**allwallfeatures\_pca300\_nov17.mtx"**

pca with 300 dimensions

ntrain = 250

nvalidation = 50

lambda\_list = list(range(5,150))

lasso

>>> test\_list

array([ 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1.,

1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,

0., 1., 0., 1., 1., 1., 0., 1.])

>>> sum(test\_list)

46.0

>>> sum(test\_list) / test\_list.size

0.76666666666666672

>>> io.mmwrite("pca300", pca)

pca300 with lambda from 0 – 10

error on test set : 0.683333333333

pca with 100 components and lambda 50-150

words guessed correctly: 45.0

percentage guessed correctly: 0.75

[ 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0.

0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1.

0. 1. 1. 1. 0. 1.]

pca with 50 dimensions

words guessed correctly: 44.0

percentage guessed correctly: 0.733333333333

lambda 50-100

dimensions – 300

accuracy improves dramatically at ~12 features

we get a max accuracy of 0.8 at about 120 features

array([ 0.26666667, 0.28333333, 0.36666667, 0.38333333, 0.43333333,

0.46666667, 0.48333333, 0.5 , 0.53333333, 0.58333333,

0.61666667, 0.71666667, 0.71666667, 0.68333333, 0.68333333,

0.7 , 0.66666667, 0.68333333, 0.68333333, 0.68333333,

0.68333333, 0.66666667, 0.65 , 0.66666667, 0.66666667,

0.66666667, 0.66666667, 0.68333333, 0.68333333, 0.66666667,

0.65 , 0.66666667, 0.68333333, 0.7 , 0.7 ,

0.68333333, 0.68333333, 0.7 , 0.7 , 0.7 ,

0.7 , 0.68333333, 0.68333333, 0.7 , 0.7 ,

0.7 , 0.66666667, 0.68333333, 0.68333333, 0.68333333,

0.68333333, 0.66666667, 0.68333333, 0.68333333, 0.7 ,

0.71666667, 0.7 , 0.7 , 0.68333333, 0.68333333,

0.7 , 0.68333333, 0.7 , 0.7 , 0.71666667,

0.73333333, 0.71666667, 0.75 , 0.73333333, 0.73333333,

0.71666667, 0.73333333, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.71666667, 0.71666667, 0.7 ,

0.71666667, 0.71666667, 0.71666667, 0.71666667, 0.71666667,

0.71666667, 0.71666667, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.73333333, 0.71666667, 0.71666667,

0.71666667, 0.71666667, 0.71666667, 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.73333333, 0.73333333, 0.75 , 0.73333333,

0.75 , 0.76666667, 0.76666667, 0.76666667, 0.76666667,

0.76666667, 0.78333333, 0.8 , 0.76666667, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.76666667,

0.75 , 0.75 , 0.75 , 0.76666667, 0.76666667,

0.76666667, 0.75 , 0.75 , 0.78333333, 0.76666667,

0.76666667, 0.76666667, 0.73333333, 0.75 , 0.76666667,

0.76666667, 0.76666667, 0.75 , 0.76666667, 0.75 ,

0.76666667, 0.76666667, 0.76666667, 0.76666667, 0.76666667,

0.76666667, 0.76666667, 0.75 , 0.75 , 0.75 ,

0.73333333, 0.73333333, 0.73333333, 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.73333333, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.73333333, 0.73333333, 0.75 ,

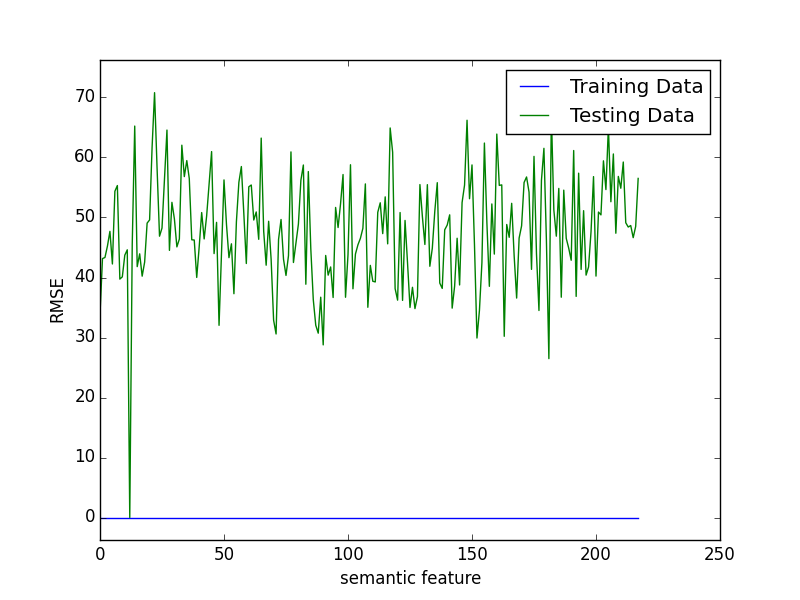
0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

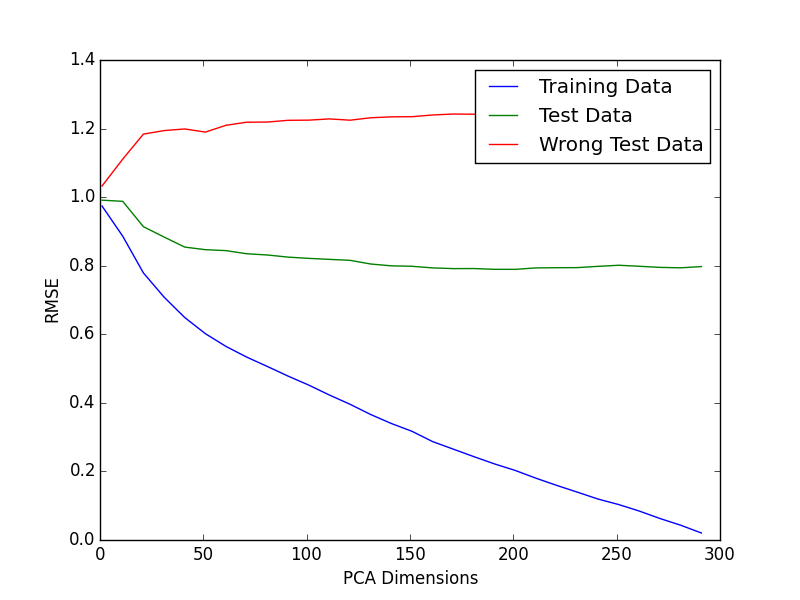
0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 ])

root mean square error for each semantic feature with training and test dat





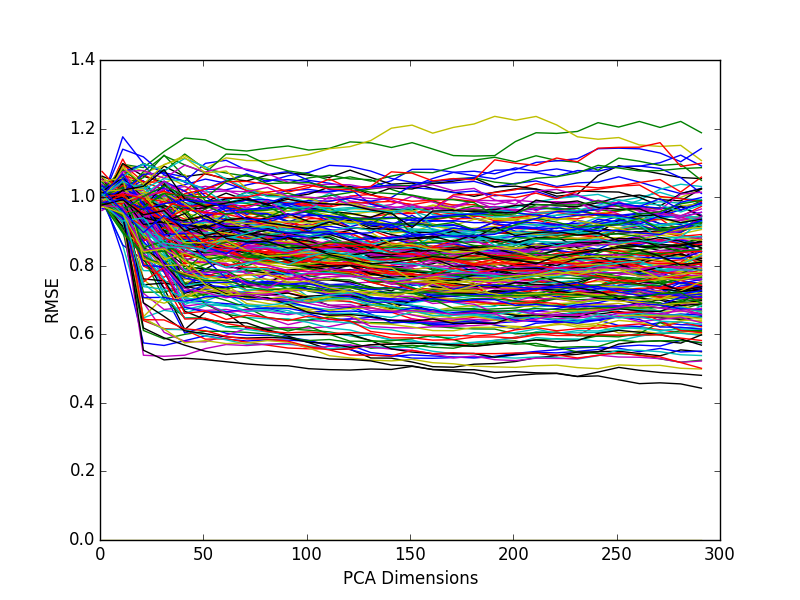
root mean square error calculated as

ypred = x.dot(w)

for each semantic feature

rmse = sqrt(1/n \* (ypred-y)^2)

total rmse = sum (rmse for each semantic feature) / num of semantic features



rmse per semantic feature

some stay high ~ 1

some drop.

need to find which ones!

dif = rmsetest\_matrix[0] - rmsetest\_matrix[29]

this should decrease with more pca dimensions.. so dif <0 is good.

if error increases it is bad!

GOOD SEMANTIC FEATURES:

Semantic Features that rmse decreased

(array([ 14, 21, 22, 27, 33, 35, 45, 65, 77, 101, 117, 118, 148,

150, 155, 160, 179, 182, 191, 205, 211]),)

worse semantic features :

>>> b = dif > 0.5

>>> b.nonzero()

(array([ 88, 90, 152, 163, 181]),)

features that have high rmse > 1

they don’t match with the ones above..

highrmse = rmsetest\_matrix[29]>1

highrmse.nonzero()

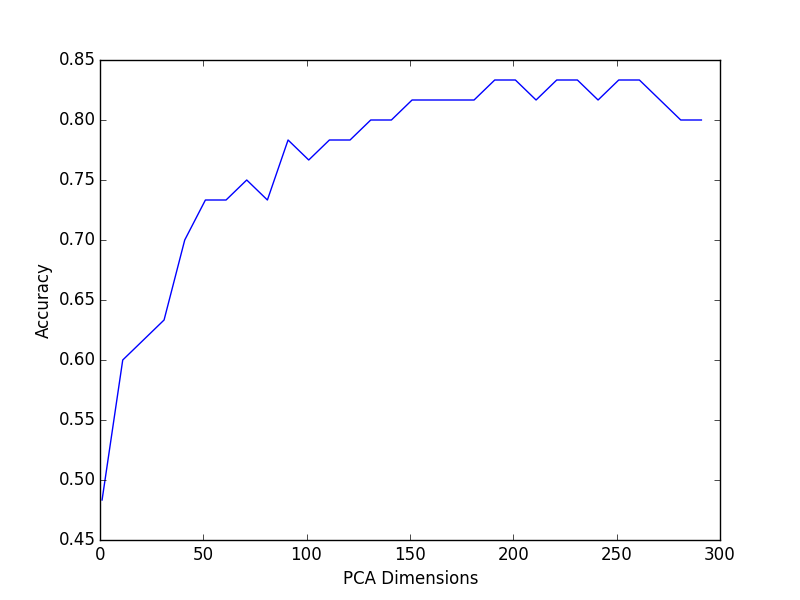
(array([ 14, 21, 22, 27, 33, 45, 65, 77, 117, 118, 148, 155, 160,

179, 182, 205]),)

# all this was done on test data. should probably be done on traingin data instead? or validation data!

#

% can I improve my prediction of getting rid of bad rmse features?

accuracy calculated as what percentage of 2 words guessed correctly by our model

pca:

[1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 201, 211, 221, 231, 241, 251, 261, 271, 281, 291]

accuracy : [0.48333333333333334, 0.59999999999999998, 0.6166666666666667, 0.6333333333333333, 0.69999999999999996, 0.73333333333333328, 0.73333333333333328, 0.75, 0.73333333333333328, 0.78333333333333333, 0.76666666666666672, 0.78333333333333333, 0.78333333333333333, 0.80000000000000004, 0.80000000000000004, 0.81666666666666665, 0.81666666666666665, 0.81666666666666665, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.80000000000000004, 0.80000000000000004]

Rank the rmse errors of all the words for each x datapoint and then find the rank of the right word!

ranks of the 60 words

[56, 44, 23, 41, 50, 17, 21, 1, 2, 31, 31, 7, 22, 56, 37, 6, 51, 46, 53, 46, 28, 56, 25, 35, 56, 26, 0, 30, 35, 19, 50, 3, 53, 57, 1, 26, 34, 43, 15, 40, 7, 32, 46, 55, 8, 55, 29, 44, 21, 39, 17, 57, 59, 54, 22, 44, 8, 46, 5, 53]

sum (rankarray <10)

11 words are ranked in the first 10 words… not very good!

100 dimensions :

[xtrainPCA,xtestPCA] = pcaData (100)

[ 0.106903 0.04334006 0.03921783 0.02457955 0.02261243 0.02033084

0.01710746 0.01473251 0.01196552 0.01153371 0.01075323 0.01019376

0.00941491 0.00901237 0.00813192 0.00782518 0.00726608 0.00695436

0.00630607 0.00623291 0.0059137 0.00567892 0.00543615 0.00526071

0.00491915 0.00488706 0.00478239 0.0047326 0.00455011 0.00447342

0.00439378 0.00437821 0.00429811 0.00420059 0.00415485 0.00401975

0.00395963 0.00387716 0.00382689 0.00377398 0.00370562 0.00363496

0.00360063 0.00354004 0.00353695 0.00347276 0.00340468 0.00340092

0.0033384 0.00331075 0.00329987 0.00321531 0.00317319 0.00313687

0.00311352 0.00310626 0.00305557 0.00303331 0.00302833 0.00299715

0.0029917 0.00297357 0.00293628 0.00291903 0.00291864 0.00285556

0.00284397 0.00284104 0.00280345 0.00280129 0.0027667 0.00275708

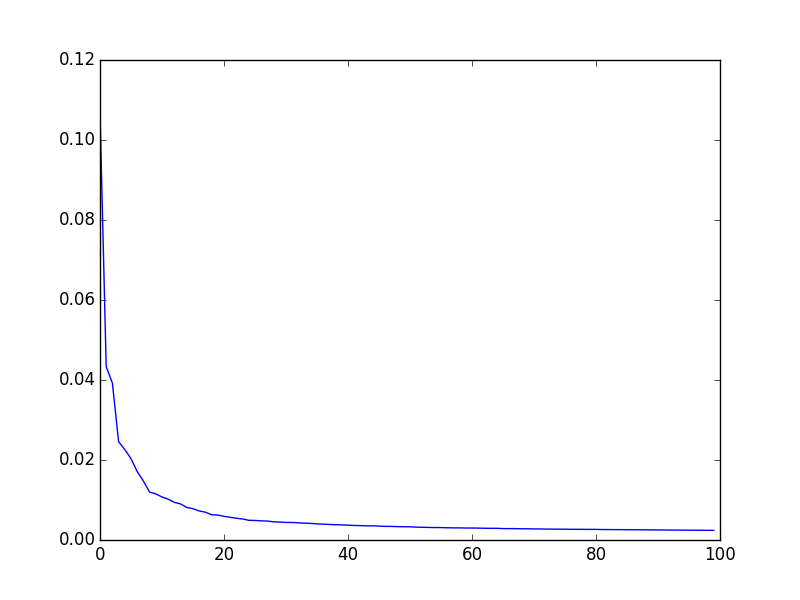
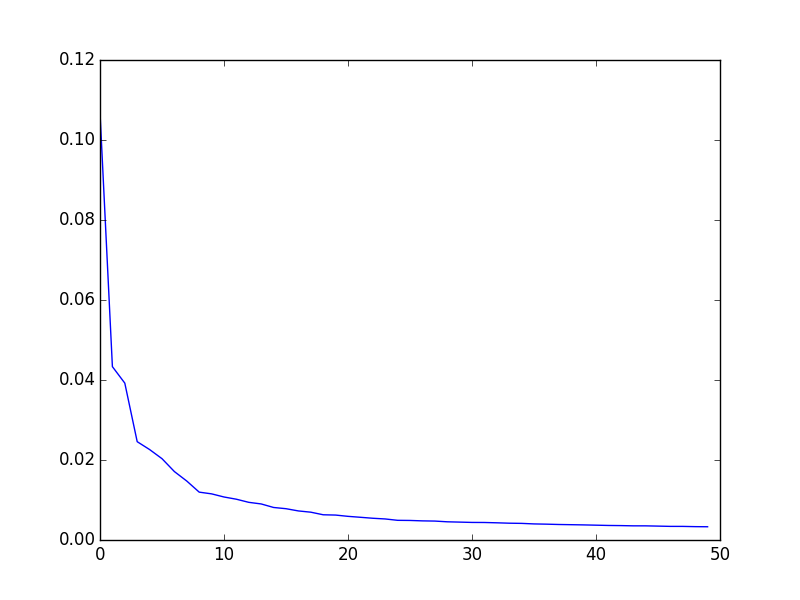
0.00272831 0.002717 0.00270268 0.00269826 0.00267912 0.00267003

0.00266353 0.0026549 0.00265318 0.00261344 0.00261154 0.00259858

0.00259032 0.00257686 0.00255975 0.00255053 0.00254799 0.00251963

0.00251352 0.00250116 0.0024787 0.00247781 0.00246315 0.00245193

0.00244606 0.00242886 0.00242076 0.00240633]



300 dimensions :

[xtrainPCA,xtestPCA] = pcaData (300)

[ 1.06902999e-01 4.33400636e-02 3.92178350e-02 2.45795544e-02

2.26124253e-02 2.03308376e-02 1.71074634e-02 1.47325118e-02

1.19655178e-02 1.15337114e-02 1.07532287e-02 1.01937555e-02

9.41490593e-03 9.01237039e-03 8.13191733e-03 7.82517674e-03

7.26608142e-03 6.95436470e-03 6.30607201e-03 6.23291149e-03

5.91370012e-03 5.67891585e-03 5.43614588e-03 5.26071388e-03

4.91915354e-03 4.88705617e-03 4.78239469e-03 4.73259912e-03

4.55010764e-03 4.47341634e-03 4.39377729e-03 4.37821147e-03

4.29811018e-03 4.20058932e-03 4.15484592e-03 4.01974554e-03

3.95962959e-03 3.87716211e-03 3.82688691e-03 3.77397936e-03

3.70561769e-03 3.63496293e-03 3.60063442e-03 3.54003940e-03

3.53694750e-03 3.47276136e-03 3.40468047e-03 3.40092337e-03

3.33839987e-03 3.31074531e-03 3.29986889e-03 3.21531059e-03

3.17319421e-03 3.13686553e-03 3.11352071e-03 3.10626042e-03

3.05557365e-03 3.03330894e-03 3.02832620e-03 2.99715485e-03

2.99170019e-03 2.97357033e-03 2.93627933e-03 2.91902774e-03

2.91863740e-03 2.85556246e-03 2.84396616e-03 2.84104328e-03

2.80344927e-03 2.80128535e-03 2.76669690e-03 2.75707759e-03

2.72831399e-03 2.71699649e-03 2.70268407e-03 2.69825852e-03

2.67911577e-03 2.67002646e-03 2.66353369e-03 2.65490127e-03

2.65317554e-03 2.61343607e-03 2.61154071e-03 2.59858158e-03

2.59031879e-03 2.57686434e-03 2.55975271e-03 2.55052806e-03

2.54798708e-03 2.51963284e-03 2.51352437e-03 2.50115647e-03

2.47869824e-03 2.47780636e-03 2.46315329e-03 2.45193199e-03

2.44606215e-03 2.42886376e-03 2.42076277e-03 2.40633230e-03

2.39987795e-03 2.39047232e-03 2.38790688e-03 2.37652233e-03

2.36078257e-03 2.35433399e-03 2.34376733e-03 2.33841245e-03

2.33092923e-03 2.32635598e-03 2.31706806e-03 2.31082692e-03

2.30073527e-03 2.28856263e-03 2.27883580e-03 2.27661167e-03

2.26067200e-03 2.24909573e-03 2.24357710e-03 2.23812429e-03

2.23020463e-03 2.22280303e-03 2.21208096e-03 2.19830344e-03

2.19286102e-03 2.18424704e-03 2.17825708e-03 2.17361103e-03

2.17256192e-03 2.15698888e-03 2.15060427e-03 2.14553917e-03

2.14285019e-03 2.13272014e-03 2.12495321e-03 2.12315161e-03

2.11327320e-03 2.10677060e-03 2.10219775e-03 2.09606680e-03

2.08936465e-03 2.08890617e-03 2.07697554e-03 2.06243748e-03

2.05848994e-03 2.05298434e-03 2.03543206e-03 2.03132739e-03

2.02423349e-03 2.02076050e-03 2.00956685e-03 2.00172711e-03

1.99732530e-03 1.99501784e-03 1.98351797e-03 1.97450685e-03

1.97178926e-03 1.96898315e-03 1.96300227e-03 1.95701758e-03

1.95150933e-03 1.94016248e-03 1.93652911e-03 1.92743566e-03

1.92191322e-03 1.91937718e-03 1.91336779e-03 1.90634545e-03

1.89591822e-03 1.88948925e-03 1.88063386e-03 1.87974494e-03

1.87432230e-03 1.86437443e-03 1.86103758e-03 1.85024930e-03

1.84849769e-03 1.84637577e-03 1.83799634e-03 1.83092832e-03

1.82698406e-03 1.82140543e-03 1.81869917e-03 1.80848210e-03

1.80600703e-03 1.80031665e-03 1.79311239e-03 1.78765792e-03

1.77788447e-03 1.77367126e-03 1.76697282e-03 1.76256901e-03

1.75996273e-03 1.75403670e-03 1.74814777e-03 1.74445017e-03

1.74010168e-03 1.73070332e-03 1.72692882e-03 1.72251396e-03

1.71230041e-03 1.70719804e-03 1.70391746e-03 1.70175381e-03

1.69693219e-03 1.69170706e-03 1.68480982e-03 1.67936777e-03

1.67321582e-03 1.66949278e-03 1.66507864e-03 1.65801323e-03

1.64838002e-03 1.64354131e-03 1.64024989e-03 1.63363500e-03

1.62851414e-03 1.62695007e-03 1.62393103e-03 1.62260827e-03

1.61068745e-03 1.60403631e-03 1.60000141e-03 1.59727602e-03

1.58773651e-03 1.57927601e-03 1.56920826e-03 1.56349589e-03

1.55517897e-03 1.55236533e-03 1.54531665e-03 1.54043283e-03

1.53471284e-03 1.52823436e-03 1.52414957e-03 1.52323715e-03

1.52021288e-03 1.51121935e-03 1.50457273e-03 1.49732106e-03

1.49278681e-03 1.48400327e-03 1.48220169e-03 1.47300439e-03

1.46651863e-03 1.45727127e-03 1.44767459e-03 1.44351115e-03

1.43829959e-03 1.43336116e-03 1.42580786e-03 1.42048824e-03

1.41448082e-03 1.41204175e-03 1.40918890e-03 1.39166310e-03

1.38677524e-03 1.38249874e-03 1.37487477e-03 1.36496817e-03

1.36221847e-03 1.35823282e-03 1.34553586e-03 1.34280270e-03

1.33645217e-03 1.33272494e-03 1.32572669e-03 1.31770482e-03

1.30520737e-03 1.30102491e-03 1.28740138e-03 1.27054809e-03

1.25761489e-03 1.25411958e-03 1.23552238e-03 1.22657064e-03

1.21464206e-03 1.19710079e-03 1.18819582e-03 1.15196179e-03

1.14612854e-03 1.13812894e-03 1.12942952e-03 1.11913461e-03

1.11247855e-03 1.10453082e-03 1.10041336e-03 1.07133527e-03

1.06724969e-03 1.05231586e-03 1.03941122e-03 1.01527354e-03

1.00636705e-03 9.93781592e-04 9.80131246e-04 9.71625174e-04

9.66583797e-04 9.58116692e-04 9.42693369e-04 3.70539101e-32]