

Supplemental Material: Nonlinear Ion Drift-Diffusion Memristance Description of TiO_2 RRAM Devices (sample code)

```
1 #!/usr/bin/env python
2 '''
3     File name: testing_params_MM1_tau.py
4     Authors: C. G. Rocha, and K. Esteki
5     Date created: 10 October 2019
6     Date last modified: 06 February 2020
7     Python Version: 2.7
8
9     This program is free software: you can redistribute it and/or modify
10    it under the terms of the GNU General Public License as published by
11    the Free Software Foundation, either version 3 of the License, or
12    (at your option) any later version.
13
14    This program is distributed in the hope that it will be useful,
15    but WITHOUT ANY WARRANTY; without even the implied warranty of
16    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
17    GNU General Public License for more details.
18
19    You should have received a copy of the GNU General Public License
20    along with this program. If not, see <https://www.gnu.org/licenses/>.
21
22    Required packages: numpy, matplotlib, scipy.
23
24    This code is generated for the purpose of testing the output parameters obtained from the
25    nonlinear fitting of systems of ODE equations. For this memristive model case (MM1+tau), the
26    optimized parameters are lambdax, lambday, eta1, eta2, eta3, eta4, tau, alpha1, beta1,
27    gamma, delta, alpha2, beta2, and x0.
28
29
30 Citation:
31     If you use this code in academic publications, please, cite our work appropriately.
32
33 Parameters:
34 -----
35     lambdap, lambdan, eta1, eta2, eta3, eta4: float
36         Nonlinear ion-drift parameters in the dynamical state equation
37
38     tau: float
39         Diffusion rate
40
41     vmax, freq: float
42         Voltage amplitude (in Volts) and frequency of the input voltage signal
43
44     alpha1, beta1: float
45         Current response function parameters (Schottky contribution)
46
47     gamma, delta: float
48         Current response function parameters (tunnelling contribution)
49
50     alpha2, beta2: float
51         Current response function parameters (rectifier contribution)
52
53     points: int
54         Number of points in the timeline (arb. units)
```

```

55
56 Returns: three plots: (i) x versus t, (ii) I versus V, and (iii) V versus t
57 '''
58
59 # importing canonical python libraries
60 import matplotlib.pyplot as plt
61 import numpy as np
62 from scipy.integrate import odeint
63
64 # State equation dx/dt to be integrated
65 def myFunc(x, t, *args):
66     '''After integrated, returns x(t).
67
68     Keyword arguments:
69     t -- time
70     x -- internal state variable
71     *args -- multiple memristive parameters: lambdap, lambdan, eta1, eta2, eta3, eta4, tau
72     *args -- input voltage signal parameters: vmax, freq
73     '''
74
75     v = vmax * np.sin(2.0 * freq * np.pi * t)
76     xdyn = x
77
78     if t <= freq/2:
79         dxdt = (lambdap * (np.exp(eta1*v) - np.exp(eta2)) - xdyn/tau) * (1.0 - (2.0*xdyn - 1.0)**2)
80     else:
81         dxdt = (lambdan * (np.exp(-eta3*v) - np.exp(eta4)) - xdyn/tau) * (1.0 - (2.0*xdyn - 1.0)**2)
82
83     return dxdt
84
85
86 # general parameters for the voltage (sinusoidal) input
87 vmax = 10.0
88 freq = 1.0
89
90 # parameters taken from the fitting procedure
91 alpha1 = 6.82991497
92 beta1 = 0.10051066
93 alpha2 = 3.42451238
94 beta2 = 0.01325593
95 gamma = 1.6138e-04
96 delta = 0.99905798
97
98 lambdap = 3.05688105
99 lambdan = 3.56661637
100 eta1 = 0.19781416
101 eta2 = 0.35545468
102 eta3 = 0.19863980
103 eta4 = 0.53302178
104 tau = 0.10820846
105 dxdt_params = (vmax, freq, lambdap, lambdan, eta1, eta2, eta3, eta4, tau)
106
107 # number of points in the timeline
108 points = 10000
109
110 # create a timeline vector
111 t = np.array([0.0, 1.0])
112 tspan_vector = np.linspace( t[0], t[1], points)
113

```

```
114 # voltage input signal
115 V = vmax * np.sin(freq * 2.0 * np.pi * tspan_vector)
116
117 # steps in the timeline
118 h = tspan_vector[1] - tspan_vector[0]
119
120 # initial conditions for the internal state variable
121 x0 = 0.9
122
123 # integration of the dynamical state equation
124 xs = odeint(myFunc, x0, tspan_vector, args=dxdt_params)
125
126 # calculating current values from the solution x(t)
127 ic = (1.0 - xs[:,0]) * alpha1 * (1.0 - np.exp(-beta1 * V[:,])) + xs[:,0] * gamma * np.sinh(delta * V[:,]) +
      alpha2 * (1.0 - np.exp(-beta2 * V[:,]))
128
129 # plotting instructions
130
131 # plotting x versus t
132 plt.figure()
133 plt.plot(tspan_vector, xs[:,0], 'r')
134 plt.xlim(t[0], t[1])
135 plt.xlabel('t', fontsize=17)
136 plt.ylabel('x', fontsize=17)
137 plt.tight_layout()
138
139 # plotting I versus V
140 plt.figure()
141 plt.plot(V, ic, 'r')
142 plt.xlabel('V', fontsize=17)
143 plt.ylabel('I', fontsize=17)
144
145 # plotting V versus t
146 plt.figure()
147 plt.plot(tspan_vector, V, 'r')
148 plt.xlabel('t', fontsize=17)
149 plt.ylabel('V', fontsize=17)
150
151 plt.show()
```

The figures below are screenshots of the output of this sample code for the sake of reference. The code plots $x(t) \times t$, $I \times V$, and $V(t) \times t$. The used parameters were optimized to fit our particular experimental data of Au-Ti/TiO₂/Ti-Au devices in vacuum and at room temperature. These parameters work better for a single voltage cycle.

