Project 1

Yashin Chen (yc3347), Chandana Dayapule (cd3037), Zhengrui Fu (zf2181), Pranathi Pothireddy(pp2665), Guoan Wang (gw2355)

**Question 1**

If we set threshold = 0.01, the output is as below.

input russell\_cov.txt 0.01

n = 947

(run algo?) break>

Running power method...

eigenvalues: (in descending order)

[0.08317999205762726, 0.02177338976668163, 0.01899304694137161, 0.01812066206869238, 0.007890670462898874, 0.006289085826981539, 0.00431284676481721, 0.003792764257924883, 0.003717711789562164, 0.003128400973338284, 0.0029248254394070674, 0.0027188018933124456, 0.0026497959063269084, 0.0023914487616908623, 0.0022571036449301612, 0.0021622513515163404, 0.0021272770471184544, 0.002109554988743372, 0.0020427181145000442, 0.0019680108221144465, 0.001903114778080716, 0.0018256982094528205, 0.0017211265470319002, 0.001684364906241053, 0.001665012855756524, 0.001612014608733486, 0.0015348433349048915, 0.001537722017492547, 0.0014979991110868067, 0.0014675363032970355, 0.0014488276251633236, 0.0014188585376301638, 0.001411546909318065, 0.0013976215025880835, 0.0013690965014921783, 0.001362540636432121, 0.0013318621755994563, 0.0013048368050926219, 0.0012736936693933903, 0.001244530684195399, 0.0012188531143053666, 0.0011920586210441976, 0.0011684735660767117, 0.0011528551024020815, 0.0011314722374586817, 0.0011189205885634359, 0.0010808575864812869, 0.0010727499322723405, 0.0010661932495857345, 0.0010586778556550768, 0.0010150373759093751, 0.0010088381971229664, 0.000994291819808489, 0.000980870452529833, 0.0009757193481508184, 0.0009474347964425622, 0.0009429640632649301, 0.0009325233467835025, 0.0009213882319905541, 0.0009132811753972472, 0.0008947351474123684, 0.0008815996918385447, 0.0008718500632490831, 0.0008603880139980444, 0.0008472705301976529, 0.000842568020060209, 0.000837247461514652, 0.0008381984571891621, 0.0008295219106805706]

------------------------------------

eigenvectors: in the order of corresponding eigenvalues

[[ 0.02780536 0.00172651 0.02018446 ... -0.01389628 -0.0282692

-0.04620073]

[ 0.02686908 0.00163878 0.01893136 ... -0.01348256 -0.02726915

-0.0433216 ]

[ 0.05021518 -0.02374549 -0.02483743 ... -0.03500571 0.07388856

0.10384036]

...

[ 0.03624454 -0.00578154 0.00829994 ... -0.0067643 -0.0084943

0.01771671]

[ 0.02517553 0.01806278 0.01335787 ... -0.02243001 0.01566766

-0.01960539]

[ 0.0234415 0.01239355 0.00363416 ... 0.03423479 0.06687312

-0.05816543]]

Power method takes 12.358064000000013 seconds. (CPU Time)

**Question 2**

How we dealt with missing data:

We first computed log prices, and then we did linear interpolation on the log prices. The reason for doing so is that we assume the return during each NA period for each asset is the same. For instance, if P1 = $1, P2 = NA, P3 = $1.21, then we would fill P2 with $1.1, so that r1 = r2 = 10%.

If threshold = 0.01, the output is as follows:

(compute covariance matrix?) break>

Computing covariance matrix...

Covariance matrix:

[[2.05135803e-04 1.90999797e-04 4.78491450e-05 ... 7.69538697e-05

6.60201221e-05 6.43064145e-05]

[1.90999797e-04 1.97151555e-04 3.60586438e-05 ... 7.43845014e-05

5.92606296e-05 7.07267641e-05]

[4.78491450e-05 3.60586438e-05 1.13604160e-03 ... 1.11966064e-04

8.97583388e-05 9.15346328e-05]

...

[7.69538697e-05 7.43845014e-05 1.11966064e-04 ... 2.28355714e-04

6.76429366e-05 5.41143659e-05]

[6.60201221e-05 5.92606296e-05 8.97583388e-05 ... 6.76429366e-05

2.48163906e-04 5.26811779e-05]

[6.43064145e-05 7.07267641e-05 9.15346328e-05 ... 5.41143659e-05

5.26811779e-05 7.72245760e-04]]

eigenvalues: (in descending order)

[0.07892084201346122, 0.021192045067352165, 0.018665213736336575, 0.017865280602172403, 0.007573548916469625, 0.00597765014108055, 0.004148625685386775, 0.0036603835751237365, 0.0035397688652730004, 0.003108265369966163, 0.0028113035146880225, 0.0026632047826357845, 0.0025695445598036, 0.002353674574900324, 0.0022322758194031167, 0.0021153521208258147, 0.002059756937075985, 0.0020432901421806766, 0.002008121567227029, 0.0019204395778187526, 0.0018205705513168033, 0.0017739241666158746, 0.0017081632225007548, 0.0016683058794054136, 0.0016289158762697188, 0.0015823851407049372, 0.0015121475978102754, 0.0014886968752252634, 0.0014640702563582444, 0.0014598233158112388, 0.0014274533280963743, 0.0013834402026451018, 0.0013688333519628405, 0.0013598630259350406, 0.0013353079285465719, 0.0013266661625149672, 0.0012991106852227614, 0.0012773421398741087, 0.001252391236873228, 0.0012181864772222912, 0.0011974979667978183, 0.0011861035515956522, 0.0011691934071362373, 0.001135476445895419, 0.0011064890086500548, 0.0010937317193551563, 0.0010866345730758055, 0.0010674049952881849, 0.0010560396674847558, 0.001031358112973715, 0.0010112678410091574, 0.0010027134088221046, 0.0009887061246605959, 0.0009899513458326444, 0.0009515998459676909, 0.000947480027983917, 0.0009348669324874292, 0.0009241062478422533, 0.0009092269418429176, 0.0008961292116798205, 0.000891753399932556, 0.000868118594863245, 0.0008590279511844481, 0.0008511298128779279, 0.0008476728828130882, 0.0008443945353764816, 0.0008289785941989647, 0.0008175402191439696, 0.0008083581601845017, 0.000798550697375727, 0.0007892648241645475, 0.0007844444879533484]

------------------------------------

eigenvectors: in the order of corresponding eigenvalues

[[ 2.76801206e-02 1.04509650e-04 1.56299790e-02 ... 5.12240328e-02

-1.25953921e-02 -1.18700975e-05]

[ 2.68302032e-02 -1.25119822e-03 1.22289910e-02 ... 4.03593723e-02

2.31678646e-03 9.13601300e-03]

[ 5.06662975e-02 -1.68988690e-02 -1.85595868e-02 ... -3.31231539e-02

-8.17672032e-02 -6.92250615e-03]

...

[ 3.60327270e-02 -6.58974579e-03 6.50928457e-03 ... -2.10787839e-02

-4.00247177e-03 7.85449432e-03]

[ 2.51423422e-02 1.77707957e-02 1.48693942e-02 ... 1.24331447e-02

6.94783859e-03 -3.84529979e-02]

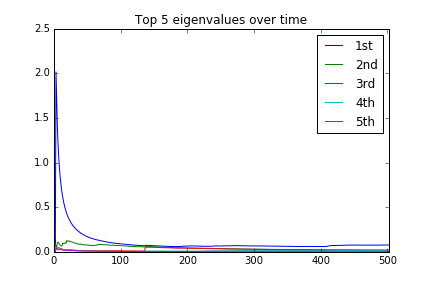
[ 2.38801349e-02 1.13736831e-02 3.07192352e-03 ... 2.42734304e-02

1.91629974e-01 -3.64126380e-02]]

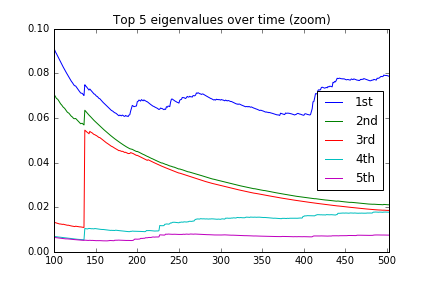
Power method takes 16.453363999999965 seconds. (CPU time)

**Question 3**

We plotted the evolution of top 5 eigenvalues over time.



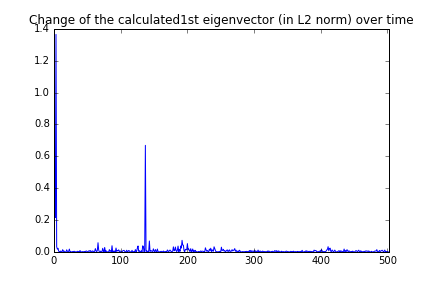
At first glance, the computed eigenvalues pretty much converge after t = 100. If we zoom in a little bit.

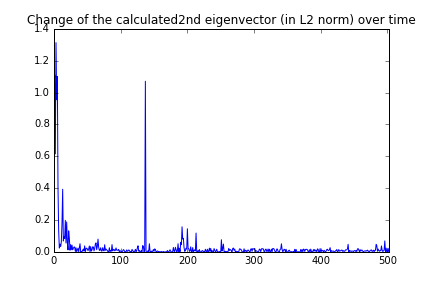


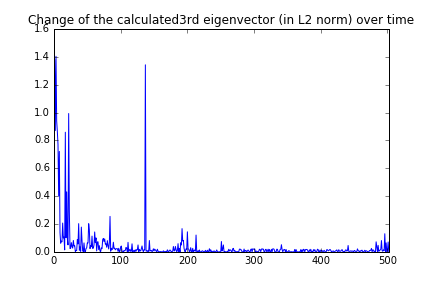
We see larger eigenvalues converge slower. For instance, the 5th eigenvalue converges when t = 100; the 2nd and 3rd eigenvalues do not converge until t ~ 400; the leading eigenvalue seems to be converging after t = 450.

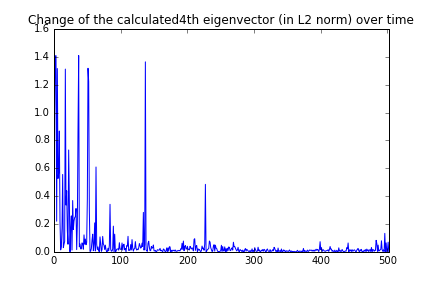
Next we show the evolution of computed eigenvectors over time. Since the eigenvectors are in a 947-dimension space, we cannot visualize them. However, to check convergence, for each eigenvector , we compute the following value in each iteration t:

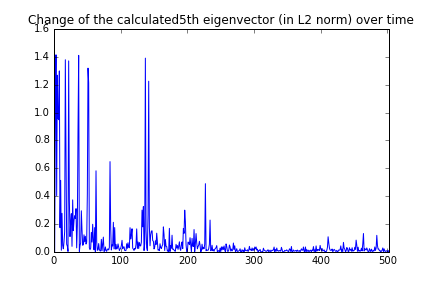
We expect this quantity would converge to zero for each i, when t approaches T (500+).











All of them pretty much stabilized around zero after t = 400, and the eigenvectors corresponding the larger eigenvalues converge a little bit faster.

**Question 4**

(start question4?) break>

Doing question4.

When k = 32 it takes 37.900589999999966 seconds. (CPU time)

When k = 64 it takes 45.02844200000004 seconds. (CPU time)

When k = 128 it takes 54.50201500000003 seconds. (CPU time)

When k = 256 it takes 63.625718000000006 seconds. (CPU time)

When k = 512 it takes 70.1645010000002 seconds. (CPU time)

When k = 1024 it takes 76.59927500000003 seconds. (CPU time)

We are not printing the eigenvalues and eigenvectors in the console this time (too many).

However, in differences.csv you can see the calculated eigenvalues using different methods compared against the result given by np.linalg.eigh().

Explanation: k here means the order that matrix has been raised to. E.g. if k = 32, we are doing M^32 \* v.

Compared to the original version, which takes only 16 minutes in CPU time (in Question 2), the new version obviously take longer time. This is because the dimension of the covariance matrix, n, is quite large (947), which makes matrix multiplication very costly.

Besides speed, let’s compare accuracy. We compare the computed eigenvalues (original power method, k = 32 to 1024) with the eigenvalues given by np.linalg.eigh(). The result is in differences.csv.

Original version: error is around 1e-8.

K=32: error is around 1e-6.

K=64: error is around 1e-7 or 1e-8.

K=128: error is around 1e-8 for large eigenvalues, 1e-6 for small eigenvalues.

K=256: error is below 1e-11 for large eigenvalues, 1e-7 for small eigenvalues.

K=512: error is below 1e-15 for large eigenvalues, around 1e-10 for small eigenvalues.

K=1024: error is below 0 for large eigenvalues, around 1e-15 for small eigenvalues.

Conclusion: as k becomes larger, errors become smaller. But the new method is very time costly given a large n.