transformer_hysteresis

May 15, 2018

```
In [1]: %pylab --no-import-all inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: plt.rcParams['figure.dpi'] = 200
```

```
In [3]: import xarray_dsp as dsp
```

```
In [4]: import golem_db_access as gdb
```

```
In [5]: from scipy import integrate, optimize, interpolate
```

1 Analysis of vacuum discharge 26768

This discharge had a high $U_{cd} = 800$ V in order to saturate the transformer.

```
In [6]: s = gdb.Shot(26768)
```

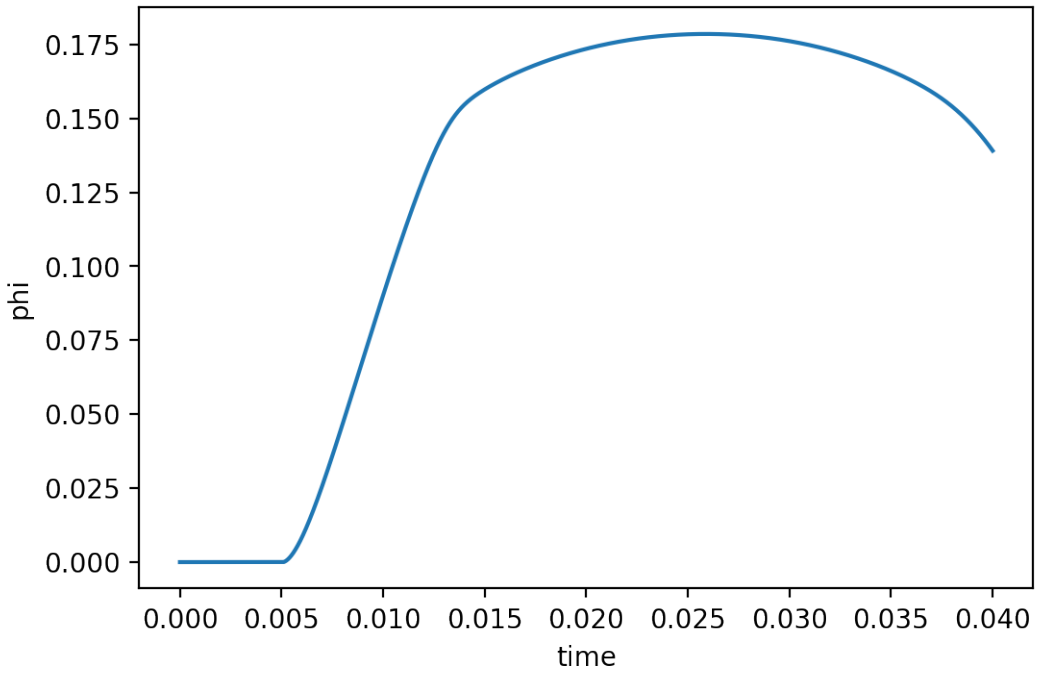
```
In [7]: u_l = s.get('loop_voltage')
```

The loop measures the derivative of the **total** magnetic flux ϕ through the iron core.

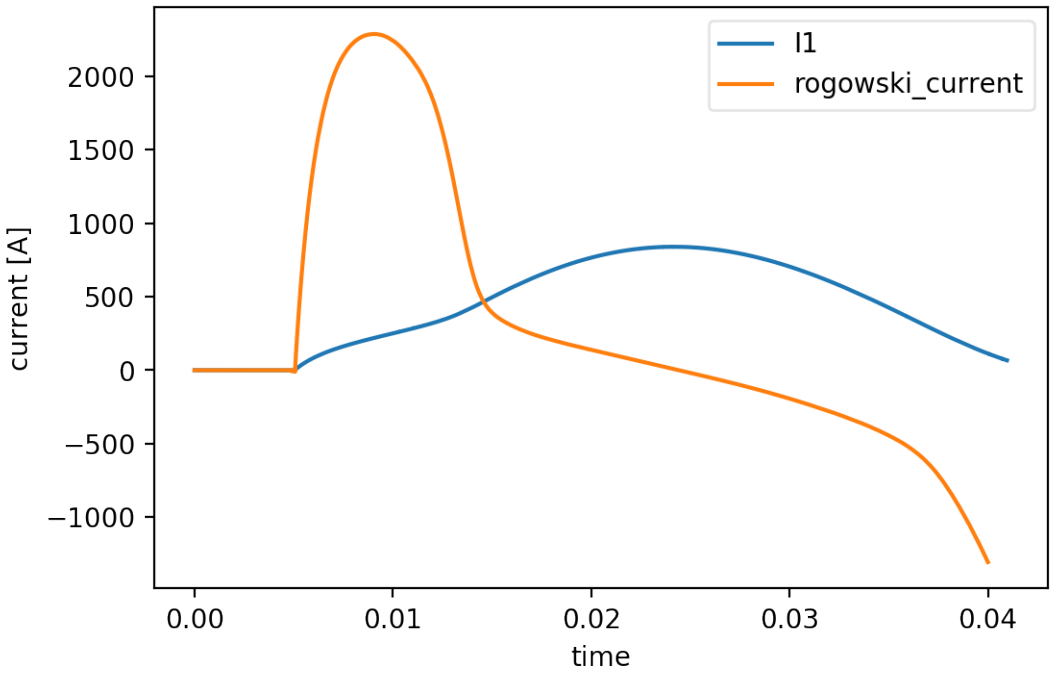
```
In [8]: phi = integrate.cumtrapz(u_l, u_l.time, initial=0)
        phi = u_l.__array_wrap__(phi).rename('phi')
```

```
In [9]: phi.plot()
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x7fbc3af02ef0>]
```




```
In [10]: I1 = s.get('papouch_za').sel(channel=0)* (-200) # 0.5 mV/A
         I1.name = 'I1'

In [11]: Ich = s.get('rogowski_current')

In [12]: I1.plot()
         Ich.plot()
         plt.legend()
         plt.ylabel('current [A]');
```

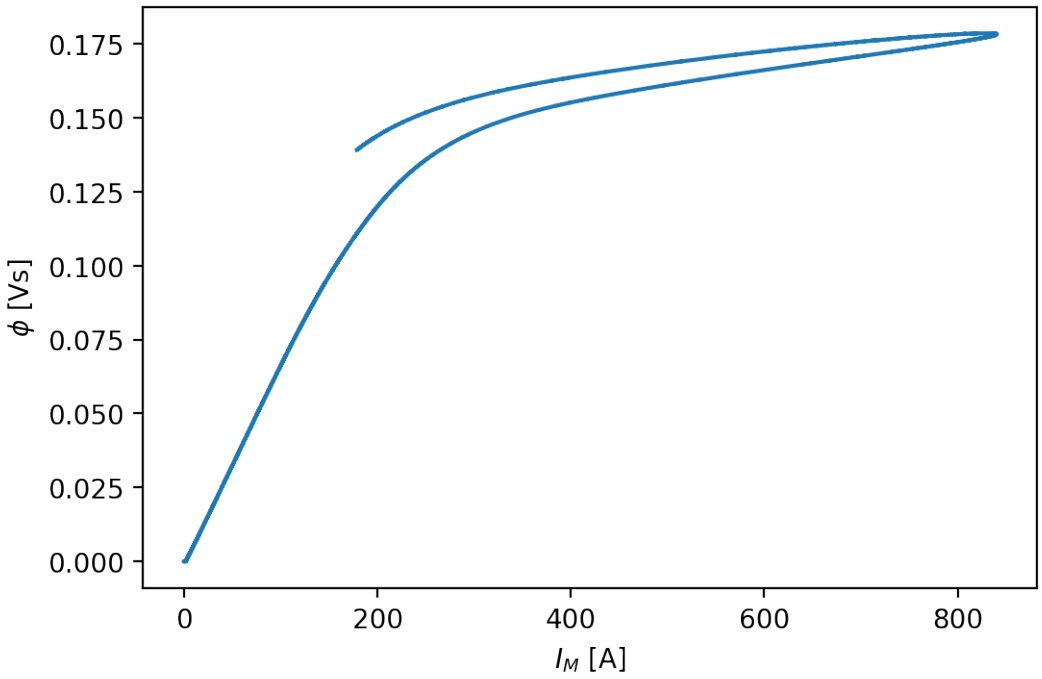


According to Valovic90 the total flux should be compared with the magnetization current $I_M = I_1 - I_{ch}/N$ where I_1 and I_{ch} are the currents in the primary winding and the chamber, respectively. N is the number of turns in the primary winding.

The magnetization current can be intepreted as giving rise to the total magnetic flux ϕ as the difference of the magnetic flux from the primary winding and the chamber $\phi = \phi_1 - \phi_{ch}$, because due to Lenz's law the chamber flux tries to reduce the primary flux inducing the chamber current.

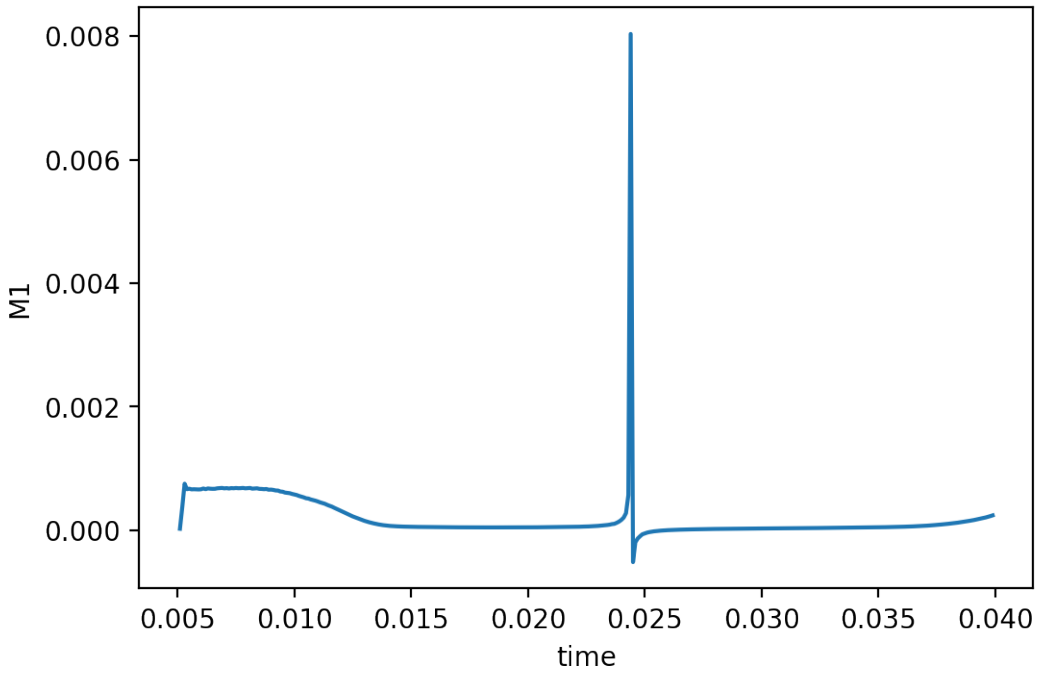
For this reason Valovic and Stockel add a secondary inductance $L_2 = M/N$ into the chamber and/or the plasma circuit. **The transformer theory should be revisited.**

```
In [13]: sl_pl = slice(5e-3, 40e-3)
         N1 = 5*4 # turns of primary winding ?
         #(should be 6*4, but one bunch may be disconnected?)
         I_M = I1.loc[sl_pl] - Ich.loc[sl_pl].values / N1
         I_M.name = 'I_M'
         phi_mag = phi.loc[sl_pl]
         plt.plot(I_M, phi_mag)
         plt.xlabel('$I_M$ [A]')
         plt.ylabel('$\\phi$ [Vs]')

Out[13]: Text(0,0.5,'$\\phi$ [Vs]')
```

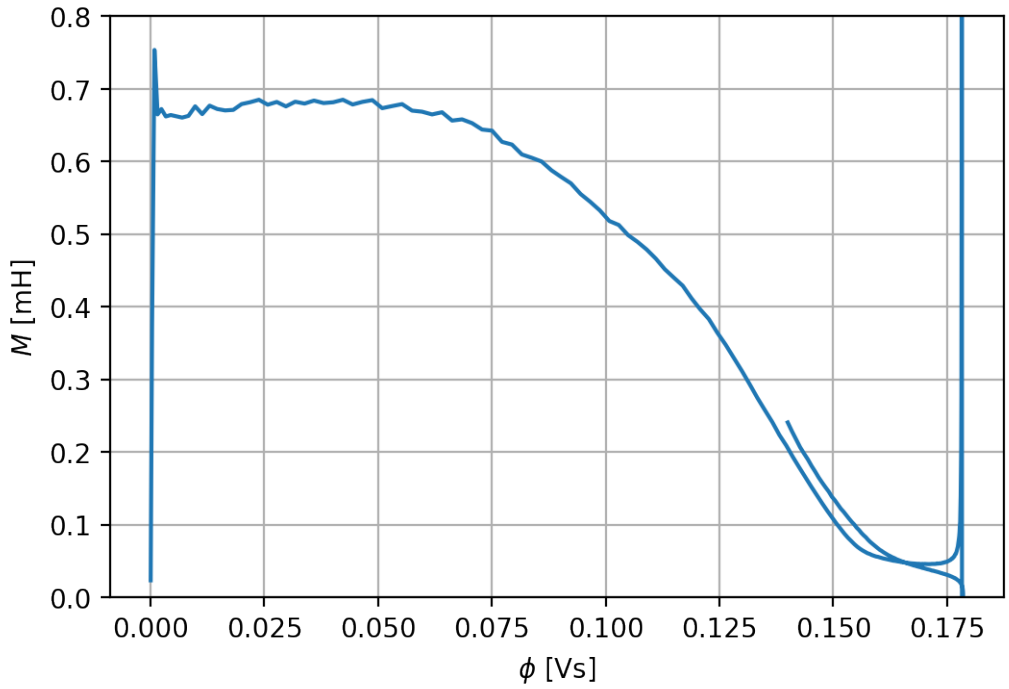


With the proper choice of N the slope is constant at the beginnig. If $I_M = I_1$ there would be a bump at the beginning and the slope would rapidly change (which is not physical).

```
In [14]: # decimate and smooth signals for derivative calculation
         q_dec = 100
         phi_clean = dsp.decimate(phi_mag, q_dec)
         I_M_clean = dsp.decimate(I_M, q_dec)

In [15]: Ich_clean = dsp.decimate(Ich.loc[sl_pl], q_dec)

In [16]: M1 = phi_clean.diff(dim='time') / I_M_clean.diff(dim='time').values
         M1.name = 'M1'

In [17]: M1.plot();
```




```
In [18]: plt.plot(phi_clean[1:], M1*1e3);
         plt.ylim(0, 0.8)
         plt.ylabel('$M$ [mH]')
         plt.xlabel('$\phi$ [Vs]')
         plt.grid()
```



The unstaturated level of mutual inductance is about $M \approx 0.68$ mH. Beyond $\phi = 0.05$ Vs the transformer begins to **gradually** saturate towards $M \approx 0.05$ mH. The function could be likely easily approximated by

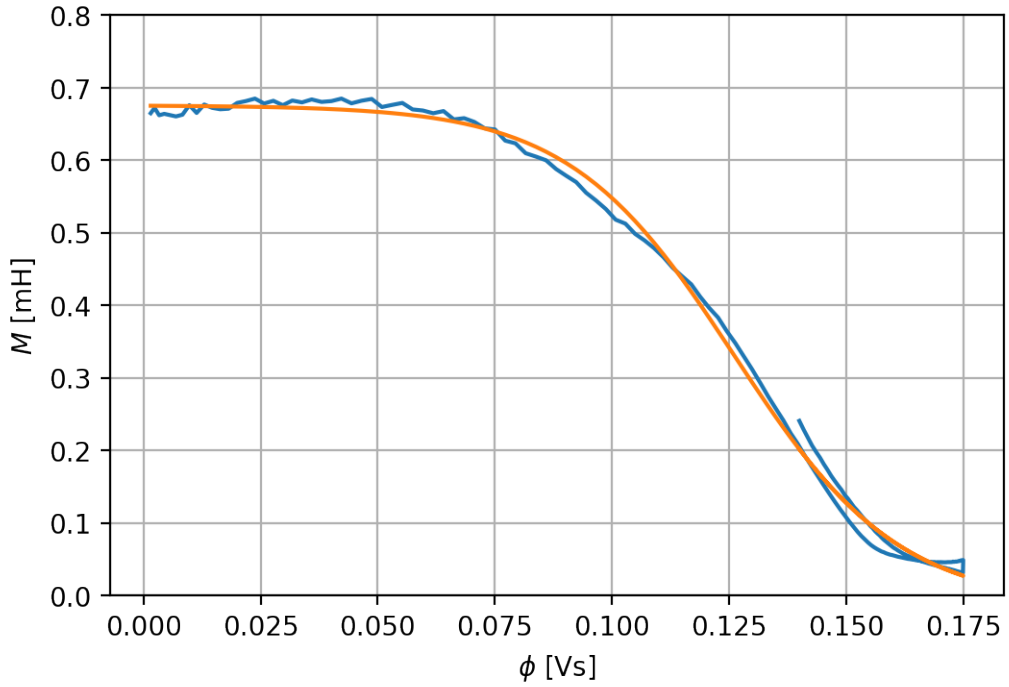
$$M(\phi) = \frac{M_0}{1 + \exp\left(a(\phi - \phi_s)\right)} + M_s$$

```
In [19]: def M_fitfunc(phi, M_0, a, phi_s, M_s):
             return M_0 / (1 + np.exp(a * (phi - phi_s))) + M_s

In [20]: from scipy.optimize import curve_fit

In [21]: fit_win = (0.001 < phi_clean[1:]) & (phi_clean[1:] < 0.175)
         phi_fit = phi_clean[1:][fit_win]
         M1_fit = M1[fit_win] * 1e3

In [24]: p, cov = curve_fit(M_fitfunc, phi_fit, M1_fit, p0=[0.68, 0.1, 0.05, 0.05])

In [25]: sd = np.sqrt(np.diagonal(cov))

In [27]: import pandas as pd

In [28]: df_fit_res = pd.DataFrame({'fit': p, 'std': sd},
                                   index=['M_0', 'a', 'phi_s', 'M_s'])
         df_fit_res
```

Out[28]:

	fit	std
M_0	0.688543	0.004886
a	56.800941	1.068281
phi_s	0.126061	0.000343
M_s	-0.012678	0.003643

In [26]:
```
plt.plot(phi_fit, M1_fit);
plt.plot(phi_fit, M_fitfunc(phi_fit, *p))
plt.ylim(0, 0.8)
plt.ylabel('$M$ [mH]')
plt.xlabel('$\phi$ [Vs]')
plt.grid()
```



Proper determination of M and L_2 and N and their optimization could prolong the discharge!