```
>> dt  {Starting with the dt matrix corresponding to the database/statistics running example}

dt =

   24   21    9    0    0    3
   32   10    5    0    3    0
   12   16    5    0    0    0
    6    7    2    0    0    0
   43   31   20    0    3    0
    2    0    0   18    7   16
    0    0    1   32   12    0
    3    0    0   22    4    2
    1    0    0   34   27   25
    6    0    0   17    4   23

>> dd=dt*dt'  {Computing the dd matrix—the doc-doc correlation matrix which is dt*dt' (where dt' is the
transpose of dt) Notices it is a square positive symmetric matrix of size 10x10; because it is symmetric, it
will have +ve real-valued eigen values}

dd =

  Columns 1 through 9

        1107        1023         669         309        1863          96           9          78          99
        1023        1158         569         272        1795          85          41         108         113
         669         569         425         194        1112          24           5          36          12
         309         272         194          89         515          12           2          18           6
        1863        1795        1112         515        3219         107          56         141         124
          96          85          24          12         107         633         660         462        1203
           9          41           5           2          56         660        1169         752        1412
          78         108          36          18         141         462         752         513         909
          99         113          12           6         124        1203        1412         909        2511
         213         204          72          36         270         714         592         454        1267

  Column 10

         213
         204
          72
          36
         270
         714
         592
         454
        1267
         870

  >> eig(dd)
```

{Sure enough, if I find the eigen values of dd, I get the following. Look ma, keeping it all real..}

```
ans =

  1.0e+003 *

   -0.0000
    0.0000
    0.0000
    0.0000
    0.0230
    0.1455
    0.1813
    0.5260
    4.8336
    5.9845

>> sqrt(eig(dd))
```

{And if I take the +ve square root of these, I get the *singular values*; and these are the ones I had showed on the slide… Intuitively, notices that dd involves multiplying d-t by d-t’ or in some way, “squaring” it, so it is reasonable to have singular value of d-t be the square-roots of the eigen values of d-d}

```
ans =

        0 + 0.0000i
   0.0000
   0.0000
   0.0000
   4.7964
  12.0632
  13.4662
  22.9342
  69.5242
  77.3599
```

```
>> [df dl]=eig(dd)
```
{actually, the full eigen decomposition of dd gives both the eigen vectors (which is the orthonormal basis for the documents, and the eigen values—which are the squares of singular values). The eigen vectors correspond to the d-f matrix of our eventual svd}

```
df =

  Columns 1 through 6

    0.7179    0.1775    0.2545    0.0358    0.3249   -0.1874
   -0.0935    0.0204    0.0210   -0.1078    0.3492    0.2941
   -0.5655    0.2866   -0.1937   -0.0998    0.5388   -0.1528
    0.0070   -0.9337   -0.1013   -0.0086    0.2816   -0.0616
   -0.1659   -0.0639   -0.0563    0.0640   -0.6129    0.0565
   -0.3082   -0.0947    0.7912   -0.1857   -0.0534   -0.2494
    0.1377    0.0132   -0.0978   -0.5464   -0.0445   -0.0995
   -0.0810    0.0132    0.0071    0.7536    0.0699   -0.3898
    0.0484    0.0184   -0.0861    0.2092    0.0785    0.6018
    0.0752    0.0216   -0.4907   -0.1684   -0.1070   -0.5117

  Columns 7 through 10

    0.2327    0.0730   -0.1653    0.3994
   -0.7699    0.0090   -0.1551    0.3918
    0.4114   -0.0367   -0.1110    0.2327
    0.1423   -0.0141   -0.0512    0.1083
    0.1571   -0.0750   -0.2964    0.6813
   -0.0267    0.2256    0.3152    0.1397
   -0.0142   -0.6901    0.4029    0.1488
   -0.2055   -0.3819    0.2555    0.1231
    0.2312    0.1940    0.6483    0.2575
   -0.2060    0.5261    0.3139    0.1837


dl =

  1.0e+003 *
  Columns 1 through 6

   -0.0000         0         0         0         0         0
         0    0.0000         0         0         0         0
         0         0    0.0000         0         0         0
         0         0         0    0.0000         0         0
         0         0         0         0    0.0230         0
         0         0         0         0         0    0.1455
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

  Columns 7 through 10

         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
         0         0         0         0
    0.1813         0         0         0
         0    0.5260         0         0
         0         0    4.8336         0
         0         0         0    5.9845

>> tt=dt'*dt  {Now let's do the term-term correlation matrix, tt which is given by dt'*dt. Notice, once
again that it is a positive symmetric matrix. So we expect +ve real valued eigen values}

tt =

        3679        2391        1308         238         302         273
        2391        1807         953           0         123          63
        1308         953         536          32          87          27
         238           0          32        3277        1584        1573
         302         123          87        1584         972         887
         273          63          27        1573         887        1423
>> eig(tt) {sure enough the eigen values are all positive!}
```

```
ans =

  1.0e+003 *

    0.0230
    0.1455
    0.1813
    0.5260
    4.8336
    5.9845
```

>> sqrt(eig(tt)) {and taking their +ve square roots, we see that these correspond to the same singular values as we got for dd}

```
ans =

    4.7964
   12.0632
   13.4662
   22.9342
   69.5242
   77.3599
```

>> [tf tl]=eig(tt) {If we capture the full eigen decomposition—i.e., both the eigen vectors and eigen values, we get tf as the eigen vectors—which will correspond to the tf in the svd}

```
tf =

    0.0653    0.0831   -0.6074    0.0413   -0.2788    0.7352
    0.3976   -0.1757    0.7156   -0.0604   -0.2351    0.4900
   -0.9121   -0.0060    0.2758   -0.0741   -0.1215    0.2677
    0.0005   -0.3717   -0.0815   -0.4745    0.7417    0.2826
    0.0568    0.8916    0.1783   -0.0473    0.3663    0.1831
   -0.0496   -0.1704    0.0671    0.8728    0.4097    0.1854


tl =
  1.0e+003 *

    0.0230         0         0         0         0         0
         0    0.1455         0         0         0         0
         0         0    0.1813         0         0         0
         0         0         0    0.5260         0         0
         0         0         0         0    4.8336         0
         0         0         0         0         0    5.9845
```

{Now let's do SVD. We expect to get three matrices, df, ff, tf, where df would be the eigen vectors we found for dd, tf would be the eigen vectors we found for tt, and ff would be the square-root of the eigen value diagonal matrix (for either tt or dd—as they will both be same. }

```
>> [df ff tf]=svd(dt)  {Let's do it. And see, we did get the three matrices as expected}

df =

  Columns 1 through 6

   -0.3994    0.1653    0.0730   -0.2327    0.1874   -0.3249
   -0.3918    0.1551    0.0090    0.7699   -0.2941   -0.3492
   -0.2327    0.1110   -0.0367   -0.4114    0.1528   -0.5388
   -0.1083    0.0512   -0.0141   -0.1423    0.0616   -0.2816
   -0.6813    0.2964   -0.0750   -0.1571   -0.0565    0.6129
   -0.1397   -0.3152    0.2256    0.0267    0.2494    0.0534
   -0.1488   -0.4029   -0.6901    0.0142    0.0995    0.0445
   -0.1231   -0.2555   -0.3819    0.2055    0.3898   -0.0699
   -0.2575   -0.6483    0.1940   -0.2312   -0.6018   -0.0785
   -0.1837   -0.3139    0.5261    0.2060    0.5117    0.1070

  Columns 7 through 10

   -0.1329   -0.3190    0.6715   -0.2067
    0.1193   -0.0165   -0.0808   -0.0136
    0.4346    0.3616   -0.2800    0.2264
   -0.6309   -0.3656   -0.5904   -0.0439
   -0.0303    0.1173   -0.1510    0.0375
    0.2735    0.0070   -0.2508   -0.7916
    0.2944   -0.4700   -0.0391    0.1347
   -0.4419    0.5975    0.1401   -0.0521
  -0.1505    0.1306    0.0910    0.0765
   0.0291   -0.1572   -0.0242    0.4991


ff =

  77.3599         0         0         0         0         0
        0   69.5242         0         0         0         0
        0         0   22.9342         0         0         0
        0         0         0   13.4662         0         0
        0         0         0         0   12.0632         0
        0         0         0         0         0    4.7964
        0         0         0         0         0         0
        0         0         0         0         0         0
        0         0         0         0         0         0
        0         0         0         0         0         0


tf =

  -0.7352    0.2788    0.0413    0.6074   -0.0831   -0.0653
  -0.4900    0.2351   -0.0604   -0.7156    0.1757   -0.3976
  -0.2677    0.1215   -0.0741   -0.2758    0.0060    0.9121
  -0.2826   -0.7417   -0.4745    0.0815    0.3717   -0.0005
  -0.1831   -0.3663   -0.0473   -0.1783   -0.8916   -0.0568
  -0.1854   -0.4097    0.8728   -0.0671    0.1704    0.0496

>> df*df' {Now, notice that df is the *ortho-normal* basis for dd, so, multiplying it by its transpose should give us an identity matrix. Let's check. Yeah—it works.}

ans =

  Columns 1 through 6

   1.0000   -0.0000    0.0000    0.0000    0.0000   -0.0000
  -0.0000    1.0000   -0.0000   -0.0000    0.0000   -0.0000
   0.0000   -0.0000    1.0000   -0.0000    0.0000    0.0000
   0.0000   -0.0000   -0.0000    1.0000    0.0000   -0.0000
   0.0000    0.0000    0.0000    0.0000    1.0000    0.0000
  -0.0000   -0.0000    0.0000   -0.0000    0.0000    1.0000
  -0.0000   -0.0000    0.0000   -0.0000         0   -0.0000
  -0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000
   0.0000   -0.0000   -0.0000   -0.0000   -0.0000    0.0000
  -0.0000    0.0000    0.0000   -0.0000    0.0000    0.0000

 Columns 7 through 10

  -0.0000   -0.0000    0.0000   -0.0000
  -0.0000   -0.0000   -0.0000    0.0000
   0.0000    0.0000   -0.0000    0.0000
  -0.0000   -0.0000   -0.0000   -0.0000
        0    0.0000   -0.0000    0.0000
  -0.0000   -0.0000    0.0000    0.0000
   1.0000    0.0000    0.0000   -0.0000
   0.0000    1.0000    0.0000    0.0000
   0.0000    0.0000    1.0000   -0.0000
  -0.0000    0.0000   -0.0000    1.0000

>> tf*tf' {Same for tf—as tf is the orthonormal basis for the terms}

ans =

   1.0000    0.0000   -0.0000    0.0000   -0.0000   -0.0000
   0.0000    1.0000   -0.0000         0   -0.0000    0.0000
  -0.0000   -0.0000    1.0000   -0.0000    0.0000   -0.0000
   0.0000         0   -0.0000    1.0000   -0.0000    0.0000
  -0.0000   -0.0000    0.0000   -0.0000    1.0000    0.0000
  -0.0000    0.0000   -0.0000    0.0000    0.0000    1.0000

>> df*ff*tf'  {and if we multiply df*tf*tf’, we do get the original matrix back..}

ans =

  24.0000   21.0000    9.0000   -0.0000   -0.0000    3.0000
  32.0000   10.0000    5.0000   -0.0000    3.0000   -0.0000
  12.0000   16.0000    5.0000    0.0000   -0.0000   -0.0000
   6.0000    7.0000    2.0000   -0.0000   -0.0000   -0.0000
  43.0000   31.0000   20.0000   -0.0000    3.0000   -0.0000
```

```
    2.0000   -0.0000   -0.0000   18.0000    7.0000   16.0000
   -0.0000   -0.0000    1.0000   32.0000   12.0000    0.0000
    3.0000   -0.0000   -0.0000   22.0000    4.0000    2.0000
    1.0000   -0.0000   -0.0000   34.0000   27.0000   25.0000
    6.0000    0.0000   -0.0000   17.0000    4.0000   23.0000

>> ff {now looking at the ff matrix, we see the eigen values arranged in the decreasing order}

ff =

   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0   22.9342         0         0         0
         0         0         0   13.4662         0         0
         0         0         0         0   12.0632         0
         0         0         0         0         0    4.7964
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> nf=ff {We want to take just the top 2 dimensions lets, say. We copy ff into nf…}

nf =

   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0   22.9342         0         0         0
         0         0         0   13.4662         0         0
         0         0         0         0   12.0632         0
         0         0         0         0         0    4.7964
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> nf(3,3)=0 {and set all the diagonal entries of nf other than 1,1 and 2,2 to 0}

nf =
   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0         0         0         0         0
         0         0         0   13.4662         0         0
         0         0         0         0   12.0632         0
         0         0         0         0         0    4.7964
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> nf(4,4)=0

nf =

   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0   12.0632         0
         0         0         0         0         0    4.7964
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> nf(5,5)=0

nf =

   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0    4.7964
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> nf(6,6)=0

nf =

   77.3599         0         0         0         0         0
         0   69.5242         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0

>> df*nf*tf' {So now  if we multiply df*nf*tf' we will get the best two-rank approximation to dt}

ans =

   25.9199   17.8427    9.6675    0.2091    1.4473    1.0204
   25.2903   17.3881    9.4241    0.5707    1.6004    1.2030
   15.3839   10.6340    5.7557   -0.6345    0.4696    0.1767
    7.1509    4.9418    2.6749   -0.2741    0.2289    0.0939
   44.4916   30.6704   16.6119   -0.3885    2.1006    1.3289
    1.8343    0.1435    0.2308   19.3079   10.0062   10.9826
    0.6514   -0.9451   -0.3215   24.0282   12.3682   13.6107
    2.0500    0.4915    0.3921   15.8701    8.2527    9.0460
    2.0795   -0.8335   -0.1419   39.0637   20.1602   22.1631
    4.3631    1.8334    1.1534   20.2056   10.5978   11.5782

>> df*nf {Now while doing LSI, we don't need to compute the approximation—we just need the reduced coordinates of the documents, which is given by df*nf—and corresponds to just the first two columns of the original df*ff matrix}

ans =
  -30.8998   11.4912        0        0        0        0
  -30.3131   10.7801        0        0        0        0
  -18.0007    7.7138        0        0        0        0
   -8.3765    3.5611        0        0        0        0
  -52.7057   20.6051        0        0        0        0
  -10.8052  -21.9140        0        0        0        0
  -11.5080  -28.0101        0        0        0        0
   -9.5259  -17.7666        0        0        0        0
  -19.9219  -45.0751        0        0        0        0
  -14.2118  -21.8263        0        0        0        0
```

>> df*ff {here is df*ff matrix so you can check that df*nf is just the first two columns of this one}

```
ans =

  -30.8998   11.4912    1.6746   -3.1330    2.2603   -1.5582
  -30.3131   10.7801    0.2064   10.3670   -3.5474   -1.6751
  -18.0007    7.7138   -0.8413   -5.5394    1.8438   -2.5841
   -8.3765    3.5611   -0.3232   -1.9162    0.7432   -1.3505
  -52.7057   20.6051   -1.7193   -2.1159   -0.6817    2.9396
  -10.8052  -21.9140    5.1744    0.3599    3.0090    0.2561
  -11.5080  -28.0101  -15.8265    0.1919    1.1998    0.2137
   -9.5259  -17.7666   -8.7594    2.7675    4.7017   -0.3354
  -19.9219  -45.0751    4.4501   -3.1140   -7.2601   -0.3766
  -14.2118  -21.8263   12.0655    2.7734    6.1728    0.5132
```

>> dbq=[50 0 0 0 0 0] {Now if we got a query that corresponds to 50 instances of the word database and nothing else, its representation in the original space will be this…}

```
dbq =

    50     0     0     0     0     0
```

>> dbq*tf {To get its representation in the transformed space, we just need to multiply the query by tf; to take just the 2-d projection of the query, take the first two columns in the following. Notices that -36 and 13 are "sort of close" to the database docs which are (-30, 11), (-30, 10) etc—see df*nf}

```
ans =

  -36.7581   13.9393    2.0671   30.3701   -4.1554   -3.2649
```

```
>>>> sqlq=[0 50 0 0 0 0]
```
{Now if you have another query which is just 50 instances of sql, its representation is as below. Now, notice that the dot product of sqlq and dbq will be zero—since there is no dimension where they are both simultaneously non-zero. So, cosine similarity in the original space would have said that these queries are *completely* dissimilar!—even though you would think database and sql are “related” through co-occurrence.}

```
sqlq =

     0    50     0     0     0     0
```

```
>> sqlq*tf
```
{However, if we get the transformed coordinates of sqlq—which is given by sqlq*tf, we get the following vector. Now, if we again take just the first dimensions, we get (-24.5, 11). This, as you can see is pretty close (in cosine-theta terms) to the reduced transformed representation of dbq which is (-36, 13). }

```
ans =

  -24.5010   11.7533   -3.0220  -35.7785    8.7842  -19.8787
```

```
>>[-24.5 11.75]*[-36.75 13.93]' / (norm([-24.5 11.75])*norm([-36.75 13.93]))
```
{Indeed, their cosine theta metric, computed as their dot product divided by their norms, comes out very close to 1}

```
ans =

   0.9964
```

```
>> (sqlq*tf) * (dbq * tf)'
```
{Notice that if we *do not reduce* the dimensionality after transformation, then dbq*tf and sqlq*tf will still be orthogonal in the 6-D space. Only their 2-D projections are not orthogonal! This is not hard to see since the dot product of two vectors v and u is given by v*u’. Thus (sqlq*tf) * (dbq *tf)’, when simplified (and realizing that transpose over multiplication reverses the multiplication order), gives sqlq*tf*tf’*dbq’. Since tf*tf’=I, we are just re-computing sqlq*dbq’—or the dot product in the original space—which is still zero!).

```
ans =

     0
```

{So dimensionality reduction is important to realize the intrinsic similarity between these documents!..

But on the other hand, if you reduce it too low—say to 1-d, then *everything* will be fully similar to everything on the cosine-theta metric—do you see that?..}