Mathieu's equation revisited

1 Introduction

As simple as it looks at first glance, MATHIEUS's equation exhibits highly interesting behaviour under control of its two parameters a and q. It was already the subject of an application note[1] dating back to 2017. This application note itself was mostly based on an even more classic one by EAI, see [EAI 7.4.4a]. On page 4 in this source a stability diagram is shown, which sparked the idea of creating such a map using a hybrid computer setup. It should be noted that the definitive source on the overall topic of MATHIEU's equation and MATHIEU functions is [MCLACHLAN 1947], which is highly recommended.

Please note that the following just describes an experiment – there are open questions and a lot could and should be discussed, so the actual implementation and the results should be taken with a grain of salt. Several implementation details seem quite idiosyncractic and are the result of the available hard- and software at the time of this writing. These experiments have been done out of curiosity, not with some special application in mind. Accordingly, fast run times were not even on the task list, in fact the run time is excruciatingly long.

MATHIEU's equation is of the form

$$\ddot{m} + (a - 2q\cos(2t))\, m = 0 \tag{1}$$

where a and q are parameters, which will satisfy $0 \leq a \leq 8$ and $0 \leq q \leq 5$ in the following implementation.

The basic idea of computing a stability map is pretty simple: Set values a and q under control of the digital computer, let the analog computer run for a certain time and check for an overload.

First of all, a forcing function $\cos(2t)$ is required which could be generated easily with two integrators and a summer in a loop. However, this approach requires some means of amplitude stabilisation, e. g. by using a small positive feedback in conjunction with two 10 V

[1] See [ULMANN 2017], retrieved 19.09.2024.

Z-diodes. Another approach is to use a VAN DER POL oscillator[2] as the source of this forcing function.

This oscillator is described by

$$\ddot{y} + \mu \left(y^2 - 1\right) \dot{y} + y = 0$$

with the parameter μ controlling the amplitude stabilisation. With small values for μ, such as $\mu \approx 0.005$, the resulting output signal is spectrally very clean, especially with an initial value of -1 at the second integrator, yielding a clean $\cos 2t$ signal with proper scaling.

This forcing function is then fed into the implementation of equation (1), while m is observed for an overload condition. Since the behaviour of the MATHIEU equation can be very complex, quite some time is required for ensuring that an overload will actually show up. Depending on the actual OP-time per analog computer run it is still possible that some points of instability show up as being stable if m did not reach an overload condition at the end of a run.

It should be noted that overloads involving $\dot{m}$ were ignored in the computation of the stability map.[3]

2 Implementation

The overall analog computer setup is shown in figure 1. The program consists of three parts. First is a sub-circuit implementing a VAN DER POL oscillator generating a clean $\cos(2t)$ forcing function signal. The value μ is chosen so that the amplitude stabilisation just kicks in which should be around 0.005.

This signal is fed to the next sub-circuit implementing the MATHIEU equation itself. The two potentiometers denoted by `a/10` and `q/10` are digital potentiometers under control of the attached digital computer.

Unfortunately, the Perl library `IO::HyCon`[4] does not handle overload conditions well as

[2] See [ULMANN(2023), p. 122 ff.].
[3] Further investigations could take also take $\dot{m}$ into account.
[4] See `https://metacpan.org/pod/IO::HyCon`, retrieved 19.09.2024.

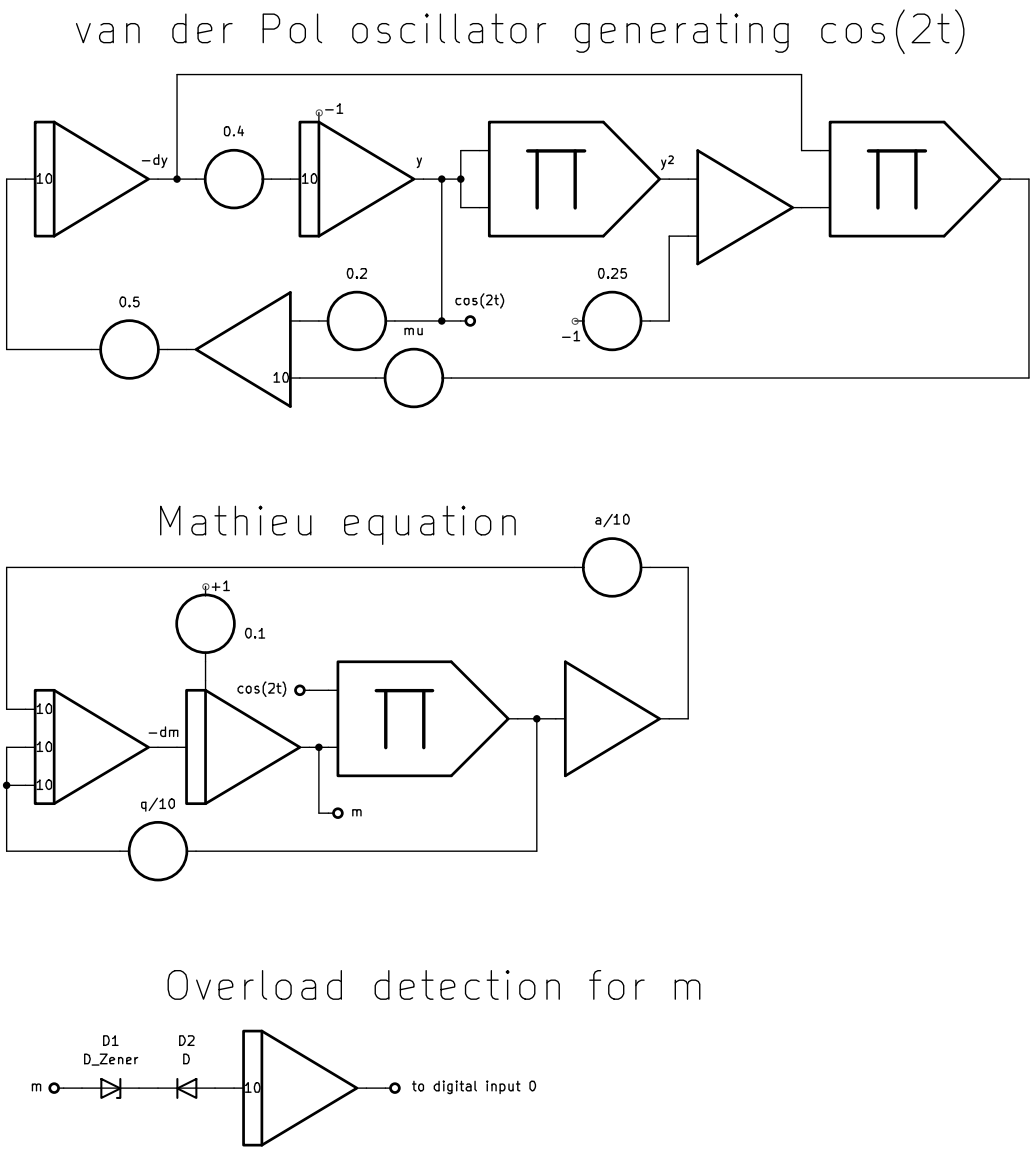

Figure 1: Setup for the MATHIEU stability map problem

of now. So the basic idea is to perform analog computer runs with constant OP-time and check for an overload condition only for the variable m at the end of each run. This requires some "memory" for an overload, shown in the sub-circuit on the bottom of figure 1. m is fed to a 10 V Z-diode followed by a normal diode, so only signals with a negative amplitude exceeding one machine unit can pass. These are integrated by means of an integrator set to the highest possible time scaling factor k_0. This integrator will go into overload very quickly as soon as a signal passes this diode chain. Its output line is connected to a digital input line of the hybrid controller.[5]

The Perl control program is shown in figure 2. It basically consists of two nested loops iterating `q/10` and `a/10` over the desired interval. Inside these loops the two digital potentiometers are set accordingly and the analog computer is set to initial condition mode for a short time, followed by OP mode for some maximum time.[6]

At the end of each run digital input `0` is read. If it is set to `1` an overflow occurred which is then reflected by printing a $*$ while stable points are denoted by a white space character.

The required configuration file is shown in figure 3. It contains the serial port parameters, the computing element type definitions, and the definition of the two digital potentiometers used in the setup.

3 Results

Capturing the results of running the program shown in figure 2 yields the stability map shown in figure 4. A bit unexpected is the little appendix of instability right to the leftmost cusp while the rest of the map looks reasonable. Nevertheless, the whole topic needs more investigation and thus should only serve as an example of a little hybrid computer setup.

Happy hybrid computing

[5] If overloads could be detected in a selective fashion, i. e., only an overload for m but not one for $\dot{m}$, this would drastically reduce the overall run time as a particular analog computer run could be directly terminated at the occurrence of such an overload. It is to be expected that this feature will be implemented in the near future in `IO::HyCon`.

[6] The longer the OP time is chosen the slower the overall program will run but it will detect instability points where m rises very slowly.

```
use strict;
use warnings;

$| = 1;

use Time::HiRes qw(usleep);
use IO::HyCon;

my $op_time = 2000;
my ($a_max, $q_max, $a_step, $q_step) = (8, 5, .05, .05);

my $ac = IO::HyCon->new();
$ac->set_op_time(10000);

for (my $q = 0; $q <= $q_max; $q += $q_step) {
    for (my $a = 0; $a < $a_max; $a += $a_step) {
        $ac->set_pt('a', $a / 10);
        $ac->set_pt('q', $q / 10);

        $ac->ic();
        usleep(100 * 1000);

        $ac->op();
        usleep($op_time * 1000);

        print $ac->read_digital()->[0] ? '*' : ' ';
    }
    print "\n";
}
```

Figure 2: Perl program controlling the analog computer

```
serial:
    port: /dev/cu.usbserial-DN050L1O
    bits: 8
    baud: 250000
    parity: none
    stopbits: 1
    poll_interval: 10
    poll_attempts: 20000
types:
    0: PS
    1: SUM8
    2: INT4
    3: PT8
    4: CU
    5: MLT8
    6: MDS2
    7: CMP4
    8: HC
elements:
    a: 0x0030/0
    q: 0x0030/1
```

Figure 3: Configuration YAML file

Figure 4: Stability map generated by the hybrid computer, a running from 0 to 8 in increments of $\frac{1}{10}$ along the horizontal axis, q running vom 0 to 5 in $\frac{1}{10}$ steps from top to bottom; dots denote value combinations for which $\dot{m}$ causes an overload, i. e. unstable points

References

[EAI 7.4.4a] Electronic Associates, Inc., "Solution of Mathieus' Equation on the Analog Computer", in *EAI Applications Reference Library, General Purpose Analog Computation, Educational*, see `http://bitsavers.org/pdf/eai/applicationsLibrary/7.4.4a_Solution_of_Mathieus_Equation_on_the_Analog_Computer_1964.pdf`, retrieved 19.09.2024

[McLachlan 1947] N. W. McLachlan, *Theory and Application of Mathieu Functions*, Oxford at the Clarendon Press, 1947

[Ulmann 2017] Bernd Ulmann, Mathieu*'s equation*, `https://analogparadigm.com/downloads/alpaca_10.pdf`, retrieved 19.09.2024

[Ulmann(2023)] Bernd Ulmann, *Analog and hybrid computer programming*, DeGruyter, 2nd edition, 2023