Fast Simulation of Linear Operations: Modeling Analog and Digital Filters with Vectorized State Equations

by Granino A. Korn
ECE Department, University of Arizona
7750 South Lakeshore Road, Chelan, WA 98816
sites.google.com/site/gatmkorn gatmkorn@aol.com

Abstract

We present new compact programs for computer simulation of linear operations defined in ordinary transfer-function form. This new technique applies to analog and digital control systems and filters, which used to require a substantial amount of code for multiple integration or delay operations. The open-source Desire program for Windows or Linux formulates filter operations as short, easily readable vector expressions that compile automatically into multiple assignments. Desire's vector operations model **n** cascaded integrators, unit delays, or gamma delay lines with a single program line. Built-in FFT operations can transform the resulting time histories into frequency-response plots. We show example programs and results for digital and analog filters.

1. Introduction

We shall describe fast and compact computer programs modeling linear operations such as analog and digital filters. Sections 2 and 3, based on textbook references,[3,4] quickly review dynamic-system modeling with differential and/or difference equations in scalar and vector form. The Desire interactive-simulation program[1] translates human-readable linear or nonlinear differential equations into fast machine code and permits vector operations for vectorized Monte Carlo simulation and neural-network models.[1] This report introduces new vector operations that model cascaded integrators or delay lines for linear operators such as filters.

Sections 4 and 5 apply the new programming scheme to analog filters represented in standard transfer-function form; Sections 6 and 7 apply our vectorization technique to the cascaded delays needed to model digital filters. Section 8 deals with filter combinations; cascaded small filters let you experiment with different pole/zero combinations.

2. A Simulation Language for Interactive Dynamic-system Modeling

Desire simulation programs define dynamic-system models with assignments and/or differential equations like

[1] The Open Desire interactive-simulation package is free open-source software dowloadable from **sites.google.com/site/gatmkorn**. References 3 and 4 are textbooks describing many applications. Very fast double-precision floating-point routines solve differential and/or linear or nonlinear difference equations (up to 40,000 state variables) in scalar or vector form). There are 13 integration rules.

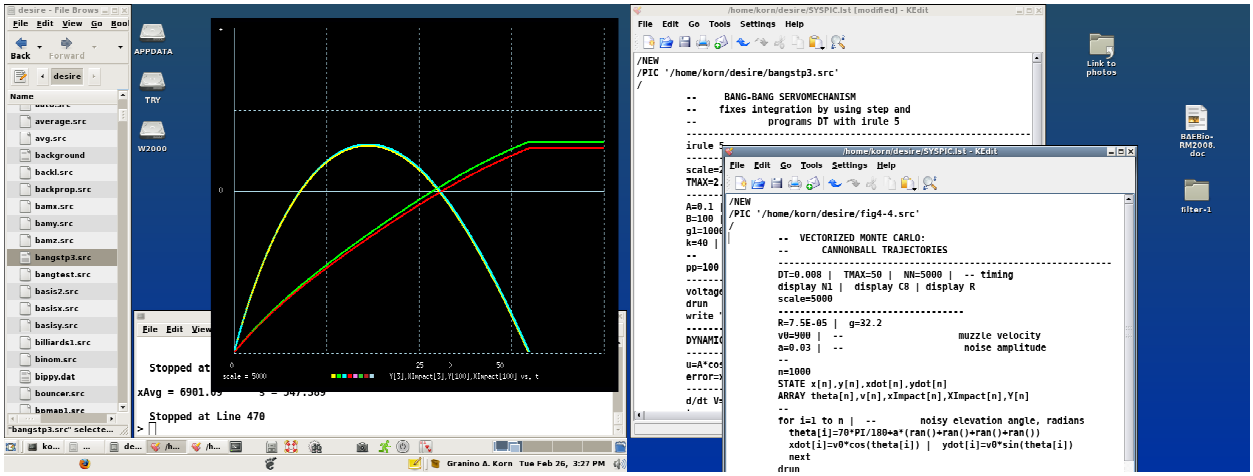

Figure 1. Dual-screen display showing Desire running with two screen-editor windows. Programs in either editor window can be run to compare models. The program works under Windows and Linux.

u = alpha* sin(w * t + beta) + c
d/dt x = xdot
d/dt xdot = - a * x – b * xdot

which are screen-edited into a ***DYNAMIC program segment*** (Fig. 1). Simulation studies are controlled by typed interactive commands and/or by an ***experiment-protocol script***. Experiment-control commands set and changes parameters and initial conditions and then call a ***simulation runs*** to exercise the dynamic-system model, as in

t0 = 0 | t = t0
a = - 5.00 | x = 17.1
drun

| is a statement delimiter. **t0** is the initial value of the ***simulation time*** **t** and usually defaults to 0. When the experiment protocol encounters the **drun** statement the DYNAMIC segment is compiled with a fast runtime compiler and runs immediately to produce time-history displays (Fig. 1). More elaborate experiment protocols can call multiple simulation runs with modified parameters and different DYNAMICsegments. [2,3]

3. Vector Operations and Delay-line Models

Desire experiment-control scripts can declare ***vectors*** like **x ≡ (x[1], x[2], …, x[n])** and ***matrices*** like **W ≡ (W[1,1], W[1,2], …, W[n, m])** with single or multiple **ARRAY** statements such as

ARRAY x[n], a[m], b[n], c[n], y[m], W[m, n], u[n], v[n], …

DYNAMIC program segments can then use the vectors and matrices in ***vector assignments***, and ***vector differential equations*** , say

Vector x = a + alpha * b * c
Vector y = tanh(W * x)
Vectr d/dt x = beta * cos (t + c)

which automatically compile into multiple scalar operations

x[i] = a[i] + alpha * b[i] * c[i] (**i** = 1, 2, …, **n**)

$$y[i] = \tanh\left(\sum_{k=1}^{m} W[i, k] * x[i]\right) \qquad (i = 1, 2, \ldots, n)$$

$$d/dt\ x[i] = alpha * \cos(t + c[i]) \qquad (i = 1, 2, \ldots, n)$$

There is no vector-loop overhead. We can also compute inner products

$$p = \sum_{k=1}^{n} u[k]\ v[k]$$

with ***inner-product assignments*** **DOT p = u * v**, again without program-loop overhead.

Given a vector **x ≡ (x[1], x[2], ..., x[n])** and an integer **k**, the ***index-shifted*** vector **x{k}** is the vector **(x[1+k], x[2+k], …, x[n+k])** where components with indices less than 1 and greater than **n** are simply set to 0. In particular, repeated execution of the assignments

Vector x = x{-1} | x[1] = u

neatly models shifting successive samples of a function **u(t)** into a simple ***tapped delay line*** with tap outputs **x[1], x[2], …, x[n].** Note that the assignment **x[1] = u** overwrites the **Vector** operation's assignment to **x[1]**.

Desire vector operations have been used for vectorized Monte Carlo simulation and for modeling neural networks, fuzzy-logic controllers, and systems involving partial differential equations.[3] We shall now apply them to create efficient models of analog and digital filters.

4. Modeling Analog Filters

An **n**th-order linear filter[1,5] with the classical transfer function[2]

$$H(s) = \{bb\ s^n + b[n]s^{n-1} + b[n-1]s^{n-2} + \ldots + b[1]\}/\{s^n + a[n]s^{n-1} + a[n-1]s^{n-2} + \ldots + a[1]\} \qquad (1)$$

can be represented by the block diagram in Fig. 2. [5] Practical filters are often realized as cascaded and/or parallel combinations of simpler filters.[1] This transfer function represents a differential-equation system, which is modeled by the easily readable Desire program

```
input = (given function of the time variable t)
output = x[n] + bb * input

d/dt x[1] = b[1] * input – a[1] * output
d/dt x[2] = x[1] + b[2] * input – a[2] * output
. . . . . .
d/dt x[n] = x[n-1] + b[n] * input – a[n] * output          (2)
```

Each of these assignments relates directly to the block diagram in Fig. 2. Execution starts with given initial conditions for each state variable **x[i]** on the right-hand side. The initial values **t0** and **x[i]** normally default to zero.

---

[2] Different texts index the coefficients **a[i]**, **b[i]**, and **bb** in different ways. Reference 5, for instance uses

b_n = **b[1]**, b_{n-1} = **b[2]**, …, b_1 = **b[n]**, b_0 = **bb** and a_n = **a[1]**, a_{n-1} = **a[2]**, **…**, a_1 = **a[n].**

The indexing chosen here is the most efficient for vector programming. If necessary, Desire experiment-protocol scripts can readily loop to relabel coefficients.

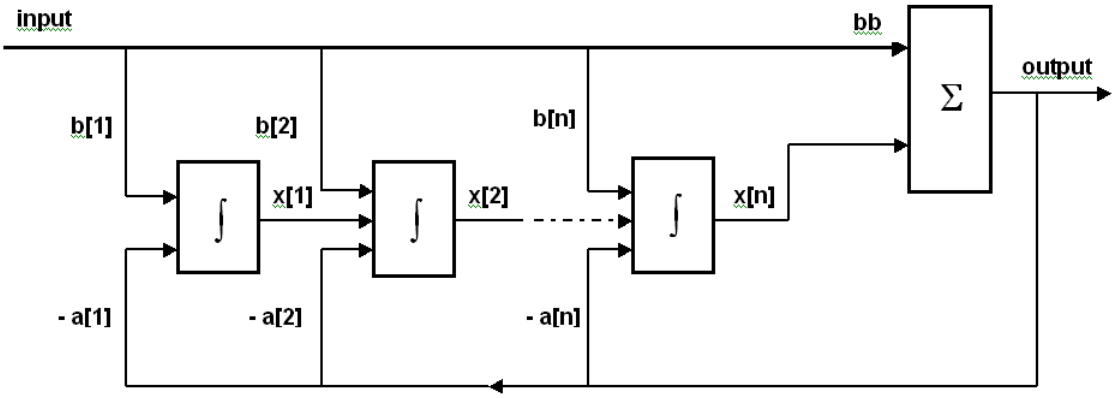

Figure 2. Block diagram of an analog filter with the transfer function

$$H(s) = \{bb\ s^n + b[n]s^{n-1} + b[n-1]s^{n-2} + \ldots + b[1]\}/\{s^n + a[n]s^{n-1} + a[n-1]s^{n-2} + \ldots + a[1]\}$$

5. Vectorization Makes the Program Much Simpler

Instead of programming **n** scalar differential equations, Desire can declare **n**-dimensional vectors **x**, **a**, and **b,**

```
STATE x[n]   |   ARRAY a[n], b[n]
```

noting that differential-equation state vectors like **x** must be declared as separate **STATE** arrays. ***We can then program our complete filter model, for any order*** **n,** ***in only three lines:***

```
input = (given function of t)
output = x[n] + bb * input
Vectr d/dt x = x{-1} + b * input – a * output                    (3)
```

These three lines compile automatically into the program (2). Note that the last statement neatly models a chain of **n** ***cascaded integrators***. There is no runtime vector-loop overhead.

To obtain the impulse response of the filter we program **input = 0** and set the initial value of **x[1]** to 1. The amplitude/phase frequency response is then obtained with Desire's built-in FFT routine. Figure 3 shows actual programs and results.

6. Modeling Digital Filters

An **n**th-order linear digital filter[1,5] with the classical z transfer function

$$H(z) = \{bb\ z^n + b[n]z^{n-1} + b[n-1]z^{n-2} + \ldots + b[1]\}/\{z^n + a[n]z^{n-1} + a[n-1]z^{n-2} + \ldots + a[1]\} \qquad (4)$$

can be represented by the block diagram in Fig. 4.[5] The time **t** is read at the sampling points **t0, t0 + COMINT, t0 + 2 COMINT, …**; the initial time **t0** usually defaults to zero.

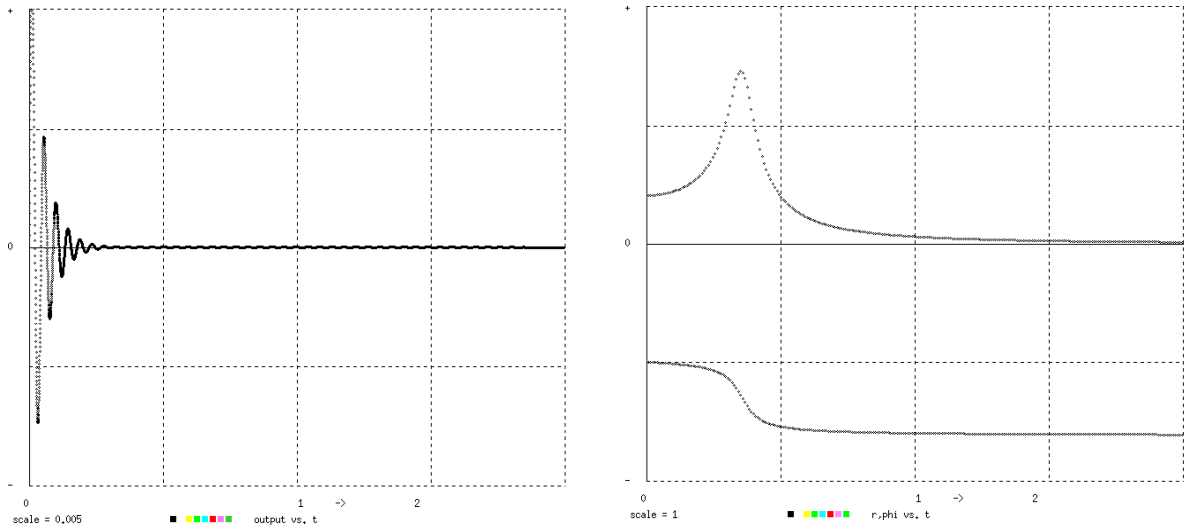

Figure 3a. Desire stripchart-type displays showing the Impulse response and amplitude/phase frequency response for a simple analog bandpass filter with the tranfer function

$$H(s) = 1/(s^2 + 40s + 20000)$$

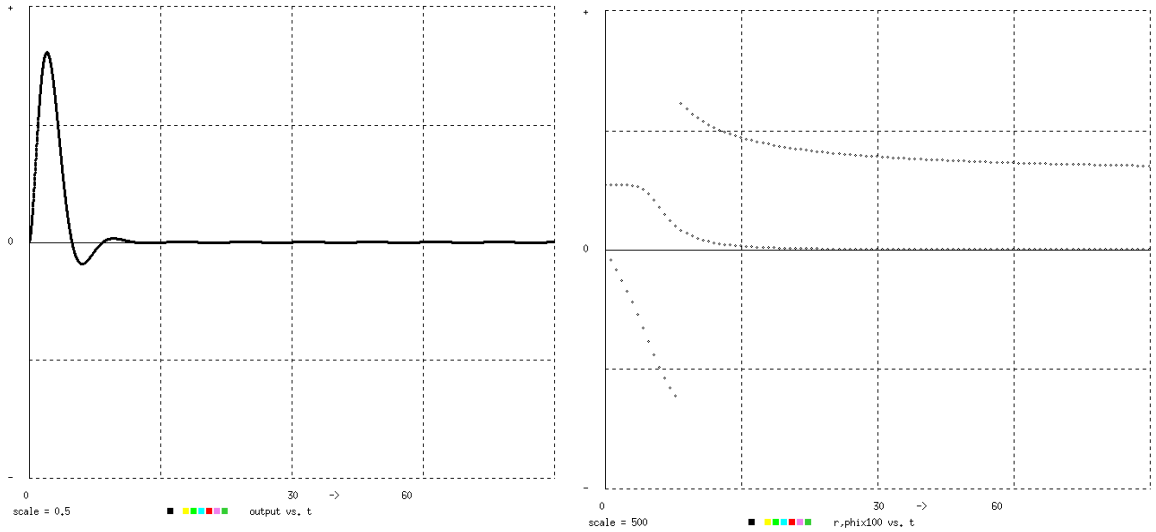

Figure 3b. Impulse response and amplitude/phase frequency response for a 3rd-order Butterworth lowpass filter with the tranfer function

$$H(s) = 1/(s^3 + 2s^2 + 2s + 1)$$

Computer simulations software solve such difference equations by ***successive substitutions***, starting with given initial conditions for each state variable **x[i]** on the right-hand side. A Desire program would represent the filter with the readable assignments

```
--      SIMPLE BUTTERWORTH LOW-PASS FILTER
--           H = 1/(s^3 + a[1]s^2 + a[2]s + a[3])
-----------------------------------------------------------------
display N1 |  display C7 |  display Q |  --     display
NN=8192 |  DT=0.001 |  TMAX=150
--
n=3 |  STATE x[n] |  ARRAY a[n],b[n]
--
--  array OUTPUT gets output samples for FFT
ARRAY OUTPUT[NN],OUTPUTy[NN]
-------------------------------------------------
--           specify the filter parameters
a[1]=1 | a[2]=2 | a[3]=2 | b[1]=1 |  -- other a[i], b[i] default to 0
bb=0 |  --       feedforward coefficient
-------------------------------------------------
t=0 |  --   (default initial t would be t=1)
x[1]=1 |  -- to get impulse response
scale=0.5 |  --       display scale
drunr  |  --          drunr resets t=0
write 'type go for FFT' |  STOP
-----------------------------------
FFT F,NN,OUTPUT,OUTPUTy
scale=100  |  NN=101
drun FFT
-----------------------------------------------------------
DYNAMIC
-----------------------------------------------------------
input=0 |  --            for impulse response
output=x[n]
Vectr d/dt x=x{-1}+b*input-a*output
dispt output
----------
store OUTPUT=output |  --   fill FFT array
--
----------------------------   AMPLITUDE/PHASE DISPLAY
  label FFT
get xx=OUTPUT |  get yy=OUTPUTy |  --    FFT arrays
r=sqrt(xx*xx+yy*yy)
phix10=10*atan2(yy,xx)
dispt r,phix10
```

Figure 3c. Complete program for the simple Butterworth filter. The experiment protocol script sets up arrays, parameters and initial conditions and calls a simulation run. The filter-output time history is displayed and also stored in the FFT array **OUTPUT**. If prompted by the user, the experiment-protocol script then computes the FFT and calls a second DYNAMIC program segment labeled **FFT** to display the frequency response.

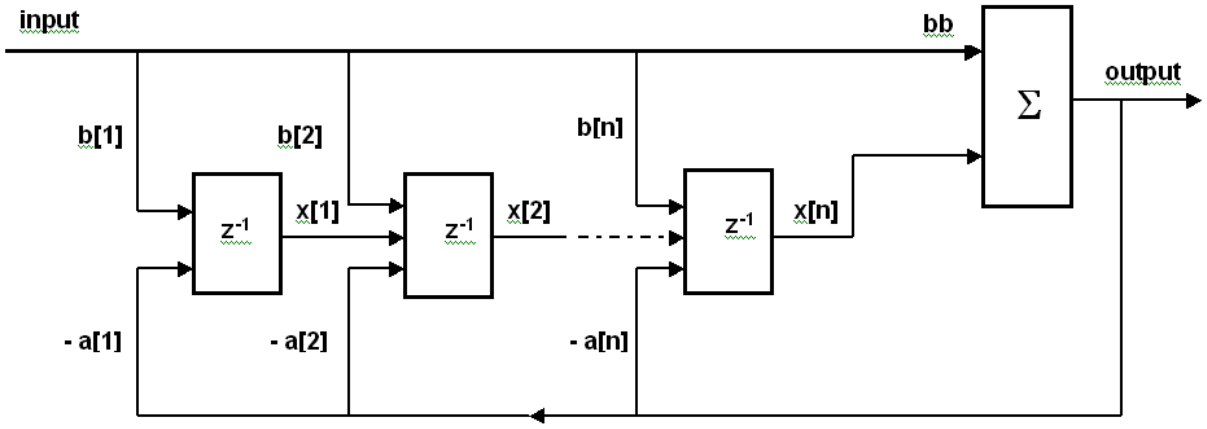

Figure 4. Block diagram of a digital filter with the z transfer function

$$H(z) = \{bb\ z^n + b[n]z^{n-1} + b[n-1]z^{n-2} + \ldots + b[1]\}/\{z^n + a[n]z^{n-1} + a[n-1]z^{n-2} + \ldots + a[1]\}$$

```
input = (given function of the time variable t)
output = x[n] + bb * input
                                   (place display or store commands here)
x[1] = b[1] input – a[1] output)
x[2] = x[1] + b[2] input – a[2] output
. . . . . .
x[n] = x[n-1] + b[n] input – a[n] output                    (4)
```

Each of these assignments again relates directly to the block diagram in Fig. 4. The program repeatedly executes these assignments in order, with **t** successively set to **t = t0, t0+COMINT, t0+2 COMINT, …** on the right-hand side of each assignment. This updates the assignment targets on the left for the next sampling time. For **t = t0**, the right-hand expressions are initialized with the given initial values of **x[1], x[2], …, x[n]**, which usually default to 0.

We remark that the order of difference-equation assignments must be carefully observed.[3] Placing display, print, or store commands (such as **dispt v, x[1]**) just ahead of the first state-variable updating assignment (**x[1] = input – output**) will ensure that state variables **x[i]** and defined variables like **input** are both sampled at the same sampling time **t**.

7. Vectorization Again Simplifies the Program

Simulating, say, a 50th-order digital filter would require programming **n** + 2 = 52 assignments (4), but there is a better way. We again formulate the state variables **x[i]**, and the filter coefficients **a[i]** and **b[i]** as **n**-dimensional ***vectors*** **x = (x[1], x[2], …, x[n]), a = (a[1], a[2], …, a[n]), b = (b[1], b[2], …, b[n])** with

```
ARRAY x[n], a[n], b[n]
```

[3] Some simulation languages order scalar differential-equation systems automatically, but difference equations must always be ordered by the programmer.

We then invoke the index-shift operation defined in Sec. 3 to replace our **n** + 2 assignments with ***only three program lines***

input = (*given function of* t)
output = x[n] + bb * input
Vector x = x{-1} + b * input – a * output (5)

Our runtime-compiled vector operations again cause no assignment-loop overhead.

Figures 5a and b show a complete program simulating a 20^{th}-order digital filter. We obtain the filter response to a unit impulse at **t** = **t0** = 0, we programmed **input = swtch(1 - t)**. The impulse response equals 1 for **t = 0, t = COMINT, …, t = (n - 1) COMINT** and then goes to 0. The Desire experiment-protocol script can also call a fast Fourier transform to produce the frequency response of the filter (Fig. 5c).

6. Combining Simple Filters

Practical filters are often realized as cascaded and/or parallel combinations of simpler filters.[1] As special cases of the program (5), we can model a ***simple recursive filter*** with

$$H(z) = 1/\{z^n + a[n]z^{n-1} + a[n-1]z^{n-2} + \ldots + a[1]\}$$

ARRAY x[n], a[n]

input = (*given function of* t)
output = x[n]
Vector x = x{-1} – a[1] * output | x[1] = input – a[1] * output (6)

Note that the final assignment to **x[1]** overwrites the **x[1]** component of the **Vector** assigment.

A ***FIR (finite-impulse-response) filter*** is modeled with

$$H(z) = \{bb\ z^n + b[n]z^{n-1} + b[n-1]z^{n-2} + \ldots + b[1]\}/z^n$$

ARRAY x[n], b[n]

input = (*given function of* t)
output = x[n] + bb * input
Vector x = x{-1} + bb * input (7)

Compact filter models like (5), (6), and (7) can be cascaded, and filter outputs can be added to represent parallel combinations of filters. Cascading several small filters is useful for creating filters with different pole/zero combinations.[1]

8. Concluding Remarks

Our new programming technique permits very convenient interactive simulation of control or communication systems that include digital or analog filters. Our programs are not meant to replace specialized programs[1] for optimal filter design

Interestingly, the new programming scheme is not restricted to linear and time-invariant filters. ***The compact programs (3) and (5) work just as well when the filter coefficients*** **a[i]**, **b[i]** ***are not constant parameters but simulation-program variables***. This is an interesting topic for future research.

```
DYNAMIC
-------------------------------------------------------------------------------------------------------
input=swtch(1-t)  | --  for impulse response;  substitute a desired input signal
output = x[n] + bb * input
Vector x=x{-1} + b * input – a * output
dispt output
```

Figure 5a. A complete DYNAMIC program segment modeling any desired digital filter with the z transfer function

$$H(z) = \{bb\ z^n + b[n]z^{n-1} + b[n-1]z^{n-1} + \ldots + b[1]\}/\{z^n + a[n]z^{n-1} + a[n-1]z^{n-2} + \ldots + a[1]\}$$

input = swtch(1 – t) is a unit impulse at **t** = **t0** = 0 and produces the filter inpulse response.

```
--                                                DIGITAL FILTER
-------------------------------------------------------------------------------
display N1  |  display C8  |  display R  |  --  display colors
NN=4096   | --                                    number of samples
--
--                  parameters for  H = (z^n + 1)/[z^n - z^(n-1)])
--
n=20   |   ARRAY x[n], a[n], b[n]
--
a[n] = -1 | b[1] = 1  |  bb = 1 | -- all other a[i], b[i] default to 0
-------------------------------------------------------------------------------
t = 0  | --                                          initial value of t
drun  | --                                        make a simulation run
```

Figure 5b. This experiment-protocol script declares vector arrays and sets the parameters **n**, **a[i]**, **b[i]**, and **bb** for the 20th-order filter with he z transfer function $H = (z^n - 1)/(z^n - z^{n-1})$.

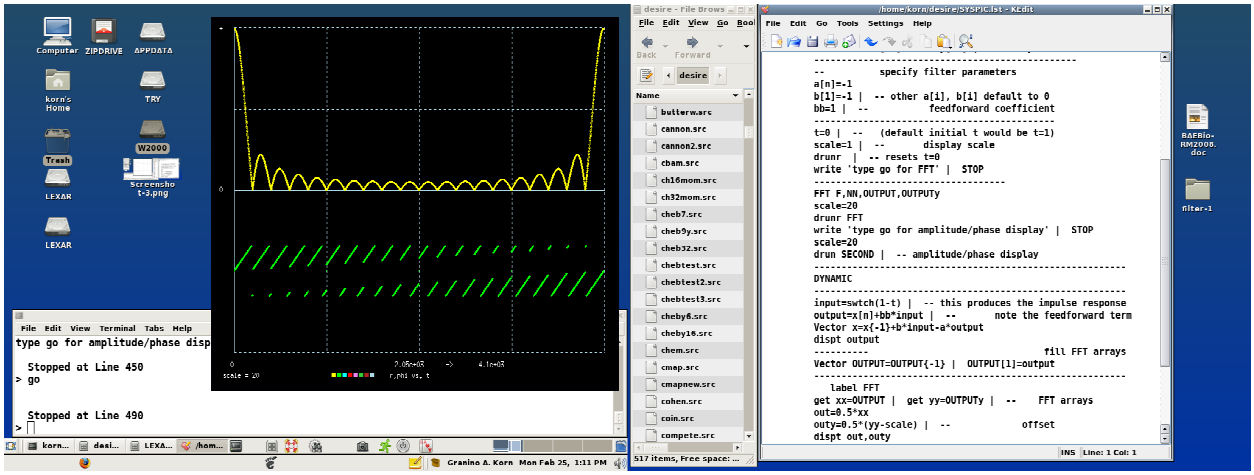

Figure 5c. Dual-screen display showing console, editor, and filemanager windows and graphs of the amplitude and phase response for a filter with the z transfer function

$$H[z] = (z^n - 1)/(z^n - z^{n-1})$$

Desire computed the frequency response as the fast Fourier transform of its impulse-response function. The order **n** of the filter is 20.

References

[1] Smith, J.: *Introduction to Digital Filters*, complete course text on the Web at *http://ccrma.stanford.edu/~jos/filters/*

[2] Korn, G.A.: *Interactive Dynamic-system Simulation under Windows*, Gordon and Breach, London, 1998.

[3] --: *Advanced Dynamic-system Simulation: Model-replication Techniques and Monte Carlo Simulation*, Wiley, Hoboken, N.J., 2007.

[4] --: *Neural Networks and Fuzzy-logic Control on Personal Computers and Workstations*, MIT Press, Cambridge, MA, 1995.

[5] Papoulis, A.: *Signal Analysis*, McGraw-Hill, New York, 1977.

[6] Principe, J., et al.: *Neural and Adaptive Systems*, Wiley, Hoboken, N.J., 2001.