Think Different – Think Analog

Bernd Ulmann
ulmann@vaxman.de

Kolloquium des Lehrstuhls Medientheorien
HU-Berlin, 27-MAY-2009

Outline

Overview

- The goal of this talk is to show the advantages of using (indirect and even indirect digital) analogs to solve problems.
- Therefore we first need to talk about analogs in general and indirect analogs in detail.
- After these preliminaries a couple of examples will be given showing how analogs may be applied even using conventional digital computers.
- The conclusion will sum up the main points of this talk and provide an outlook to possible future developments.

Analogs and *analog computers*

The term *analogy* is used in the following in the sense of "similarity of relation without identity" (cf. [TSE][p. 333]) with the extension of structural similarity:

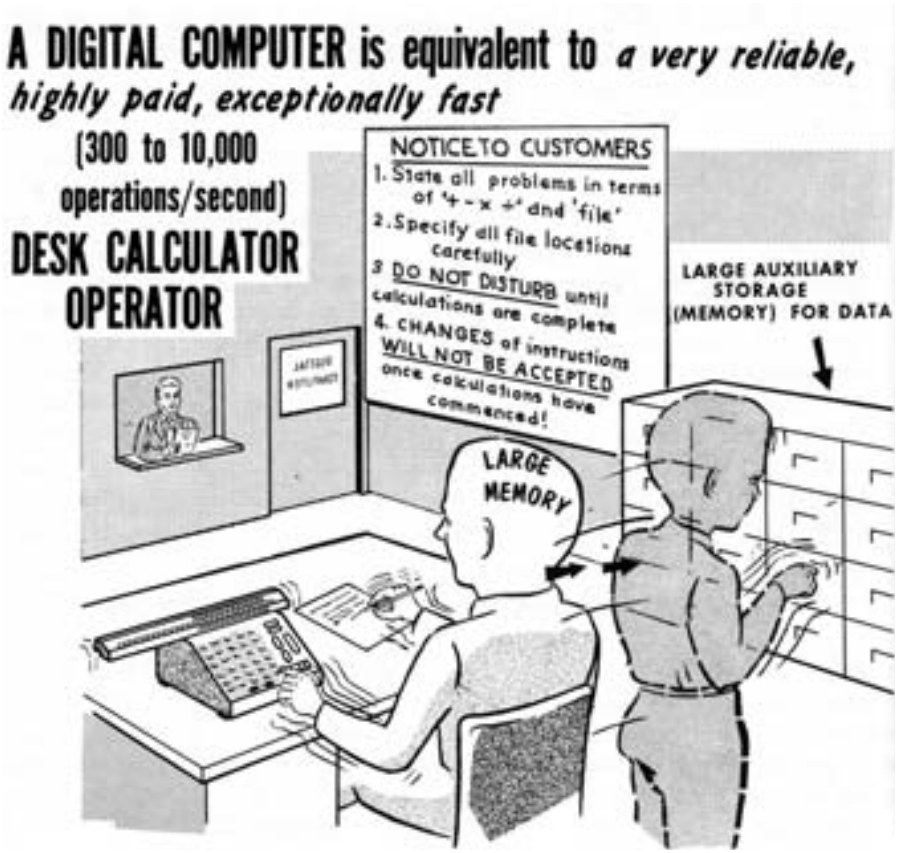
Definition of an Analog:

An *analog* is similar to a given system with respect to is relations and structure but not necessarily similar with respect to size, time etc.

So an *analog computer* is a computer that allows modelling systems by adapting to their structure and their internal relations rather than by utilizing an algorithmic approach as in conventional digital computers.

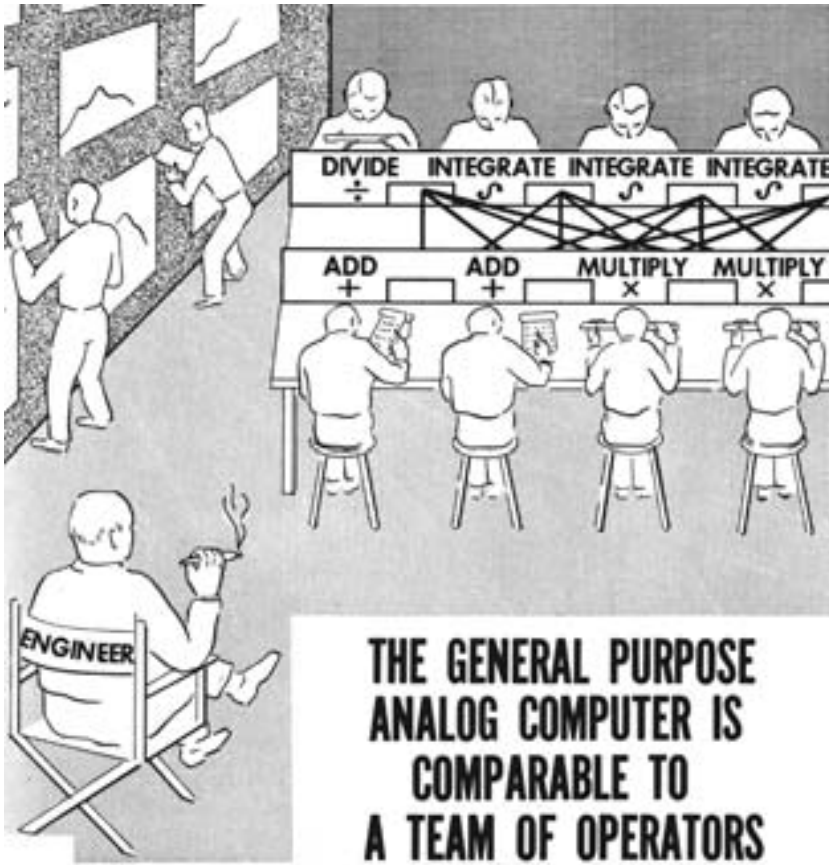
Analog vs. stored program digital computer

The following picture (cf. [TRUITT][p. 1-40]) shows the currently dominant algorithmic approach to computing:



Analog vs. stored program digital computer

The central structural equivalence typical for analog computers is shown in the following picture (cf. [TRUITT][p. 1-41]):



The two competing models

Stored program digital computer: This machine type solves Problems by a stepwise approach controlled by an algorithm normally stored in memory. The structure of the underlying machine does depend on the problem to be solved but remains fixed.

Analog computer: This machine type is programmed by changing its structure in a way to create a model, an analog, of the problem to be solved, so its structure is not fixed but changes with different problems.

Direct and indirect analogies

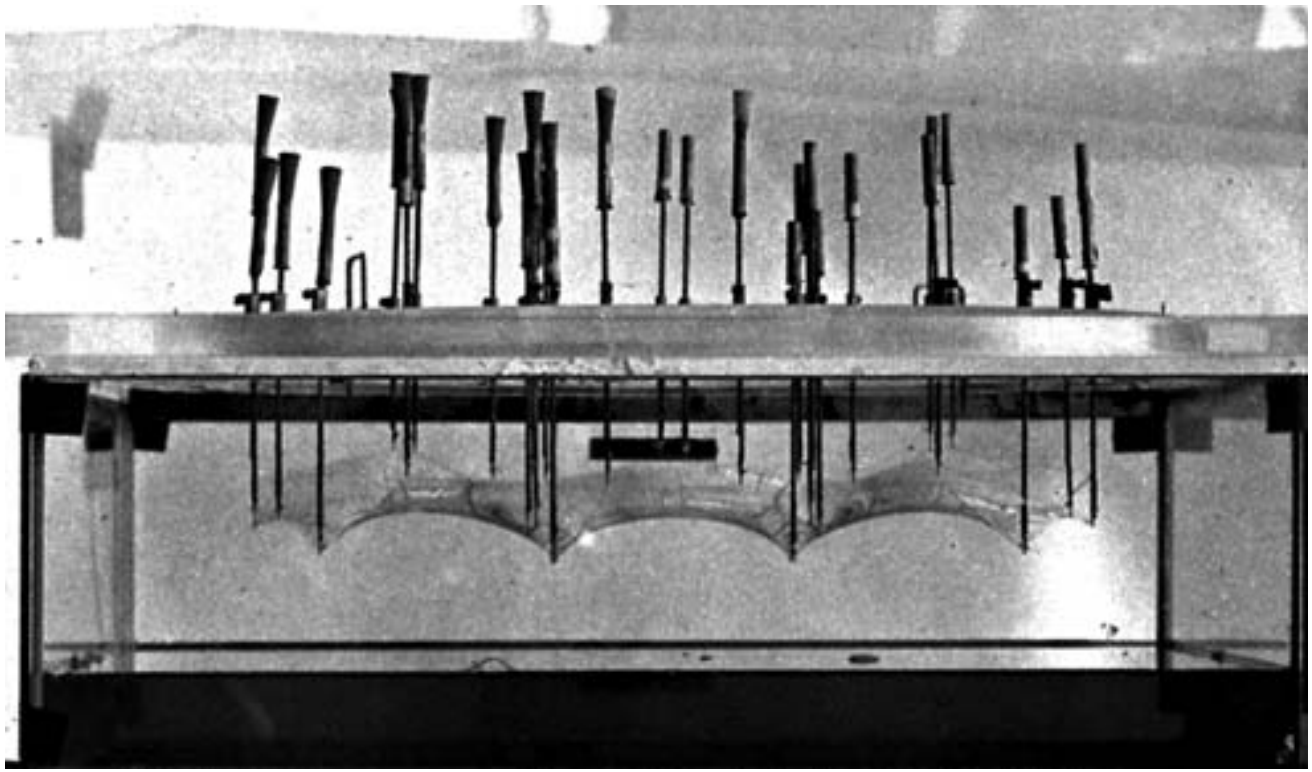
We have to distinguish between *direct* and *indirect* analogies since the focus of the following is on the latter type:

Direct analogies: Typical for this type is that a physical foundation similar to the original problem is used. Examples are soap bubble models for minimal surfaces, model rivers for hydrology, wind tunnels etc.

Indirect analogies: These analogs do not have any restrictions regarding their underlying physical representation. A mass spring damper system might be investigated using an electronic model consisting of a capacitor, a resistor and a coil.

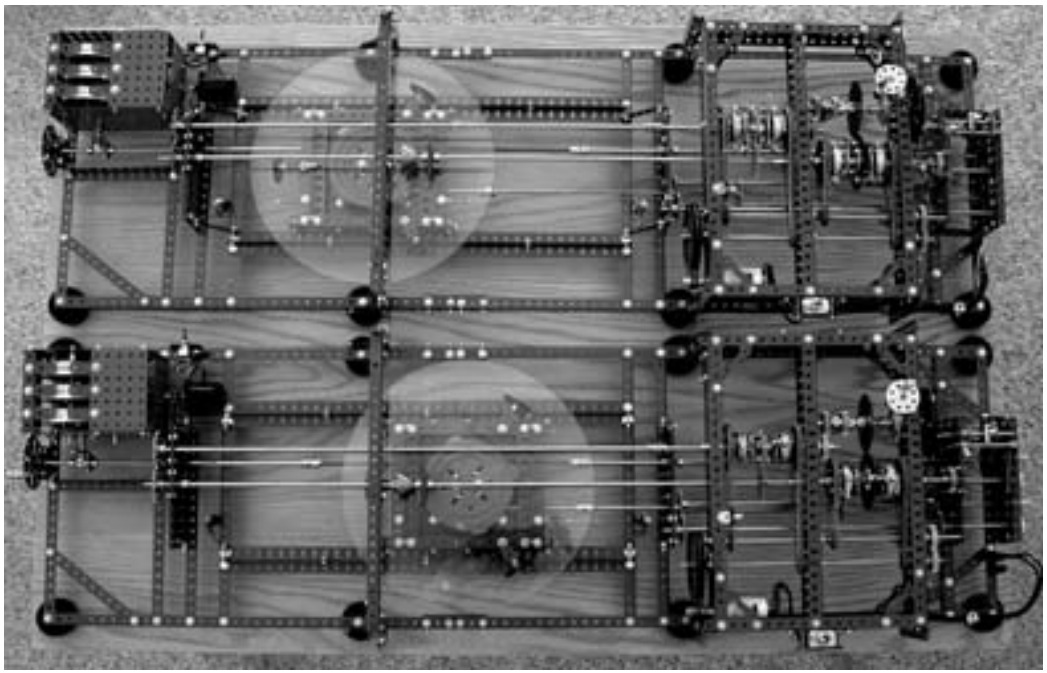
A direct analogy model

The following picture shows a typical direct analogy – the model shown was used to develop the ceiling of Munich's olympic stadium (cf. [DRESSLER][p. 52]):



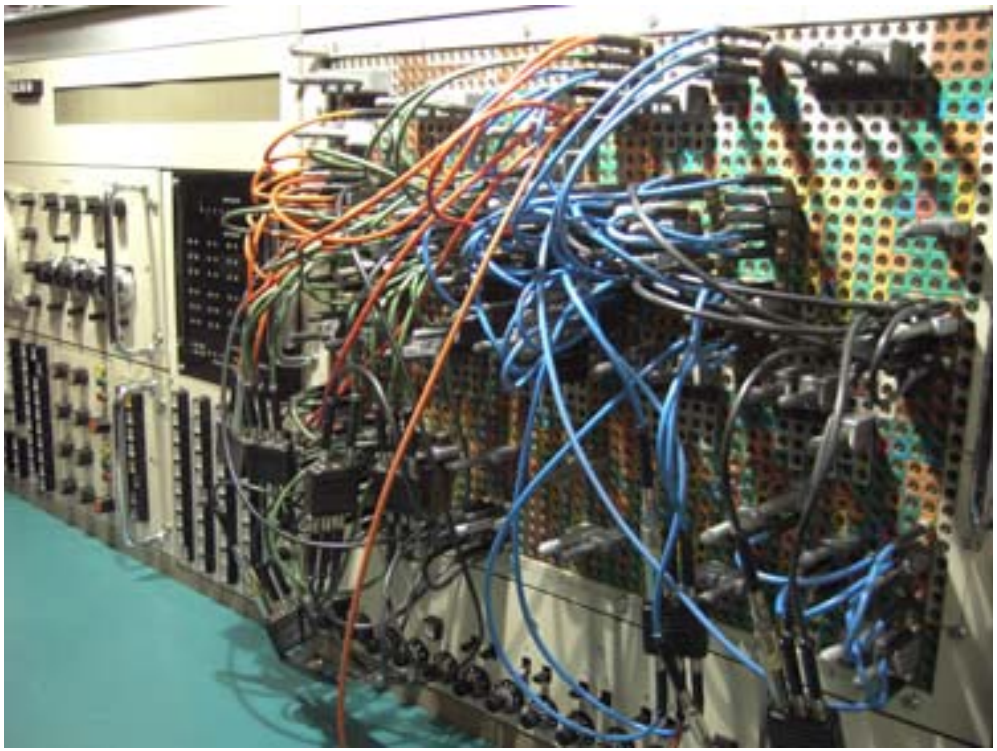
Two examples for indirect analogies

The following picture shows a mechanical analog computer (a so called *differential analyzer*) built by Tim Robinson and set up to solve a DEQ of second degree representing an oszillator (mass spring damper system or the like):



Two examples for indirect analogies

The picture below shows an analog computer (RA 770) setup to simulate the flow of air around a so called Joukowski wing – obviously there is no similarity on the phyiscal layer at all between computer and problem any more:



Different implementation variants

The following slides introduce some basic implementation variants of indirect analog computers with their particular pros and cons (mechanical implementations are not covered since these are way too clumsy for any practical application of today).
All variants have in common that the resulting system will be an indirect analog computer, i.e. a machine whose structure has to be adapted to the problems being under investigation.

Analog electronic implementations

Advantages: No need to worry about quadrature formulas/numerical stability since integration is a *natural* operation.

Disadvantages: Obsolete, hard to maintain, very limited precision (10^{-4} at maximum), limited bandwidth, expensive, values limited to ± 1 machine units, only time as free variable (no way to solve PDEs directly without tricks and approximations), generation of arbitrary functions is really hard (especially when functions of more than one variable are necesary), stability of amplifiers is hard to achieve, ...

Digital implementations

Advantages: A perfect match for modern devices like FPGAs etc. Any variable can be the free variable (so solving PDEs is not a problem any longer). Function generation is simple and mostly a question of memory and available macro cells.

Disadvantages: Problem of selecting the proper quadrature algorithm, numerical stability is an issue, necessity of choosing the *right* number representation (integer, float, double etc.), so all the problems known from numerical mathematics are back.

Software implementations

Advantages: Cheap and thus fine for evaluation/teaching purposes. Generation of arbitrary functions is easy. Could be a really great idea in conjunction with GPUs as number crunshers (cf. [CT] – this should be investigated more thoroughly in the future!).

Disadvantages: Comparably slow. Careful selection of underlying algorithms, number formats etc. is necessary (as with digital implementations, so all problems from numerical mathematics are back again).

What type of analog computers will we deal with now?

In the following only indirect analog computers will be of interest (direct analogies are left to the empiricists :-)). Such a computer is – regardless of its actual implemention – at first just a collection of some basic computing elements which can be implemented in any way (as with analog electronic circuits or digital techniques or even in software!):

- Coefficient devices
- Summers
- Integrators
- Multipliers (dividers, square root devices, . . .)
- Arbitrary function generators
- Decision elements like comparators and switches

Basic properties of an analog computer

- These basic computing elements can be used as building blocks to setup models. Obviously such an analog computer is highly parallel in its very nature and does not suffer from most of the problems known from parallel digital computers (synchronization, memory access techniques and the like).
- If a problem is larger than the analog computer at hand, there is no time/complexity-tradeoff as with digital computers but, on the other hand, the analog computer can be easily extended with additional computing elements until its size matches the size of the problem to be modeled.

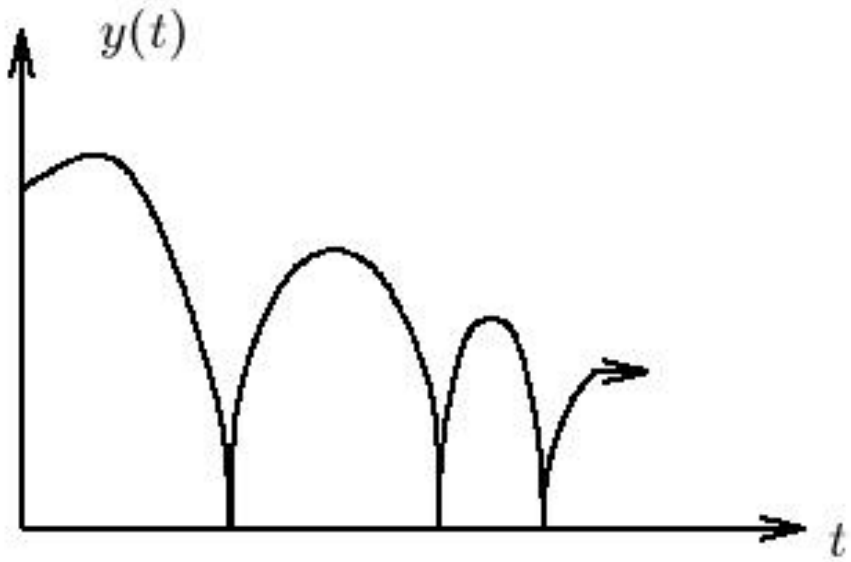
Simplifying assumptions

In the following we will make some simplifying assumptions to keep things simples. In particular these are:

- We will not care about a possible limitation of values in a model, so we will not care about scaling.
- The representation of variables is assumed to be error free (we will assume to be able to compute in $\mathbb{R}$).
- We will employ a simple graphical representation for the basic computing elements which will be mostly self explaining when we deal with digital analogs. In all other cases we will use the traditional graphical symbols typical for analog computing.

Introductory example

To show the basics of setting up an indirect analogy to model the behaviour of a more or less complex dynamic system a ball bouncing completely elastically on a rigid plate will be treated in the following.
Assuming a constant velocity of the ball in the x direction the graph showing its overall movement looks (quite) like this:



The equations of motion

How is this bouncing ball described mathematically? Its y-position at any given time is determined from its velocity by

$$y = y_0 + \int_0^T \dot{y} \, \mathrm{d}t \tag{1}$$

while its velocity is determined by

$$\dot{y} = \dot{y}_0 + \int_0^T \ddot{y} \, \mathrm{d}t. \tag{2}$$

The equations of motion

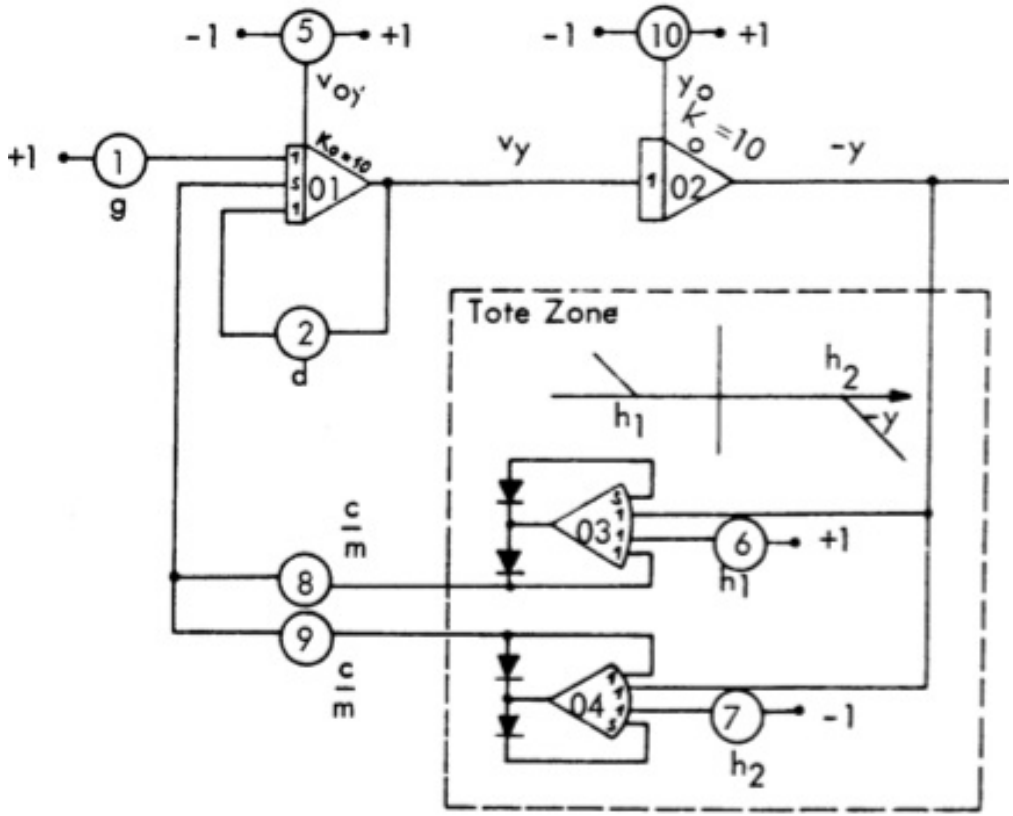
So the question remains how the ball's acceleration, $\ddot{y}$, is determined. With g being the earth's acceleration, d a damping coefficient due to air friction, m the mass of the ball and c a constant determining the elastic rebound we get

$$\ddot{y} = -g + d\dot{y} \begin{cases} +\frac{c}{m}\left(|y| + 1\right) \text{ if } y < -1 \\ \\ -\frac{c}{m}\left(y - 1\right) \text{ if } y > 1. \end{cases} \tag{3}$$

The upper term is the elastic rebound when the ball hits the floor while the lower term gets active when the ball hits the ceiling. To simplify things we will assume only a floor and no ceiling, so the equation above reduces accordingly.

Analog circuit

A complete computer setup for generating y has been given in [N.N.]:



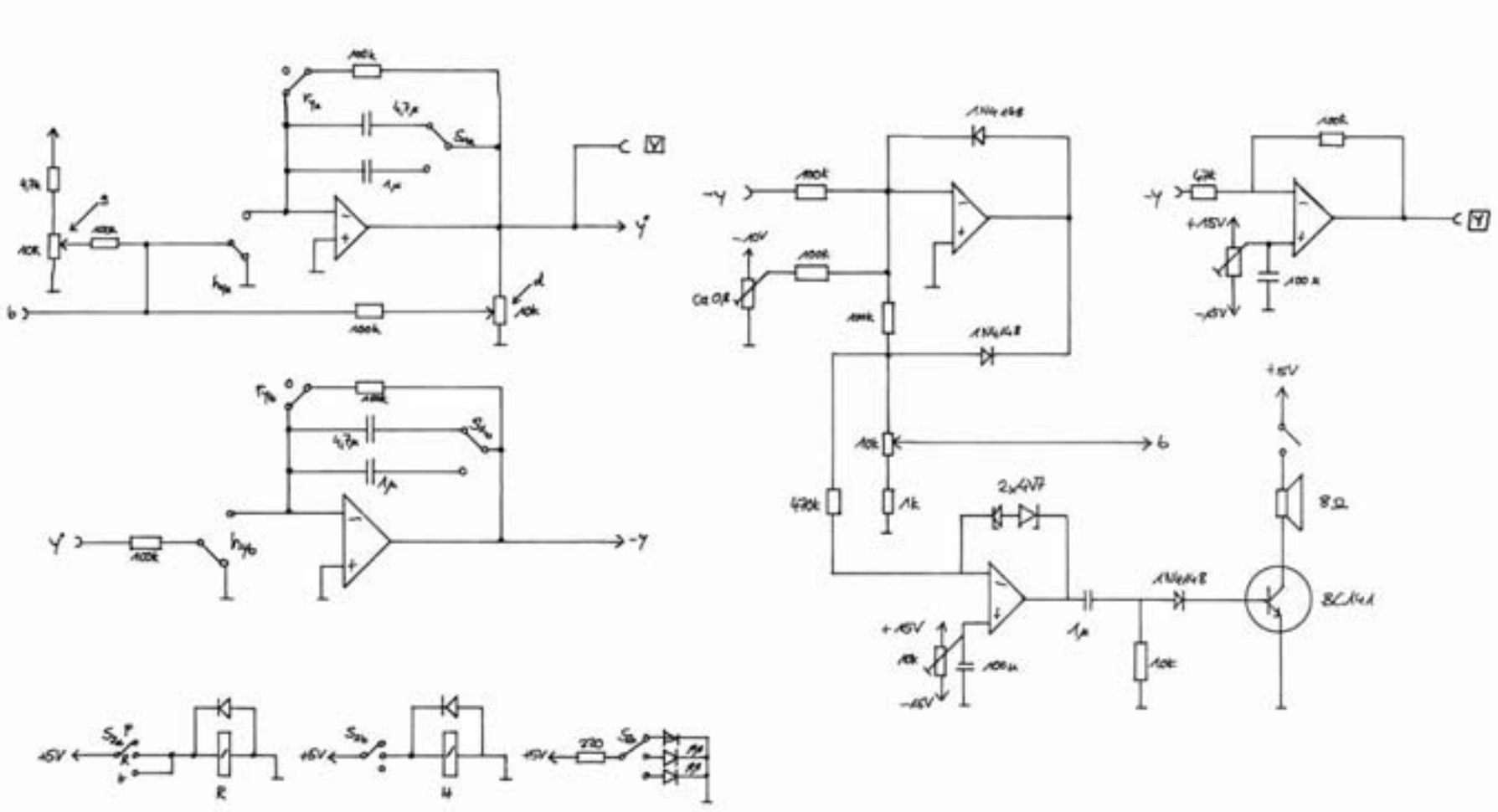
Analog circuit

The two integrators on top of this schematic are used to yield $\dot{y}$ and y from the central $\ddot{y}$ which was assumed to be known at the beginning of deriving a circuit from the equations (1) to (3). Using y which is available at the output of the second integrator the right hand side of equation (3) describing the rebounce of the ball can be generated using a trick circuit known as dead zone which is shown in the lower right half of the schematic.

A discrete implementation of the bouncing ball

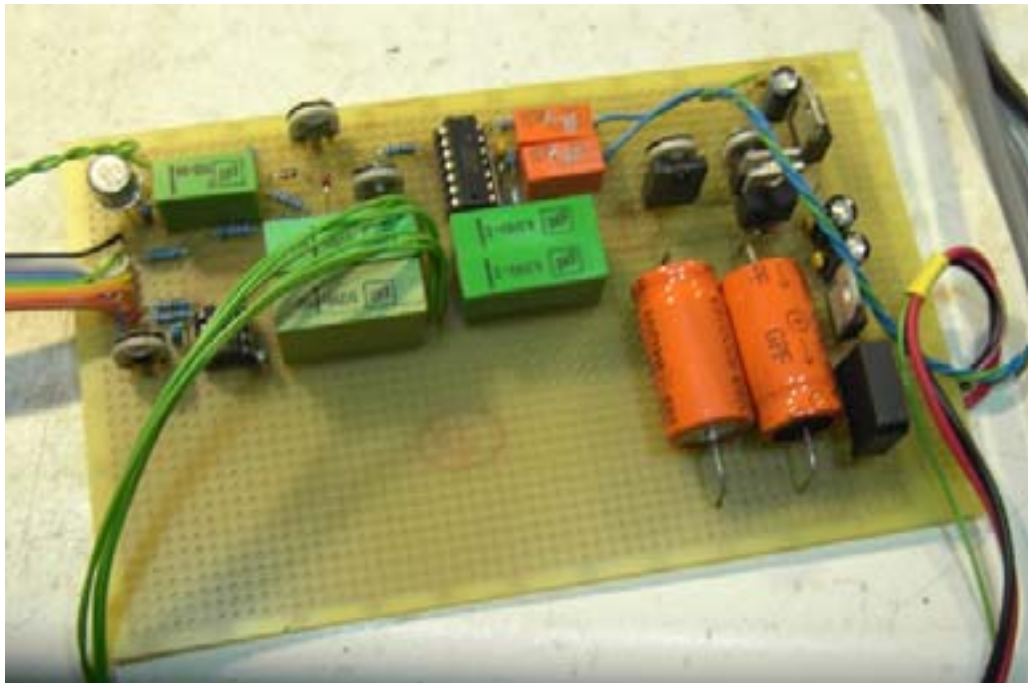
Since I have no analog electronic analog computer which is small enough to be carried on a plane and brought safely from my home to the location of this talk, a discrete implementation of this indirect analogy was done which shows the behaviour of the model very well.
Only standard circuit elements were employed and a couple of hours is enough to build a working replica of this circuit which is shown on the following slide.

The bouncing ball simulator



The bouncing ball simulator

The completed circuit looks like this (the similarity to a bouncing ball is not even marginal as was to be expected with an indirect analog :-)):



The bouncing ball simulator

The overall setup of the bouncing ball simulator looks like this using the typical output device of nearly any analog electronic analog computer, a (very simple in this case) oscilloscope:

Approaching digital computers

After this simple introductory example of how to build an analog of a given problem it will now be shown how to employ this basic idea of working with indirect analogs can be used to approach traditional stored program digital processors as well.
Therefore a simple example is used to show the power of analogies when it comes to programming digital processors especially when dynamic systems are to be controlled, simulated or otherwise treated.

A simple example – generate a sine oscillation

The simple example to be used in the following is the task of generating the output of a harmonic undamped oscillator, namely a sine oscillation.
The traditional approach for a digital computer programmer would be resorting to some (Taylor-)approximation and generating successive sine values within a loop. Although this approach works fine, it has some drawbacks:

- The structure of the original problem is completely lost – nothing in the program resembles anything close to an oscillator at all.
- Depending on the series used to generated approximated sine values the computational overhead can be quite large.

The traditional approach

A typical way to generate a sine function would be employing a simple Taylor approximation like

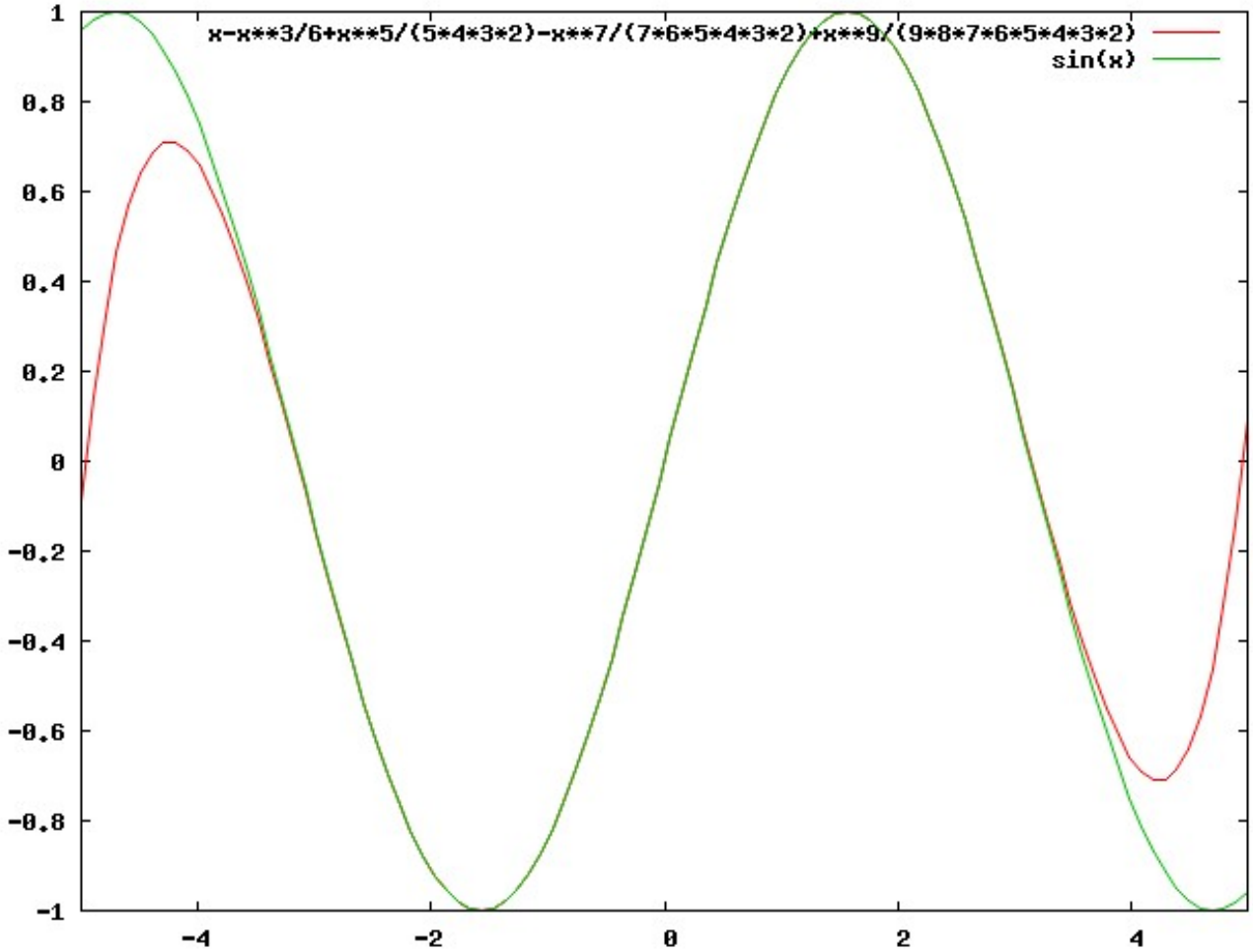
$$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$$

This is not a too bad approximation for the sine function in the interval

$$\left[-\frac{-\pi}{2} : \frac{\pi}{2} \right]$$

as the following picture shows. Bigger intervals either require more terms or additional logic for reducing input values.

The traditional approach



The traditional approach

```
#include <stdio.h>
#include <math.h>

int main()
{
  double x;
  int i;

  for (x = 0; x < 6.28; x += .01)
    printf("%12g", sin(x));

  return 0;
}
```

The traditional approach

Although this implementation seems quite straight forward the drawbacks mentioned before are evident – the main problem being the high CPU overhead necessary to evaluate the Taylor expansion over and over again, hundreds of times (even more sophisticated techniques of computing trigonometric functions like the CORDIC-algorithm – cf. [VOLDER] – require quite a lot of CPU time to evaluate).
Taking an analog point of view yields a completely different approach which not only much more closely resembles the original problem but also needs only a small percentage of the CPU power necessary for the Taylor approximation.

The analog approach

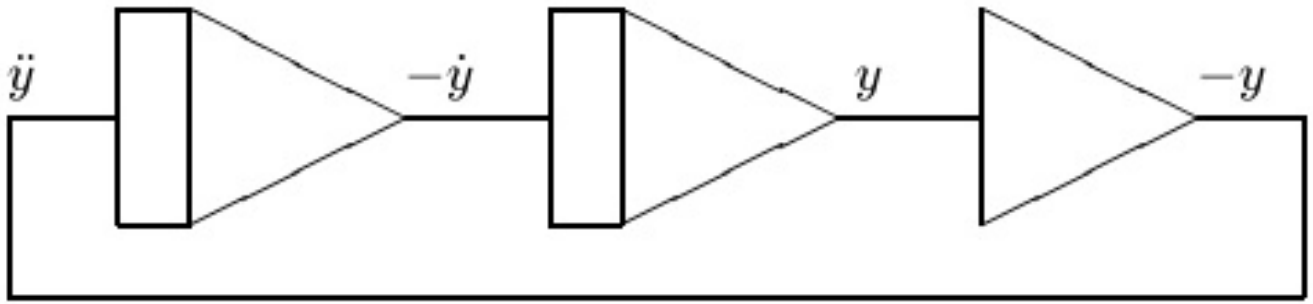
Let us now try this analog approach to the problem of the undamped oscillator's output. First of all it is noticed that $sin(t)$ is a particular solution of the following differential equation of second degree which describes an undamped oscillator:

$$\ddot{y} = -y.$$

Taking into account that we have something like an integrator at hand in an analog computer, this equation can be easily transformed into a circuit consisting of some basic computing elements, all working in parallel, as shown on the following slide:

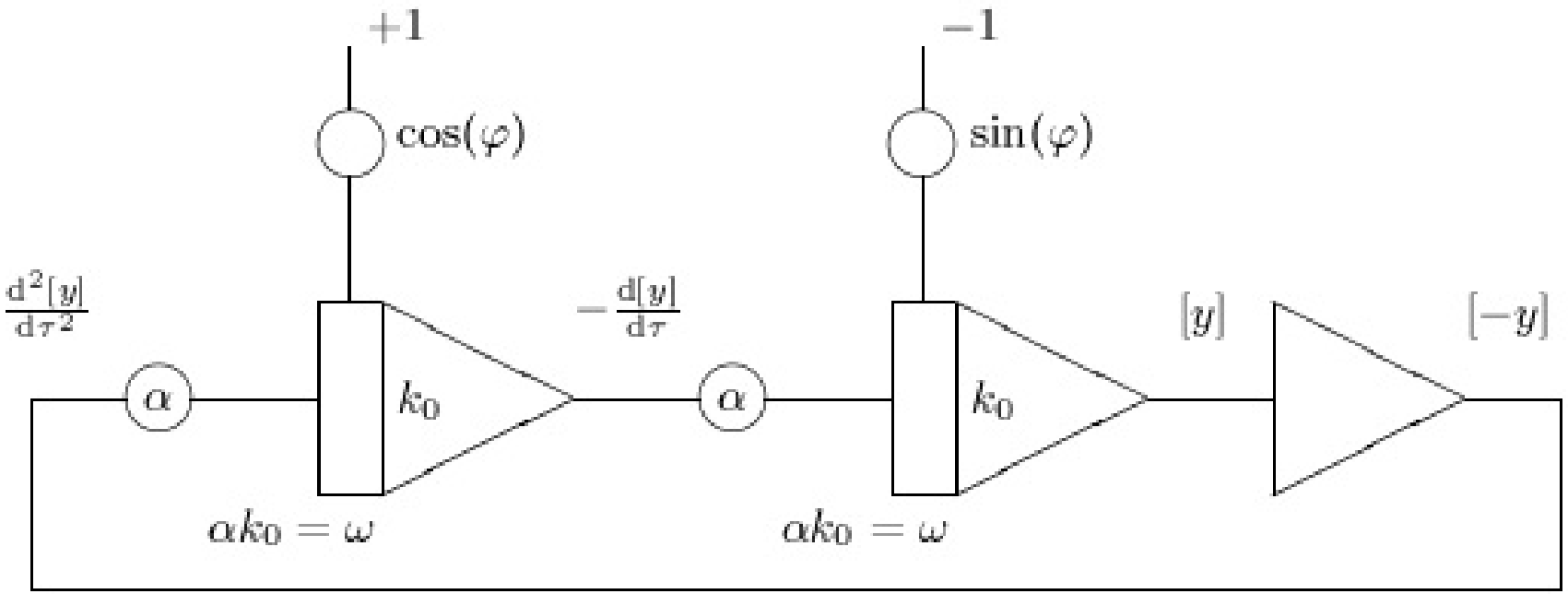
An analog electronic implementation

The basic idea to derive a computing circuit from a differential equation has been developed by Kelvin in 1876 (cf. [THOMSON]). Generally it is assumed that the highest derivative is known in advance which then serves as the starting point for generating all other lower derivatives using integrators etc. At the end of such a sequence of integrators all necessary signals for generating this initial highest derivative are available and thus a closed loop can be set up as shown below:



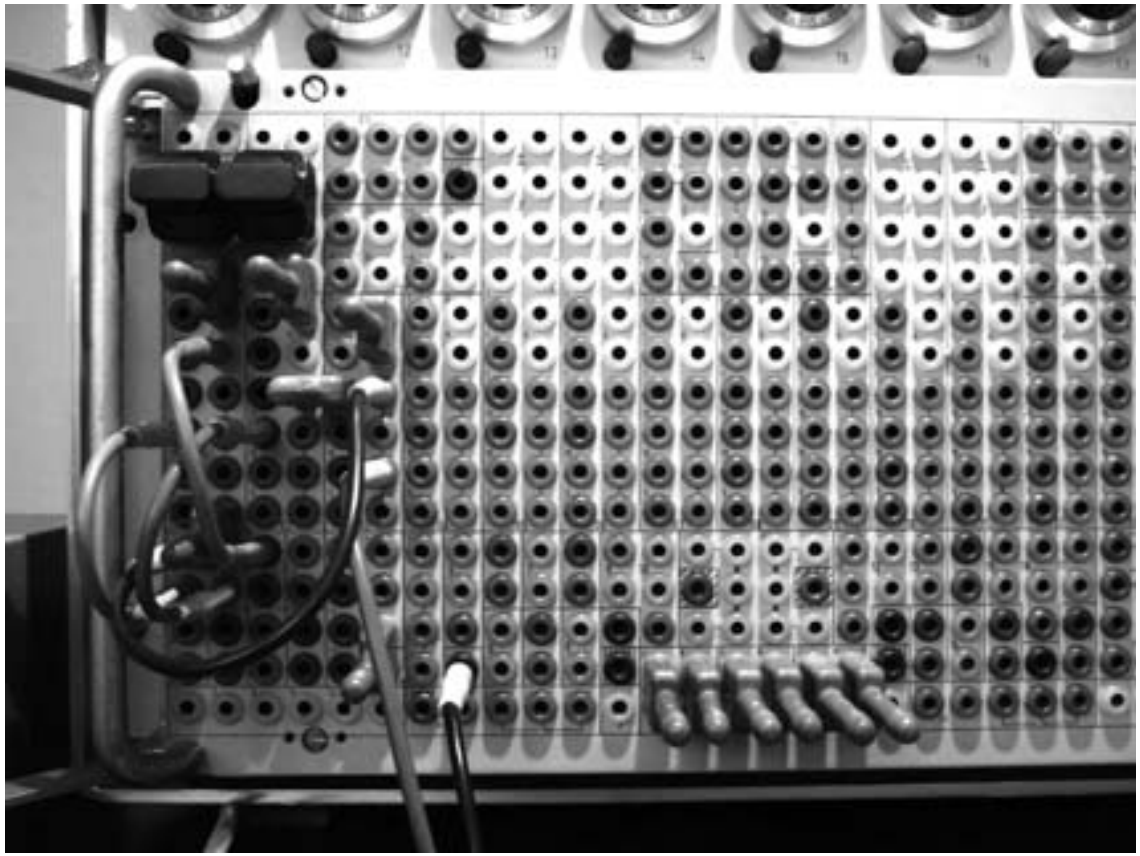
An analog electronic implementation

A complete setup for an analog electronic analog computer taking into account coefficient setting and initial values is shown here:



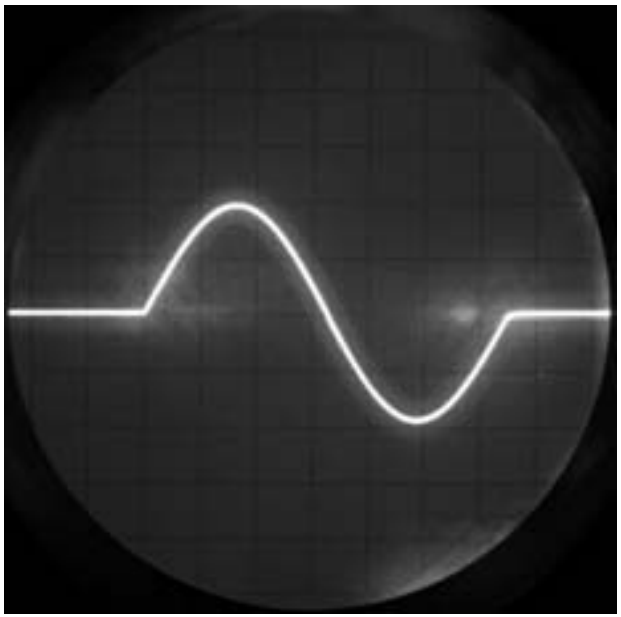
An analog electronic implementation

The actual setup on the patch panel of such an analog electronic analog computer (Telefunken RA 741) looks like this:



An analog electronic implementation

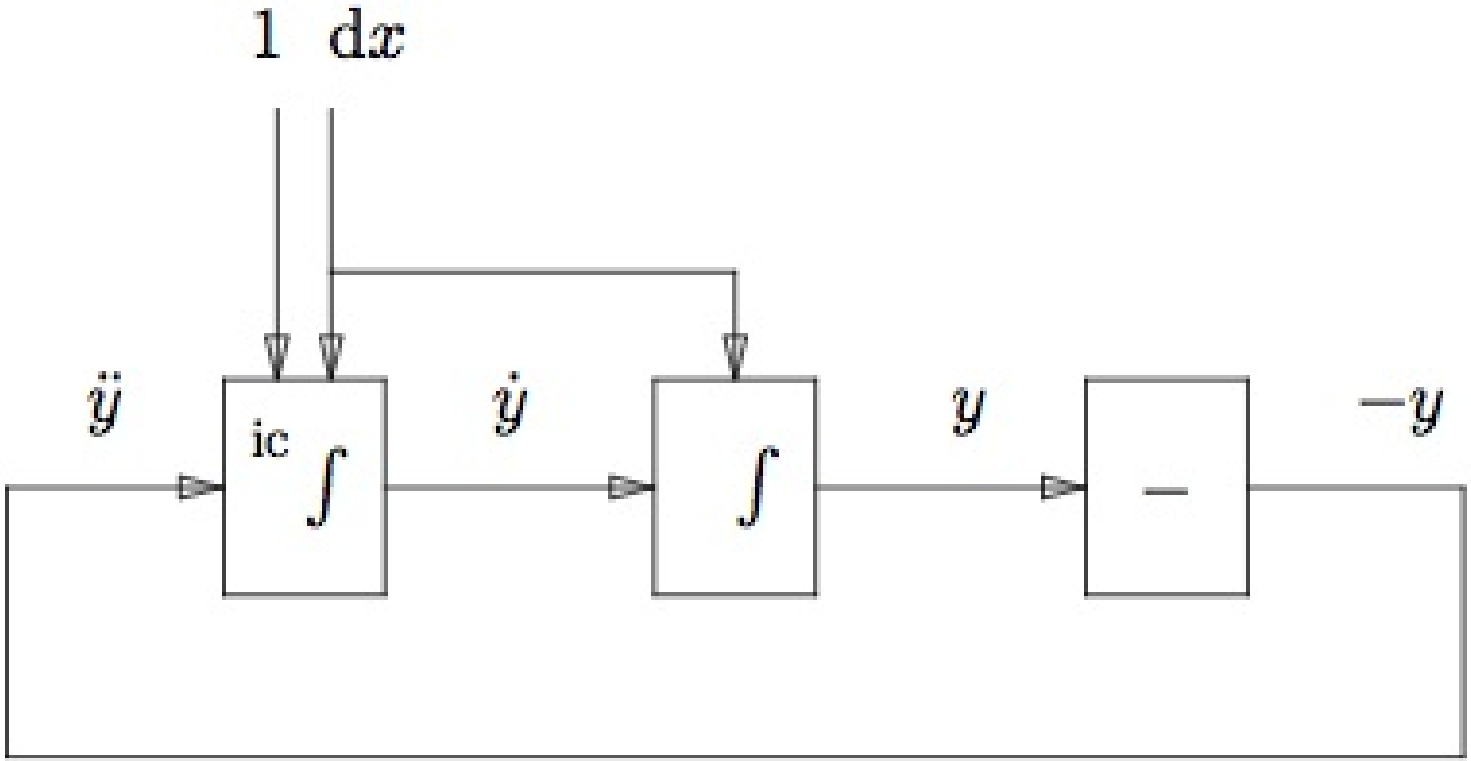
With proper setting of the coefficients and initial values and a carefully controlled computing time this analog yields the following output which is exactly what was desired in the initial problem statement:



Towards a digital implementation

Let us now see how this analog approach can be transferred to a conventional stored program digital computer.
First of all we have to abandon the idea of series approximations and concentrate on the essence of the underlying differential equation of second degree.
Redrawing the schematic used in the analog electronic implementation yields the following setup (note that now dx is an additional input to the integrators whereas analog electronic integrators can only integrate over time as the free variable):

An indirect digital analog to generate a sine signal



A software implementation

This analog of an undamped oscillator may now be used as the basis for developing a software implementation which has very low overhead.
The basic idea behind this is to model the central integrators as simple *triads* like

```
result = result + integrand * time_step;
```

Using this idea the software implementation of the sine generator looks like this:

A software implementation

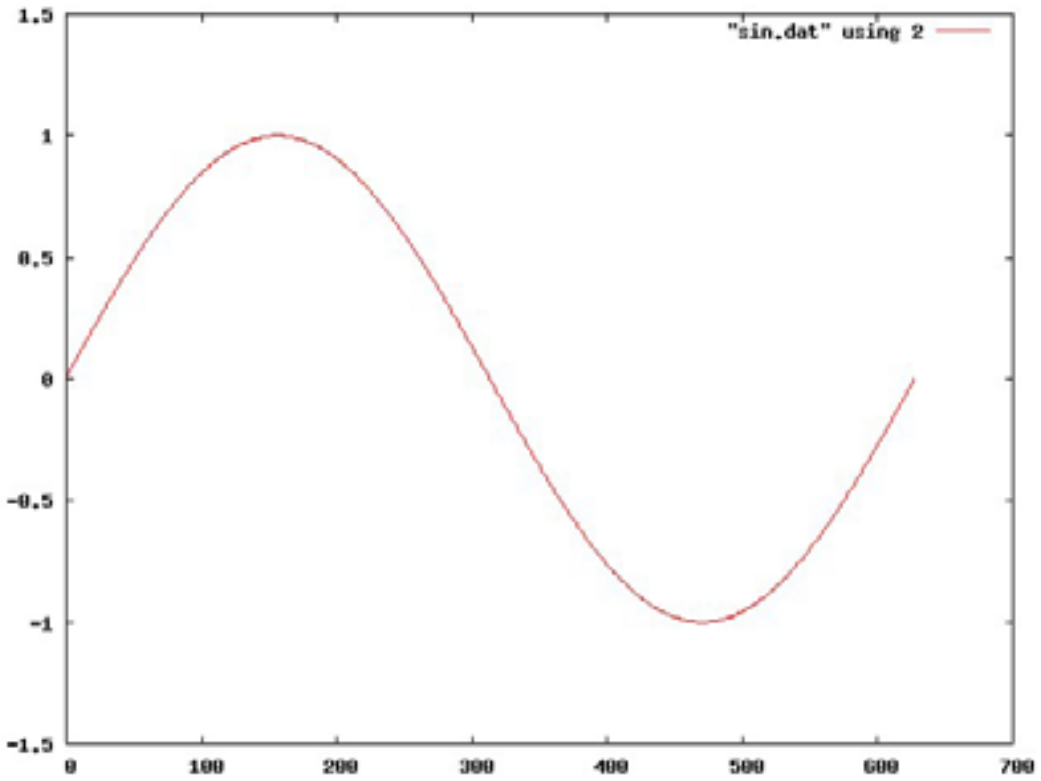
```
#include <stdio.h>
int main()
{
  double i_0 = 1, i_1 = 0, dx_0, dx_1;
  int i;

  dx_0 = dx_1 = .01;

  for (i = 0; i < 628; i++)
  {
    i_0 += -i_1 * dx_0;
    i_1 += i_0 * dx_1;
    printf("%12g", i_1);
  }
  return 0;
}
```

Result of the software implementation

The output values of this simple program yield the following graph:



Generalization of this approach

Obviously it is cumbersome to derive an algorithm for the approximation of some differential equation(s) using an analog point of view by hand.
Therefore a simple compiler was developed which accepts differential equations employing a notation of nested functions and generates C code which integrates over these underlying differential equations.

Undamped oscillator revisited

The following code is the description of the second order DEQ $\ddot{y} = -y$ from above. The `int`-function takes three parameters:

1. Integrand,
2. increment and
3. initial value.

Please note that the representation of the equation in question is already in the form yielded by Kelvin's feedback method:

sine.dda

```
dt = const(0.01)
ddy = neg(int(int(ddy, dt, 1), dt, 0))
```

dda2c.pl

Using the Perl written compiler `dda2c.pl` this source file is then transformed into C source code which implements the simple integration scheme employed before.
Issuing the command

```
dda2c.pl sine.dda 628 y
```

yields a C source file named `sine_dda.c` which will iterate over the differential equation described and print out the value of the internal variable y after each iteration.
Plotting those values yields the desired sine function which was now generated using only its describing differential equation instead of a clumsy series approximation which is just too sophisticated when an oscillation is to be generated instead of only few distinct trigonometric values.

The Volterra-Lotka-DEQs

In the following a more complex example in form of the well known Volterra-Lotka differential equation is being treated first with a historic analog electronic analog computer and then with the simple compiler `dda2c.pl`.
In 1925 and 1926 Alfred James Lotka and Vito Volterra both derived a system of two coupled differential equations describing the behavour of a dynamic system consisting of prey (rabbits) and predators (lynxes).

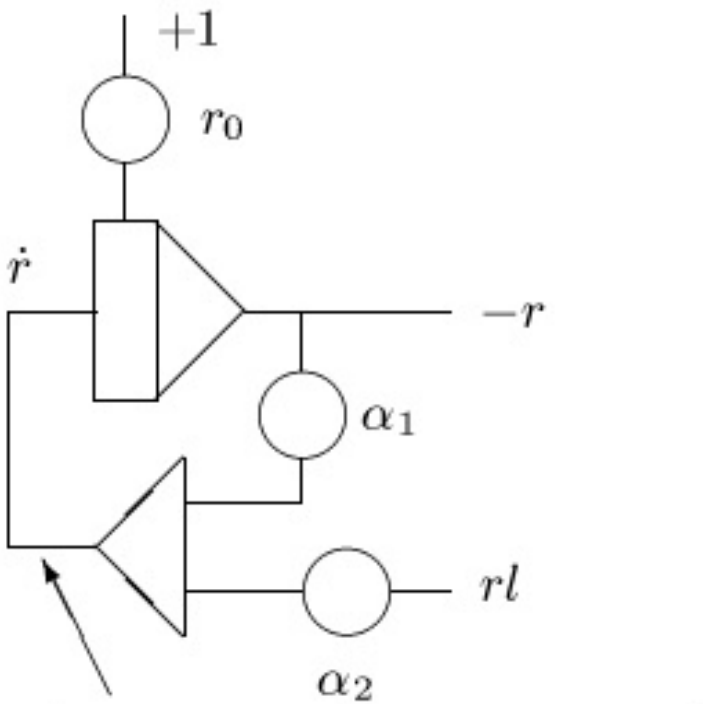
The Volterra-Lotka-DEQs

With r representing the number of rabbits, l being the number of lynxes and with α_1 the rate of birth for rabbits, α_2 the rate of rabbits killed by lynxes, β_1 the rate of death for lynxes and, finally, β_2 the increase of the lynx population due to food, the Volterra Lotka DEQs look like this:

$$\dot{r} = \alpha_1 r - \alpha_2 rl$$
$$\dot{l} = -\beta_1 l + \beta_2 rl.$$

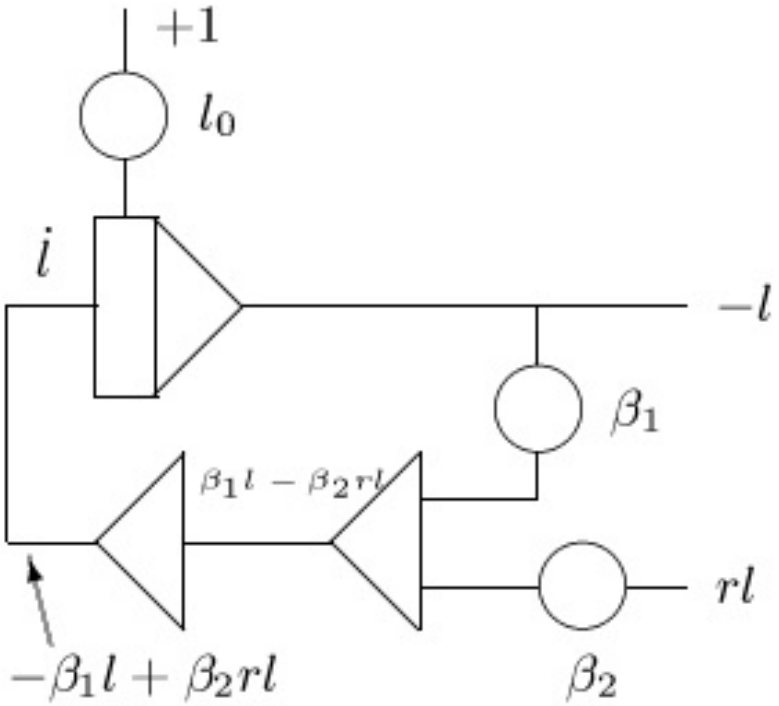
Deriving an indirect analog

The first equation, namely $\dot{r} = \alpha_1 r - \alpha_2 rl$, yields the following computer circuit:



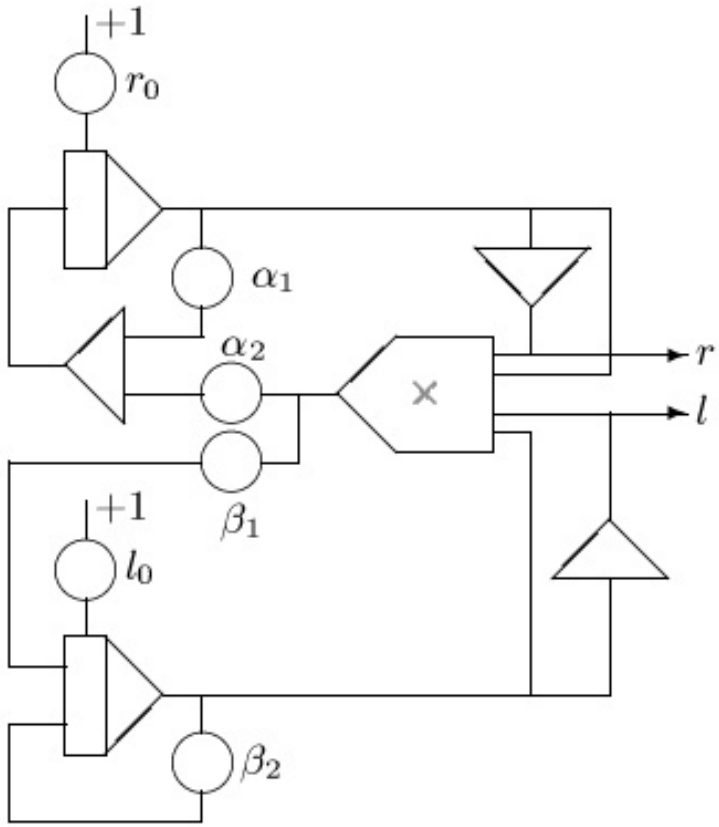
Deriving an indirect analog

The second equation, $\dot{l} = -\beta_1 l + \beta_2 rl$, yields this computer circuit:



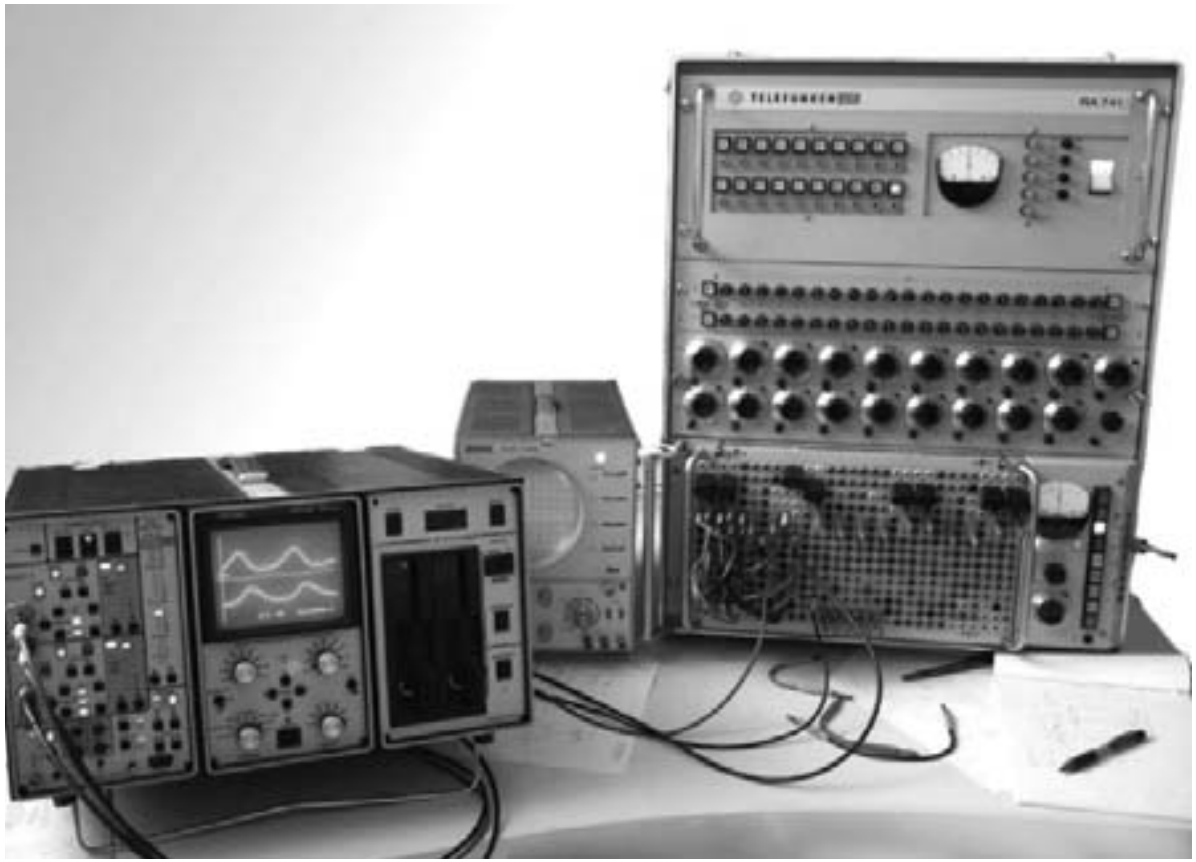
Deriving an indirect analog

Combining both partial computer circuits yields the following overall schematic representing the two coupled DEQs:



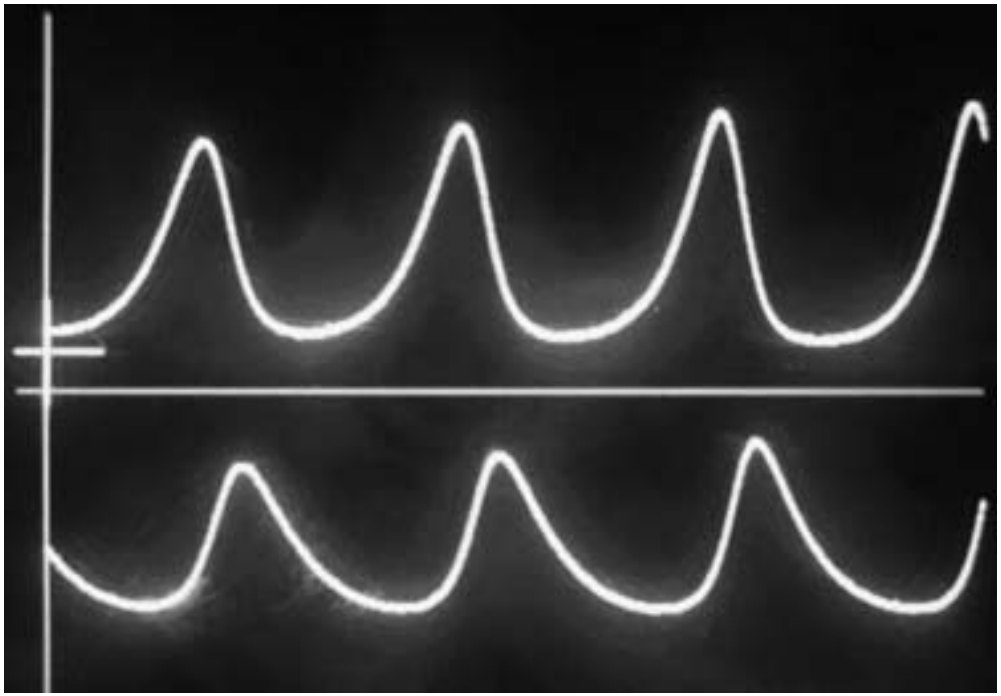
An analog electronic analog computer setup

The complete setup of this computer circuit on a historic analog electronic analog computer (Telefunken RA 741) looks like this:



Result of the analog electronic simulation

Using a proper parameterset for α_1, α_2, β_1 and β_2 the following result was obtained using this analog electronic approach to the Volterra-Lotka differential equations:



The digital approach

Using the already mentioned simple compiler `dda2c.pl` we will now tackle the Volterra-Lotka differential equations using the analog computer circuit shown above on a stored program digital processor.
Therefore we have to describe the analog computer circuit in the same way as it was done before in the oscillator example which yields the following source code:

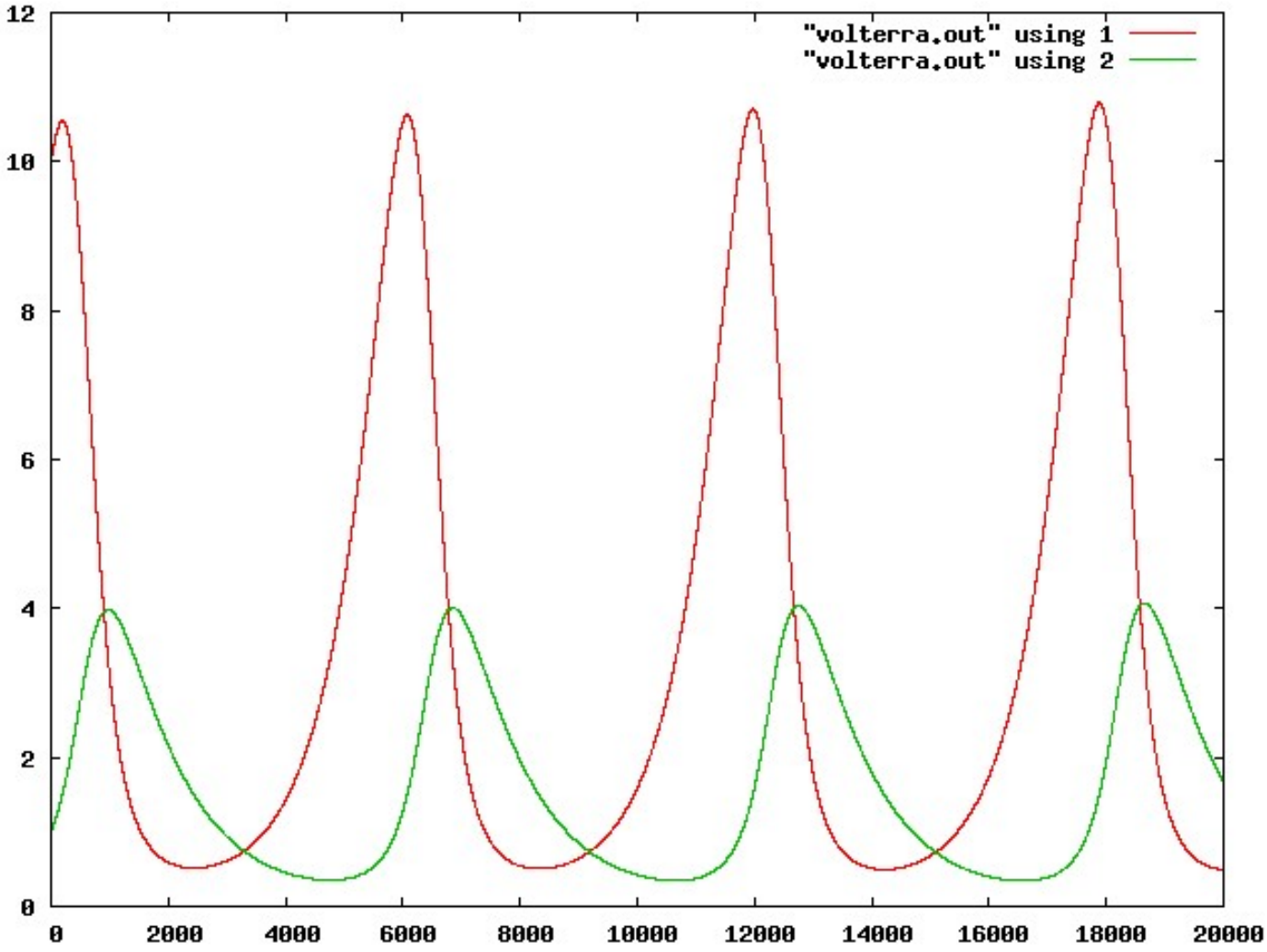
The digital approach

volterra.dda

```
dt = const(.01)
a1 = const(.2)
a2 = const(-.1)
b1 = const(-.2)
b2 = const(.03)
#
r = int(sum(mult(r, a1), mult(r, l, a2)), dt, 10)
l = int(sum(mult(l, b1), mult(r, l, b2)), dt, 1)
```

Feeding this source code into `dda2c.pl` yields a C program which readily simulates the predator-prey-system described by the Volterra-Lotka equations. The output of a typical simulation run is shown on the following slide:

Result of the digital simulation



Extending this approach

Obviously the use of analogs for the investigation of dynamic systems is very fertile since it's focus lies on the problem and not on the machine and its particular idiosyncrasies being used to solve it.
Using conventional stored program digital computer this is about the only advantage of this approach, but one can do way better.
The real power of this approach will become apparent when it will be combined with modern programmable logic circuits like FPGAs (short for *F*ield *P*rogrammable *G*ate *A*rrays). Using elements like this the old dream of a usable digital analog computer will come true (this dream dates back to the days of MADDIDA and TRICE – cf. [REED] and [AMELING] – and was spoiled by the crude digital technology available in the 1950s and 1960s).

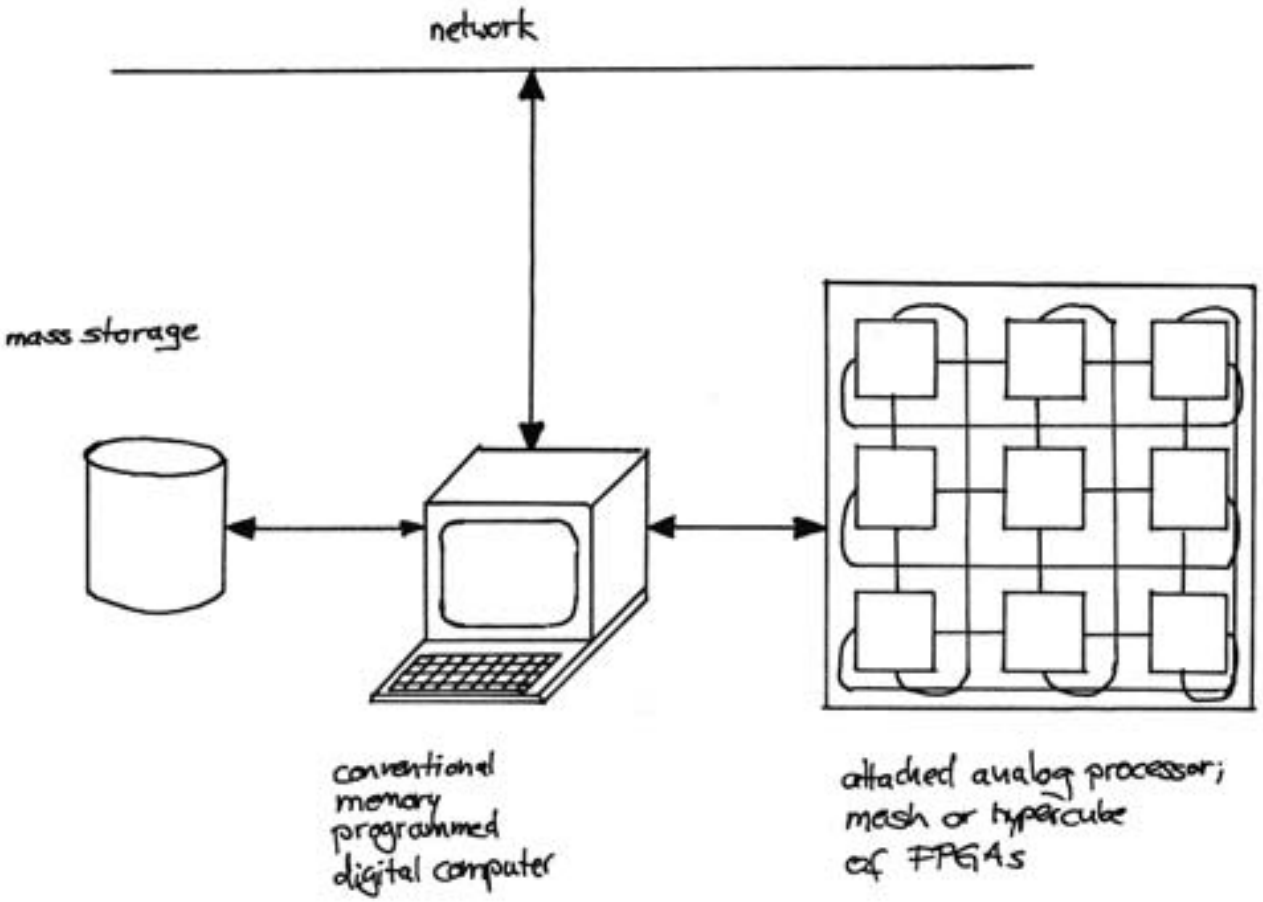
A modern digital analog computer

The following picture shows a hypothetical system being completely digital in its implementation but being taylored to the simulation of dynamic systems using the traditional yet powerful analog approach described above.
On the left side a conventional digital stored program processor with attached storage and network connection is shown which serves as the host system for the specialized attached processor shown on the right.
This attached processor will consist of a mesh or maybe better a hypercube of FPGAs which will be changed in their internal structure from one problem to the next thus resembling the structure of the problem under investigation.

A modern digital analog computer

To explore a given dynamic system it will be described in terms of an analog circuit as with historic analog electronic computers. This particular analog computer setup will then be used as the source code by a specialized compiler which will generate structural data for the FPGAs of the attached processor.
After loading the attached processor with its configuration and structural data it will resemble a digital analog of the initial problem which can then be simulated at very high speed while the host processor can supply input data, gather output data or even be part of the simulation by generating complex functions etc.

A modern digital analog computer



Another modern digital analog computer

Recent develops show that is might even be worthwhile to try a much cheaper approach to building a purely digital analog computer using state of the art GPUs as number crunshers for the analog portion of a digital-digital hybrid computer.
These GPUs would, of course, not be able to exhibit the same amount of fine grain parallelism as might be possible using FPGAs but considering their comparably low price this approach might result on a truly low-budget all-digital hybrid supercomputer.

Advantages of the analog approach

It may be concluded that the analog approach is very powerful in terms of building models for systems described by differential equations or sets thereof.
Using state of the art technology like FPGAs or modern GPUs it seems be possible to create an all-digital hybrid supercomputer system consisting of a conventional stored program digital processor with an attached processor consisting of FPGAs or GPUs which will take care of the dynamic part of the system simulation.
Due to the inherently high parallelism of analog computers most of the problems known from massively parallel conventional digital processors can be avoided thus putting a high portion of the computing power of the attached processor to useful simulation work.

The heritage of the past is the seed of the future.

Bibliography

- [AMELING] W. Ameling, *Aufbau und Arbeitsweise des Hybrid-Rechners TRICE*, in *Elektronische Rechenanlagen*, 5 (1963), Heft 1, pp. 28–41
- [CT] Manfred Bertuch, *Parallel-Werkzeuge*, c't 11 2009, pp. 142
- [DRESSLER] Fritz Dressler, *Das Dach*, in *hobby – Das Magazin der Technik*, Nr. 8/72, pp. 50
- [N.N.] N. N., *Demonstrationsbeispiel Nr. 5, Ball im Kasten*, AEG Telefunken
- [REED] Irving S. Reed, *The Dawn of the Computer Age*, in *Engineering & Science*, No. 1, 2006, pp. 7–12

Bibliography


[THOMSON] William Thomson, *Mechanical Integration of linear differential equations of the second order with variable coefficients*, Proceedings of the Royal Society, Volume 24, No. 167, pp. 269-270, 1876


[TRUITT] Thos. D. Truitt, A. E. Rogers, *Basics of Analog Computers*, John F. Rider Publisher, Inc., New York, December 1960


[TSE] Francis S. Tse, Ivan E. Morse, Rolland T. Hinkle, *Mechanical Vibrations*, Allyn and Bacon, Inc., Boston, Second Printing, August 1964


[VOLDER] Jack E. Volder, *The CORDIC Trigonometric Computing Technique*, in IRE Trans. Electron. Comput. EC-8:330–334 (1959)