Modeling of Filamentary Resistive Memory by Concentric Cylinders with Variable Conductivity

Supporting Information

Python Code for Simulations

Andrew J. Lohn\*, Patrick R. Mickel

Simulation Code in Python:

"""

This program simulates the electrical response of memristors or RRAM based on the physical parameters of the device.

Written by Andrew J. Lohn and Patrick R. Mickel

Created on Tue May 20 15:26:10 2014

Related theory is described in "Isothermal Switching and Detailed Filament Evolution in Memristive Systems" published in Advanced Materials.

"""

import numpy as np

import pylab as pl

######## Set the Device Parameters ###########

delT = 1186 #Activation temperature in degrees C from the ambient temperature

Trt = 300 #Ambient temperature in degrees C

de = 50 \* 10\*\*-9 #Electrode thickness

do = 10 \* 10\*\*-9 #Oxide thickness

ke = 363.12 #Effective electrode thermal conductivity - Usually higher than bulk values due to spreading effects

Lwf = 2.44 \* 10\*\*-8 #Wiedemann-Franz constant

sigSat = 9.81 \* 10\*\*5 #Saturation Conductivity in the Filament

maxConc = 100 #The maximum concentration of oxygen vacancies or dopants in arbitrary units

minConc = 50 #The minimum concentration of oxygen vacancies or dopants in arbitrary units

a = 0.00828 #Amplitude coefficient for Poole-Frenkel contribution

b = 4.514 #Nonlinearity coefficient for Poole-Frenkel contribution

Ar = 2\*ke\*do/(sigSat \* de) #Coefficient for radius change equation is composed of previously defined parameters

Rmin = ke / (4\*3.14\*(sigSat\*\*2)\*Lwf\*(Trt+delT)\*de) #Minimum resistance value is composed of previously defined parameters

numberOfShells = 200 #Set the number of shells for the simulation

simRadiusRange = 20 \* 10\*\*-9 #The full spacial range of the simulation

shellSize = simRadiusRange/numberOfShells #Calculate grid spacing radially

shellRadii = [(i+1)\*shellSize for i in xrange(numberOfShells)] #Build an array of outer edge locations for each shell

shellGeometries = [] #Build an array of aeometries for each shell to be used with conductivity to calculate resistance

for radius in shellRadii:

shellGeometries.append(3.14\*(radius\*\*2 - (radius-shellSize)\*\*2) / do)

shellConcentrations = np.ones(numberOfShells) #Initialize the shell concentrations

#Forming

formingRadius = (2 \* 10\*\*-9)/shellSize #Set the forming radius

for i in xrange(len(shellConcentrations)):

if i < formingRadius:

shellConcentrations[i] = 80 #Set the post-forming concentration - This can be a low value if the device is turned off after forming

else:

shellConcentrations[i] = 0 #Set the concentration to 0 outside the formed region. The minimum concentration is higher than this region which is as-deposited.

########## Establish Functions for Changing Resistance #############

def findShellConds(shellConcentrations,voltageVal): #Calculate the shell conductivities from the shell concentrations and the applied voltage

shellConductivities = []

for shellConc in shellConcentrations:

if shellConc == 0:

shellConductivities.append(0) #Enforce that the as-deposited region does not contribute.

else: #Otherwise use a combination of Ohmic and Poole-Frenkel weighted by concentration

shellConductivities.append(sigSat \* abs(minConc-shellConc)/minConc + sigSat \* (1 - abs(minConc-shellConc)/minConc) \* abs(voltageVal) \* a\*np.exp(b\*(abs(voltageVal)\*\*0.5)))

shellConductivities = np.array(shellConductivities)

return shellConductivities

#Define a function for sourcing current

cSourceVoltages = np.linspace(0,2,100) #Voltage is needed to calculate conductivity. These values are stepped through to find the voltage for that calculation.

def CurrentSourcing(currentVal):

global shellConcentrations

if currentVal < 0: #Negative current implies conductivity change

for voltageVal in cSourceVoltages: #Find the voltage that corresponds to the sourced current by stepping through values

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1/np.sum(shellConductivities\*shellGeometries)

currentTest = -voltageVal/resistance

if currentTest <= currentVal: #If the calculated current is more negative than the sourced current then stop stepping through voltages

break

for i in xrange(len(shellConcentrations)): #Find the location of the edge of the inner-most filament

if shellConcentrations[i] < np.max(shellConcentrations):

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the resistance of the inner-most filament

rop = (i+1)\*shellSize #Approximate operating radius as current filament radius

Asig = 8\*(do\*\*2)\*Lwf\*(Trt+delT)/(rop\*\*2) #Calculate new Asig with new operating radius

Rmax = 4\*(do\*\*2)\*Lwf\*(Trt+delT)\*de/(3.14\*ke\*(rop\*\*4)) #Calculate new Rmax with new operating radius

while (currentVal\*\*2)\*resistance >= Asig\*delT/(Rmax - resistance) and resistance < Rmax: #While sufficient power is supplied for activation

shellMaxConc = np.max(shellConcentrations)

if shellMaxConc <= minConc:

break

for i in xrange(len(shellConcentrations)): #Decrease the conductivity by one step for every shell in the inner-most filament

if shellConcentrations[i] == shellMaxConc:

shellConcentrations[i] += -1

else:

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1])

rop = (i+1)\*shellSize #Calculate new operating radius and associated parameters

Asig = 8\*(do\*\*2)\*Lwf\*(Trt+delT)/(rop\*\*2)

Rmax = 4\*(do\*\*2)\*Lwf\*(Trt+delT)\*de/(3.14\*ke\*(rop\*\*4))

if currentVal >= 0: #Positive current implies radius change

for voltageVal in cSourceVoltages: #Find the voltage that corresponds to the sourced current by stepping through values

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1/np.sum(shellConductivities\*shellGeometries)

currentTest = voltageVal/resistance

if currentTest >= currentVal: #If the calculated current is more negative than the sourced current then stop stepping through voltages

break

for i in xrange(len(shellConcentrations)): #Find the location of the edge of the inner-most filament

if shellConcentrations[i] < np.max(shellConcentrations):

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the resistance of the inner-most filament

while (currentVal\*\*2)\*resistance > Ar\*delT / (resistance-Rmin) and resistance > Rmin:

for i in xrange(len(shellConductivities)):

if shellConcentrations[i] < maxConc:

shellConcentrations[i] = maxConc #Find the first shell that is not saturated and move it to saturation

break #Break so only one shell saturates at a time

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the new resistance of the inner-most filament

if i == len(shellConductivities)-1: #If all the shells are saturated then exit

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities \* shellGeometries) #Calculate the resistance of the entire filament, not just the inner-most filament

return currentVal \* resistance #Return the voltage

#Define a function for sourcing voltage

def VoltageSourcing(voltageVal):

global shellConcentrations

if voltageVal <= 0: #Negative voltage implies conductivity change

for i in xrange(len(shellConcentrations)): #Find the location of the edge of the inner filament

if shellConcentrations[i] < np.max(shellConcentrations):

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the resistance of the inner-most filament

rop = (i+1)\*shellSize #Approximate operating radius as current filament radius

Asig = 8\*(do\*\*2)\*Lwf\*(Trt+delT)/(rop\*\*2) #Calculate new Asig with new operating radius

Rmax = 4\*(do\*\*2)\*Lwf\*(Trt+delT)\*de/(3.14\*ke\*(rop\*\*4)) #Calculate new Rmax with new operating radius

while (voltageVal\*\*2)/resistance >= Asig\*delT/(Rmax - resistance) and resistance < Rmax: #While sufficient power is supplied for activation

shellMaxConc = np.max(shellConcentrations)

if shellMaxConc <= minConc:

break

for i in xrange(len(shellConcentrations)): #Decrease the conductivity by one step for every shell in the inner-most filament

if shellConcentrations[i] == shellMaxConc:

shellConcentrations[i] += -1

else:

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1])

rop = (i+1)\*shellSize #Calculate new operating radius and associated parameters

Asig = 8\*(do\*\*2)\*Lwf\*(Trt+delT)/(rop\*\*2)

Rmax = 4\*(do\*\*2)\*Lwf\*(Trt+delT)\*de/(3.14\*ke\*(rop\*\*4))

if voltageVal > 0: #Positive voltage implies radius change

for i in xrange(len(shellConcentrations)): #Find the location of the edge of the inner-most filament

if shellConcentrations[i] < np.max(shellConcentrations):

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the resistance of the inner-most filament

while (voltageVal\*\*2)/resistance > Ar\*delT / (resistance-Rmin):# and resistance > Rmin: #While sufficient power is supplied for activation

for i in xrange(len(shellConductivities)):

if shellConcentrations[i] < maxConc:

shellConcentrations[i] = maxConc #Find the first shell that is not saturated and move it to saturation

break #Break so only one shell saturates at a time

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities[0:i+1] \* shellGeometries[0:i+1]) #Calculate the new resistance of the inner-most filament

if i == len(shellConductivities)-1: #If all the shells are saturated then exit

break

shellConductivities = findShellConds(shellConcentrations,voltageVal)

resistance = 1 / np.sum(shellConductivities \* shellGeometries) #Calculate the resistance of the entire filament, not just the inner-most filament

return voltageVal / resistance #Return the current

######### Create Input Signals and Generate Output Signals #########################

Voltages = [] #Create empty lists to be populated during the voltage or current sweep

Currents = []

#Switch On

CurrSpacing = np.linspace(0,6\*10\*\*-3,50) #Use current sourcing to sweep from 0 to 6 mA

CurrentSweep = CurrSpacing

for val in reversed(CurrSpacing):

CurrentSweep = np.hstack([CurrentSweep,val]) #Also sweep from 6 mA back to 0

for currentVal in CurrentSweep:

voltage = CurrentSourcing(currentVal) #Calculate the voltage by calling the current sourcing function

Voltages.append(voltage)

Currents.append(currentVal)

#Switch Off

VoltSpacing = np.linspace(0,-1.0,50) #Use voltage sourcing to sweep from 0 to -1.0 V

VoltageSweep = VoltSpacing

for val in reversed(VoltSpacing):

VoltageSweep = np.hstack([VoltageSweep,val]) #Also weep from -1.1 V to 0

for voltageVal in VoltageSweep:

current = VoltageSourcing(voltageVal) #Call the voltage sourcing function to find the current at every voltage step

Voltages.append(voltageVal)

Currents.append(current)

pl.plot(Voltages,Currents)

pl.show()

Python Code for Fitting ON switching data to find device parameters:

"""

This program finds the saturation conductivity, activation temperature, electrode thermal conductivity and filament radius from the ON switching data of an I-V hysteresis loop

Written by Andrew J. Lohn and Patrick R. Mickel.

Created on Tue May 20 15:26:10 2014.

Related theory is described in "Isothermal Switching and Detailed Filament Evolution in Memristive Systems" published in Advanced Materials.

"""

import numpy as np

from scipy.optimize import curve\_fit

import pylab as pl

import csv

de = 50 \* 10\*\*-9 #Electrode thickness in meters

do = 10 \* 10\*\*-9 #Oxide thickness in meters

Lwf = 2.44 \* 10\*\*-8 #Wiedemann-Franz constant

Trt = 300 #Ambient temperature in Kelvin

def onSwitch(R,ke,sigSat,Tc): #This function calculates Powers for a list of Resistances given the device properties

Ar = 2\*ke\*do/(sigSat\*de) #This is a prefactor in the power-resistance relation

Rmin = ke / (4\*3.14\*(sigSat\*\*2)\*Lwf\*Tc\*de) #This is the minimum resistance

P = Ar \* (Tc - Trt) / (R - Rmin) #Calculate power as a function of resistance

if ke > 10 and ke < 500 and sigSat > 1\*10\*\*3 and sigSat < 1\*10\*\*6 and Tc > 500 and Tc < 3000: #if the device parameters are reasonable then return the power

return P

else: #If the parameters are unreasonable then return a large number so the error will be large in the fitting routine.

return 1\*10\*\*6

def PRconvert(IV): #This function converts current-voltage data to power-resistance data

PR = []

for pair in IV:

power = pair[0]\*pair[1] #Power is current times voltage

resistance = pair[1]/pair[0] #Resistance is voltage divided by current

PR.append([power,resistance])

return PR

########### Load and Fit Data ##############

IVdata = csv.reader(open('onSwitch.csv','rb')) #A csv file is used where the columns are voltage then current.

IV = [] #This data is only the ON switching while the resistance is changing.

for row in IVdata:

IV.append([float(row[1]),float(row[0])]) #This reverses the order to match with the order in the variable name - IV is I then V

IV = np.array(IV)

PR = PRconvert(IV) #Convert the current-voltage data to power-resistance data

PR = np.array(PR)

pinit = [105.5,5\*10\*\*5,1750] #Initialize the values for [electrode thermal conductivity (ke), saturation conductivity (sigSat), activation temperature (Tc)]

popt,pconv = curve\_fit(onSwitch,PR[:,1],PR[:,0],p0=pinit) #Fit the data to the power-resistance relation to determine the ke, sigSat, and Tc values

print popt

PRfit = onSwitch(PR[:,1],popt[0],popt[1],popt[2]) #Create a list of power values from the fitting parameters for comparison to data

resistance = IV[-1][1]/IV[-1][0]

print (do/(popt[1]\*3.14\*resistance))\*\*0.5 #print out the radius at the end of the ON switching

pl.plot(PR[:,1],PR[:,0],'bo') #plot the data and the fit

pl.plot(PR[:,1],PRfit,'b')

pl.show()

Python Code for Fitting OFF state data to find Poole-Frenkel a and b parameters:

"""

This program finds the Poole-Frenkel coefficients to describe nonlinearity in the simulator

Written by Andrew J. Lohn and Patrick R. Mickel.

Created on Tue May 20 15:26:10 2014.

"""

import numpy as np

from scipy.optimize import curve\_fit

import pylab as pl

import csv

f = open('Nonlinearity.csv','rb') #This is the experimental current-voltage data from only the OFF state.

expData = csv.reader(f) #The resistance should be changing due to nonlinearity only, not state change.

expVs = []

expIs = []

for row in expData:

expVs.append(float(row[0]))

expIs.append(float(row[1]))

expVs = np.array(expVs)

expIs = np.array(expIs)

f.close()

#These should all be the same as will be used in the simulation code

sigSat = 9.81 \* 10\*\*5 #Saturation conductivity was previously found from fitting the ON switching data

numberOfShells = 200 #Number of Grids in the Radial Direction

thickness = 10 \* 10\*\*-9 #Vertical Thickness in the z direction

simRadiusRange = 20 \* 10\*\*-9 #The full spacial range of the simulation

shellSize = simRadiusRange/numberOfShells #Calculate GridSpacing

rmax = 6.4 \* 10\*\*-9 #This is the radius that was previously found from fitting the ON switching data

numAtrmax = int(rmax/shellSize)

shellRadii = [(i+1)\*shellSize for i in xrange(numAtrmax)]

shellGeometries = [] #Build an Array of Geometries for Each Shell

for radius in shellRadii:

shellGeometries.append(3.14\*(radius\*\*2 - (radius-shellSize)\*\*2) / thickness)

shellConcentrations = 50\*np.ones(numAtrmax) #This sets the filament to the minimum concentration for all shells up to the largest radius during ON switching

minConc = 50

a = 0.001 #These are initialized values with no real significance

b = 7

def findShellConds(shellConcentrations,voltageVal): #This function finds the shell conductivities given their concnentrations and the applied voltage

shellConductivities = []

for shellConc in shellConcentrations:

if shellConc == 0:

shellConductivities.append(0)

else:

shellConductivities.append(sigSat \* abs(minConc-shellConc)/minConc + sigSat \* (1 - abs(minConc-shellConc)/minConc) \* abs(voltageVal) \* a\*np.exp(b\*(abs(voltageVal)\*\*0.5)))

shellConductivities = np.array(shellConductivities)

return shellConductivities

def nonLinFunc(Voltages,a,b): #This function calcuates currents the way the simulator would for a range of voltages and known a and b values

Currents = []

for voltageVal in Voltages:

shellConductivities = []

for shellConc in shellConcentrations:

if shellConc == 0:

shellConductivities.append(0)

else:

shellConductivities.append(sigSat \* abs(minConc-shellConc)/minConc + sigSat \* (1 - abs(minConc-shellConc)/minConc) \* abs(voltageVal) \* a\*np.exp(b\*(abs(voltageVal)\*\*0.5)))

shellConductivities = np.array(shellConductivities)

resistance = 1 / np.sum(shellConductivities \* shellGeometries)

Currents.append(voltageVal/resistance)

if a > 1\*10\*\*-21 and a < 1\*10\*\*3 and b > 1\*10\*\*-1 and b < 1\*10\*\*3: #Return the currents if the a and b values are in a reasonable range.

return Currents

else: #If the a and be values are not reasonable then return a large value to create a large error during the fitting routine.

return 1\*10\*\*6

########## Fit the Data ###########

pinit = [1\*10\*\*-9,1] #Initialize the a and b values for fitting

popt,pconv = curve\_fit(nonLinFunc,expVs,expIs,p0=pinit) #Fit to find the a and b values

print popt

fitIs = nonLinFunc(expVs,popt[0],popt[1]) #Create a list of fitted currents for comparison to experimental data

pl.plot(expVs,expIs,'bo') #Plot the experimental and fitted curves.

pl.plot(expVs,fitIs,'b-')

pl.show()