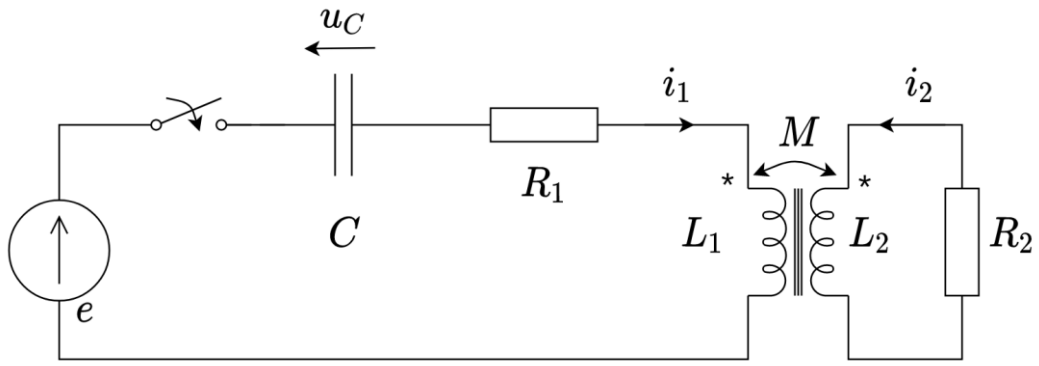
Mutual inductance circuit project

Zbigniew Michalak

The project involved developing a simulator in Matlab of a simple electrical circuit containing mutual inductance between coils L1 and L2. The simulator generates time traces for the capacitor voltage uC, and the currents i1 and i2 in the circuit.



Used methods description

Euler's method:

It's an explicit method for numerically solving ODEs. It uses a simple evaluation of the derivative at the current point to find the next value using the formula: y(n+1) = y(n) + h * f(x(n), y(n)) where h is the step size. Euler's method is the simplest ODE solver but also the least accurate, with truncation error of O(h^2). It is not recommended for stiff ODEs as it can become unstable.

Heun's method:

A predictor-corrector method that improves upon Euler's method. It first uses Euler's method to predict y(n+1), then evaluates the derivative again at this predicted point. The two derivative evaluations are combined to improve the final y(n+1) approximation. Heun's method has a truncation error of O(h^3)[as the step size h gets smaller, the truncation error gets smaller proportionally to h^3], so is more accurate than Euler for the same step size h. It provides a good trade-off between accuracy and computational cost.

Runge-Kutta:

A family of ODE solvers that achieve higher order accuracy by evaluating multiple increments per step. The classic RK4 method evaluates 4 increments per step to arrive at 4th order accuracy O(h^5). RK4 requires 4 evaluations of f(x,y) per step, so is more

computationally expensive than lower order methods. However, RK4 provides excellent accuracy and stability for non-stiff ODEs when using an appropriate step size. Higher order RK methods like RK5 are also available for even higher accuracy at increased computational cost. More advanced ODE solvers are required for stiff systems, for example adaptive methods and multistep methods like Adams-Bashforth, Adams-Moulton and Gears method, which we used during the laboratory.

Lagrange polynomial interpolation:

Lagrange interpolation is a method for finding a polynomial that passes through a given set of points. Given n data points (x1, y1), (x2, y2), ..., (xn, yn), we aim to construct a polynomial P(x) of degree n-1 that fits these points exactly.

Lagrange interpolation is useful for its simplicity and the fact that it does not require solving a system of equations, unlike some other interpolation methods. However, it can become numerically unstable for large datasets due to the Runge phenomenon (overfitting), where oscillations occur at the edges of the interval.

Spline

Spline interpolation is similar, but instead of using one polynomial, it uses cubic spline pieces (piecewise interpolation) that are tied together. This results in a very smooth interpolating function that follows the data points closely without any abrupt changes in curvature. It avoids the problem of Runge's phenomenon, in which oscillation can occur between points when interpolating using high degree polynomials.

Polynomial approximation using a polynomial of high degree

Polynomial approximation using lower degree polynomials like cubics (degree 3) or quintics (degree 5) attempts to find the best fit polynomial of that degree to the given data over the entire domain, not just passing point-by-point.

Integration by composite rectangles rule and composite trapezoidal rule

Integration rules like the composite rectangles and composite trapezoidal rules let us numerically estimate the integral or area under a curve, even when we don't have an explicit function to integrate analytically. They work by approximating the region using a series of rectangles or trapezoids whose areas can be easily calculated and summed.

Bisection Method

Splits interval in half, selecting subinterval where the function changes sign. It is a straightforward numerical technique to find a root of the equation f(x) = 0. It works by repeatedly dividing an interval in half and selecting the subinterval in which the function changes sign. Starting with an interval [a, b] where f(a) and f(b) have opposite signs, the method computes the midpoint c = (a + b) / 2. If f(c) is zero (or close enough), c is the root.

Otherwise, the interval is halved: if f(a) and f(c) have opposite signs, the new interval is [a, c]; otherwise, it is [c, b]. This process repeats until the interval is sufficiently small.

Secant Method

Uses two initial guesses, approximates root using secant line between points. Iterative technique to find the root of f(x) = 0 using two initial guesses, x0 and x1. Unlike the bisection method, it does not require the function to be bracketed within an interval. The method approximates the root by the formula:

x(i+1) = xi - f(xi) * (xi - x(i-1)) / (f(xi) - f(x(i-1)))

Here, xi and x(i-1) are the current and previous guesses. This process is repeated using the most recent guesses until convergence.

Quasi-Newton Method

Iterative, refines root estimate using approximated derivative, improves convergence speed. More advanced technique to find the root of f(x) = 0. It improves upon the Newton-Raphson method by approximating the derivative. Starting with an initial guess x0, the method uses:

x(i+1) = xi - f(xi) / f'(xi)

However, instead of computing the exact derivative, it updates an approximation of the inverse Jacobian matrix. The process iterates, refining the root estimate and the Jacobian approximation until convergence.

Verification of the results using a different method

For a steady-state circuit (e= 1V DC) the capacitor voltage converges to 1 in the simulator, which agrees with the fact that for DC current, the voltage over a capacitor should be the same as on the source (perhaps the currents should be 0, considering that at DC current the capacitor behaves like an open circuit).

Circuit analysis for e=sin(t)

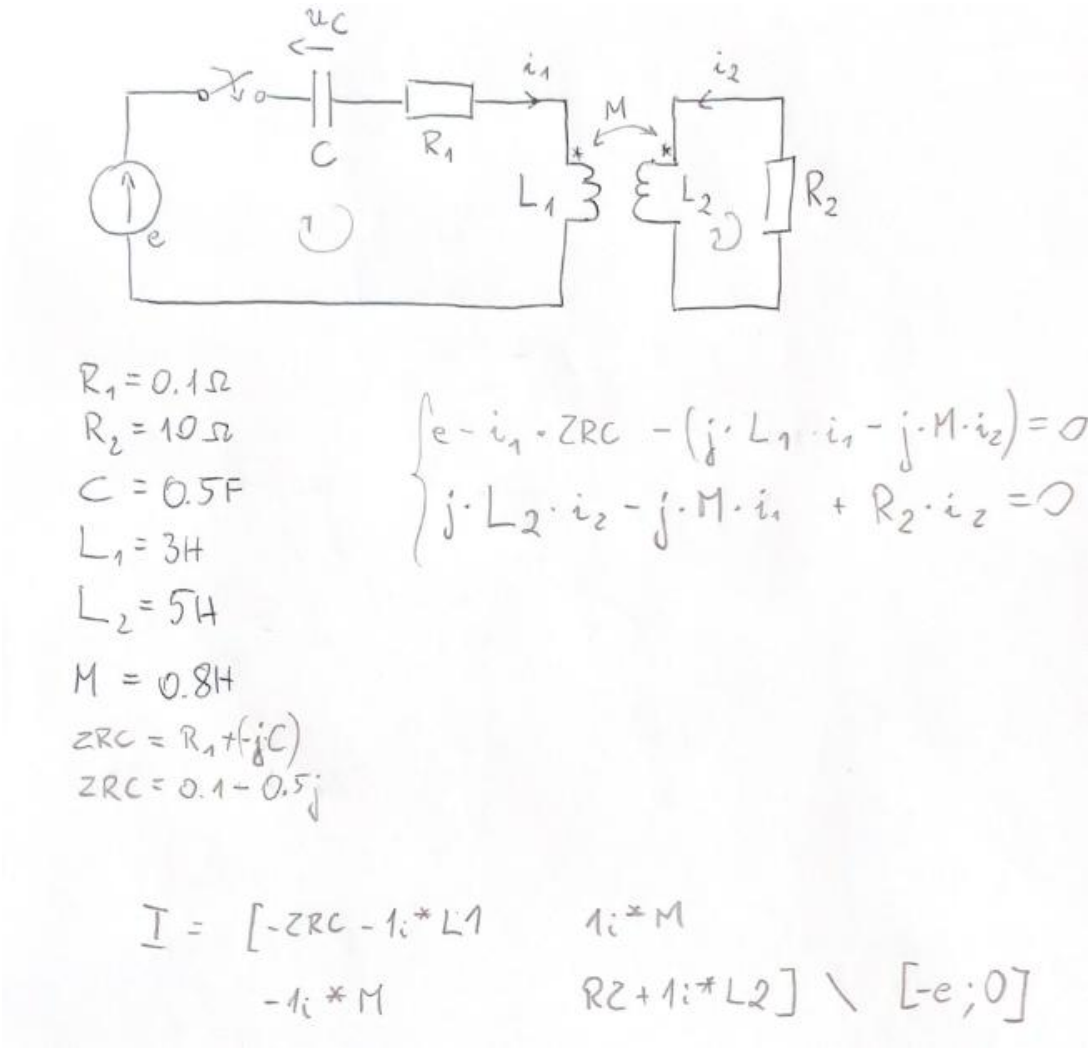


Figure 1. Simplified circuit analysis calculations for e=1*sin(t)

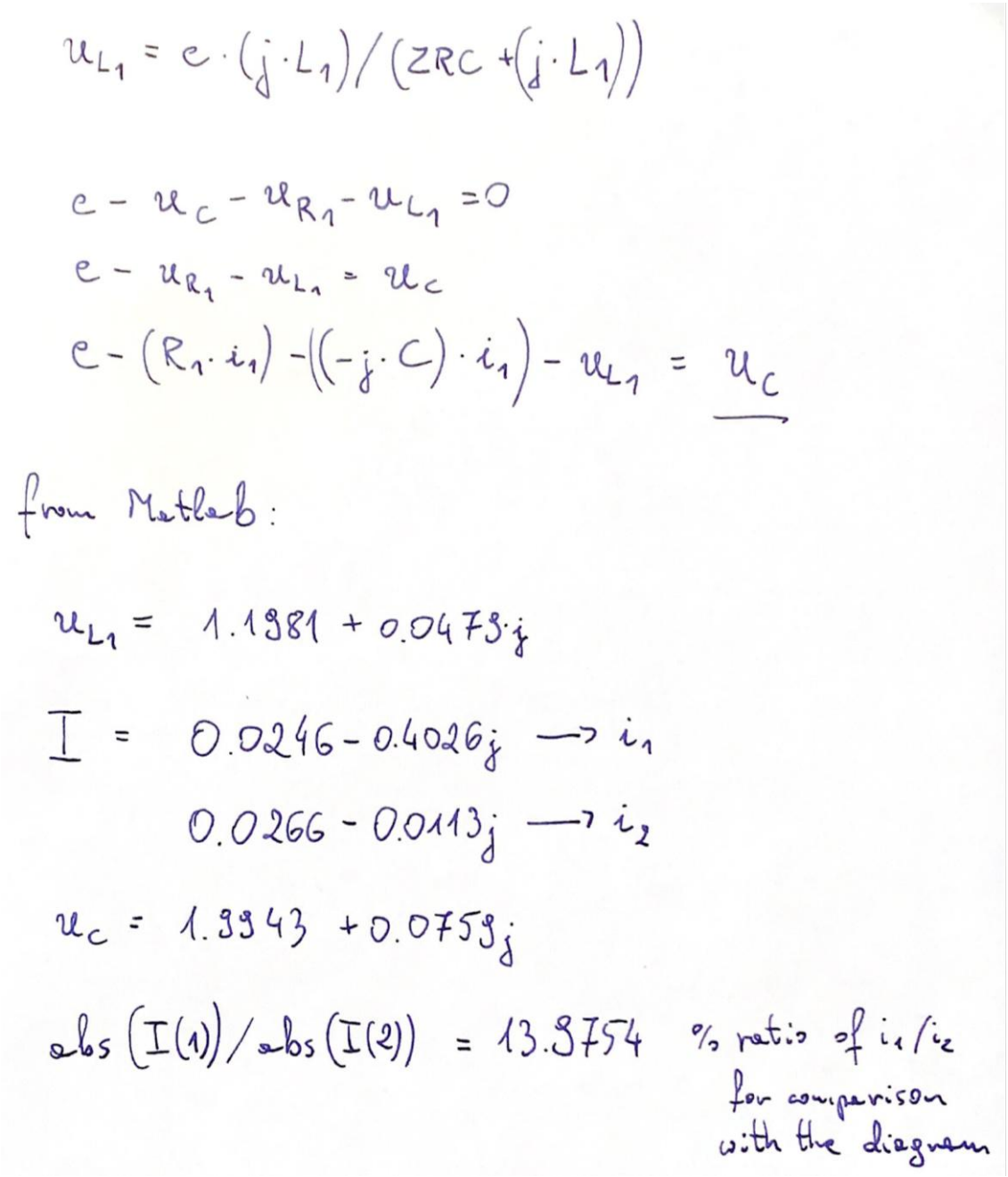


Figure 2. Circuit analysis for e=1*sin(t)

I performed a simple circuit analysis. Assumed e=1. For this kind of circuit analysis calculation e=1 means **e=1*sin(t)**. Matlab script is attached under the name *"manual_num_project_circ".*

1. The ratio of i_1 to i_2 is around 14, which agrees with simulator plots for currents (trace for i_2 is around 14 times larger, which I verified by checking the amplitude of signals.
2. The voltage on the capacitor is 2V which agrees with the simulator diagrams.

Part 1 - Transient State Simulation

The first part involved implementing a transient state simulation of the circuit using three different numerical methods for solving ordinary differential equations – Euler's, Heun's, and RK4 methods.

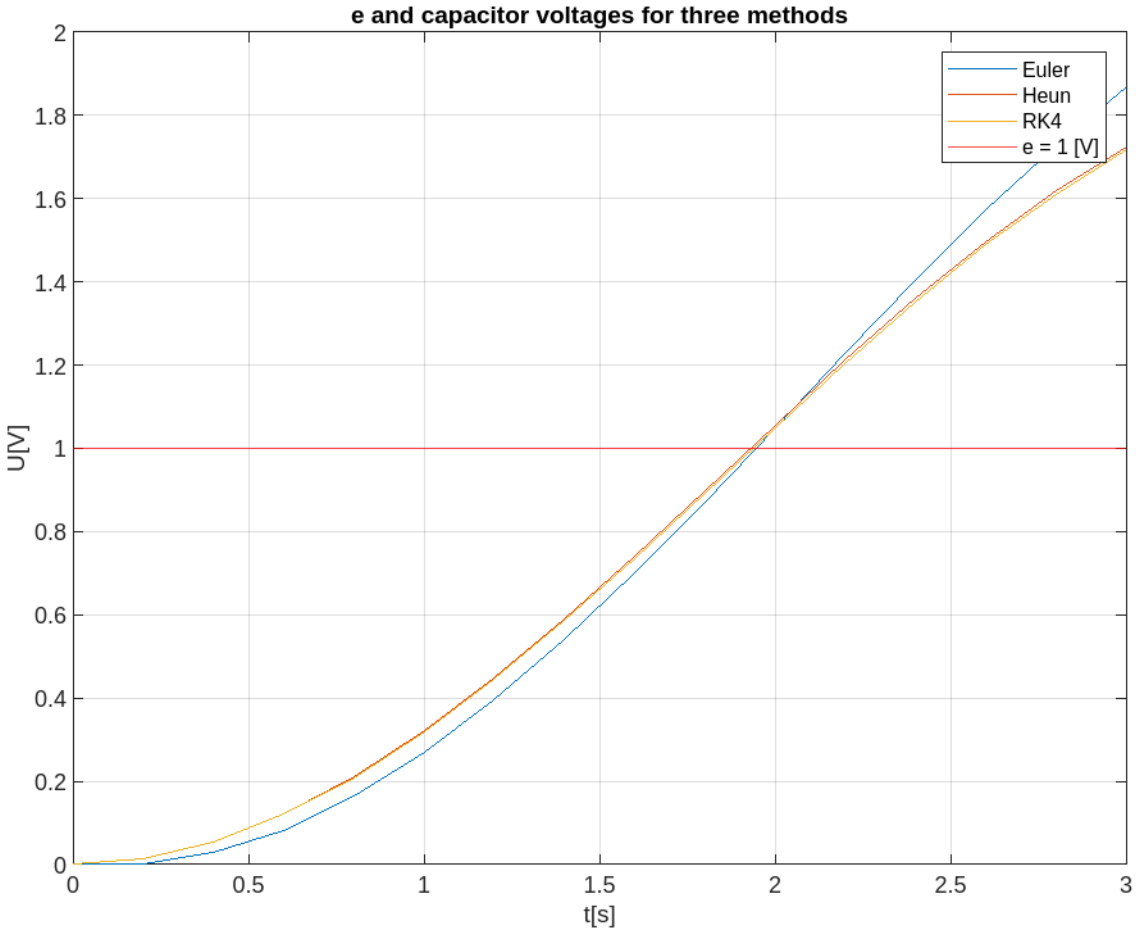


Figure 3. Voltage waveform diagrams for 1 [V], T = 3 [s]

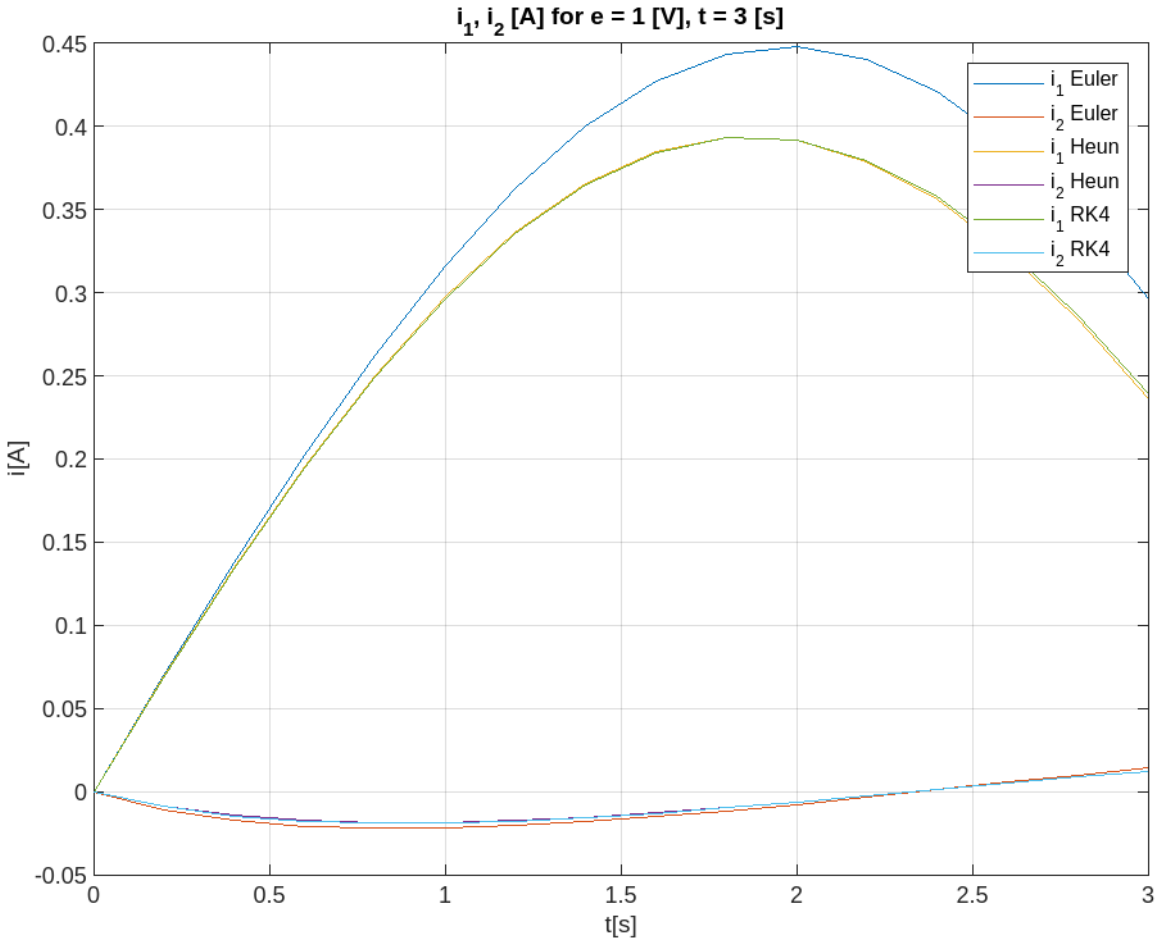


Figure 4. Currents waveform diagrams for 1 [V], T = 3 [s]

Even for a very high step of h=0.2 the waveforms look reasonable, although we can notice the discrepancies between the different methods – Euler's is clearly fast diverging from the others, indicating it might be the least accurate.

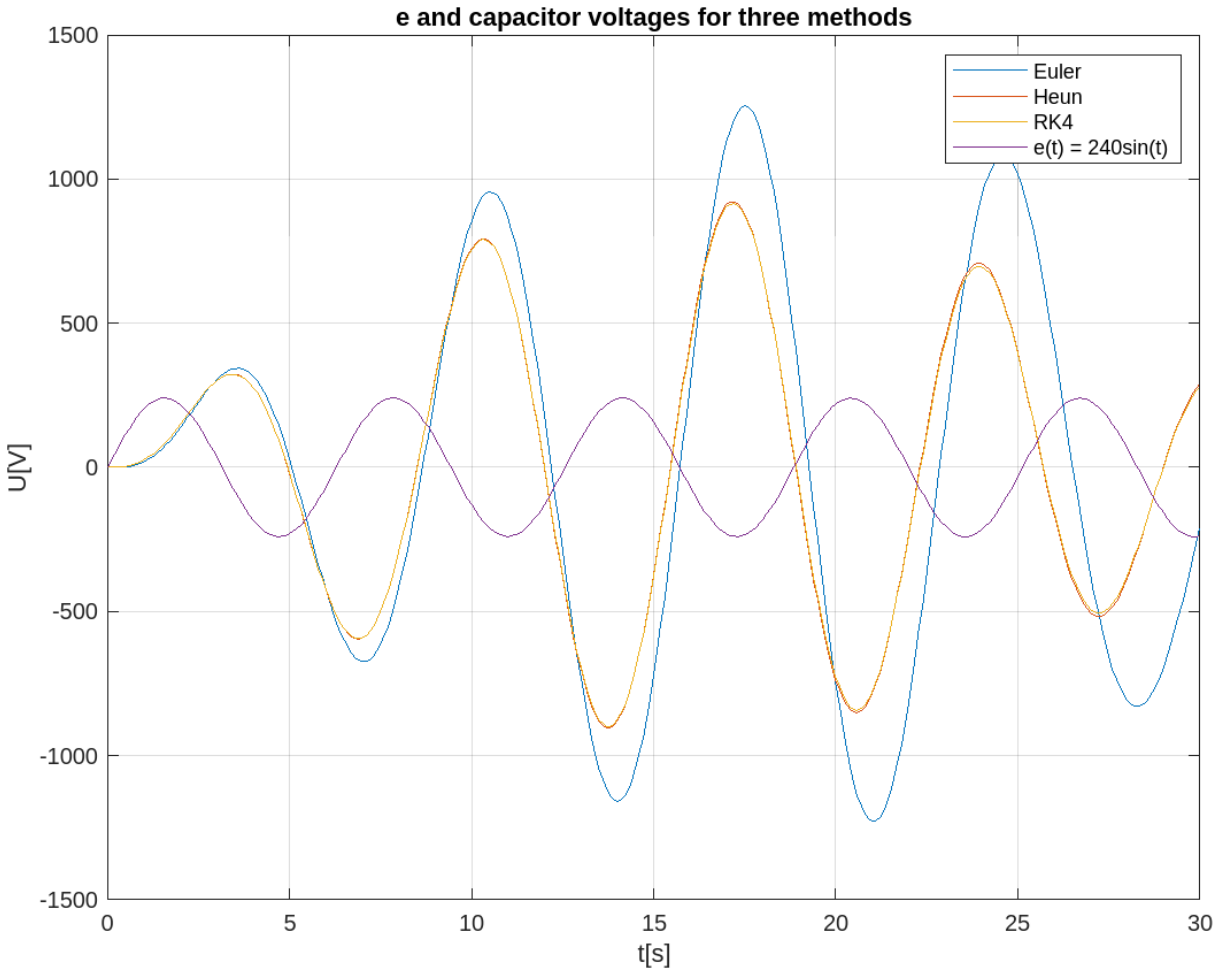


Figure 5. Voltage waveform diagrams for e(t)=240sin(t)

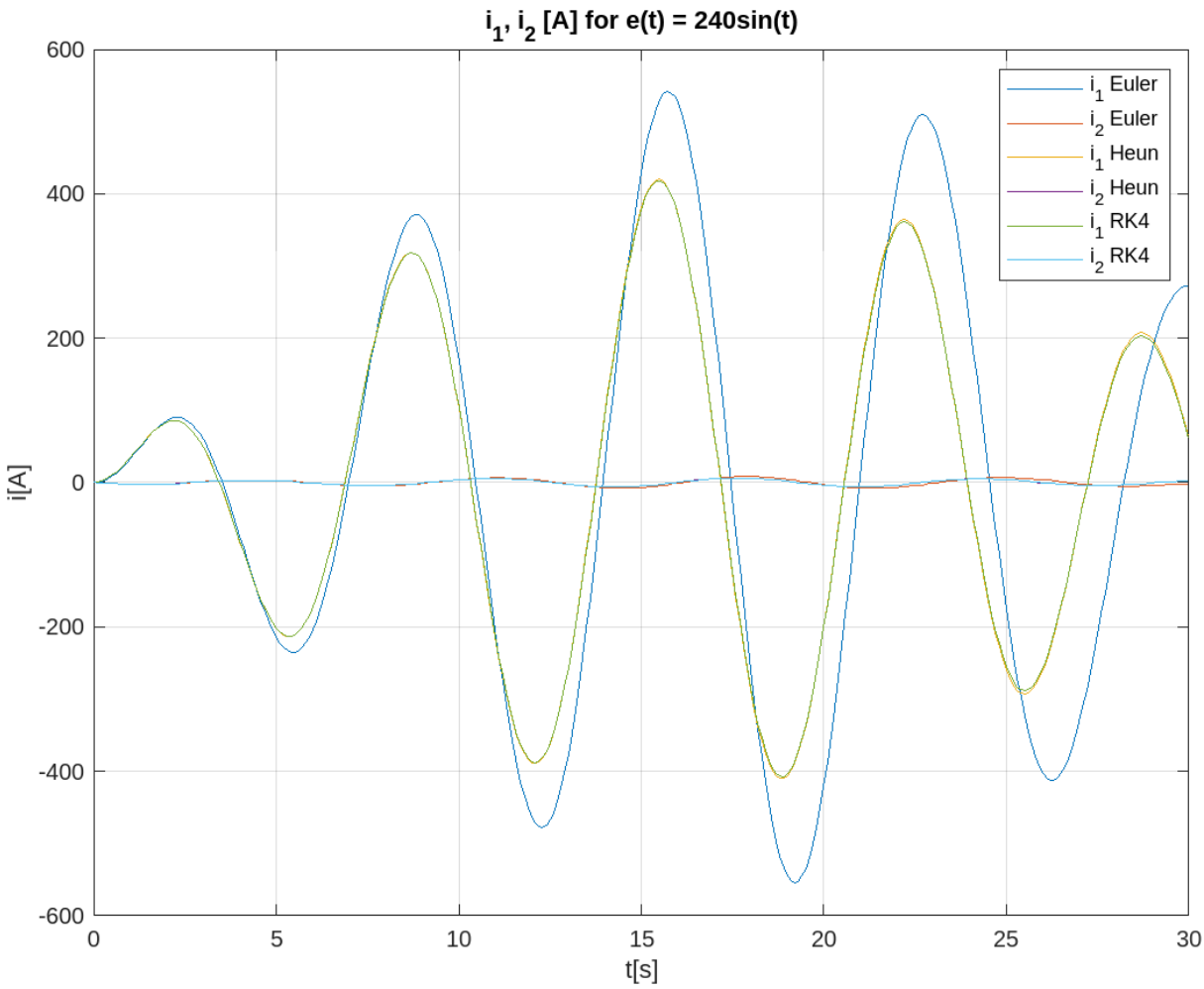


Figure 6. Currents waveform diagrams for e(t) = 240sin(t)

In the case of e(t) = 240sin(t), I used h=0.1, and again the Euler's method is the one obviously diverging with time. At least for the 30s interval Heun's and RK4 methods remain stable and produce similar result.

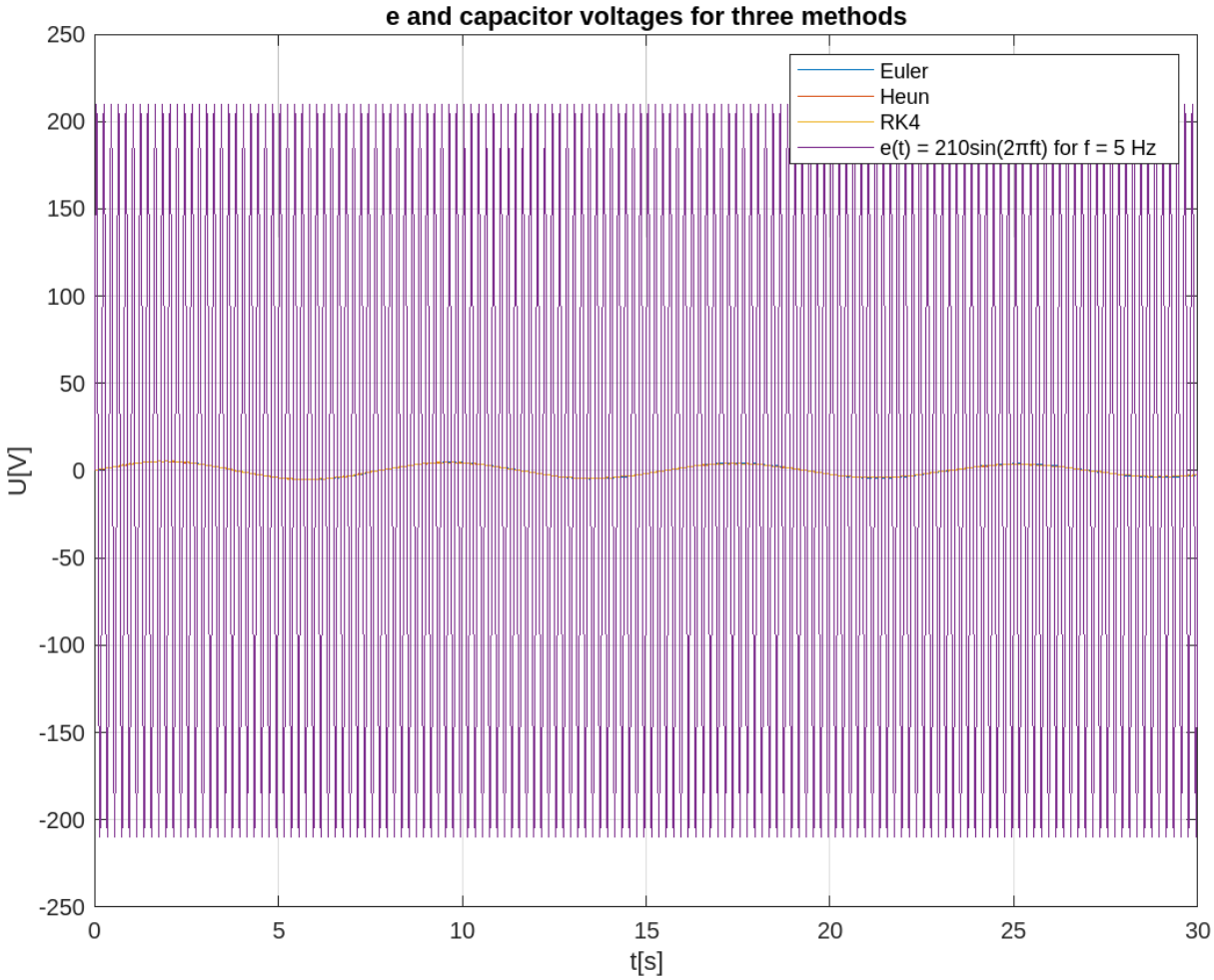


Figure 7. Voltage waveform diagrams for e(t) = 210sin(2πft), for f = 5 Hz

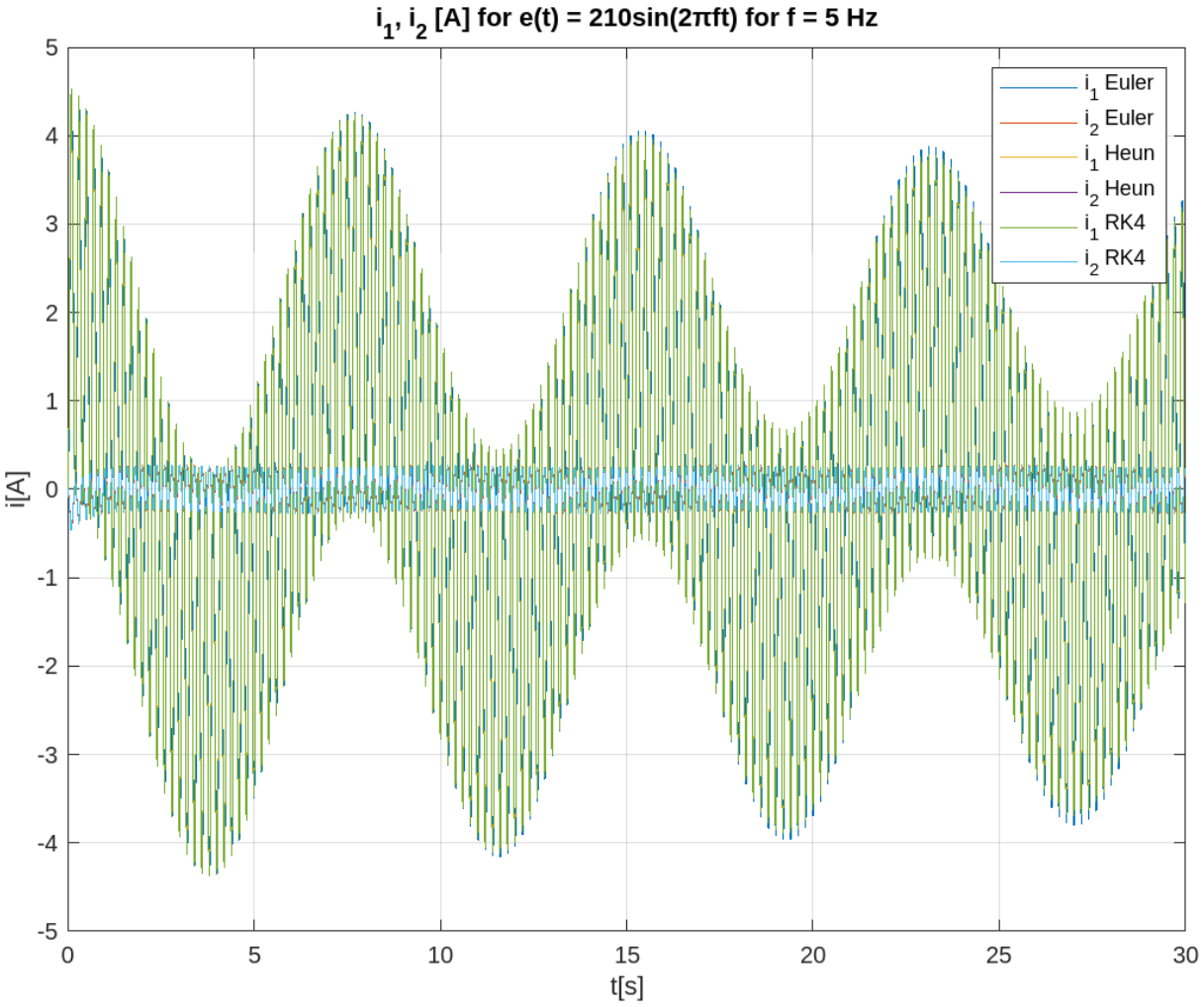


Figure 8. Currents waveform diagrams for e(t) = 210sin(2πft), for f = 5 Hz

For excitations involving frequency increasing the h was necessary to get an appropriate waveform. Since 5Hz is 5 cycles per second we need a lot of sudden changes in plotting direction already at this frequency level.

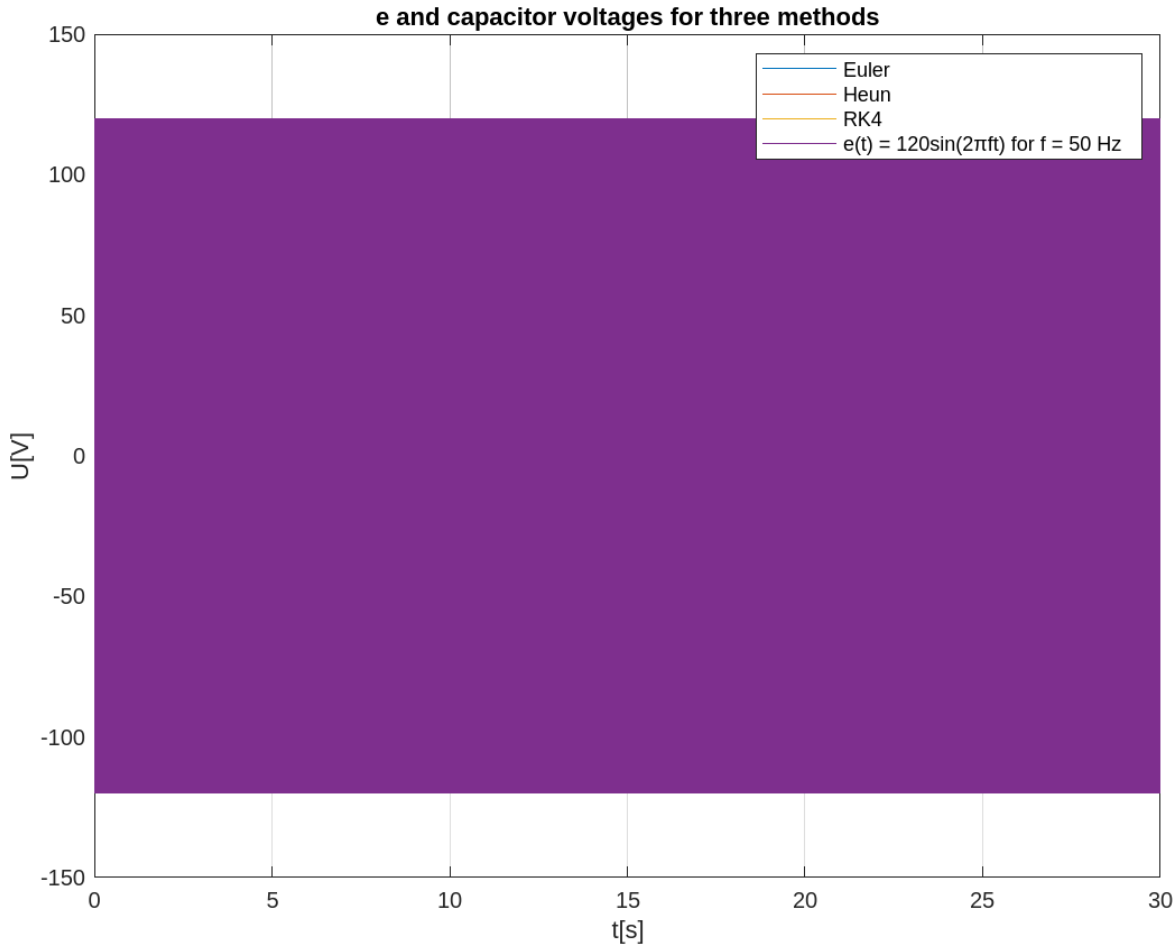


Figure 9. Voltage waveform diagrams for e(t) = 120sin(2πft), for f = 50Hz

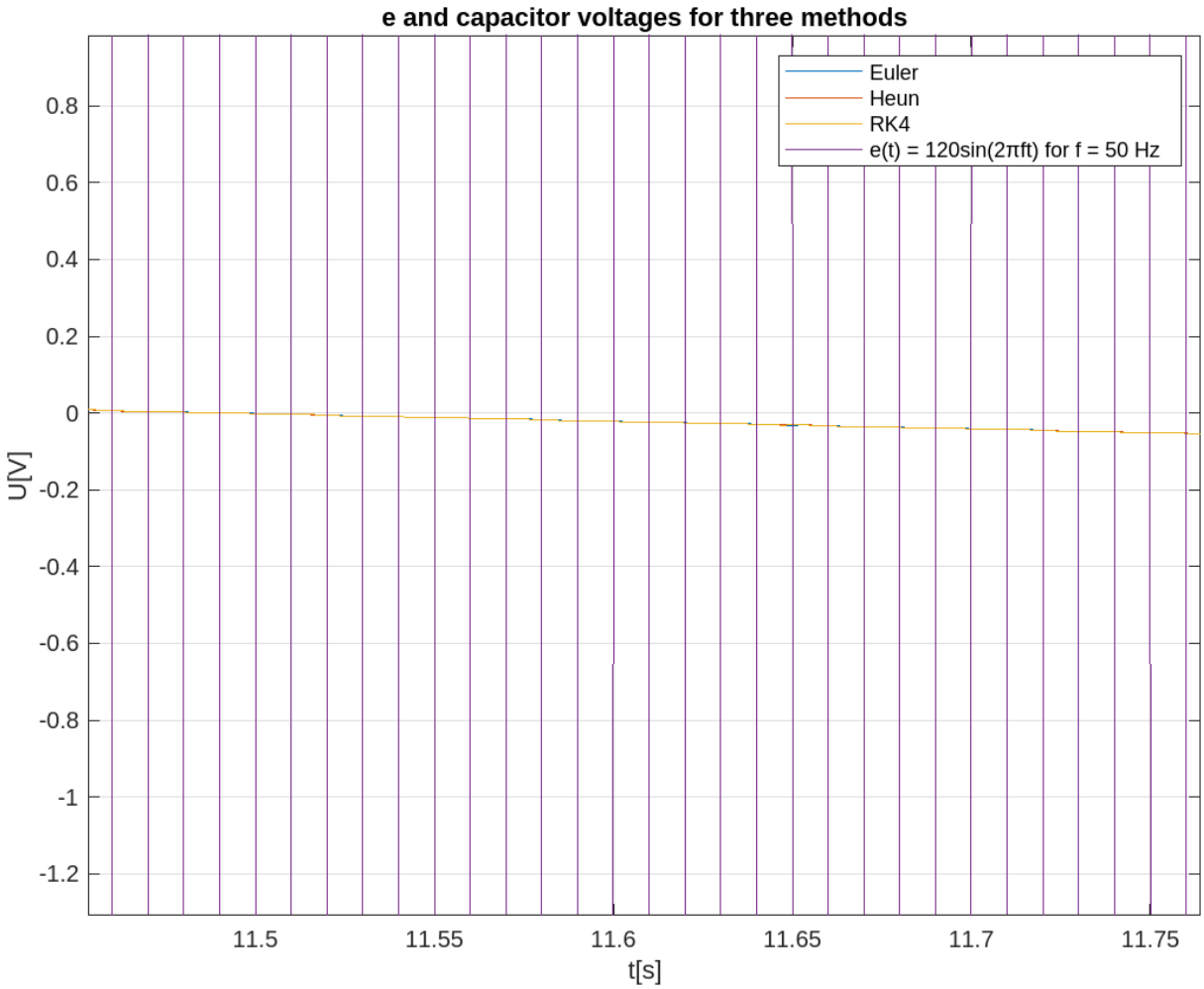


Figure 10. Voltage waveform diagrams for e(t) = 120sin(2πft), for f = 50Hz (zoomed in)

For clarity I included a zoomed in diagram which shows that the trace of e(t) and the capacitor voltages are indeed present on the plot.

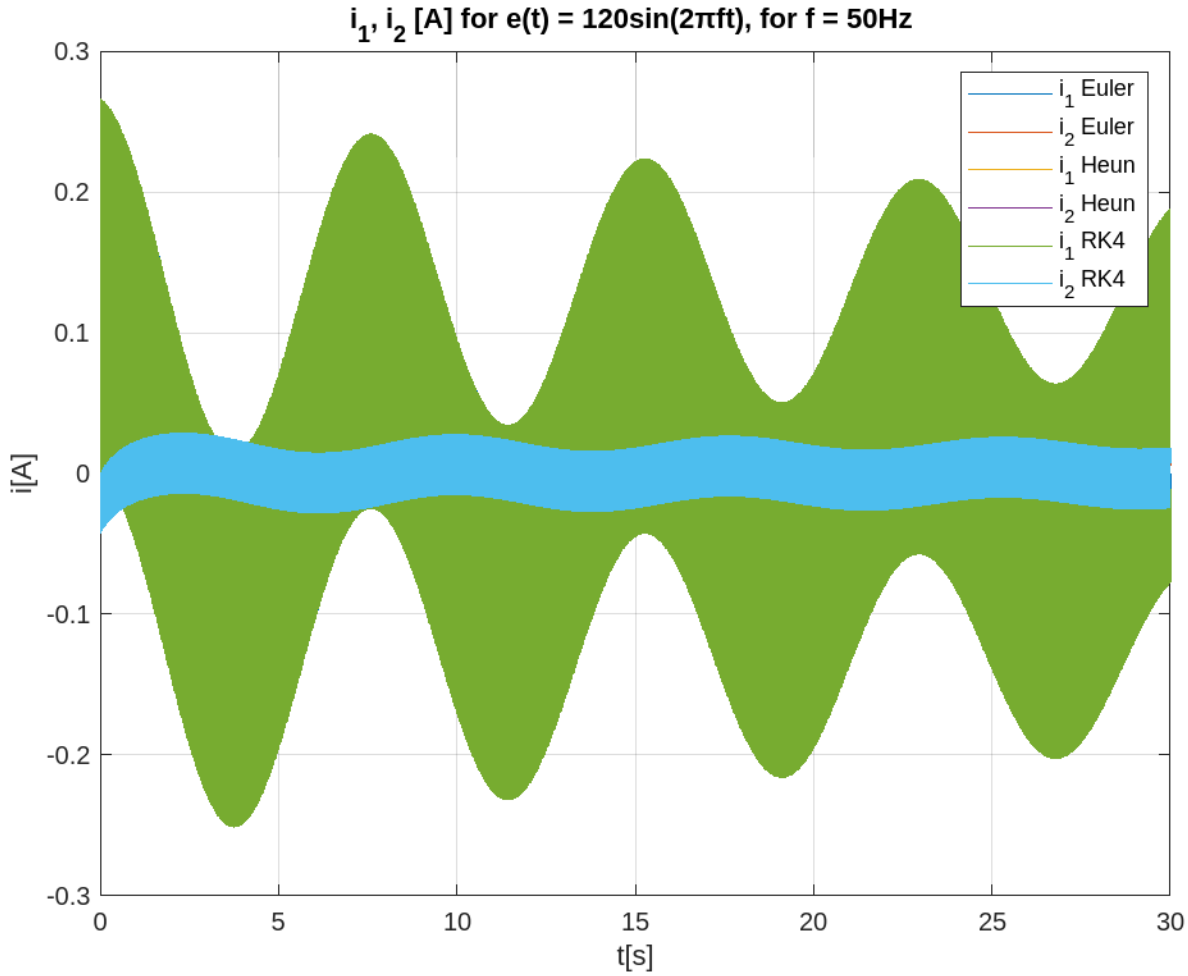


Figure 11. Currents waveform diagrams for e(t) = 210sin(2πft), for f = 5 Hz

For 50Hz the plots become artistic but are not legible. I used h= 0.001 to generate the waveforms, larger increments couldn't generate an appropriate result.

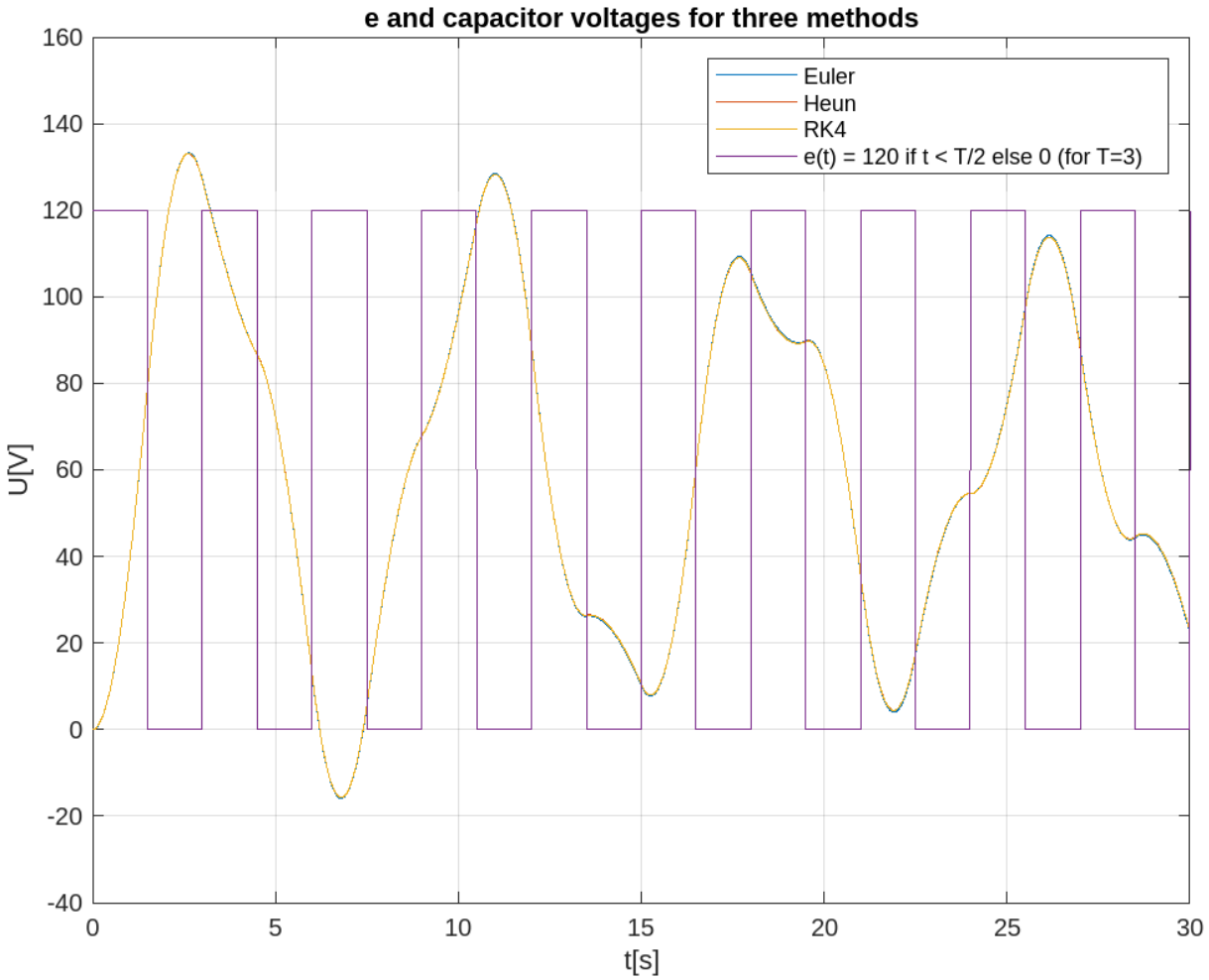


Figure 12. Voltage waveform diagrams for e(t) = 120 if t < T/2 (else 0)

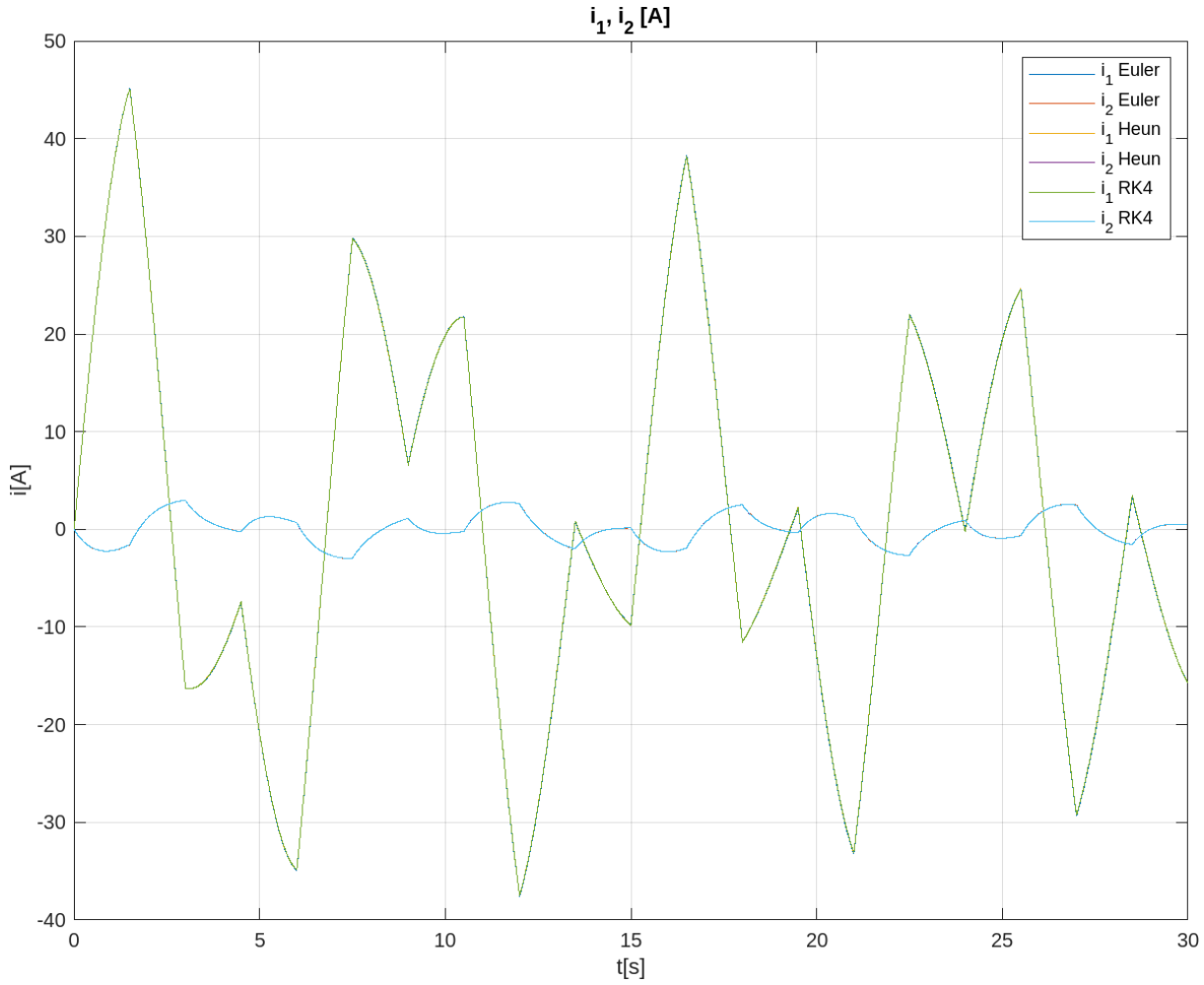


Figure 13. Currents waveform diagrams for e(t) = 120 if t < T/2 (else 0)

Part 2 - Non-Linear Mutual Inductance

This part modeled the mutual inductance M as a non-linear function of the voltage uL1 across L1, based on a given data table relating M and uL1. I will use four different interpolation/approximation methods were implemented to find M(uL1) at each time step - polynomial interpolation, spline interpolation, and 3rd and 5th order polynomial approximations. Time traces will be generated using the RK4 method.

$U_{L_1,j}$ [V]	20	50	100	150	200	250	280	300
M_j	0.46	0.64	0.78	0.68	0.44	0.23	0.18	0.18

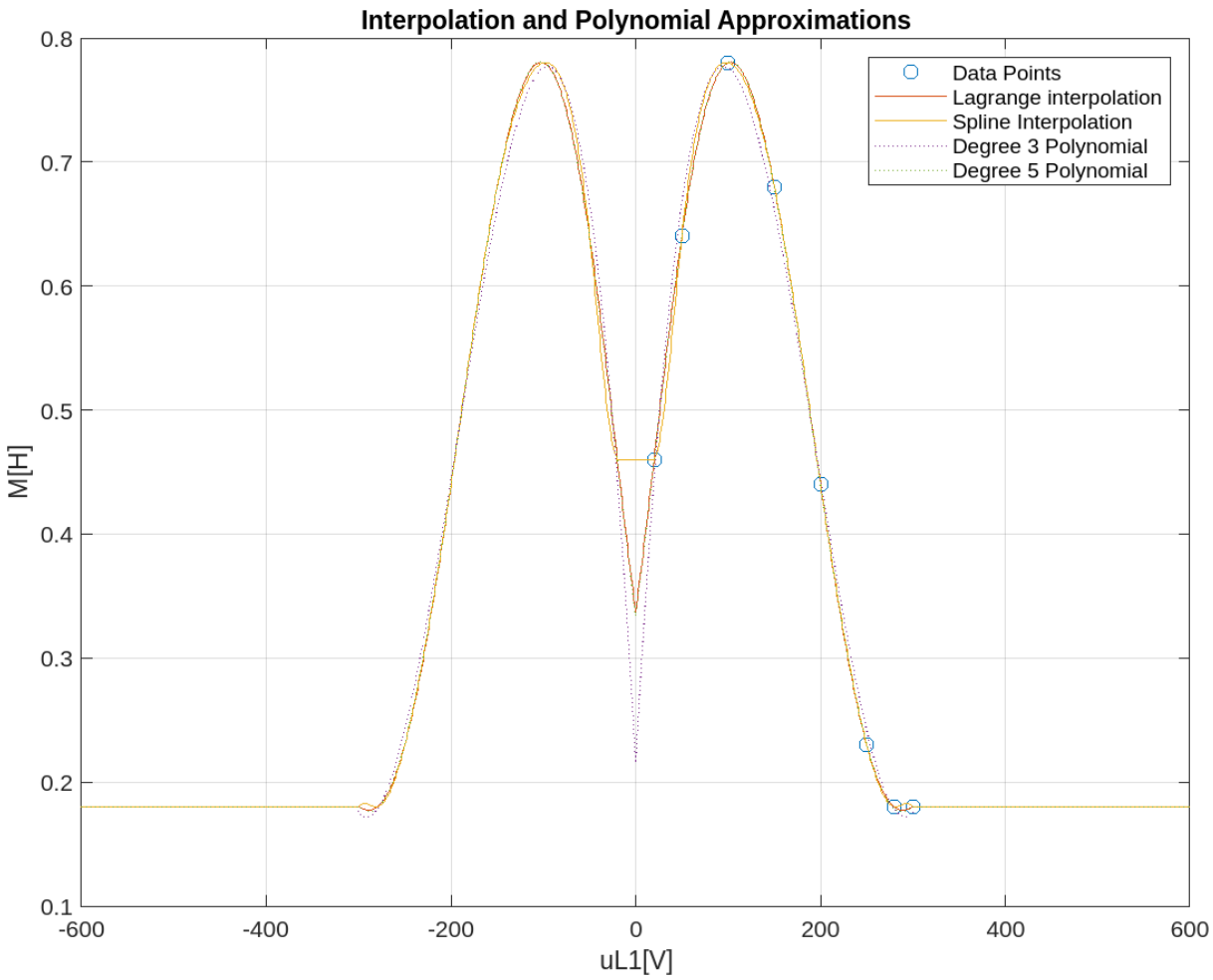


Figure 14. The traces for interpolation of the mutual inductance data

The values of M for uL1 > 300 were fixed to 0.18H. For voltage greater than 300V mutual inductance values are invalid. It's a typical problem of extrapolation (determining values of function outside the given range of measurements).

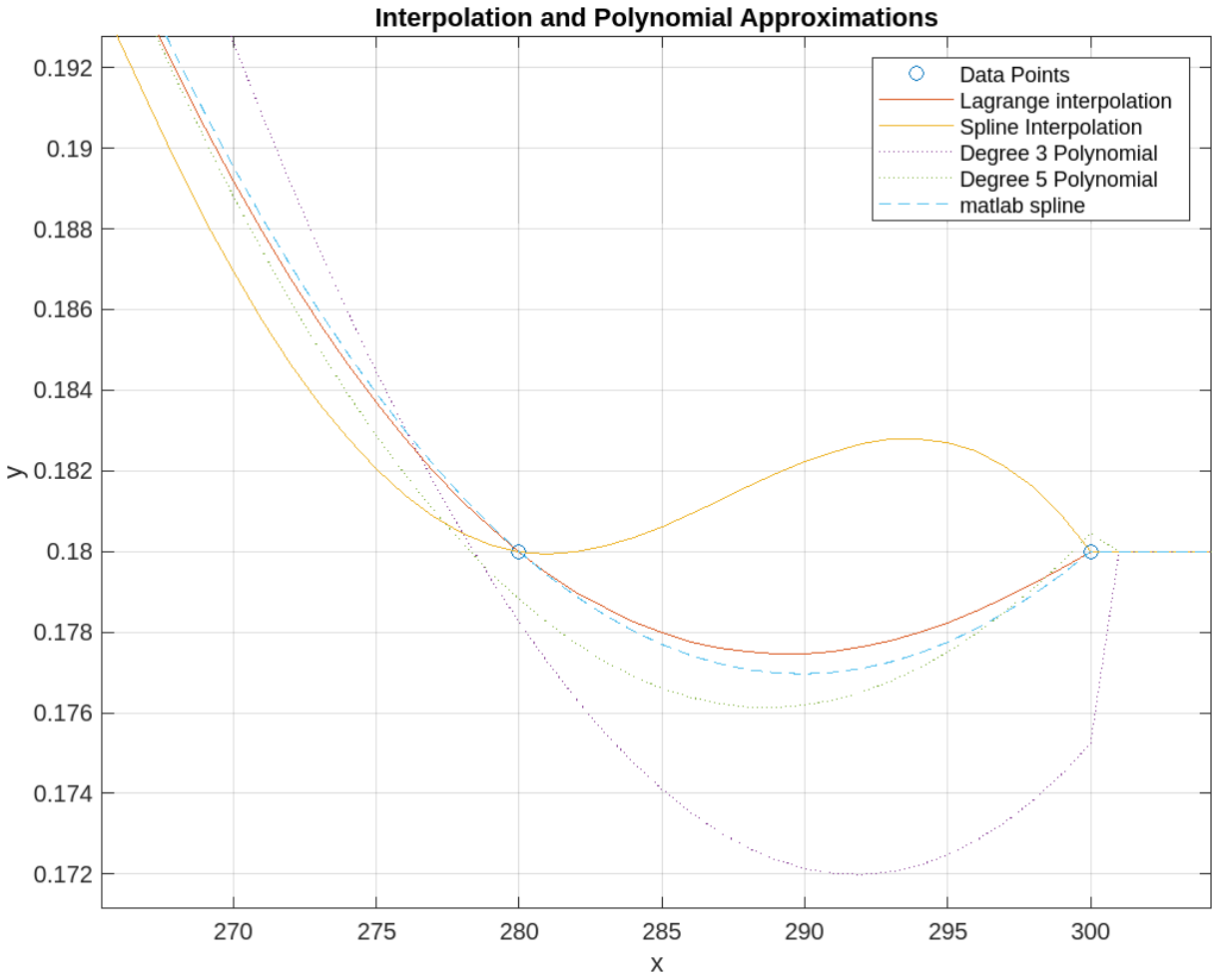


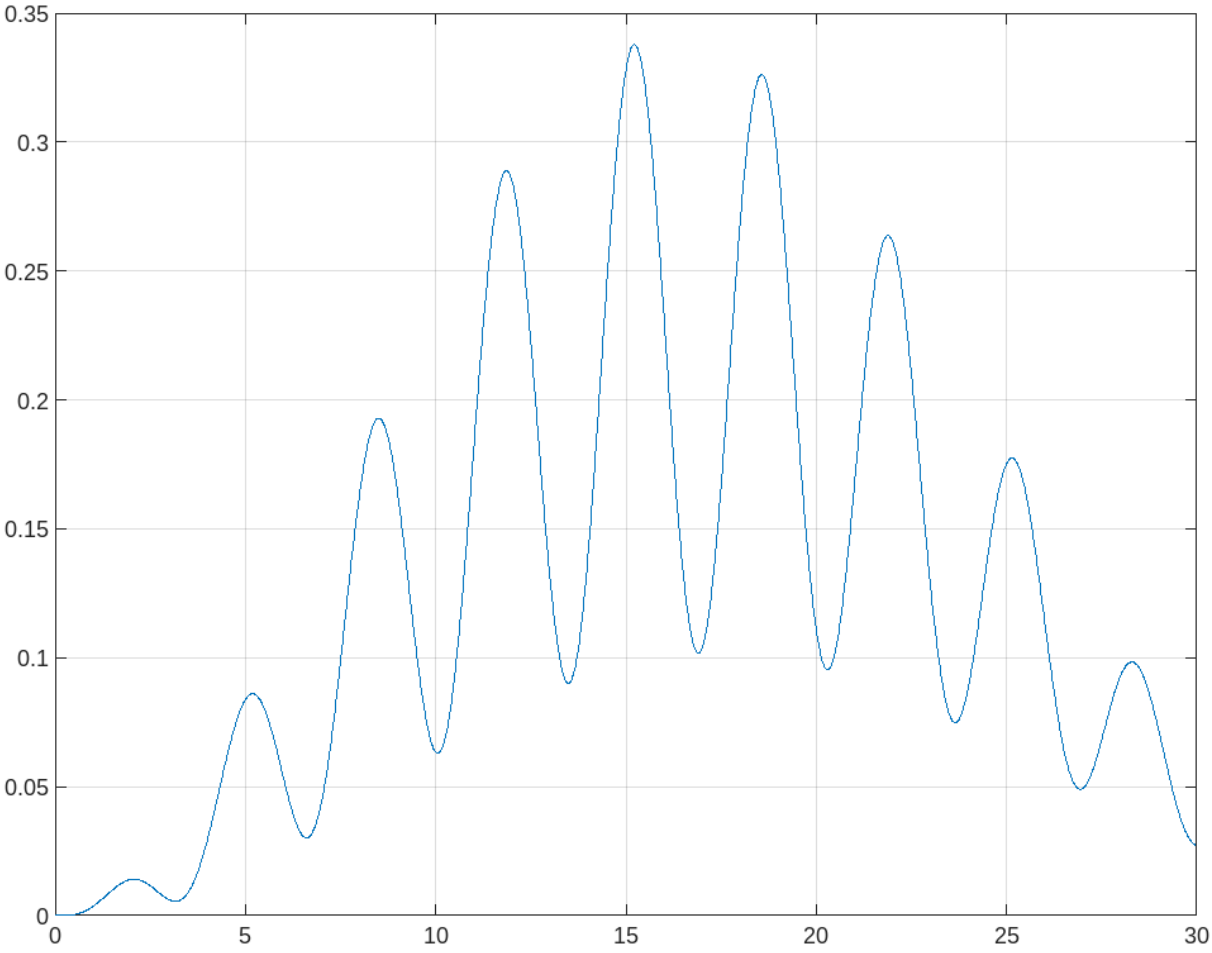
Figure 15. Zoom of the traces at boundary conditions, compared with default Maltab spline function

Part 3 - Power Dissipation Analysis

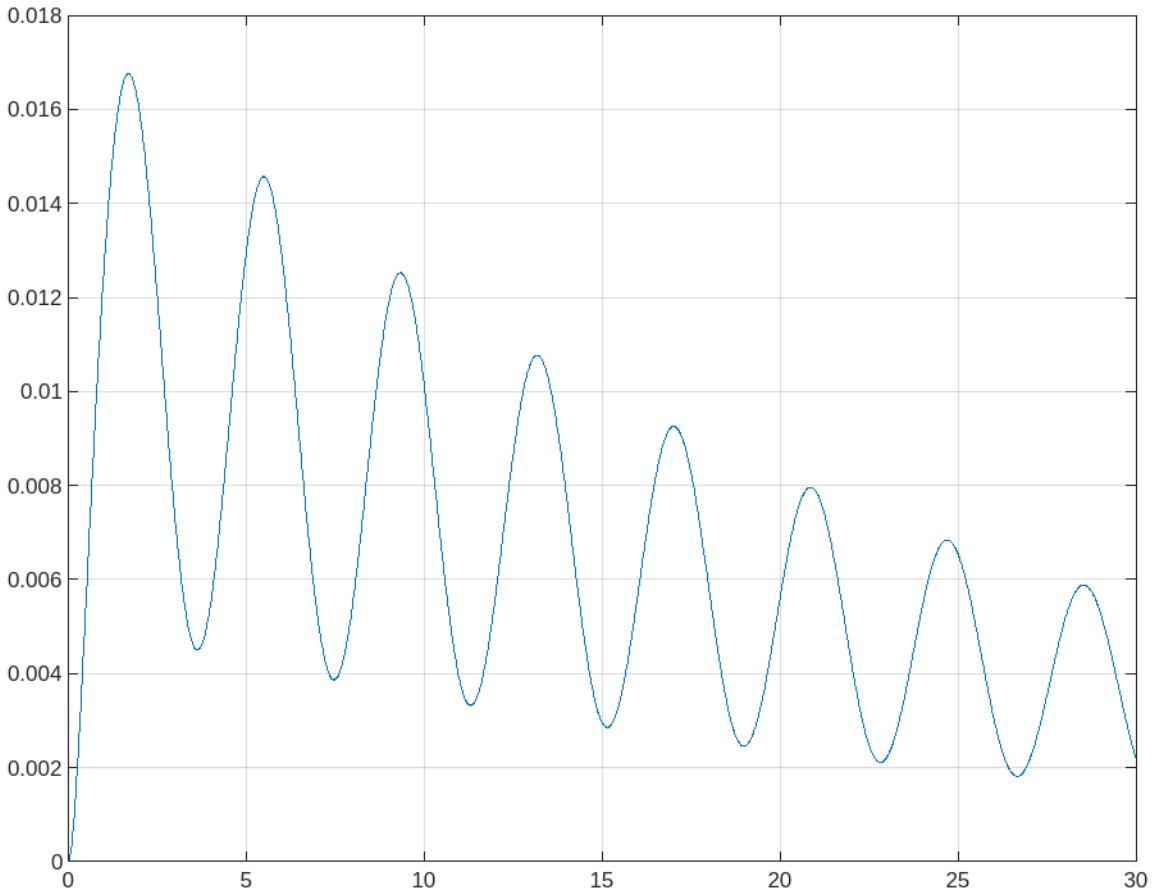
The total power dissipated in the resistors R1 and R2 was calculated by numerically integrating the instantaneous power p(t) = uR1(t)*i1(t) + uR2(t)*i2(t) from 0 to 30s, using both the composite rectangle and trapezoidal integration rules. This was done for the different excitation signals, with both the linear and non-linear M models, and using short and long integration step sizes.

		Rectangular		Trapezoidal	
		Short h	Long h	Short h	Long h
e(t) = 1 V	Linear (Constant) M	0.2031	0.4884	0.2032	0.4890
	Nonlinear M	0.1803	0.4531	0.1804	0.4539
e(t) = 240sin(t)	Linear (Constant) M	2.1337e+05	2.2234e+05	2.1337e+05	2.2235e+05
	Nonlinear M	1.5566e+05	1.6253e+05	1.5566e+05	1.6253e+05
e(t) = 210sin(2πft) V, for f = 5 Hz	Linear (Constant) M	34.8262	35.7914	34.8268	35.7972
	Nonlinear M	21.2150	21.8049	21.2154	21.8087

Table 1. Values of the dissipated power P



Quick validation of results: for e(t) = sin(t), the total power dissipated looks to be about 3W.



Plot for e(t) = 1 V. Graphically, the area under the curve seems to agree with the numerical results – below 1W of power dissipated.

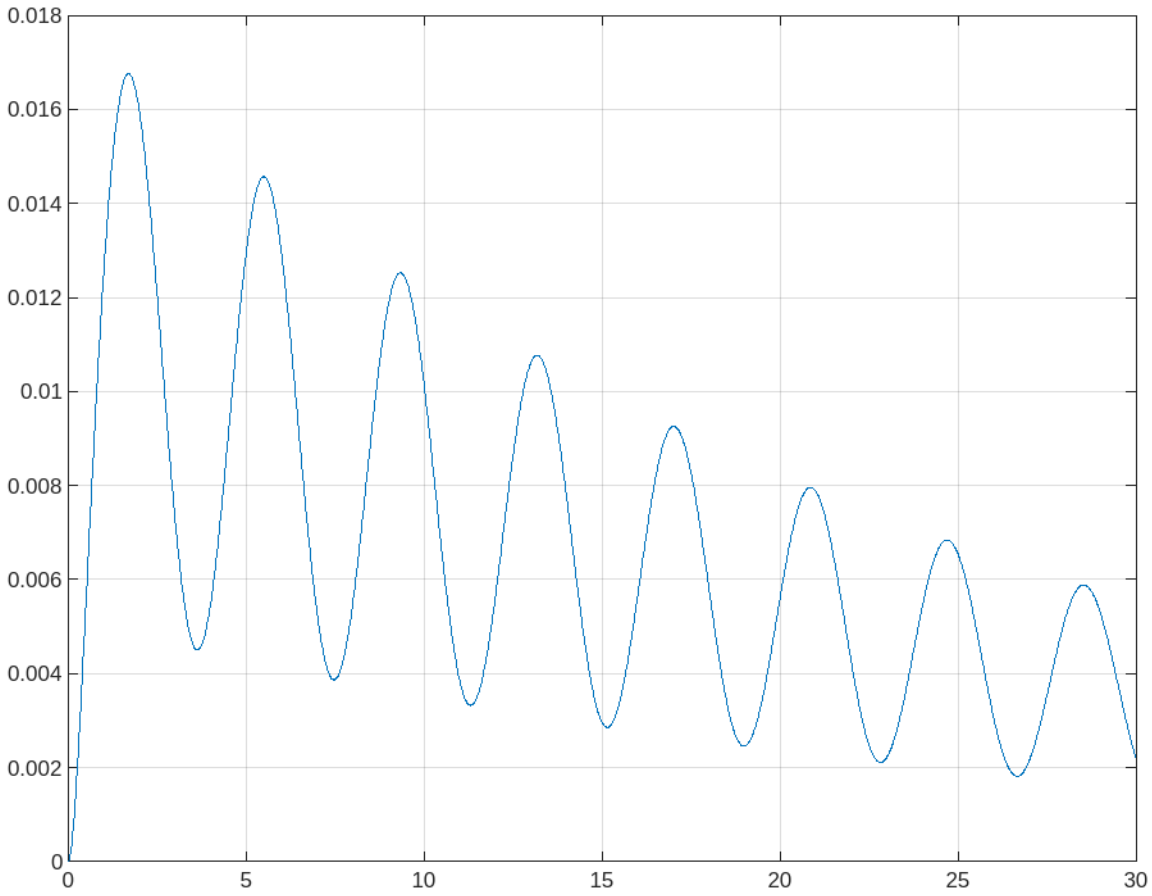
In general it seems that the rectangular and trapezoidal rule are performing similarly, although in high precision applications the small differences might be important.

The length of step h is definitely a crucial part of the experiment, for e=1V the differences in results between h=0.1 and h=0.01 are significant (over 2x).

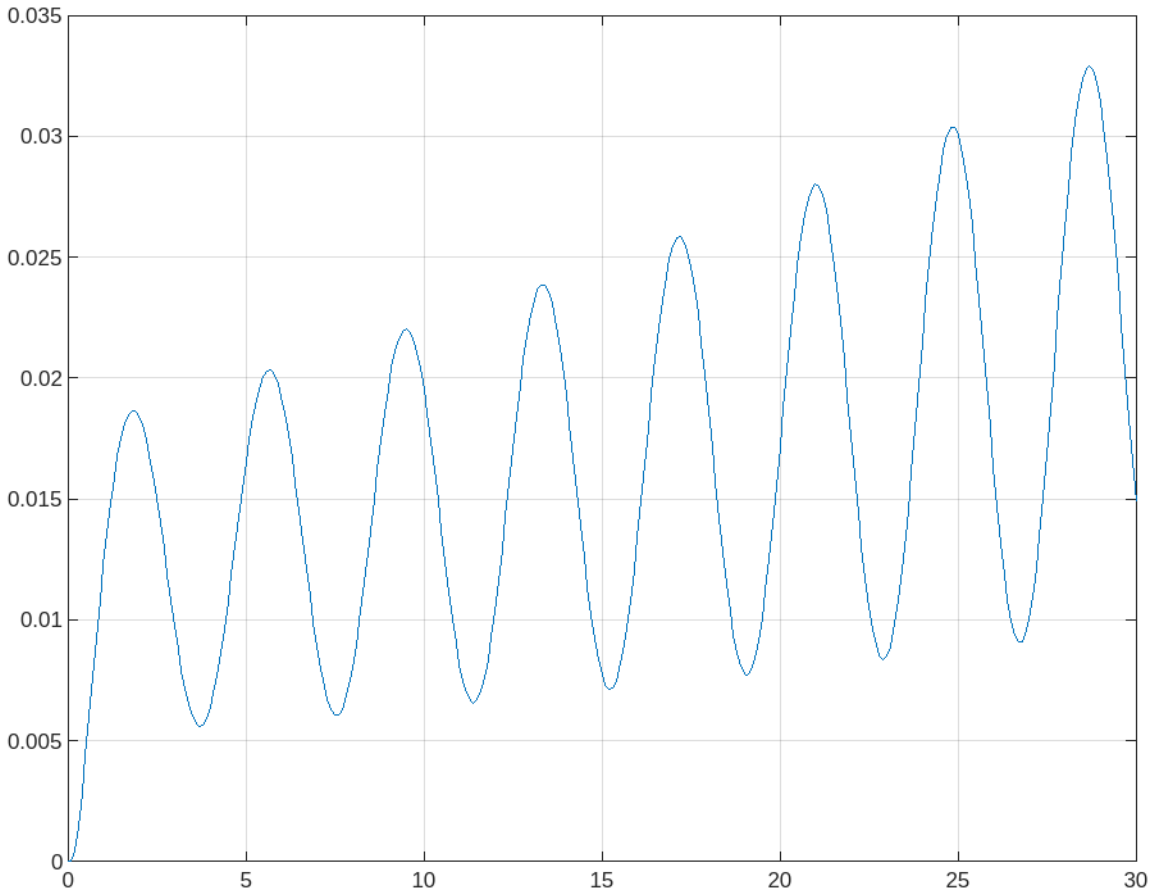
For e(t) with f=5Hz choosing a very long h resulted in absurd results since the simulator cannot properly generate at too high intervals.

The differences between linear and nonlinear M are most significant in case of including the frequency, presumably it has a big effect on power variations when the M is nonlinear. The more stable e(t), the less difference is observed (small difference for e=1V).

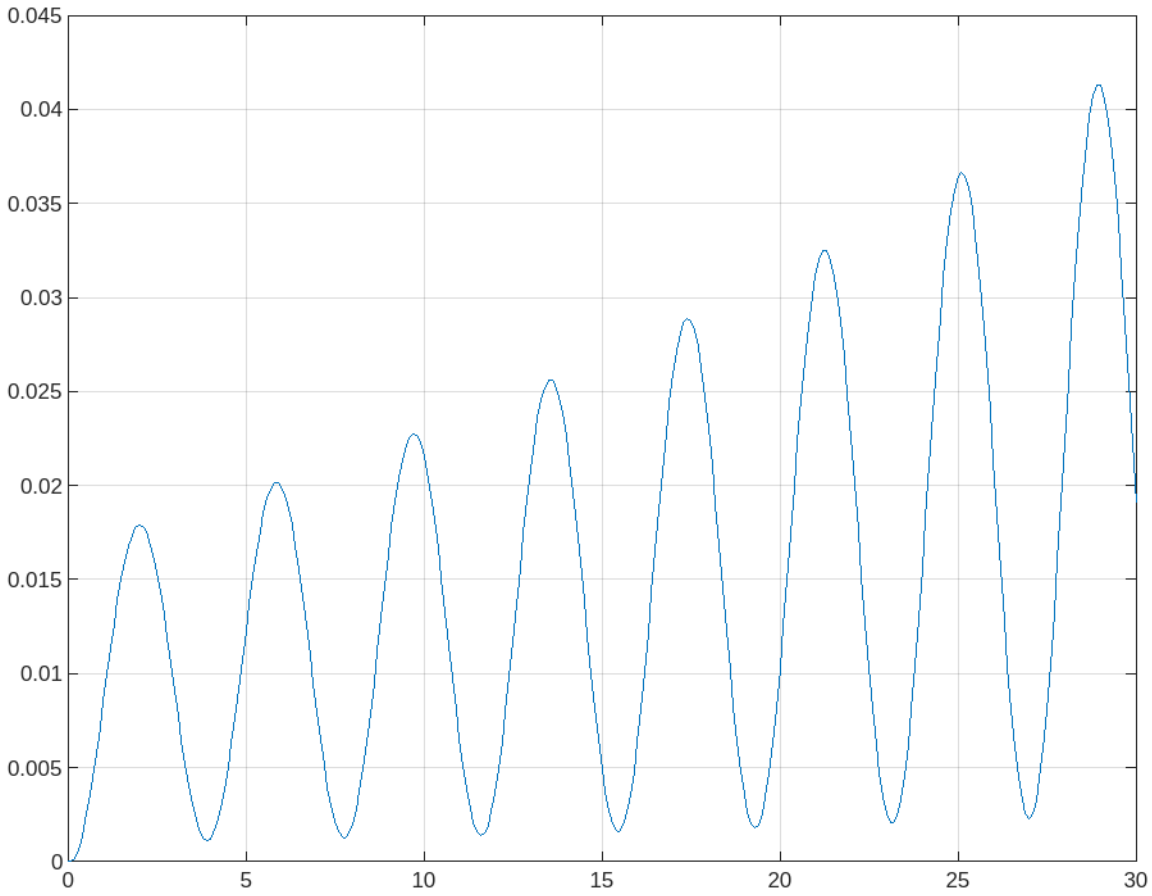
Time traces of the integrand expression ($R_1 i_1^2(t) + R_2 i_2^2(t)$) for the following cases:



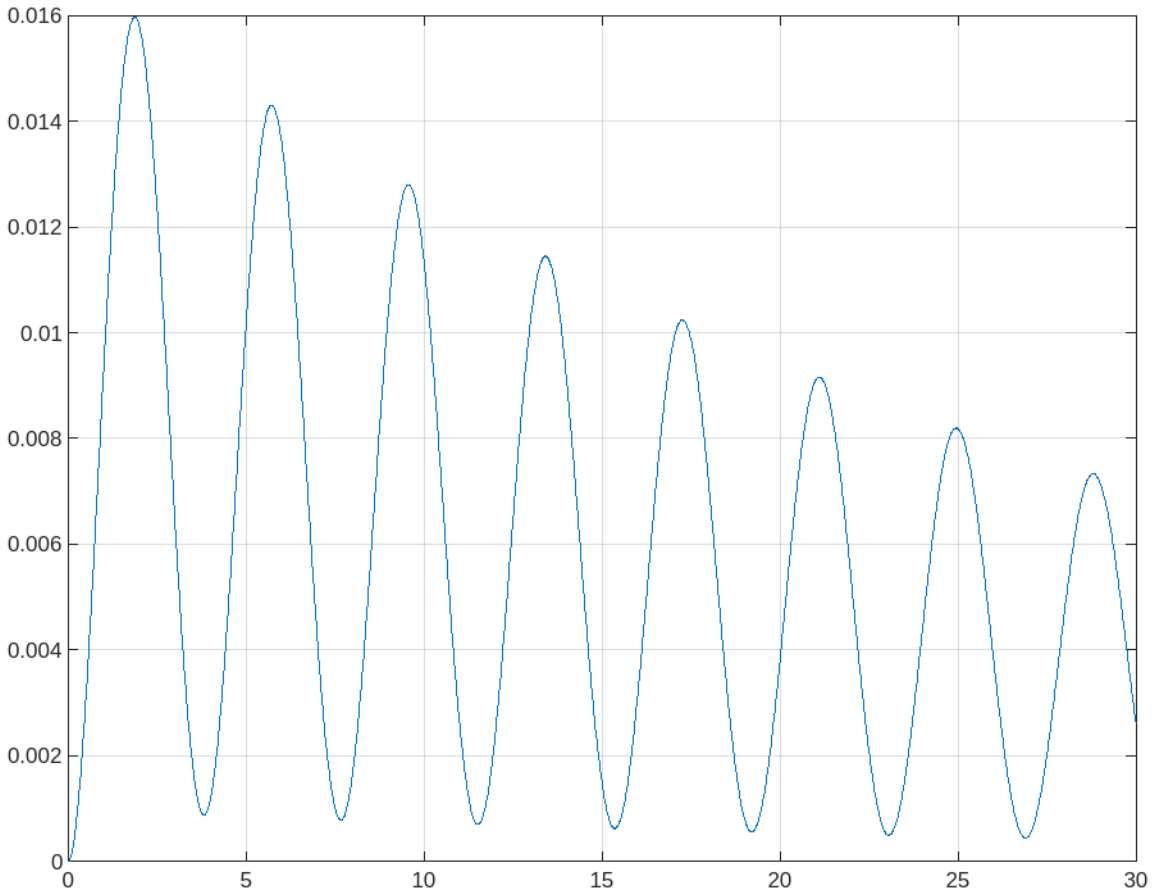
e(t) = 1 V, linear M, short H



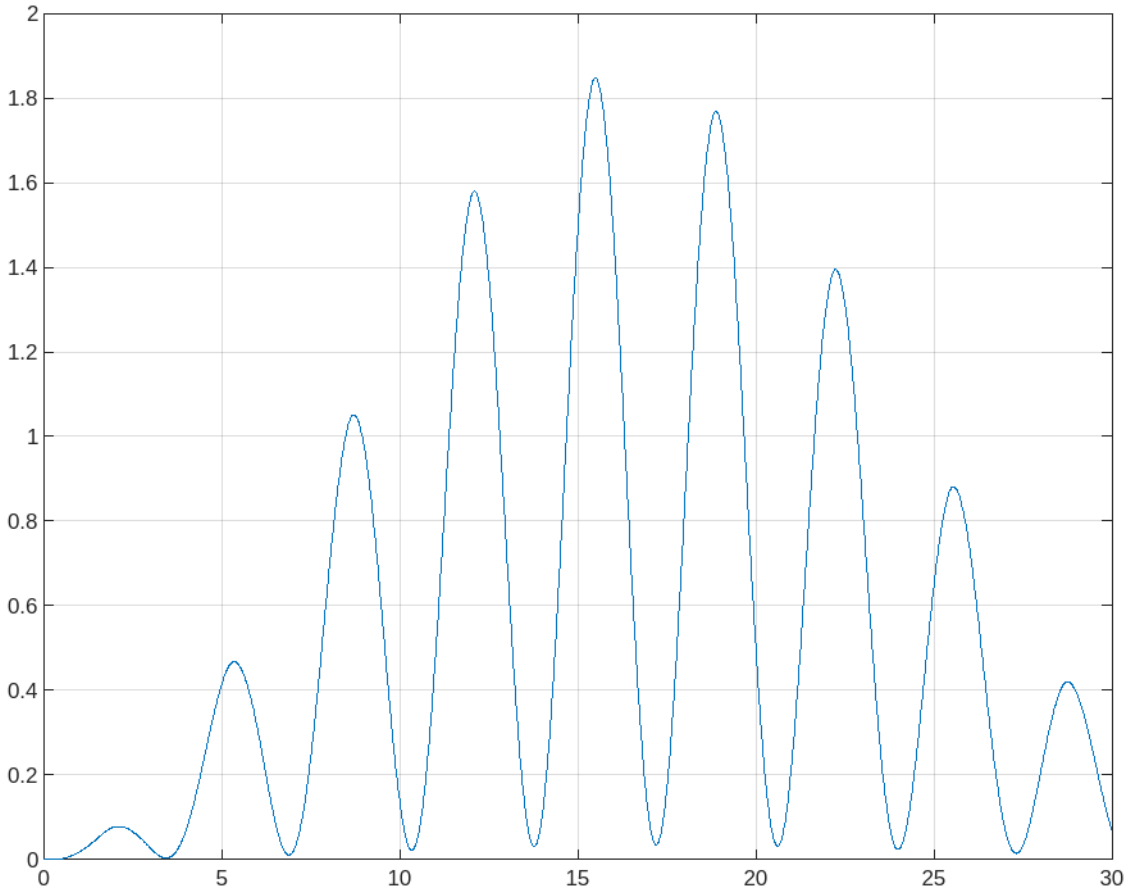
e(t) = 1 V, linear M, long H



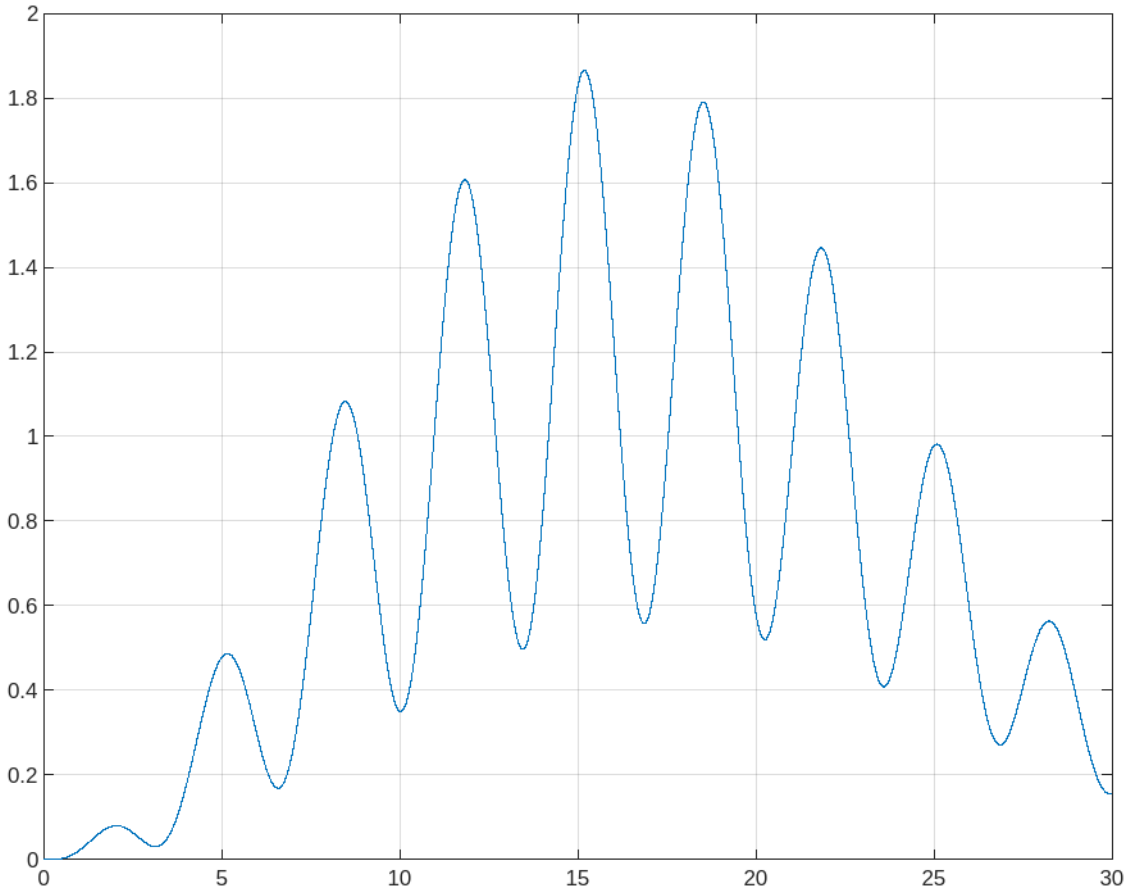
e(t) = 1 V, non-linear M, short H



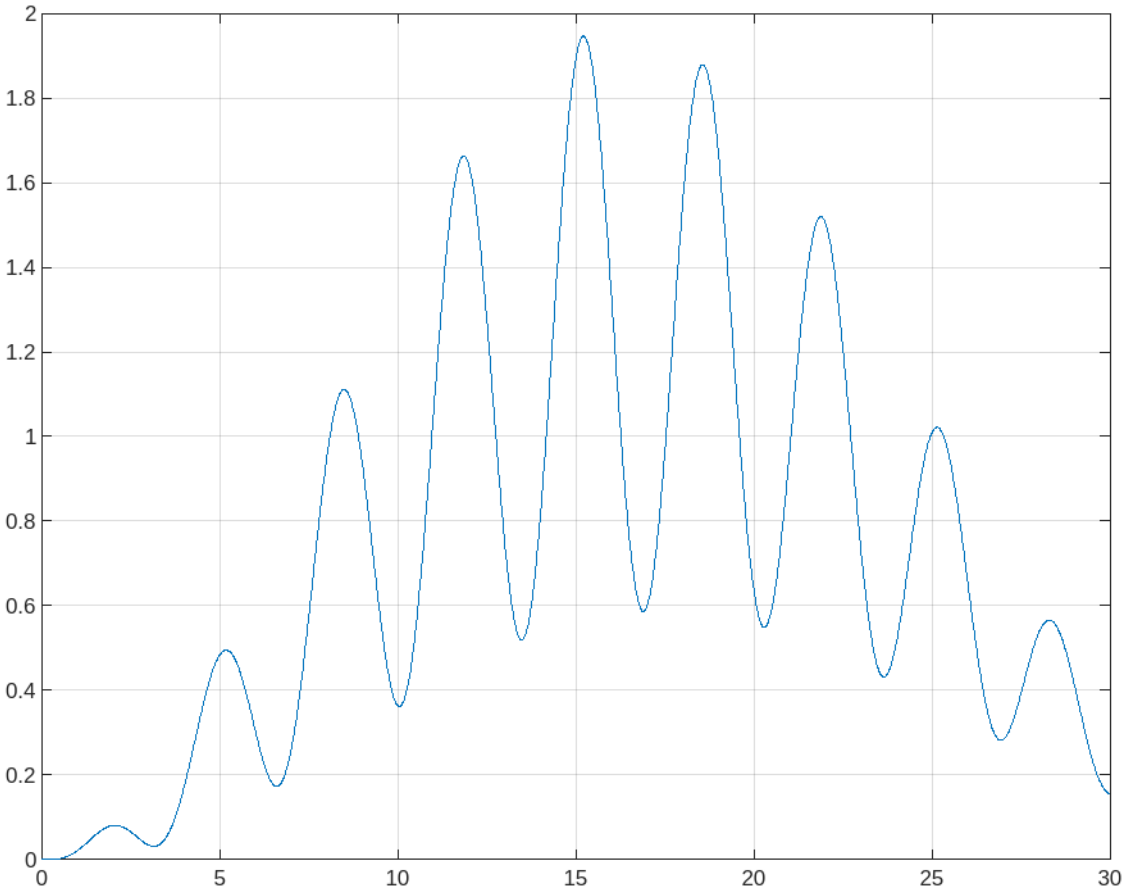
e(t) = 1 V, non-linear M, long H



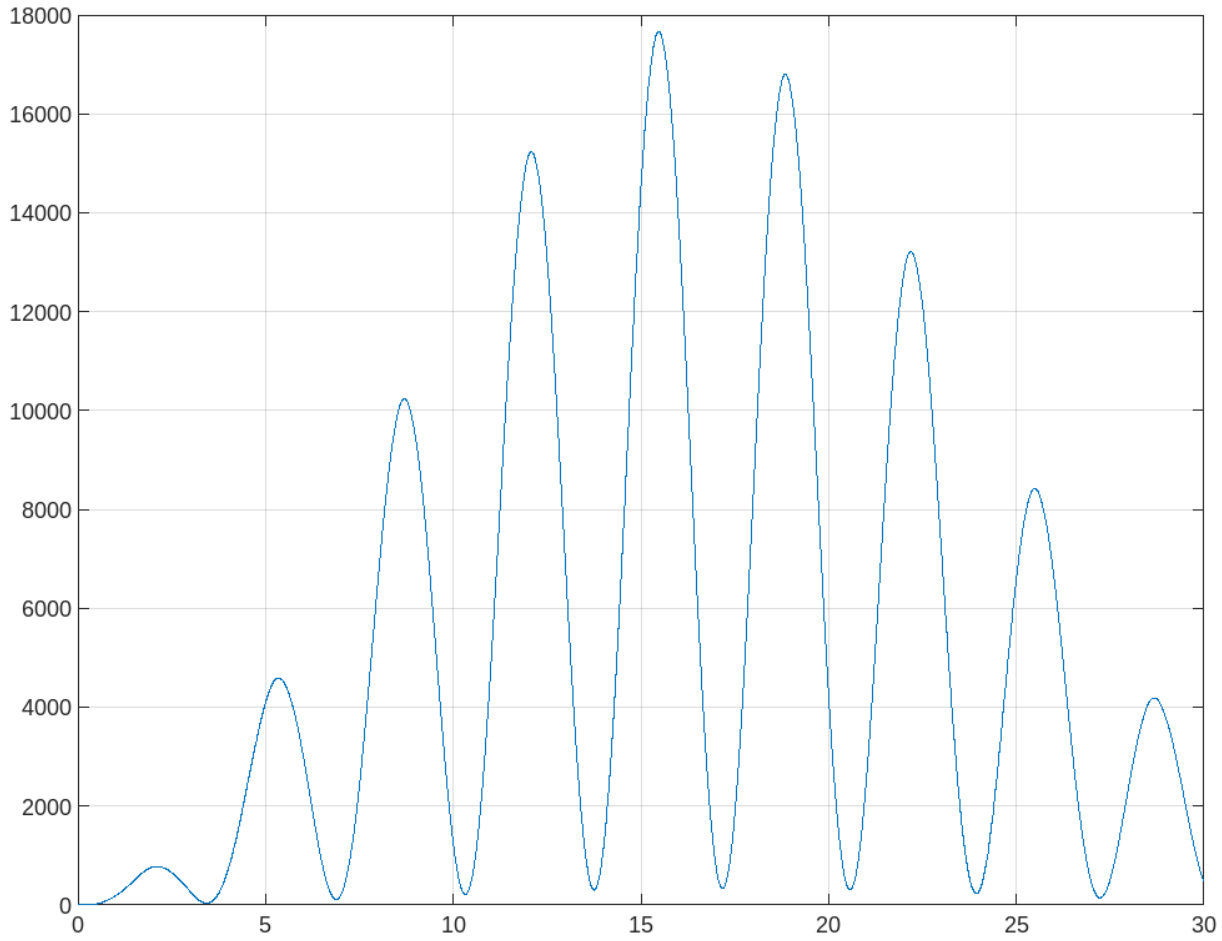
e(t) = 240sin(t) V, linear M, short H



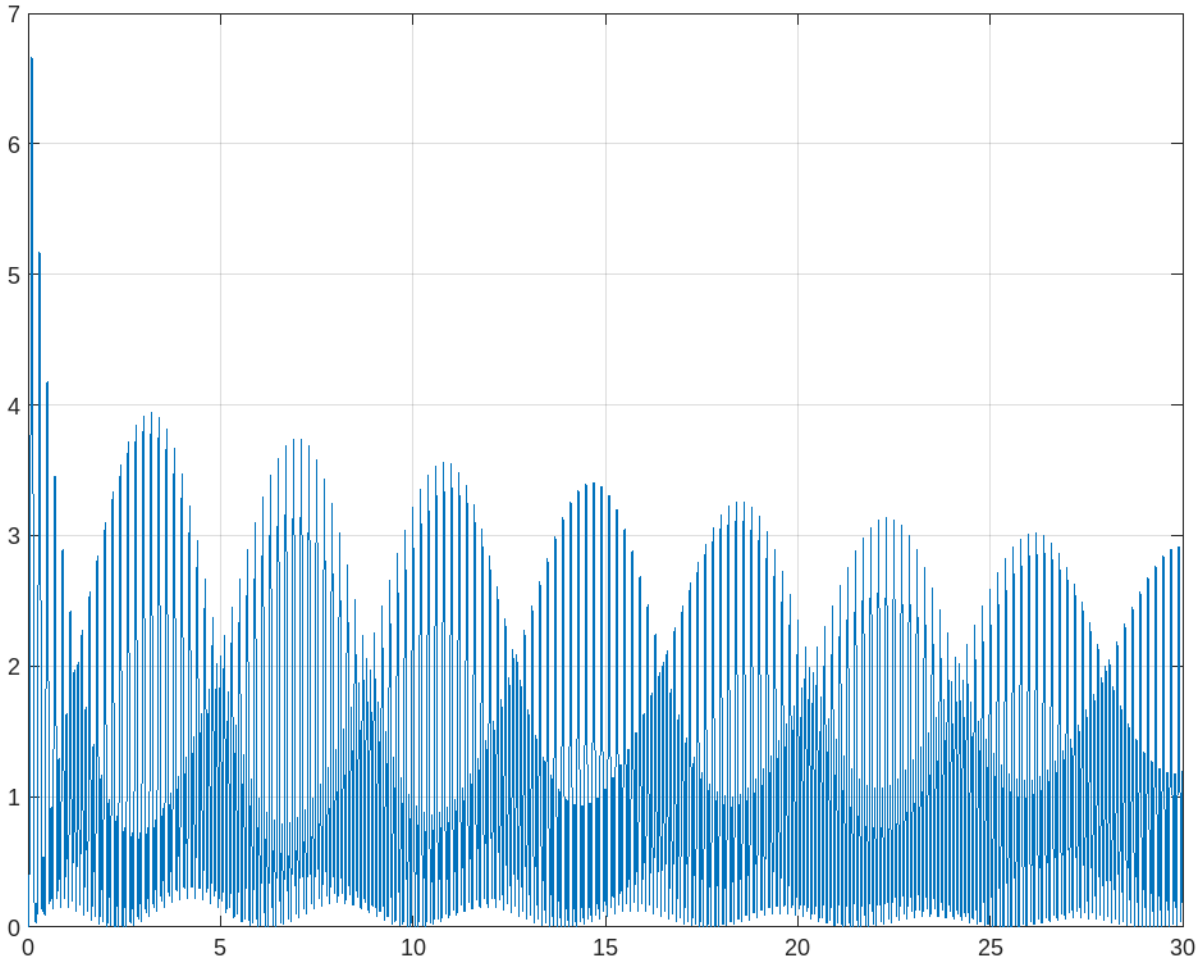
e(t) = 240sin(t) V, linear M, long H



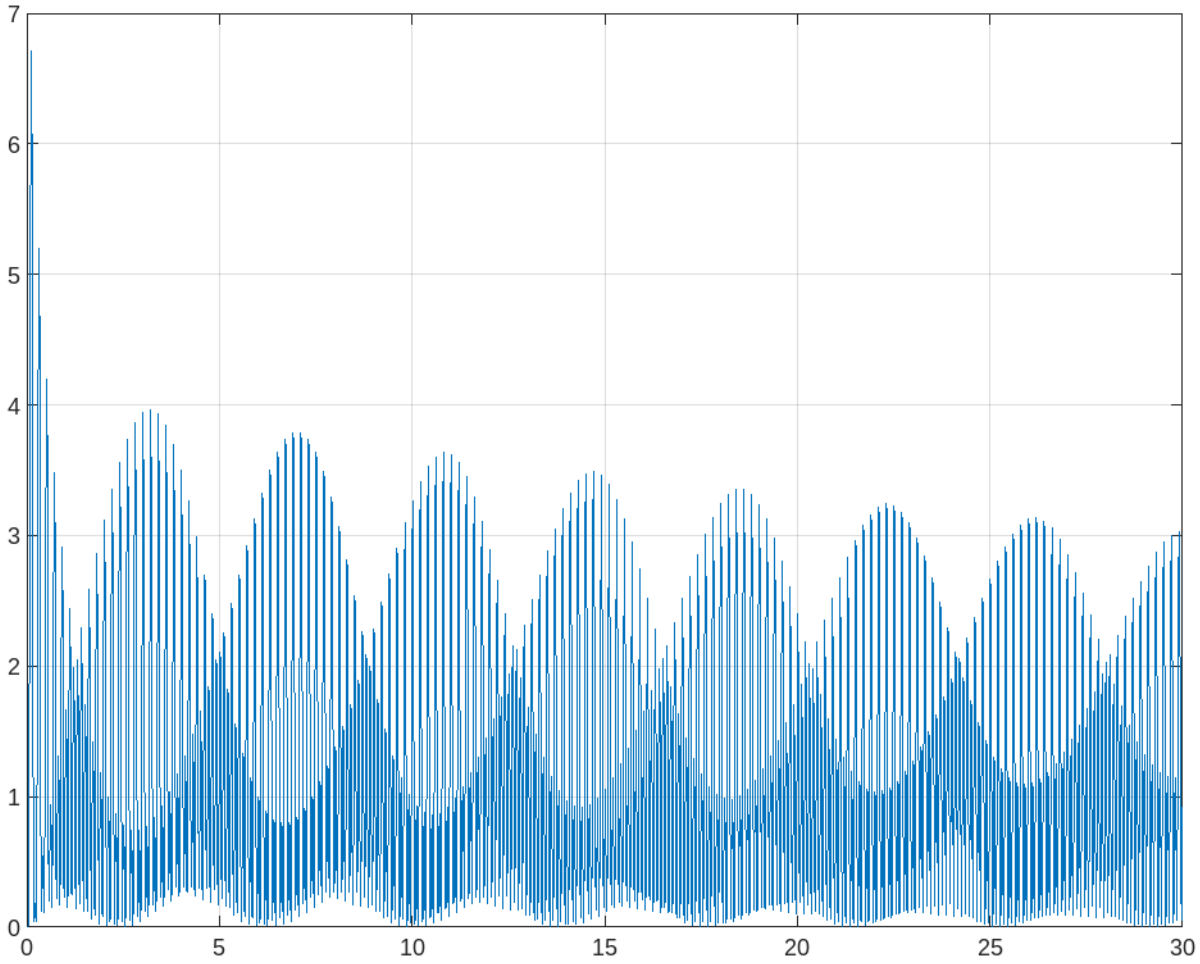
e(t) = 240sin(t) V, non-linear M, short H



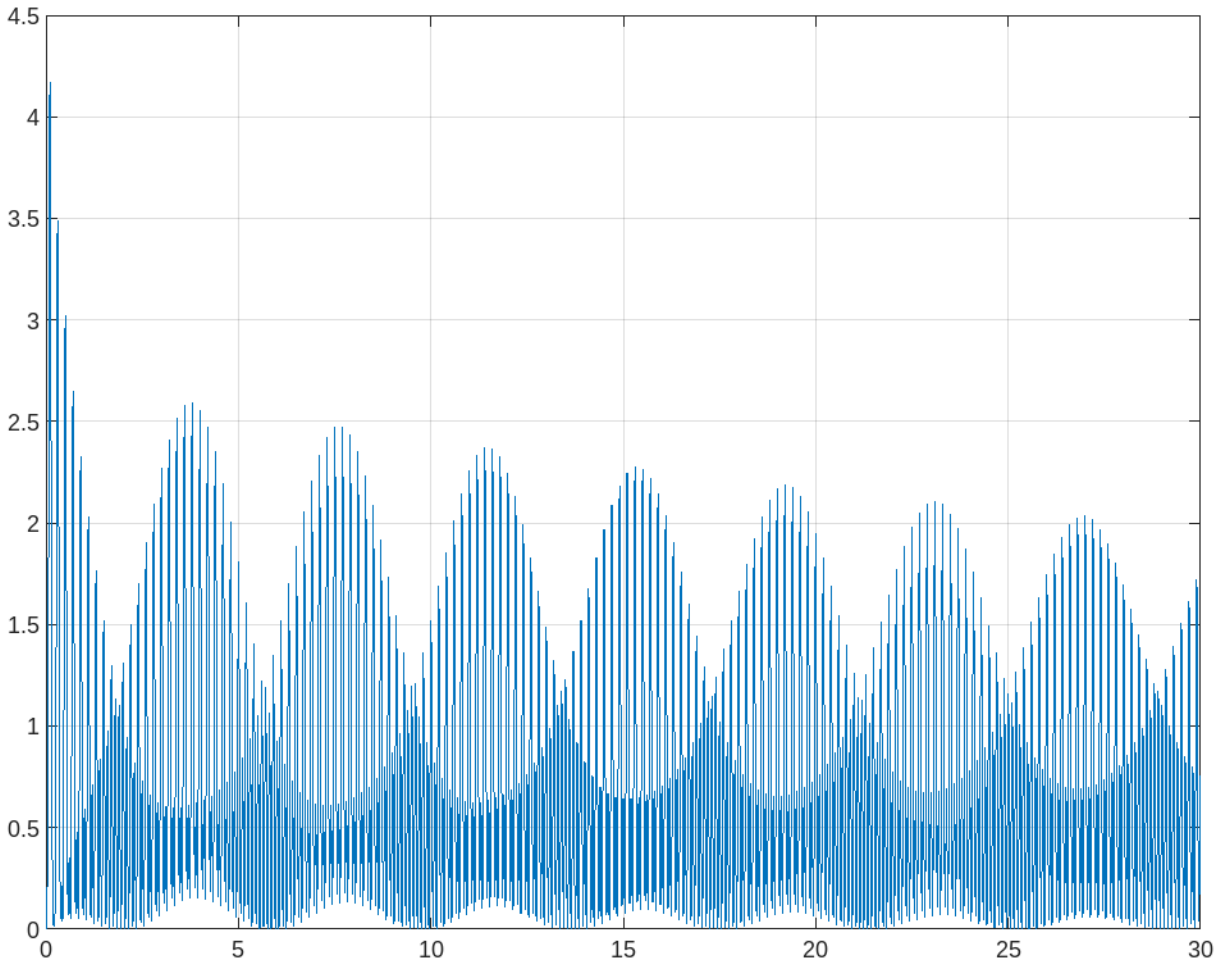
e(t) = 240sin(t) V, non-linear M, long H



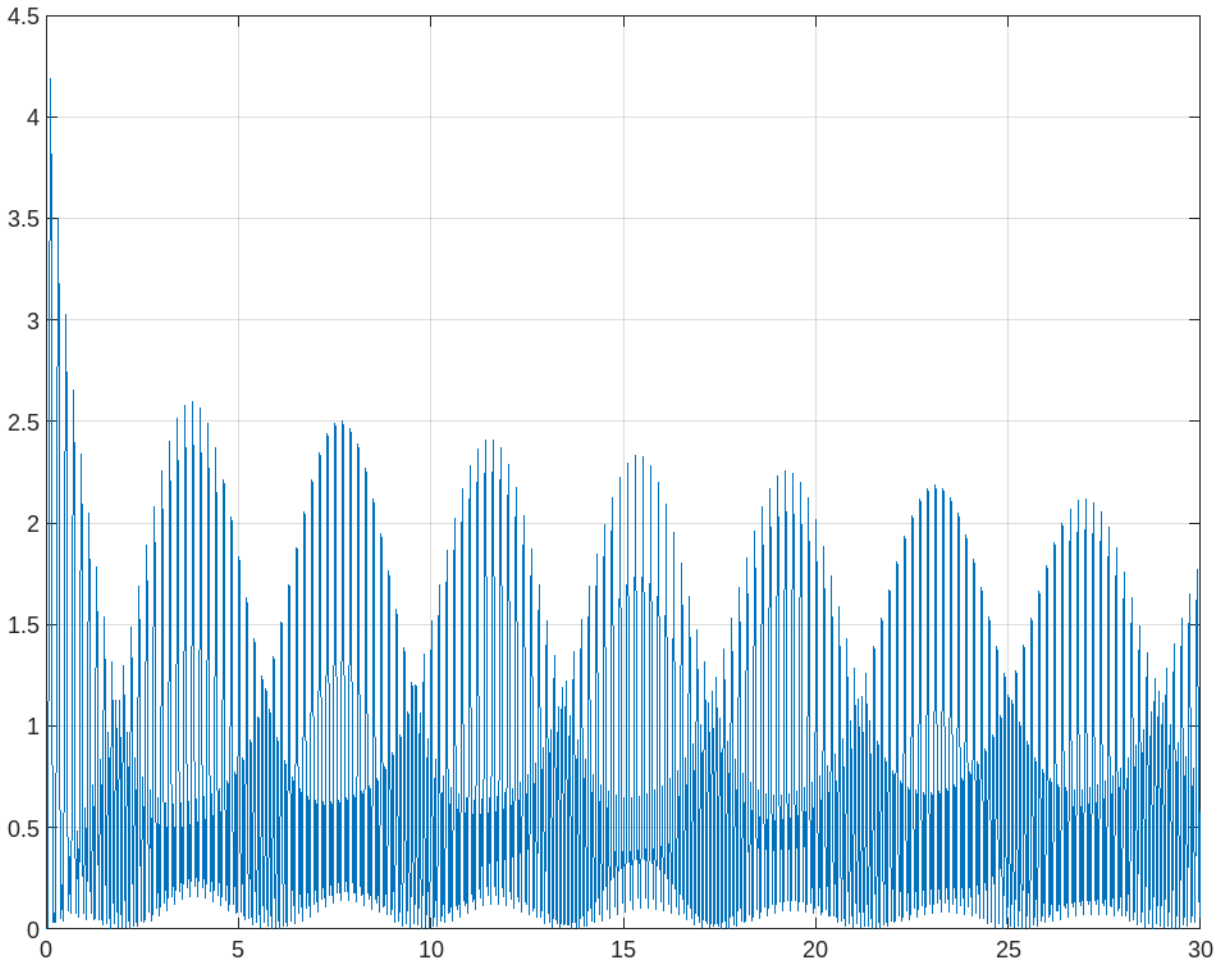
e(t) = 210sin(2πft) V, for f = 5 Hz, linear M, short H



e(t) = 210sin(2πft) V, for f = 5 Hz, linear M, long H



e(t) = 210sin(2πft) V, for f = 5 Hz, non-linear M, short H



e(t) = 210sin(2πft) V, for f = 5 Hz, non-linear M, long H

Part 4 - Finding Input Frequency for Target Power

This part solved for the input frequency f that would result in a total dissipated power of 406W from 0 to 30s. Three root-finding methods were implemented - bisection, secant, and Quasi-Newton.

For the bisection method it took 16 iterations and `43200` function evaluations to arrive at `P=406.0046` and frequency of `0.8035`.

For the secant method 6 iterations and `19200` function evaluations were enough to arrive at `P=405.9987` and frequency of `0.8035`.

For the Quasi-Newton method 15 iterations and `69600` function evaluations were needed to arrive at `P= 405.9979` and frequency of `0.8035`.

It seems that the secant method was the most efficient, requiring only 6 iterations and 19200 evaluations of P(f) to find the solution `f = 0.8035 Hz.`