Supplement to 'Trapdoor viscous remanent magnetization'

Karl Fabian
NTNU, S. P. Andersens veg 15a, Trondheim, Norway
karl.fabian@ntnu.no

October 2023

1 Description of Python routines for trapdoor VRM

The following Python routines are used to model trapdoor VRM processes. The corresponding Jupyter notebooks are provided.

1.1 Nearly uniform distribution of points on the sphere

The first function is generating a numpy array of unit vectors in three dimensions, that nearly uniformly covers the unit sphere and contains for each vector also the antipodal vector.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import SphericalVoronoi
import math
from numba import njit

def fibonacci_symsphere(samples=100):  # Points on a sphere
    distributed by golden section spiral, but with exact antipodes

    points = []
    phi = math.pi * (math.sqrt(5.) - 1.)  # golden angle in radians

    for i in range(samples):
        z = 1 - (i / float(samples - 1))   # z goes from 1 to 0
        radius = math.sqrt(1 - z * z)      # radius at z

        theta = phi * i  # golden angle increment

        x = math.cos(theta) * radius
        y = math.sin(theta) * radius

        points.append((x, y, z))         #  add point
        points.append((-x, -y, -z))      #  add antipode

    return np.array(points)  # returns 2 * samples  points !!
```

1.2 Fast calculation of trapdoor VRM

To use the numba in-place compiler '@njit' for accelerating the nested summing involved in tVRM calculation, a fast function is defined that lies outside the later defined class 'Trap_VRM_Density'. This function performs the main task in tVRM modeling by updating the occupation ρ for each parameter combination of barrier w (`E1`), magnetic moment energy q (`MH`), and magnetic moment direction $\mathbf{u}$ (`dirs[k]` weighted by `dir_weights[k]`). The updating is done by performing a time step t of the exponential decay using the individually calculated appropriate decay constant p_{12}. The loop also calculates the average magnetic moment after the time step by adding all moments $q\,\mathbf{u}$ with their correct weight and occupation density.

```
@njit
def _fast_Trap(b,m,d,fd,t,E1,MH,dirs,rho,dir_weights):
    avmag=np.zeros(3)
    for i in range(b):
        for j in range(m):
            for k in range(d):
                p12=np.exp(-(E1[i]+MH[j]*np.dot(fd,dirs[k])) )  # decay constant for specific barrier
                rho[i,j,k]*=np.exp(-p12*t)  # updating decay of state density during time t
                avmag+=dirs[k]*MH[j]*rho[i,j,k]*dir_weights[k]  # contribution to total magnetization
    return avmag
```

1.3 Trapdoor VRM density class

The main functions to define and calculate tVRM models are encapsulated in the 'Trap_VRM_Density' class. It has three parameters containing the list of energy barriers w, the list of q values and the list of magnetic moment directions $\mathbf{u}$. With initialization the lengths of these lists are calculated and also the weights of the individual directions through the area of their spherical Voronoi cell. Initially a constant density is assigned to all states, but this can be changed to the density described in the main article by calling the function 'set_density' which requires α and σ as parameters. The evolution of the state occupation ρ is performed through the function 'density_evolve' that also uses the helper function 'time_rescale' to convert the human readable time scales m,h,d,a,ka,Ma,Ga into multiples of τ_0. The computationally challenging part of 'density_evolve' is performed by calling the previously described '_fast_Trap' function.

```
class Trap_VRM_Density:
    def __init__(self, bar, mag, directions):
        self.b=np.shape(bar)[0] #  Length of energy barrier list
        self.m=np.shape(mag)[0] #  Length of magnetic field energy list (for given h)
        self.d=np.shape(directions)[0] # Number of directions on the sphere

        self.w_min=bar[0]
        self.w_max=bar[-1]


        self.t0= 1e-9 # tau0 in unit seconds

        self.E1 = bar  #  Energy barrier list  in units k T
        self.MH = mag   #  Magnetic field energy list (for given h) in units k T
        self.dirs =directions  # Direction unit vectors on the sphere
        try:
            sv = SphericalVoronoi(directions, 1, np.array([0, 0, 0]))   # Voronoi tesselation with these centers
            self.dir_weights=sv.calculate_areas()/4/np.pi  # Weights of all Voronoi cells
        except:
```

```
            self.dir_weights=np.array([1 for dd in self.dirs])/np.shape(self.dirs)[0]

        self.constant_density()


    def constant_density(self):
        self.rho = np.ones((self.b,self.m,self.d))   #  all combinations are assumed to have the same initial probability
        self.history=[['constant density']]  #  clear history

    def set_density(self,alpha,sigma ):
        self.rho = np.ones((self.b,self.m,self.d))   #  all combinations are assumed to have the same initial probability
        i=0; j=0; k=0
        for i in range(self.b):
            rw= 1/(self.w_max-self.w_min)+ alpha * (self.E1[i]-0.5*(self.w_max+self.w_min))/(self.w_max-self.w_min)/(self.w_max-self.w_min)
            rw=2*rw/(sigma)/np.sqrt(2*np.pi)  # normalization parameter depends on i only
            for j in range(self.m):
                rq=rw*np.exp(-0.5*self.MH[j]*self.MH[j]/sigma/sigma)
                for k in range(self.d):
                    self.rho[i,j,k]= rq


        self.history=[['density reset',alpha,sigma]]  #  clear history

    def time_rescale(self,t,t_unit):
        if t_unit=='a':
            t*=60*60*24*365.25
        if t_unit=='m':
            t*=60
        if t_unit=='h':
            t*=60*60
        if t_unit=='d':
            t*=60*60*24
        if t_unit=='ka':
            t*=60*60*24*365.25*1000
        if t_unit=='Ma':
            t*=60*60*24*365.25*1.e6
        if t_unit=='Ga':
            t*=60*60*24*365.25*1.e9
        t/=self.t0  #  time now in units of tau0
        return t

    def density_evolve(self,tval,t_unit,field_dir):
        t= self.time_rescale(tval,t_unit)  # time in units of tau_0
        fd=1.0*np.array(field_dir)  # non-normalized field direction e.g. [1,1,1]
        norm=np.linalg.norm(fd)
        if (norm>0.0001):   #  allows for zero field in case of viscous decay
            fd/=norm        #  unit vector in field direction
```

```
67
68          avmag=_fast_Trap(self.b,self.m,self.d,fd,t,self.E1,self.MH,
    self.dirs,self.rho,self.dir_weights)
69          self.history.append([tval,t_unit,field_dir,t,avmag])
70          return avmag
```

2 Additional figures

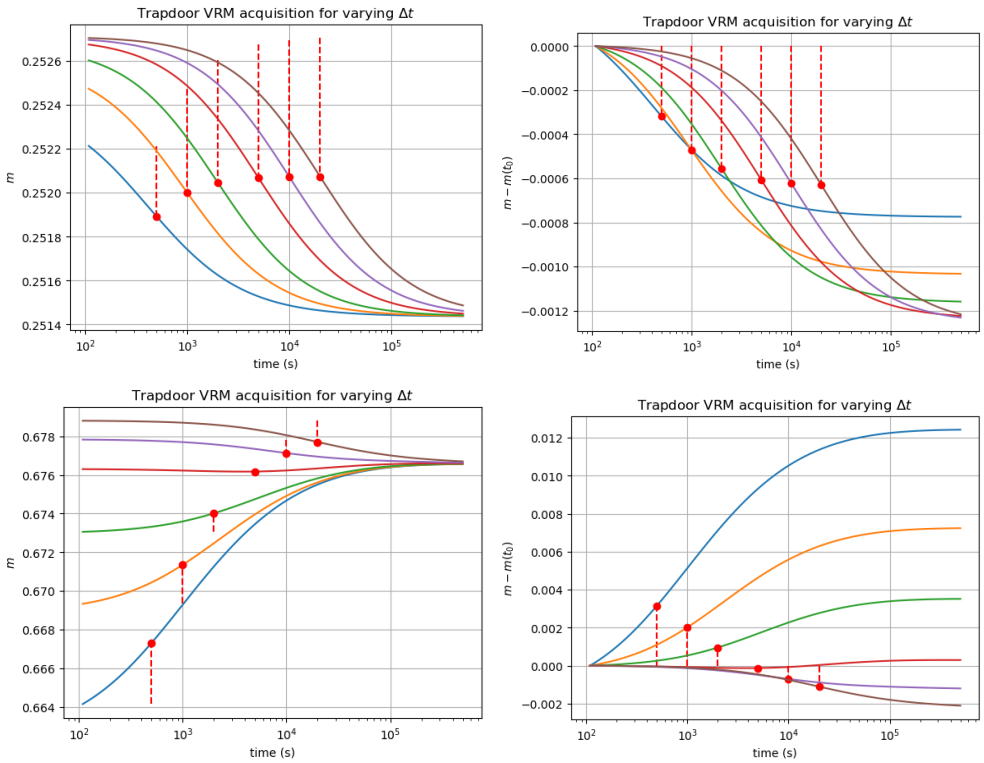

Figure 1: Top left: Modeled tVRM acquisition in an MD sample with initial density defined by $\alpha = 0$, $\sigma = 3$ and VRM acquisition in x-direction for $\Delta t = 0.5, 1, 2, 5, 10, 20 \times 10^3$s (sequence blue to brown). The plot shows the results of a repeated VRM acquisition in x-direction after this history. Note the marked non-log(t) behavior and the systematic dependence on the previous Δt marked by red dots. Top right: The same data after subtracting the initial magnetization to simplify comparison to experiments. Bottom Left: Same as top left, but with initial density defined by $\alpha = 0$, $\sigma = 3$ for $\Delta t = 0.5, 1, 2, 5, 10, 20 \times 10^3$s (sequence blue to brown). Bottom right: The same data as on the left after subtracting the initial magnetization.