Available online at: http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-482/admin-1/ (accessed 17 June 2004).

D. Tosic, B. Kovacevic and B. Reljin, ''Symbolic Analysis of Linear Dynamic Systems'', *Control and Computers*, 24, pp. 54–59, 1996.

S. Wolfram, *The Mathematica Book*, Cambridge: Cambridge University Press, 2003. Wolfram Media.