**The Influence of Isoconcentration Surface Selection in Quantitative Outputs from Proximity Histograms (Appendix)**

*Dallin J. Barton1, B. Chad Hornbuckle2, Kristopher A. Darling2, and Gregory B. Thompson1*

*1) The University of Alabama, Department of Metallurgical & Materials Engineering, Box 870202 Tuscaloosa, AL 35487-0200, USA*

*2) United States Army Research Laboratory, Weapons and Materials Research Directorate, RDRL-WMM-B Aberdeen Proving Grounds, Aberdeen Proving Grounds, MD, 21005-5069, USA*

A.1 Individual particle analysis

**Table A.1:** Zr Isovalue, volume and composition for each individual particle labelled in Figure 9(a)

|  |  |  |  |
| --- | --- | --- | --- |
| **Particle Number** | **Zr Isovalue** | **Volume (nm3)** | **Zr Composition (at.%)** |
| 1 | 3.0 | 41.17 | 14.04 ± 3.80 |
| 2 | 3.0 | 2.54 | 9.15 ± 6.93 |
| 3 | 3.0 | 76.57 | 14.78 ± 3.68 |
| 4 | 1.5 | 341.046 | 24.97 ± 7.19 |
| 5 | 1.5 | 127.416 | 30.56 ± 10.70 |
| 6 | 1.5 | 91.502 | 20.65 ± 9.35 |
| 7 | 7.0 | 229.690 | 27.08 ± 4.95 |
| 8 | 2.0 | 48.730 | 28.13 ± 11.21 |
| 9 | 2.0 | 94.446 | 26.66 ± 10.81 |
| 10 | 5.0 | 42.388 | 35.88 ± 14.07 |
| 11 | 11.0 | 3695.117 | 32.15 ± 1.13 |
| 12 | 5.0 | 145.044 | 14.26 ± 2.83 |
| 13 | 5.0 | 85.279 | 18.95 ± 5.08 |
| 14 | 3.0 | 80.85 | 33.08 ± 11.61 |
| 15 | 3.5 | 78.125 | 16.18 ± 5.50 |
| 16 | 3.5 | 94.775 | 11.60 ± 3.13 |

A.2 Chemical Randomizing Code

The first method is keeping the same positions and randomizing the mass-to-charge positions. The file format is written as .pos converted to .csv files using IVAS. These files have four columns: x position, y position, z position, and mass-to-charge ratio. Changing a file of ~75 million ions in this manner took about 30 minutes with a 2.2 GHz processor.

This is the code written in Python 3.6 and ran on a UNIX system, MacOS.

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

#import stuff

import pandas as pd

from sqlalchemy import create\_engine

import random

import sqlite3

from pandas.io import sql

import subprocess

import numpy as np

import uuid

import csv

import os

#set constants

total\_sum = 0

loading = 0

loadingcvs = 0

maxx = 0

maxy = 0

maxz = 0

minx = 0

miny = 0

minz = 0

chunksize = 10000

i = 0

j = 1

#define your file name

filename = ‘filename’

#Read in your CSV file

loading = 0

chunks = []

for chunk in pd.read\_csv(filename, chunksize=chunksize, low\_memory=False):

 chunks.append(chunk)

 loading +=1

 loadingtest = loading % 100

 if loadingtest == 0:

 print ("Loading: " + repr(loading))

df = pd.concat(chunks, axis=0)

print('Finished Loading')

#After you are finished loading, create a dataframe putting x, y, z and mass into columns 0, 1, 2, #and 3.

columnx = df.iloc[:,0]

columny = df.iloc[:,1]

columnz = df.iloc[:,2]

columnmass = df.iloc[:,3]

print('Finished column loading')

lencolumn = len(columnmass)

n = 0

newcolumnmass = []

#The random sample will pick numbers randomly from a list given and not repeat that list. Here, #the list length was taken, a number from that list length randomly chosen, the position from the #mass-to-charge ratio column was taken, then that position was added to a new column.

randomlist = random.sample(range(lencolumn),lencolumn)

while n < lencolumn:

 randomnumber = randomlist[n]

 nextentry = df.iloc[randomnumber,3]

 newcolumnmass.append(nextentry)

 n += 1

 ntest = n % 100000

 if ntest == 0:

 print('Random Number Loading: ' + repr(n))

#Reload the chosen mass into a new column

df.iloc[:,3] = newcolumnmass

print('Finished column randomizing')

#Write back to a CSV file.

df.to\_csv('APT\_shuffled.csv', sep=',',index=False)

#end of code

A.3 Complete Spatial Randomness.

The code for complete spatial randomness (CSR) is based off of the equation found in (Stephenson et al., 2007),

$P\_{K}\left(r,ρ\right)dr=\frac{3}{\left(K-1\right)!}\left(\frac{4π}{3}ρ\right)^{K}r^{3K-1}exp\left(-\frac{4π}{3}ρr^{3}\right)dr.$ [1]

Large exponents and factorials require too much computational power. A natural log was applied to the equation eliminating the exponent and allows for Stirling’s approximation to be used.

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

#import stuff

from math import sqrt, log, exp

import random

import numpy as np

import decimal

import pandas as pd

D = decimal.Decimal

#initialize stuff

total\_sum = 0

loading = 0

loadingcvs = 0

maxx = 0

maxy = 0

maxz = 0

minx = 0

miny = 0

minz = 0

chunksize = 10000

i = 0

j = 1

filename = 'filename’

#Load the CSV file into Python.

loading = 0

chunks = []

for chunk in pd.read\_csv(filename, chunksize=chunksize, low\_memory=False):

 chunks.append(chunk)

 loading +=1

 loadingtest = loading % 100

 if loadingtest == 0:

 print ("Loading: " + repr(loading))

df = pd.concat(chunks, axis=0)

print('Finished Loading')

columnx = df.iloc[:,0]

columny = df.iloc[:,1]

columnz = df.iloc[:,2]

columnmass = df.iloc[:,3]

print('Finished column loading')

lencolumn = len(columnmass)

n = 0

newcolumnmass = []

randomlist = random.sample(range(lencolumn),lencolumn)

while n < lencolumn:

 randomnumber = randomlist[n]

 nextentry = df.iloc[randomnumber,3]

 newcolumnmass.append(nextentry)

 n += 1

 ntest = n % 100000

 if ntest == 0:

 print('Random Number Loading: ' + repr(n))

df.iloc[:,3] = newcolumnmass

print('Finished mass assignment randomizing')

volumeold = 0

n = 0

howmanyatoms = len(columnx)

K = 2

r = 0

dr = 0.001

roh = 74

Pi=3.141592653589793238462643383

xstor = []

ystor = []

zstor = []

mass = []

k = 1

# Step one, step up atom and step up radius

K += 1

r += dr

#Step two, calculate probability

for n in range(howmanyatoms):

# Define probability

 randomnumber = 1

 P = 0

 volumeold = 0

 while randomnumber > P:

 randomnumber = random.random()

 storageK = []

 storageP = []

 #Expanded version of the probability with a log taken and Stirling’s approximation applied

 firststep = D(log(3))

 secondstep = D((K-1)\*(log(K-1)))-D(K-1)

 fourthstep = D(K\*log(4\*Pi/3\*roh))

 fifthstep = D((3\*K-1)\*log(r))

 sixthstep = D(4\*Pi/3\*roh\*pow(r,3))

 logofP = firststep-secondstep+fourthstep+fifthstep-sixthstep

# Python cannot take exponents of negatives

 try:

 P = exp(logofP)

 except:

 try:

 logofP = -logofP

 P = exp(logofP)

# If all else fails, punt

 except: P = 0.001

 if randomnumber > P:

#This means the probability is not high enough, so we need to increase the radius to #increase the probability.

 r +=dr

 else:

#This means the probability is high enough, so we add another atom at the same radius #and see if we can do it again.

# We randomly pick a number for x and then use 3-D Pythagorean theorem to #find y #and z. The problem with this method is that as x is randomly chosen it limits the #possible options for y and z to be an extreme e.g. 1/n chance for x to be 0.0. 1/n^2 #chance for x and y to be 0.0 allowing z to be 1. To overcome this, I just brute forced all #of the possible combinations of randomly choosing a point in space. The dataset is #large enough to get over the statistical variance this causes.

 if k == 1:

 x = r \* random.random()

 if random.random() < 0.50:

 x=-x

 y = sqrt(pow(r,2) - pow(x,2)) \* random.random()

 if random.random() < 0.50:

 y=-y

 z = sqrt(pow(r,2) - pow(x,2) - pow(y,2))

 if random.random() < 0.50:

 z=-z

 if k == 2:

 x = r \* random.random()

 if random.random() < 0.50:

 x=-x

 z = sqrt(pow(r,2) - pow(x,2)) \* random.random()

 if random.random() < 0.50:

 z=-z

 y = sqrt(pow(r,2) - pow(x,2) - pow(z,2))

 if random.random() < 0.50:

 y=-y

 if k == 3:

 y = r \* random.random()

 if random.random() < 0.50:

 y=-y

 x = sqrt(pow(r,2) - pow(y,2)) \* random.random()

 if random.random() < 0.50:

 x=-x

 z = sqrt(pow(r,2) - pow(x,2) - pow(y,2))

 if random.random() < 0.50:

 z=-z

 if k == 4:

 y = r \* random.random()

 if random.random() < 0.50:

 y=-y

 z = sqrt(pow(r,2) - pow(y,2)) \* random.random()

 if random.random() < 0.50:

 z=-z

 x = sqrt(pow(r,2) - pow(y,2) - pow(z,2))

 if random.random() < 0.50:

 x=-x

 if k == 5:

 z = r \* random.random()

 if random.random() < 0.50:

 z=-z

 x = sqrt(pow(r,2) - pow(z,2)) \* random.random()

 if random.random() < 0.50:

 x=-x

 y = sqrt(pow(r,2) - pow(x,2) - pow(z,2))

 if random.random() < 0.50:

 y=-y

 if k == 6:

 z = r \* random.random()

 if random.random() < 0.50:

 z=-z

 y = sqrt(pow(r,2) - pow(z,2)) \* random.random()

 if random.random() < 0.50:

 y=-y

 x = sqrt(pow(r,2) - pow(z,2) - pow(y,2))

 if random.random() < 0.50:

 x=-x

 k += 1

 if k > 6:

 k = 1

 K += 1

 xstor.append(x)

 ystor.append(y)

 zstor.append(z)

 randommass = random.random()\*300

 mass.append(randommass)

 Ktest = K% 100000

 if Ktest == 0:

 print ('K: ' + repr(K))

#This is for IVAS to work with. The extremes are now at the origin.

xstor = np.array(xstor)

xstor = xstor - min(xstor)

ystor = np.array(ystor)

ystor = ystor - min(ystor)

zstor = np.array(zstor)

zstor = zstor - min(zstor)

#Change the positions. The mass has already been taken care of.

df.iloc[:,0] = xstor

df.iloc[:,1] = ystor

df.iloc[:,2] = zstor

print ("Column Loading Finished")

#Save to a new CSV

df.to\_csv(‘newfilename’, sep=',',index=False)

#end of code

Works Cited for Supplementary Material

Stephenson, L. T., Moody, M. P., Liddicoat, P. V & Ringer, S. P. (2007). New Techniques for the Analysis of Fine-Scaled Clustering Phenomena within Atom Probe Tomography (APT) Data. *Microscopy and Microanalysis* **13**, 448–463.