Introduction

Timings : Ungrouped

Timings: Grouped

Functions

Speeding up R for Calculating Euclidean Distances

Gibril

Introduction

The task is to calculate euclidean distances between rows in a matrix. It is divided into two subproblems, one with groups and without groups. The following `R` packages have been used:

- `Rcpp`
- `microbenchmark` : for timings

A base `R` function was used as a benchmark for the timings. Three `C` source codes were used which were sourced into `R` via it's `C` API and a `C++` script via `Rcpp` library.

Function	Language
naive_dist	`R`
dot_dist	`R`
fastdist2	`C++`
eucDist	`C` & `R`
r_euc_dist	`C` & `R`
cDist	`C` & `R`

Timings : Ungrouped

Below are sample outputs from the functions for a matrix with 2000 entries for the ungrouped problem and relative timings of each of the functions for different matrix sizes. All timings were in milliseconds.

```
Unit: milliseconds
          expr        min         lq       mean     median         uq
 naive_dist(M) 115.139840 120.060930 130.419957 125.704581 132.208132
       dist(M)   2.432668   3.308361   6.356756   3.926424   5.074132
 r_euc_dist(M)   2.616436   3.330798   7.379464   3.833894   4.486725
   dot_dist(M) 673.753718 723.242430 768.162506 753.764015 796.262480
  fastdist2(M)  10.874602  12.146634  17.410744  13.472862  15.964630
    eucDist(M)   2.605100   3.456320   6.764453   3.846511   4.416576
      cDist(M)   1.877975   2.455699   6.573165   3.195958   3.880205
        max neval
  197.57561   100
   55.02963   100
   52.26835   100
 1017.22427   100
   59.76397   100
   52.69272   100
   54.42619   100
```

```
[1] "naive_dist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "dist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "r_euc_dist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "dot_dist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "fastdist2"
```

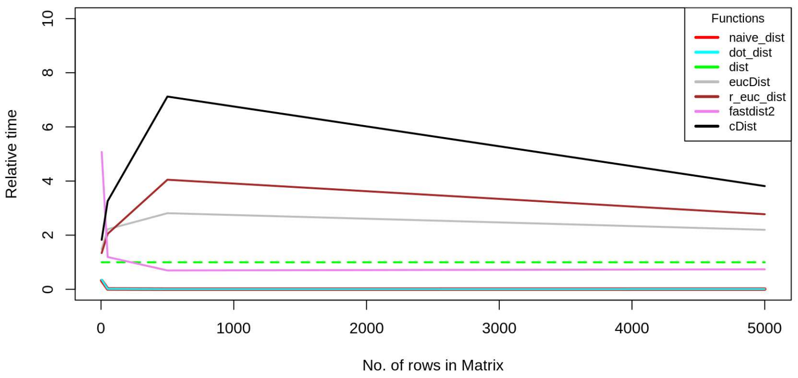
```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "eucDist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```

```
[1] "cDist"
```

```
[1]  8.104245 20.166204 24.063068 14.315379 28.043333 11.600828
```



Timings: Grouped

Output

Below are sample outputs from the functions `dist`, `cDist`, `naive_dist`, `r_euc_dist` and `eucDist` for the grouped problem. The outputs from all the functions are the same so only one is shown. The functions `dot_dist` and `fastdist2` could not be modified to allow for a vectorised calculation of distances within each group.

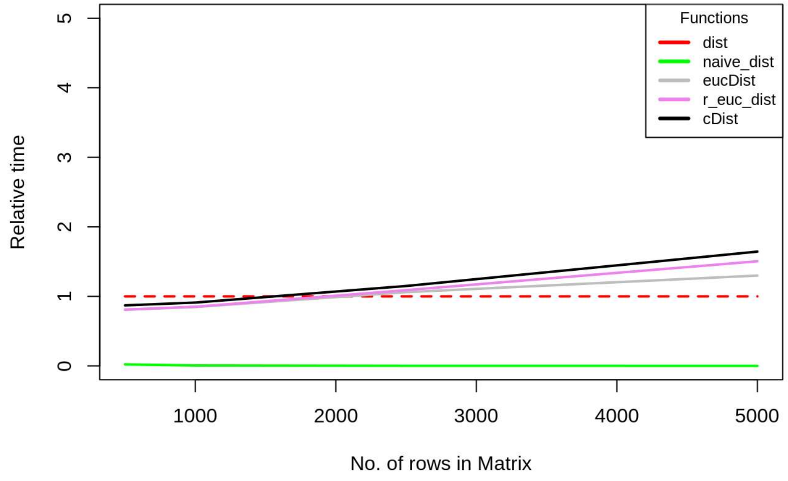
```
INDICES: 1
  [1] 17.953014 29.055633 20.280481 36.992981 30.461330 13.499654 25.497846
  [8] 23.251852  6.954638 16.455878 25.955105 25.275923 31.722903  4.423357
 [15] 28.973370 10.539937 11.934160 18.790168 21.836891 27.875367 22.281440
 [22] 21.972519 11.312633 12.047727 21.163439 12.516000 28.284468 17.196294
 [29] 17.983686 17.824807 24.551286 16.328599 11.391780 23.801526 15.540253
 [36] 11.791640 27.457483 29.623682 16.318462 14.364219 42.274637  4.641332
 [43]  5.686311 19.321377 11.759840  3.663682 39.515878 22.332847 20.439148
 [50] 29.064728 34.947336 20.782440 14.123785 27.434811 26.137581  2.586903
 [57] 37.957892 40.858410 21.349812 21.089434 53.301610  8.157001  7.588540
 [64] 30.933881 18.090438 24.649020  5.776565 29.720994 15.778034 17.257108
 [71]  5.756167  7.004466 12.595475 20.521479 21.068989 30.803808 29.225970
 [78] 27.347882  2.349000 36.501819 11.637599 17.158372 14.240403 48.986535
 [85] 34.087414 20.143698 38.627851 45.686806 32.519508 25.860807 38.994485
 [92] 33.242297  9.876369 44.173504 48.891321 23.299129 32.791032 63.246715
 [99] 19.402523 15.676941 39.989178 20.257810 23.948055 29.642258 34.587270
[106] 18.595866 12.031678 24.786872 27.948018  6.199463 39.863651 42.035636
[113] 24.577264 19.563353 53.372232  8.353472 10.237007 27.838746 36.742003
[120] 10.459902  9.820404 29.058261 31.388995 32.145094 17.921645 40.050324
[127] 16.321148  8.184070 32.289758 24.998375 14.511431 31.689095 33.376306
[134] 35.125437 20.235691 19.130024  1.783070  8.251995  6.829934 26.116372
[141] 24.457322 35.919360 33.407953 32.968731  3.709927 38.127436 15.671773
[148] 21.764781 28.033662 37.462474 34.310834 29.039445 41.781881 18.829433
[155] 18.324429 26.853645 33.795420  4.588265 32.069956 51.108088 20.880775
[162] 14.816763  9.813387 21.035600 21.935863 25.916068 10.309388 29.596191
[169] 16.185385 13.461859 23.894291 16.800369 24.629930 21.307082 22.975012
[176] 20.584030 24.261555 22.644066 20.116445 36.058097 23.696190 17.371455
[183] 33.491747 16.918318 19.264459 26.792544 30.180154  6.659574  7.474232
[190] 26.276118 22.965813 36.448634 34.348428 32.352984  4.235299 39.680138
[197] 14.416900 20.619470 13.578560 24.463387 16.334162 35.734367 35.339899
[204] 27.697942  7.837808 43.503276  8.324463 14.652728 32.693475 29.779383
[211] 41.959849 38.549007 39.752621 10.412682 40.518098 21.732659 28.003803
[218] 25.756364 11.922318 15.754527 14.369898 22.407267 32.288422 20.116466
[225] 18.753026 37.410967 40.875412 19.619360 22.965571 54.061317  9.532075
[232]  7.016702  9.171035 22.362524 32.293850 27.805307 32.013216 30.471149
[239] 29.207501 30.124721 18.692785 33.727117 33.888535 29.678092 46.602315
[246] 19.983015 14.599927 36.160540 13.668712 19.377657 45.261491 47.577840
[253]  6.328346
--------------------------------------------------------
INDICES: 2
 [1] 25.070363 18.635368 18.419634 10.360622 23.919160 10.530893 12.605604
 [8]  9.364406  7.710103 19.359355  7.535748 22.657248  1.171562 21.630424
[15] 29.083264 34.180305 32.291281 12.829916  9.003334 18.437643  8.402657
[22] 29.871398 24.060701 22.111228 15.122401  6.400935 14.096427 24.829443
[29] 27.189959 25.244507 21.522147  1.032235 22.618631 15.089336 13.123231
[36] 20.496852 28.144042 33.015840 31.124347 22.564180 15.828815 13.843382
[43] 17.002818 16.696480  2.013010
--------------------------------------------------------
INDICES: 3
  [1] 11.0827883 11.2511375 22.2275477  7.9307743 26.3689306 11.3689916
  [7] 12.8371395 11.8389850 19.4063979 27.6747030  7.5432076 19.4293864
 [13] 15.3750493 29.8261897 13.2493712 16.4617449 18.4489620 10.9548684
 [19] 22.9719569  4.9598925 17.9302442 17.8668528  8.2669248 16.3415157
 [25] 19.2145139 26.2384374  4.5580805 14.1629337 14.4030162 27.1831741
 [31]  8.5114796 22.6073444  7.5537603 12.3762816  6.6035482 17.1133020
 [37]  8.9680268  4.3016728 22.2843098  8.8777218 16.8465729  7.3990575
 [43]  8.8221535  4.2692489 18.7024781  4.5046454 12.6294179 17.1597696
 [49] 18.9688981 20.0319074 12.9845851 14.8855374 33.9594638  3.8410106
 [55]  6.6020478 19.7726672 12.3609812  8.5726560  9.8577685 14.7479481
 [61] 11.2291641 27.9498979 18.5550901 12.9107964  5.6550226 16.7570249
 [67] 15.3875093 23.0548752  0.8371191 12.4625532 10.6033305 24.5129460
 [73]  6.0472908 17.6617656 12.2697668 25.6055171 13.7596575 34.2463491
 [79] 17.1572135 18.1390211 19.1259524  8.8839330 15.5599254 16.6424830
 [85] 13.3693705 27.7487329 16.5679686 13.2169380 22.9725925 11.6666240
 [91] 19.4059144 13.3535667 16.8203716 10.2678407 22.3270685 13.4549138
 [97]  5.1531622 25.0568490 22.4110628 11.0692951 17.9980288  6.4439084
[103]  6.8171980  6.3616798 19.1303751  0.4131294 16.8941594 13.3374891
[109] 30.9322458 39.0980863 15.9712944 29.2045664 26.5513661 40.9530954
[115] 22.8000637 27.8974991 20.8857800  8.2885130 16.2074994  8.8954327
[121]  4.8281122 10.8049795 10.9202450 11.5388163 24.1114354 23.8913252
[127] 13.0060877 12.5838285  3.4511276 17.7314225 17.8031134 29.7976503
[133] 13.2350309 11.4308830 25.3467179  6.8297461 18.1885273 12.0232829
[139]  6.6780989 13.2559063  6.4178691 18.8847458 16.8059603 14.4706135
[145]  6.2687077 12.2741209 19.6255371 18.8121539 21.0851930 29.9134411
[151] 17.0436816 13.3742778 29.5367086
--------------------------------------------------------
INDICES: 4
 [1] 18.727833 22.609260 27.706955 24.859736 16.414001 15.526164 25.097087
 [8] 20.918897 45.855401 27.639818  9.852353 14.212768 20.301051 23.184338
[15] 20.312290 22.273968 26.039438 44.164350 20.004735  5.219458 11.637192
[22] 19.358671 16.435289 13.975179 20.651912 34.847863 10.383092 11.842779
[29] 22.497874 19.761169 14.328803 22.860720 32.758197  8.834138 13.173946
[36] 11.319732  2.489941 11.941703 23.958802  4.216174  2.979977 11.940428
[43]  4.506454 29.775759 17.278540 10.573791  6.218556 30.332264 15.218778
[50] 10.021825 22.343335  6.323417 25.399193 16.147233 24.503278
--------------------------------------------------------
INDICES: 5
  [1] 29.3820180 11.4639083 24.1979599 21.8866912 46.7052459 17.6851670
  [7] 43.3289224 12.2789476 15.3897251 17.8626056  7.0978137  5.2571201
 [13] 18.0184777 16.7435287  6.3671569 11.0055382 37.4873787 33.1199864
 [19] 18.5596701 17.1871578 11.5217044 22.9447865 10.7973528 22.2529770
 [25] 23.2830048 14.8127120 33.6906403 26.3834083 33.9471117 38.5737617
 [31] 26.5032481 43.4321584 36.7128367 22.5811579 27.5551988 25.5222960
 [37] 29.2273410 26.5585478 23.4273057 22.0491337 33.2214712 24.9879629
 [43] 14.6143605 18.2678795 18.0388703 29.9679217 45.8288147 13.7184464
 [49] 13.7586233 35.8709396 11.7362133 32.2464719 19.7118815 21.1000355
 [55] 13.9400221  6.7624872  6.6169679  7.1861052  8.0090392  5.3171894
 [61]  0.4590424 26.3700509 22.3332831  7.5270581  8.7262830  7.0680719
 [67] 11.8182094  6.6881920 22.5801393 19.7916429 22.5148888 21.2487864
 [73] 25.5658964 27.9947603 34.6013398 24.3615241 17.4682075 20.0944248
 [79] 12.2637850 16.4518607 18.8015533 14.0955035 13.6373616 18.6910613
 [85] 11.5447786  7.7846471 13.7317167  3.4061844 19.5831308 34.1067757
 [91] 36.5296985  5.0129049 24.0086952 32.7234089 21.9472064  7.4781601
 [97] 20.1010877 16.8938311  8.2592015  5.7696643 16.2039063 14.0149797
[103] 26.8084389 13.7234873  8.7175955 19.9791166 20.7128763 16.4076335
[109] 11.1006319 16.4788155 40.0921579 23.0536793 49.4318703 55.8573254
[115] 43.2773658 39.9564026 42.4272180 32.0213667 35.9801580 41.1223944
```

```
[121] 36.2877861  9.8688857 25.9945344 31.2148554 29.9712495 36.0233490
[127] 24.0674312 40.8553072 52.9130273 28.9674742 29.1697766 17.0716154
[133]  3.2185331 17.1662916 13.0547136  8.9886625  4.8235065 12.6383405
[139] 11.8429996 30.2437269 18.6097538  9.7321266 19.5171646 18.7598281
[145] 18.0711013  7.0432119 13.0417173 51.3570079 45.9444394 31.3967185
[151] 38.7369546 38.0882473 25.3149131 27.2112493 36.9692443 32.6916875
[157] 18.0973234 10.4481035 24.8097381 32.0913652 36.7325675 23.8525568
[163] 33.5683476 39.0923059 26.4193895 29.8396564 12.9494022 16.1240431
[169] 26.8819119 27.0800240 16.5841370 19.3249207 41.5001112 41.8545889
[175] 27.1526831 20.2435520 14.7276060 28.2329070 22.1301657 34.4350233
[181] 14.5691763 21.0715379 15.8772066 23.8653995 20.1382672 16.9016016
[187] 20.7982288 46.0292893 35.6571828 24.6658481 29.5028762 25.1959296
[193] 32.1513066 15.0933219 10.7473139 18.5839202 13.8400055 12.1309011
[199]  7.9116820 13.7193835 13.9623203 33.4371578 21.1874388 12.8923887
[205] 22.1641281 20.7601315 21.2277436  8.1091991  9.8314858  5.5047211
[211] 13.9382924 14.3557368  5.0291300  6.3783459 30.9910771 29.0025755
[217] 14.2338987 10.1581436  4.4421491 16.6674281 10.4747757 25.5698087
[223] 12.7921620 11.5629363  1.3058150  6.1781117 32.9772921 27.8636154
[229] 13.3588693 13.9043031  9.3355975 18.4110689  6.0115023 20.1265393
[235]  4.2322455 11.6551110  7.5955129 22.1737123 15.2142767  0.8098070
[241] 11.7506709 13.3167155  9.0973987  9.1205122 21.9327186 10.7177080
[247]  8.2517386 26.1120195 16.7886181  5.0125972 14.9454733 15.0195742
[253] 13.3262166  6.3658514 17.7011916  4.8758072 31.6724437 26.7949416
[259] 12.1926440 12.7229096  8.4469981 17.1054772  5.6689213 20.5713655
[265] 26.8020399 22.7617283  7.9545851  8.9156665  6.9206750 12.2433583
[271]  6.5467990 22.4785705 17.5415815 21.3693016 21.4015194 27.3680063
[277] 14.5744608 31.0757107 43.1145208 14.8071566 23.7277011 27.5490692
[283] 16.0488582 23.1531244 29.4971701 11.4223336 13.3594859  8.3394338
[289]  9.8788491 22.7048906  6.0521204  8.2779137 15.4086453 31.2172640
[295] 13.5496165 12.9140676 28.7211420 17.0582168 31.0267910 15.9523373
--------------------------------------------------------
INDICES: 6
  [1]  4.0441042  5.9045997 11.6579123  7.8531216 14.6201625 18.4785922
  [7] 25.1311458 10.8363133 11.9394824  2.7796588  6.8988923  5.3268879
 [13] 10.4180541 21.6636383  5.3571082 10.7979049 10.3773852 16.8631753
 [19]  9.8541898 15.3318590  4.9481560 10.5985193 21.7533541 21.1240035
 [25] 14.4423352 15.9428503  2.8668424  3.8886815  9.0881655 13.9086512
 [31] 25.1182209  6.5779947 14.1666285  8.3660870 20.0205368  8.5361791
 [37] 12.5624580 20.2371956 15.9023971 30.6467642  8.1145784  6.1136667
 [43]  8.4434415 11.8331936  1.3425374  5.2339824 18.4409289  6.9672350
 [49]  8.8733033 13.8705814 14.7072702 19.4997718 25.7374022  7.3666387
 [55] 36.2320355  0.9921101  7.6855031 12.7208366 18.5567135  9.8375187
 [61] 11.7658212 10.1182198 15.1267156  2.2362520 21.6558834  6.2546180
 [67]  8.4596838 26.2916429 18.3525159 18.6885419 18.5382097  7.7509815
 [73]  1.0602809 11.4258506 15.2070279 29.5148970  6.4978654 18.6209929
 [79]  3.7145823 24.6308522 31.6460186 10.5314781 24.8052692 26.3478919
 [85] 13.0200317  8.7626881 19.2821552 23.5218360 35.1826849 14.8997806
 [91] 24.3335012 10.2962635 29.8085542 41.9239516  7.8959735 13.9205541
 [97] 18.9441433 25.2930757 17.2031901 18.7183711  3.7107038 22.4090067
[103]  7.6811032 28.7455155  1.9129938 35.2897675 36.7350906 23.5314135
[109] 18.9027721 29.6250696 33.5446554 45.5184498 24.8194806 34.7602501
[115] 18.8837168 40.0551083  8.0109527 11.7972897 17.7313085  9.3767311
[121] 11.7300145 10.8494759 14.5142554  1.4571632 20.9380646  6.5953082
[127] 14.2888751 17.8694376  7.1216521  5.2674312 15.3660518 12.4714854
```

```
 [73] 16.9163478 11.0154663 11.6839607 22.3880203 15.8594851  3.0805736
 [79] 16.2239634  4.2999055 43.4922993 18.9016926 37.1978306 32.1953631
 [85] 26.7416460  9.1984805 16.0028349 31.1615014 29.5055651  9.0856763
 [91] 20.7530208 17.4025562 26.6868509 30.5369920 41.0260916 12.2008324
 [97] 18.4563845 35.5593728 30.5159279 14.8964045 29.2538813 34.8921374
[103] 22.9264451 31.3277567 18.6091228 21.2950308 18.3611286 20.5101089
[109]  9.9521364 19.1393139 15.8199561 10.8198502 10.8228251 14.4398688
[115]  1.4992856 20.6108049 31.5319506 39.1936914 29.9426687 40.3940772
[121] 28.0741716 13.1346657 31.2717285 34.4804264 22.4003096 39.3509599
[127] 11.6063212 23.7702777 21.4605912  3.4596227 18.7151739 23.2905940
[133] 12.7791648 19.1276311 11.8116278 20.5952205 12.1917354 13.7576698
[139] 26.1215865 19.3933134  6.7292914 20.4230256  0.2054087 13.3133517
[145] 22.3555967 20.7482884  1.6624094 13.9374842  8.4704960 20.6001299
[151] 22.0768478 28.6540879 11.6681927  8.8144981 18.1903751 12.0693842
[157] 15.2717791 22.1031602 13.2661367 16.7582437 13.9596975 21.6381430
[163] 21.8086185 12.2822051 26.2841058 12.8246573  9.3901902 19.3864820
[169] 14.1235593  6.7607724 20.5104029
--------------------------------------------------------
INDICES: 50
  [1] 19.776367 19.842431  4.825416  7.272741 16.443856 24.392573 13.159657
  [8] 14.516723 17.915316 18.202602 25.700908  6.870605 29.675683  8.731881
 [15] 13.971889 26.172183 14.963268 12.760197 30.997687 16.191113 11.405456
 [22] 34.269936  9.691621  4.867070 25.357890 23.470902 24.785717 28.261851
 [29] 14.592502 20.111909 18.716173 11.300265 16.175102 15.027124 26.832926
 [36] 17.058877 21.522704  8.705245 14.461980 14.925285 24.679821 11.654361
 [43]  2.936604 19.373589 21.251935  9.750491 19.308410 14.105315 13.677821
 [50] 24.462561 10.082305 27.542086 13.369993 11.564800 19.648786 18.388858
 [57]  6.907180 21.743142 11.169646 10.935550 22.232627 11.016503 24.934672
 [64] 15.994083  9.069975 25.916760 19.946258 17.131902 23.920317 27.222992
 [71] 20.005347  9.588683 26.169709 17.077713 17.543076 11.501556 37.019750
 [78]  7.835830 11.755360 10.883701 23.339524  8.625715 32.515690 10.605339
 [85] 26.784463  4.886938  7.278911 16.288727 13.860228 18.222818 21.658095
 [92]  3.409690 31.671365 32.604115 34.945083 13.748695 40.319489  6.252806
 [99] 26.485648  4.824734 15.695057 18.643440 15.856245 26.498468  6.377268
[106] 20.504221 20.597306 20.225212 26.929220 10.131885 21.730779  6.377538
[113] 32.026096 13.262573 26.739458 10.321077 12.935558 36.828722 15.987264
[120] 21.916120
```

Relative times

The graph below shows the relative timings of each of the functions for different matrix sizes. All timings are in milliseconds.



Functions

The function `naive_dist()` is written purely in R using the formula `dist = sqrt((xi-xj)^2 + (yi-yj)^2)`, `dist()` is the base `R` implementation, `dot_dist()` is also written purely R but uses the dot product method of calculating distances, `fastdist2()` is a `C++` implementation of the calculation and `r_euc_dist()`, `eucDist()` and `cDist` are `R` wrapper functions for `C` implementaions of the calculation.

`naive_dist()`

```
naive_dist <- function(x){

#TODO: function documentation

    result = vector('numeric', length = nrow(x) * (nrow(x) - 1)/2)
    k = 1
    for (i in 1:(nrow(x)-1)){
        for (j in (i+1):nrow(x)){
          result[k] = sqrt((x[i,1]-x[j,1])^2 + (x[i,2] - x[j,2])^2)
          k = k + 1
        }
    }
    result
}
```

dot_dist()

```
dot_dist <- function(x){
# TODO: function documentation
  result = vector('numeric', length = nrow(x) * (nrow(x) - 1)/2)

  i = 1
  m = 1
  while (i <= (nrow(x)-1)){
    for (j in (i+1):nrow(x)){
      k         = x[i,] - x[j,]
      result[m] = sqrt(k %*% k)
      m         = m + 1
    }
    i = i + 1
  }
  result
}
```

fastdist2()

```
// filename: fastdist2.cpp

#include <Rcpp.h>
using namespace Rcpp;

//[[Rcpp::export]]

NumericVector fastdist2 (const NumericMatrix &x){
    unsigned int outrows = x.nrow(), i = 0, j = 0, k = 0;

    Rcpp::NumericVector out(outrows * (outrows - 1)/2);

    for (i = 0; i <= outrows - 2; i++){
        for (j = i+1; j <= outrows - 1; j++){

            out(k) = sqrt(sum(pow(x.row(i) - x.row(j), 2.0)));
                    k++;
}
    }
    return out;
}
```

r_euc_dist() and euc_dist()

```
# filename: r_euc_dist.R
# last edited: 7-AUG-2019


r_euc_dist <- function(matrix, cols_interest = 1:2){

# R wrapper function for euc_dist.c
# check euc_dist.c for source code and function documentation
# input:
#       matrix: an N by 2 numeric matrix
#       cols_interest : if input is a data.frame, the cols to be used.

# output: a vector of euclidean distances between rows
    cpMat   = matrix[, cols_interest]
    len_mat = dim(cpMat)[1] * dim(cpMat)[2]
    n_rows  = nrow(cpMat)
    vec     = as.vector(t(cpMat))

    .C("euc_dist",
       as.double(vec),
       as.integer(len_mat),
       as.double(vector("double", n_rows * (n_rows - 1)/2)))[[3]]

    }

/*
filename: euc_dist.c
last edited: 23-JUL-2019
*/


#include <R.h>
#include <math.h>

/*
Calculates the euclidean distance between rows of a matrix

input:
    vec: a one dimensional array of length N * 2 where N is the number of rows
       in the input matrix.
    len_vec: N * 2
output:
    result: a one dimensional array of length N * 2 containing the calculated distance
s
*/

void euc_dist(double *vec, int *len_vec, double *result)
{
    int k = 0, i, j, iter1 = *len_vec - 3, iter2 = *len_vec - 1;

    for (i = 0; i < iter1; i += 2){
        for (j = i + 2; j < iter2; j += 2){
            result[k] = sqrt((vec[i] - vec[j]) * (vec[i] - vec[j])
                        + (vec[i+1] - vec[j+1]) * (vec[i+1] - vec[j+1]));
            k++;
        }
      }

}
```

eucDist() and eucDist()

```
# filename: eucDist.R
# wrapper function for eucDist.c
# TODO: error correcting code

eucDist <- function(matrix, cols_interest = 1:2){

    cpMat   = matrix[, cols_interest]
    len_mat = dim(matrix)[1] * dim(matrix)[2]
    n_rows  = nrow(matrix)
    n_cols  = ncol(matrix)
    vec     = as.vector(t(matrix))


    .C('eucDist',
       as.double(vec),
       as.integer(n_rows),
       as.integer(n_cols),
       as.double(vector("double", n_rows * (n_rows - 1)/2))
       )[[4]]
}

#include <math.h>
#include <R.h>
#include <Rinternals.h>

/* Euclidean distance */
/*q=.C("eucdist",as.integer(c(1,2,3,4,5,6,7,8)),as.integer(4),as.integer(2),as.double
(vector("double",6))*/
void eucDist(double *x, int *m, int *n, double *d)
{
  /* Arguement:
    1.  x is a matrix of dimension n by m
    2.  m is the number of rows
    3.  n is the number of coloums
    4.  d is the pointer for output */
  /*
   d = sqrt(sum((XI-XJ).^2,2));            % Euclidean
   */
  int i,j,k; /* **pointer; /* Indexers */
  int local_m, local_n;
  local_m = *m, local_n = *n;
  double theSum; /* size_t is an unsigned integer of size 16 bits */
  /*
  XI for indexing rows
  XJ for indexing columns
  XI0 unknown for now
  */
  int  XI, XJ, XI0, index; /* pointers as row indexers*/
  // d = malloc( local_m*(local_m - 1)/2);
  // XI0 =  (double *) x; /* we are not tuching x but using its memory address as XI
*/
  // x = (double) x;
  index = 0;
  for (i=0; i<local_m-1; ++i) { /* Iterating through the rows of the matrix */
    // XI0 =  XI; /* taking the memory address of the array (Refer to line 29) */
    XI = i*local_n; /* Move along memory by n ( the first coloumn */
    XI0 = XI;
    // Rprintf("XI is %d\n", XI);
    for (j=i+1; j<local_m; ++j) { /* iterating through the rows from the i_th row*/
      // XI = x + i*(*n); /* Change to XI happpens here after using it on line 28*/
      XJ = j*local_n;
      // Rprintf("XJ is %d\n", XJ);
      // XI = XI0; /* Index? */
      theSum = 0.0;
      for (k=0;k<local_n;k++,++XI,++XJ){
        theSum += pow((x[XI]- x[XJ]), 2.0);
        // Rprintf("x[XI] is %lf and x[XJ] is %lf\n", x[XI], x[XJ]);
        // Rprintf("The sum is %lf\n", theSum);
        // Rprintf("The sum is %d\n", theSum);
      }
      XI = XI0;
      d[index++] = sqrt(theSum);
      // Rprintf("d is %lf\n", d[index]);
      // XI = XI0; /* Index? */
    }
  }
}
```

cDist() and cDist()

```
/*
filename: cDist.c
last edited: 24-JUL-19
*/


/*
 exactly as euc_dist.c but modified to use R's .Call() interface

 input:
    Rvec: a one-dimensional array of length N * 2 where N is the number of rows
          in the input matrix.

    resLen: an integer (coerced to double) of the number of pairs
            calculated as N * (N - 1)/2.
 output:
    result: a one-dimensional array containing the calculated distances.
*/

#include <R.h>
#include <Rinternals.h>
#include <math.h>

SEXP cDist(SEXP Rvec, SEXP resLen){
    int k = 0, i, j, iter1 = length(Rvec) - 3, iter2 = length(Rvec) - 1;

    SEXP result = PROTECT(allocVector(REALSXP, asReal(resLen)));
    double *vec = REAL(Rvec);

    for(i = 0; i < iter1; i += 2){
        for(j = i + 2; j < iter2; j += 2){
            REAL(result)[k] = sqrt(((vec[i] - vec[j]) * (vec[i] - vec[j])) +
                         ((vec[i+1] - vec[j+1]) * (vec[i+1] - vec[j+1])));
            k++;
        }
    }

    UNPROTECT(1);
    return result;

}

# filename: cDist.R
# last edited: 7-AUG-19


# wrapper function for cDist.c
# see cDist.c for function documentation
#
# input:
#       matrix: an N by 2 numeric matrix
#       cols_interest : if input is a data.frame, the cols to be used.
#
```

```
# output: a vector of euclidean distances between the rows of the matrix.
# TODO: error-correcting code



cDist <- function(matrix, cols_interest = 1:2){
    cpMat  = matrix[, cols_interest]
    vec    = as.vector(t(cpMat))
    reslen = nrow(cpMat) * (nrow(cpMat) - 1) * 0.5

    .Call("cDist",
           as.double(vec),
           as.double(reslen))
}
```