WORKING NOTE AC83

Alan Creak
15 December 1993

THE INFINITELY VERSATILE ANALOGUE COMPUTER

This note embodies some ideas about what I might call a Generalised Analogue Computer, if it were not for the embarrassing acronym. I think they have been with me since I was in Derby, so they are of about 1970 vintage. I wish someone would make them work.

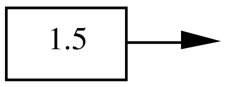
THE BASIC ANALOGUE COMPUTER.

An analogue computer is composed of an arbitrarily large set of units of various types. Each unit has one output, and zero or more inputs. A real analogue computer can be constructed in various ways, but the common feature is that the units' input and output signals are represented by some physical quantity which is continuously variable. Machines have been built in which the variable quantity is shaft rotation, shaft linear displacement, and water level, but the most common implementation uses electrical voltages. When I refer to a real analogue computer, I shall assume an electrical machine unless I explicitly state otherwise.

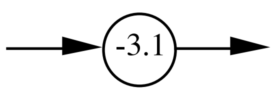
A circuit is constructed by connecting the outputs of some units to the inputs of others in such a way as to model the behaviour of some mathematical equation, or set of equations, of interest.

Four types of unit comprise the basic set. Others can be added, but the set of four units is sufficient for many purposes. All these units are linear.

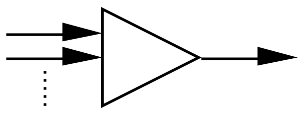
CONSTANTS have no inputs, and generate a constant output value. The constant is a parameter of the unit, and can have any value. Constants are drawn like this :



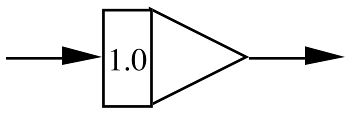
POTENTIOMETERS have one input and one output. The output is always a fixed multiple of the input. The multiplying factor is a parameter of the unit, and can have any value. (In a real electrical analogue computer, a potentiometer is implemented as a real potentiometer, so the factor is always positive and less than 1; there seems to be little point in maintaining this accidental physical feature when implementing a computer simulation.) Potentiometers are drawn like this :



ADDERS have an arbitrary number of inputs and a single output. The output is the sum of the inputs. (Real adders often provide inputs with built-in scale factors; this is purely a matter of convenience, and adds nothing to the principles involved.) Adders are drawn like this :



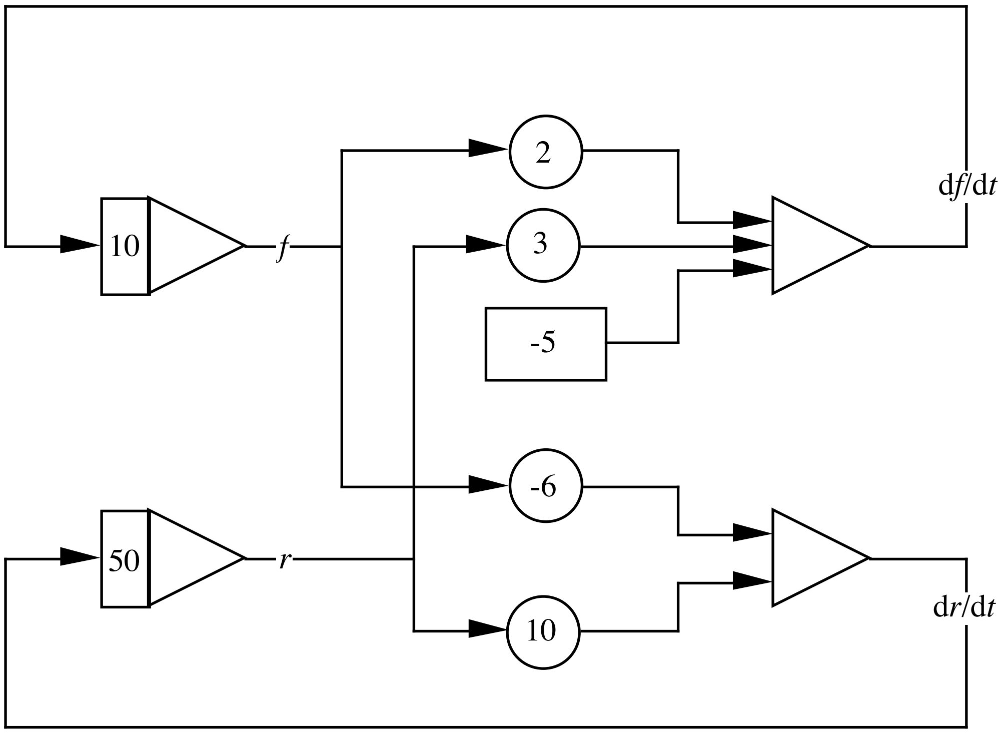
INTEGRATORS have one input and one output. The output is the integral of the input with respect to an independent variable, usually taken to be time. An integrator has one parameter, which is the initial value of its output variable. (Real integrators often have the same sort of inputs as adders; once again, this is purely for convenience, and because adders and integrators use almost identical electronics. A simulated system has an unlimited supply of simulated hardware, so we can always prefix an adder to an integrator if we so wish.) Integrators are drawn like this :



Here is a simple example to show how it all works. Suppose we have two simultaneous ordinary differential equations :

$$\mathrm{d}f/\mathrm{d}t = 3r + 2f - 5$$
$$\mathrm{d}r/\mathrm{d}t = 10r - 6f$$

(A grossly distorted version of the foxes and rabbits system - it has to be distorted, because the real problem is non-linear.) This could be simulated by this circuit, where the inital fox and rabbit populations are set to 10 and 50 respectively.



THE DIGITAL DIFFERENTIAL ANALYSER.

Analogue computers are very effective, great fun, and marvellously "intuitive" interfaces; but they are not very precise. To achieve 0.1% accuracy, it is already necessary to be very careful with the hardware, and better performance becomes exponentially more expensive. For many purposes, that's not very important, but in other cases adequate precision may be impossible to achieve. The precision problem can be solved by digital machinery, but it isn't easy to preserve the analogue computer's structure of many independent units. The obvious way is to build units containing self-contained adders, etc., and to connect them together using the same sort of wiring as with the analogue computers; but now we must use n-wire connections for n-bit precision. This is profoundly inconvenient. Alternatively, we can stick to single wires, but equip each unit with a parallel-to-serial converter and as many serial-to-parallel converters as may be required, and run the whole machine from a global clock. That's messier, and a good deal more expensive.

The digital differential analyser (DDA) retains the digital precision, the single-wire connection, and the feel of the analogue computer by an ingenious modification. Instead of conveying the values of the variables on the connecting wires, the DDA only conveys changes to the values. This can be managed in several ways. The simplest is to imagine one wire which can carry one of two distinguishable signals, one representing +1 and the other -1, or no signal. With this arrangement, a clock is no longer necessary (again like the analogue computer). The electronics can be made slightly simpler with no change in accuracy by retaining the clock and using a binary signal on the wires. "No change" is now represented by a regular alternation of *on* and *off* signals; any change of phase represents a bit of the appropriate sign.

THE TAYLOR SERIES ENGINE.

The analogue computer and DDA make an interesting pair. They both consist of a set of distinct units from which one can construct a model of a system of equations. When the model is run, it solves the equations in some sense. The units are nominally the same, the topology of the circuit is the same - but the internal structure of the units and the signals which pass along the wires are quite different. Can we get any more variety out of the same basic method ?

Yes. The Taylor series engine (TSE) again uses just the same types of unit connected in just the same way - but now the inner structure of the units is quite different once again, and the signals which pass along the wires are neither analogue nor digital electrical signals, but represent power series in the independent variable. Now each cycle of operation, instead of extending a numerical solution one step further in time, adds a further term to the end of a developing power series for each quantity represented by a line in the model. There isn't much point, so far as I can see, in even trying to build a TSE from real hardware as a set of real modules, so it remains an abstraction, but that's a detail; we are still using exactly the same high-level model as before.

PROPOSAL.

The idea is to write one programme with which we can simulate all these three sorts of machine. In all cases we set up the same circuit, and provide the same parameters. Then, depending on which mode we select, we can run an analogue computer, a DDA, or a TSE (or all three at once ?), and acquire answers of the appropriate sort.

Here are some specifications for the units, NOT guaranteed correct, but in the right sort of form.

CONSTANTS

Analogue computer : Output = parameter.

DDA : A slightly tricky one. There is no constant unit as such, because the value never changes, but it has to be worked through the circuit to set up correct initial values in other units.

TSE : Output[0] = parameter; Output[$n > 0$] = 0.

POTENTIOMETERS

Analogue computer : Output = parameter * input.

```
DDA : Accumulate constant * input;
      while accumulator > 0
            output;
            accumulator -= constant.
```

TSE : Output[n] = parameter * input[n].

ADDERS

Analogue computer : Output = $\sum$ input

```
DDA : Accumulate ∑ input;
      While accumulator > 0
            output;
            accumulator -= 1;
```

TSE : Output[n] = $\sum$ input[n]

INTEGRATORS

```
Analogue computer : State = state + input * dt;
    output = state.

DDA : accumulate input;
    state += input;
    While state > 0
        output;
        state -= constant;

TSE : Output[ $n + 1$ ] = input[ $n$ ] / $n$
```

The really pretty way to implement all this is to use a language which provides polymorphic procedures, but you can do it in Pascal with a lot of case statements.

DECORATIONS.

Input : a Macintoshy interface with which you can build up the circuit in the obvious way. (Perhaps one could steal one from the logic simulator, or something ?)

Output : draw graphs of the numerical solutions as they develop; show additional terms of the polynomial, maybe with graphs too ?

More functions : add other operators. These need a bit of care to fit them into all the different modes. Multipliers are certainly feasible. What are the conditions which must be satisfied by a candidate function ? (Something to do with always being able to calculate the next term in all cases without changes the preceding terms.)