**********DRAFT**********

Understand the SVD

Using Octave

Using C RaspBian

Using Bare Metal Running Ultibo

06/13/19

**********DRAFT**********

Requirements:

2 Raspberry Pi

one running RaspBian

one Bare Metal Ultibo

Lazarus IDE (Ultibo Edition)

Free Pascal Compiler FPC

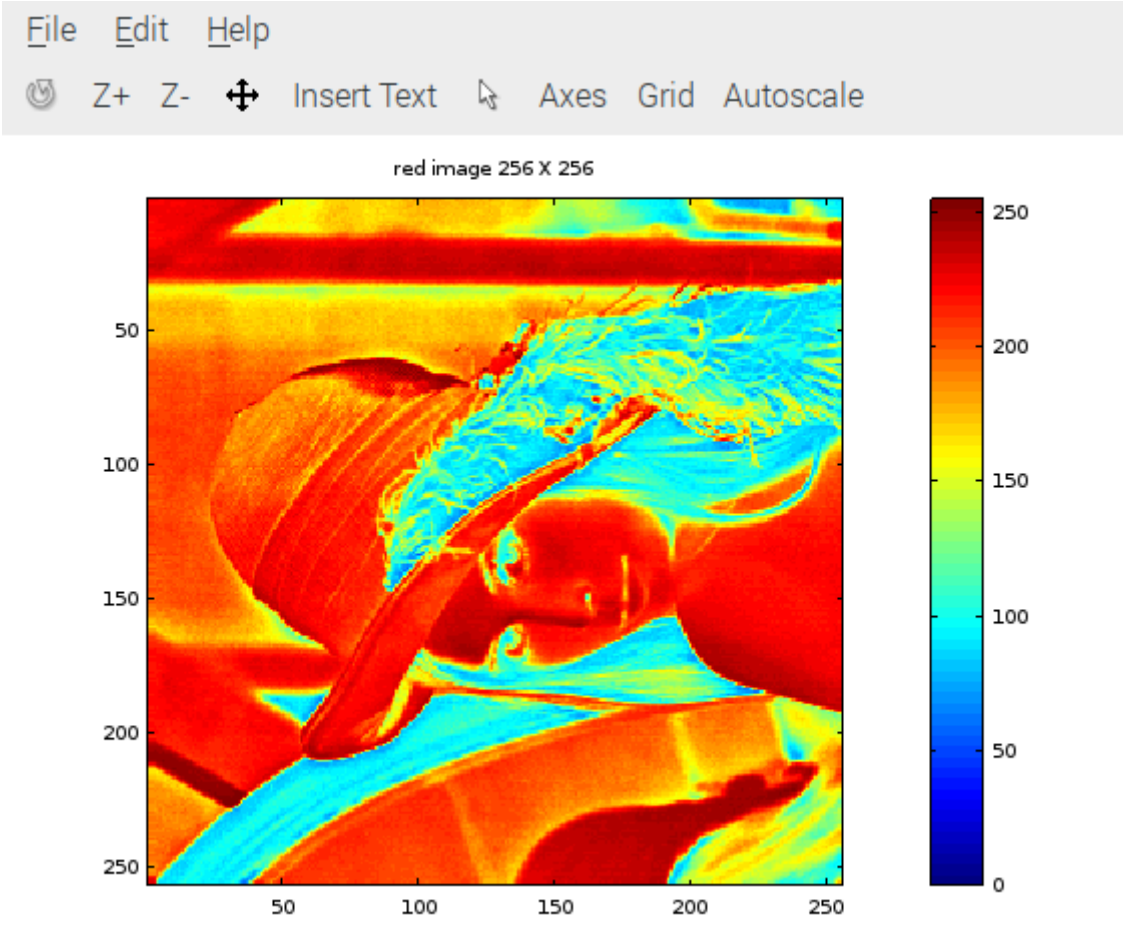
The SVD of image

```
clear
n = 10;
p = 1;


fid = fopen('red.bin','r'); im1 = fread(fid, [256,inf], 'int32');
if p == 1
  figure;
  imagesc(im1);
  colorbar;
  title "red image 256 X 256 "
end

[U,S,V] = svd(im1);
VT = V';
im2 = U*S*VT;

if p == 1
  figure;
  imagesc(im2);
  colorbar;
  title "red image 256 X 256 reconstructed from U*S*VT "
end
for i = 1:n
  for j = 1:n
    if i == j
      a(i) = S(i,j);
    end
   end
end
figure
stem(a)
```

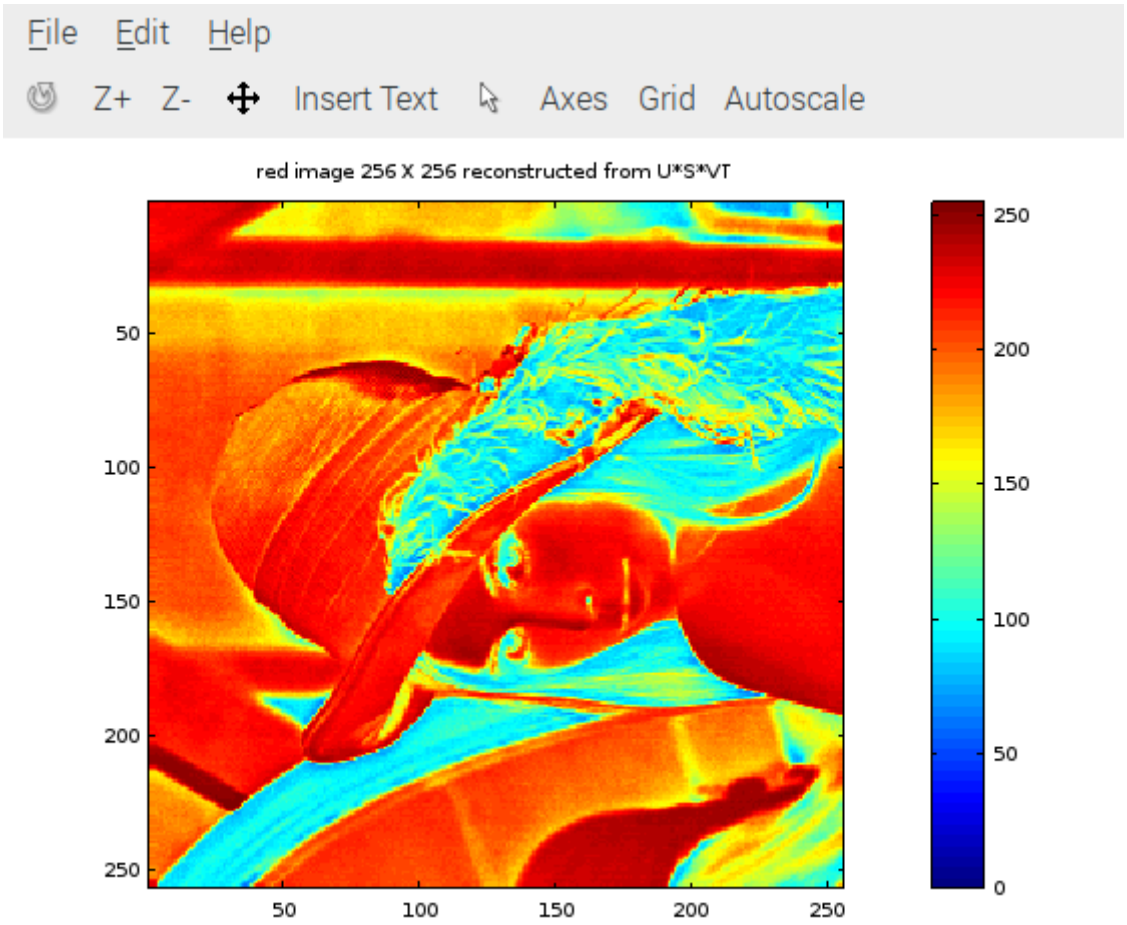
title "n S values from [U,S,V] = svd(im1)"

The RGB 256 x 256 image of lena.

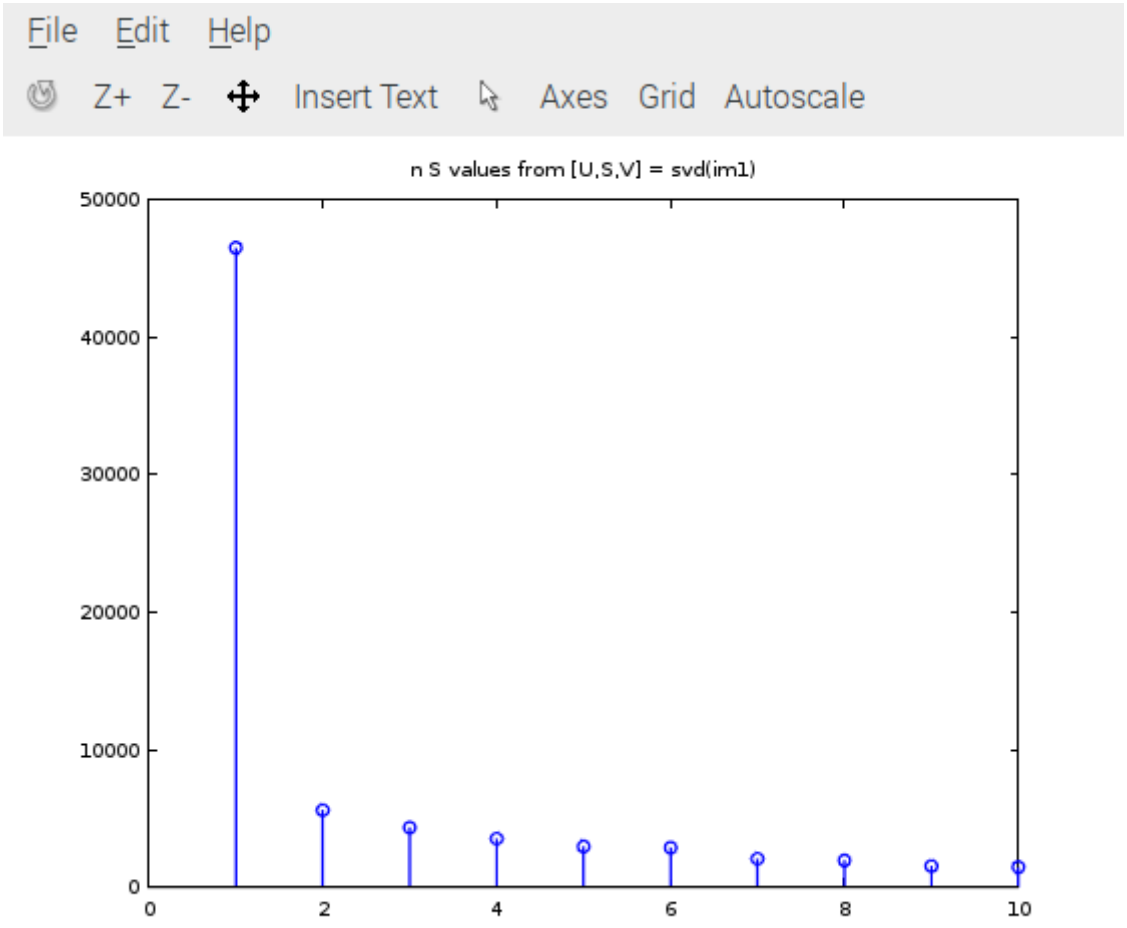
The Red sub band



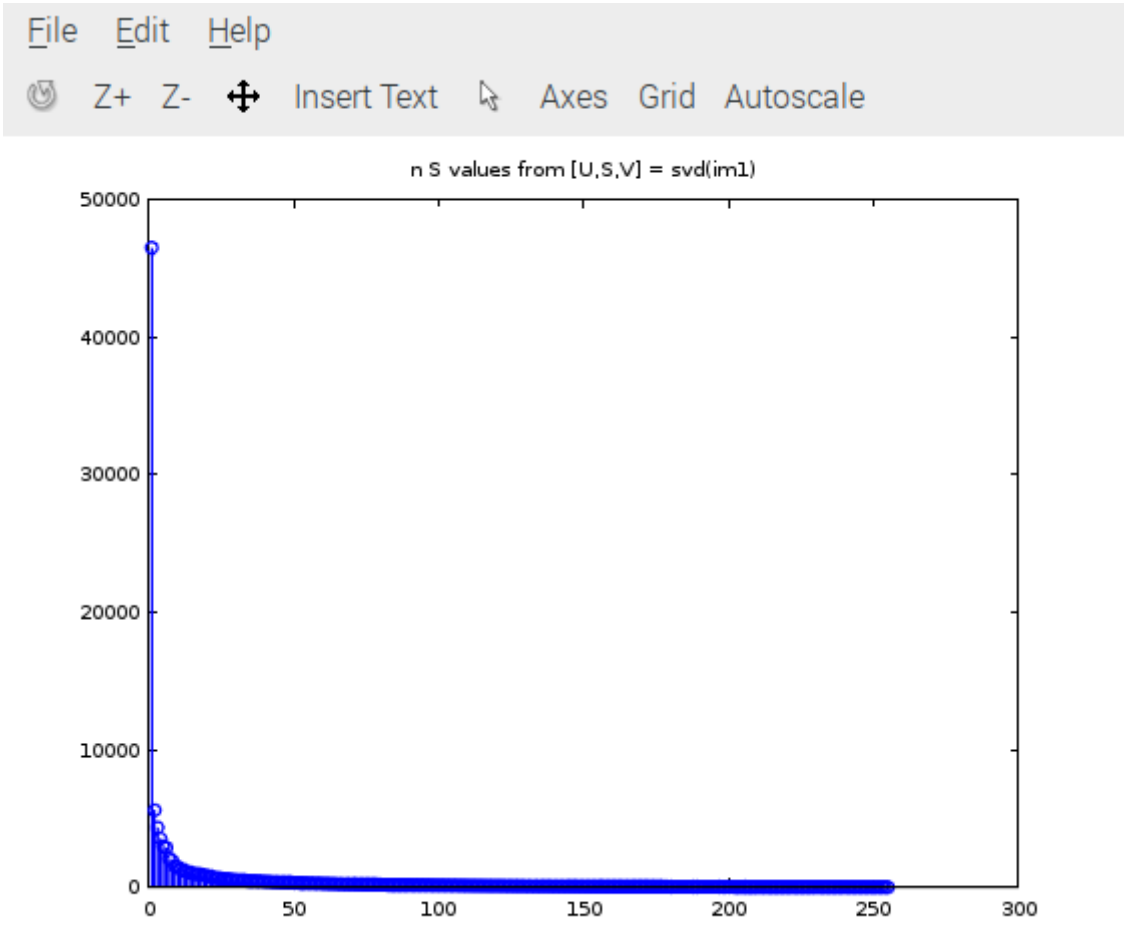
The reconstructed image



The first 10 Singluar Values extracted from S



All of singluar values



The script build_svd.sh compiles and links the C programs.

build_svd.sh

```
#!/bin/bash
gcc -g -O3 -c svd.c
gcc -g -O3 -c disp_mat.c
gcc -g -O3 -c trans_mat.c
gcc -g -O3 -c mul_mat.c

gcc -g -O3 test_svd.c svd.o disp_mat.o trans_mat.o mul_mat.o -lm -o test_svd
```

void main() is replaced with void test_svd() in ultibo_svd.c

The script buildlib.sh compiles and links the C programs into libsvd.a which is linked with svd_FS_Rpi2.lpr (uTFTP.pas which provides TFTP).

buildlib.sh

```
#!/bin/bash
#export PATH=/home/pi/ultibo/core/fpc/bin:$PATH
rm -f *.o
rm -f libsvd.a

arm-none-eabi-gcc  -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard
-c svd.c -o svd.o
```

```
arm-none-eabi-gcc  -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard
-c disp_mat.c -o disp_mat.o
arm-none-eabi-gcc  -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard
-c trans_mat.c -o trans_mat.o
arm-none-eabi-gcc  -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard
-c mul_mat.c -o mul_mat.o

arm-none-eabi-gcc  -O3 -mabi=aapcs -marm -march=armv7-a -mfpu=vfpv3-d16 -mfloat-abi=hard
-c ultibo_svd.c -o ultibo_svd.o
#gcc test_svd.c svd.o disp_mat.o -lm -o test_svd
arm-none-eabi-ar rcs libsvd.a *.o
arm-none-eabi-ar -t libsvd.a > libsvd_obj.txt
#fpc -vi -B -Tultibo -Parm -CpARMV7A -WpRPI2B @/home/pi/ultibo/core/fpc/bin/RPI2.CFG -O4
SVD_RPi2.lpr
```

test_svd.c

```
#include <stdio.h>
#include <stdlib.h>


void main() {
/*
 * Input to dsvd is as follows:
 *   a = mxn matrix to be decomposed, gets overwritten with u
 *   m = row dimension of a
 *   n = column dimension of a
 *   w = returns the vector of singular values of a
 *   v = returns the right orthogonal transformation matrix
 * dsvd(float **a, int m, int n, float *w, float **v)
*/

extern int dsvd(float **a, int m, int n, float *w, float **v);

extern int trans(float **a,float **b,int m,int n);
extern int disp(float **a,int m,int n);
extern int mul(float **a,float **b,float **c,int m,int n,int p,int q);

int m=9,n=8,i,j,p=9,q=9,result,len1,len2,len3;

float w[m],*pw;

/*
 *  a 9 x 8
 *  u 9 x 8
 *  v 9 x 8
 *  ds 9 x 8
 *  vt 8 x 9
 *  uds 9 x 8
 *  udsvt 9 X 9
*/
```

```
//Several of the arrays use 2 pointers to allocate memory.
//9 x 8 arrays
float *pv, **ppv;
float *puds, **ppuds;
float *pa,**ppa;
float *pds, **ppds;
//8 x 9 arrays
float *pvt, **ppvt;
//9 x 9 arrays
float *pudsvt, **ppudsvt;

len1 = sizeof(float *) * m + sizeof(float) * n * m;
len2 = sizeof(float *) * n + sizeof(float) * m * n;
len3 = sizeof(float *) * p + sizeof(float) * p * q;
printf("len = %d len2 = %d len3 = %d\n",len1, len2,len3);
ppv = (float **)malloc(len1);
ppuds = (float **)malloc(len1);
ppa = (float **)malloc(len1);
ppds = (float **)malloc(len1);
ppvt = (float **)malloc(len2);
ppudsvt = (float **)malloc(len3);

// pv, puds, pa, pds, pvt, and pudsvt are now pointing to the first elements of 2D arrays
pv = (float *)(ppv + m);
puds = (float *)(ppuds + m);
pa = (float *)(ppa + m);
pds = (float *)(ppds + m);
pvt = (float *)(ppvt + n);
pudsvt = (float *)(ppudsvt + p);
// for loop to point rows pointer to appropriate location in 2D array
for(i = 0; i < m; i++) {
        ppa[i] = (pa + n * i);
        ppuds[i] = (puds + n * i);
        ppv[i] = (pv + n * i);
        ppds[i] = (pds + n * i);

}

for(i = 0; i < m; i++) ppvt[i] = (pvt + n * i);
for(i = 0; i < m; i++) ppudsvt[i] = (pudsvt + q * i);




pw=(float *)&w;



/*
printf("pa 0x%x ppa 0x%x  \n",pa,ppa);
```

```
printf("pv = 0x%x ppv = 0x%x \n",pv,ppv);
printf("pvt = 0x%x ppvt = 0x%x \n",pvt,ppvt);
printf("pds = 0x%x ppds = 0x%x \n",pds,ppds);
printf("puds = 0x%x ppuds = 0x%x \n",puds,ppuds);
printf("pudsvt = 0x%x ppudsvt = 0x%x \n",pudsvt,ppudsvt);
*/

/*
 * a = { {1.0,2.0,1.0.22.0,11.0},{3.0,4.0,2.0,41.0,21.0},{5.0,6.0,3.0,63.0,33.0},
{7.0,8.0,4.0,282.0,242.0},{7.0,8.0,4.0,182.0,142.0}};
*/


ppa[0][0]=1.0;
ppa[0][1]=2.0;
ppa[0][2]=1.0;
ppa[0][3]=22.0;
ppa[0][4]=11.0;
ppa[0][5]=22.0;
ppa[0][6]=11.0;
ppa[0][7]=11.0;

ppa[1][0]=3.0;
ppa[1][1]=4.0;
ppa[1][2]=2.0;
ppa[1][3]=41.0;
ppa[1][4]=21.0;
ppa[1][5]=41.0;
ppa[1][6]=21.0;
ppa[1][7]=11.0;

ppa[2][0]=5.0;
ppa[2][1]=6.0;
ppa[2][2]=3.0;
ppa[2][3]=63.0;
ppa[2][4]=33.0;
ppa[2][5]=63.0;
ppa[2][6]=33.0;
ppa[2][7]=11.0;

ppa[3][0]=7.0;
ppa[3][1]=8.0;
ppa[3][2]=4.0;
ppa[3][3]=82.0;
ppa[3][4]=42.0;
ppa[3][5]=82.0;
ppa[3][6]=42.0;
ppa[3][7]=11.0;

ppa[4][0]=7.0;
ppa[4][1]=8.0;
ppa[4][2]=4.0;
```

```
ppa[4][3]=282.0;
ppa[4][4]=242.0;
ppa[4][5]=282.0;
ppa[4][6]=242.0;
ppa[4][7]=11.0;

ppa[5][0]=7.0;
ppa[5][1]=8.0;
ppa[5][2]=4.0;
ppa[5][3]=182.0;
ppa[5][4]=142.0;
ppa[5][5]=82.0;
ppa[5][6]=42.0;
ppa[5][7]=11.0;

ppa[6][0]=1.0;
ppa[6][1]=2.0;
ppa[6][2]=1.0;
ppa[6][3]=22.0;
ppa[6][4]=11.0;
ppa[6][5]=82.0;
ppa[6][6]=42.0;
ppa[6][7]=11.0;

ppa[7][0]=3.0;
ppa[7][1]=4.0;
ppa[7][2]=2.0;
ppa[7][3]=41.0;
ppa[7][4]=21.0;
ppa[7][5]=82.0;
ppa[7][6]=42.0;
ppa[7][7]=11.0;

ppa[8][0]=5.0;
ppa[8][1]=6.0;
ppa[8][2]=3.0;
ppa[8][3]=63.0;
ppa[8][4]=33.0;
ppa[8][5]=82.0;
ppa[8][6]=42.0;
ppa[8][7]=11.0;

printf("a row = %d col = %d \n",m,n);
result = disp(ppa,m,n);
/*
for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        {
                printf("0x%x 0x%x 0x%x %5.2f %5.2f\n",ppa,pa,&a[i][j],a[i][j],*pa);
                pa++;
        }
```

```
}
ppa=&pa;
pa=&a;
*/


//printf("pa 0x%x ppa 0x%x  \n",pa,ppa);
result = dsvd(ppa,m,n,pw,ppv);
printf("U row = %d col = %d \n",m,n);
result = disp(ppa,m,n);
printf("Singular Values\n");
for(i=0;i<m;i++) {
        for(j=0;j<n;j++) {
                ppds[i][j] = 0;
        }
}

j=0;
for(i=0;i<m;i++) {
        ppds[i][j] = w[i];
        j++;
}
printf("S row = %d col = %d \n",m,n);
result = disp(ppds,m,n);
//for(i=0;i<m;i++) printf("%5.2f \n",w[i]);

printf("V row = %d col = %d \n",m,n);
result = disp(ppv,m,n);

printf("V' row = %d col = %d \n",n,m);

result = trans(ppv,ppvt,m,n);
result = disp(ppvt,n,m);
printf("Call mul u * s  \n");
result = mul(ppa,ppds,ppuds,m,n,p,q);
printf("UDS row = %d col = %d \n",m,n);
result = disp(ppuds,m,n);
printf("Call mul u * ds * vt \n");

result = mul(ppuds,ppvt,ppudsvt,m,n,n,m);
printf("USDVT row = %d col = %d \n",p,q);


result = disp(ppudsvt,p,q);
/*
free(ppv);
free(ppuds);
free(ppa);
free(ppds);
free(ppvt);
*/
```

```
}
```

trans_mat.c

```
#include <stdio.h>
#include <stdlib.h>

int trans(float **a,float **b,int m,int n) {

        int i,j,result;


for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        {
                b[j][i] = a[i][j];
        }

}



        return(1);
}
```

disp_mat.c

```
#include <stdio.h>
#include <stdlib.h>

int disp(float **a,int m,int n) {
        int i,j,result;



for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        {
                printf("%5.8f ",a[i][j]);

        }
        printf("\n");
}


        return(1);
}
```

mul_mat.c

```
#include <stdio.h>
#include <stdlib.h>

int mul(float **a,float **b,float **c,int m,int n,int p,int q) {
        int i,j,k,result;
```

```
          float sum=0.0;
          for (i = 0; i < m; i++) {
     for (j = 0; j < q; j++) {
       for (k = 0; k < p; k++) {
                    //printf("i %d j %d k %d a %5.3f b %5.3f \n",i,j,k,a[i][k],b[i][j]);
         sum = sum + a[i][k]*b[k][j];
         //printf("sum %5.3f \n",sum);
       }

       c[i][j] = sum;
       //printf(c %5.3f \n",c[i][j]);
       sum = 0.0;
     }
   }
          return(1);
}
```

./test_svd > xx.txt

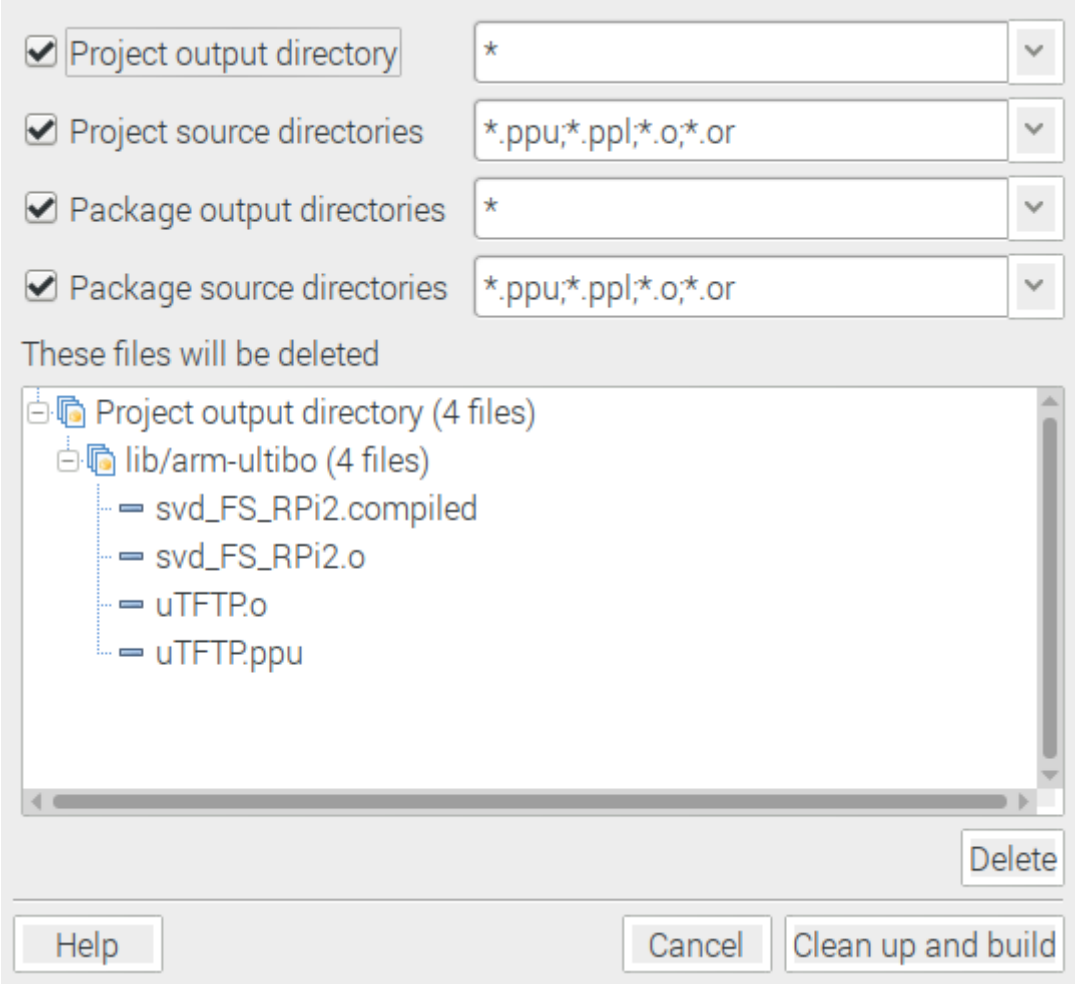
```
len = 324 len2 = 320 len3 = 360
a row = 9 col = 8
1.00000000 2.00000000 1.00000000 22.00000000 11.00000000 22.00000000 11.00000000
11.00000000
3.00000000 4.00000000 2.00000000 41.00000000 21.00000000 41.00000000 21.00000000
11.00000000
5.00000000 6.00000000 3.00000000 63.00000000 33.00000000 63.00000000 33.00000000
11.00000000
7.00000000 8.00000000 4.00000000 82.00000000 42.00000000 82.00000000 42.00000000
11.00000000
7.00000000 8.00000000 4.00000000 282.00000000 242.00000000 282.00000000 242.00000000
11.00000000
7.00000000 8.00000000 4.00000000 182.00000000 142.00000000 82.00000000 42.00000000
11.00000000
1.00000000 2.00000000 1.00000000 22.00000000 11.00000000 82.00000000 42.00000000
11.00000000
3.00000000 4.00000000 2.00000000 41.00000000 21.00000000 82.00000000 42.00000000
11.00000000
5.00000000 6.00000000 3.00000000 63.00000000 33.00000000 82.00000000 42.00000000
11.00000000
U row = 9 col = 8
-0.05541832 -0.01790087 -0.16102470 0.11920507 0.79220021 -0.37744969 0.07404913
0.32213101
-0.10343107 -0.02939794 -0.23943540 0.22800885 0.43026865 0.52036184 0.30906394
-0.49498701
-0.15958314 -0.04268976 -0.32130972 0.33856225 0.03623828 0.06008183 -0.83287090
0.05579400
-0.20621006 -0.05439396 -0.41707826 0.45471457 -0.36364248 -0.40626699 0.22812617
-0.33679608
-0.84121537 -0.16022219 0.50921464 0.08147858 0.02639885 -0.00149095 0.00645019
0.00046739
-0.37113467 0.79577464 -0.30287606 -0.37006834 -0.01107633 -0.00118669 -0.00604446
-0.01136582
```

-0.12792568 -0.44762436 -0.25567460 -0.61894733 0.07765095 -0.37730163 -0.07404581 -0.32212564
-0.15285745 -0.32233346 -0.30603474 -0.28348282 -0.06192626 0.52124029 -0.08053842 0.14728157
-0.18200766 -0.17561121 -0.35984349 0.07324401 -0.20749584 0.06356924 0.37581018 0.63960040
Singular Values
S row = 9 col = 8
623.59698486 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 113.23142242 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 59.31568146 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 12.16330910 0.00000000 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 8.49159718 0.00000000 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.20821390 0.00000000 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000150 0.00000000
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000014
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
V row = 9 col = 8
-0.02018946 0.01285760 -0.11690084 0.21011896 -0.15540643 0.60369533 -0.50010186 0.55037284
-0.02371713 0.00884463 -0.14815842 0.21198617 -0.07077976 0.61116689 0.73106879 -0.13954927
-0.01185851 0.00442234 -0.07407922 0.10599311 -0.03538979 0.30558631 -0.46191961 -0.82165217
-0.57367194 0.50138474 -0.34003848 0.45926267 -0.11076186 -0.28407273 -0.00000138 0.00000050
-0.45479280 0.46127322 0.40777835 -0.55958360 0.14070038 0.28491610 0.00000133 -0.00000049
-0.54206085 -0.58477330 -0.41484702 -0.39257458 -0.19492903 0.00045217 -0.00000010 0.00000004
-0.40941137 -0.43780005 0.62182570 0.47019655 0.18155093 -0.00087418 0.00000009 -0.00000008
-0.03880329 -0.04414316 -0.34383288 0.02054236 0.93089414 0.08227108 -0.04546362 0.05003385
0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000 0.00000000
V' row = 8 col = 9
-0.02018946 -0.02371713 -0.01185851 -0.57367194 -0.45479280 -0.54206085 -0.40941137 -0.03880329 0.00000000
0.00000000 0.00884463 0.00442234 0.50138474 0.46127322 -0.58477330 -0.43780005 -0.04414316 0.00000000
0.00000000 -0.14815842 -0.07407922 -0.34003848 0.40777835 -0.41484702 0.62182570 -0.34383288 0.00000000
0.00000000 0.21198617 0.10599311 0.45926267 -0.55958360 -0.39257458 0.47019655 0.02054236 0.00000000
0.00000000 -0.07077976 -0.03538979 -0.11076186 0.14070038 -0.19492903 0.18155093 0.93089414 0.00000000
0.00000000 0.61116689 0.30558631 -0.28407273 0.28491610 0.00045217 -0.00087418 0.08227108 0.00000000

0.00000000 0.73106879 -0.46191961 -0.00000138 0.00000133 -0.00000010 0.00000009
-0.04546362 0.00000000
0.00000000 -0.13954927 -0.82165217 0.00000050 -0.00000049 0.00000004 -0.00000008
0.05003385 0.00000000
Call mul u * s
UDS row = 9 col = 8
-34.55869675 -2.02694130 -9.55129051 1.44992816 6.72704506 -0.07859027 0.00000011
0.00000005
-64.49930573 -3.32877016 -14.20227432 2.77334213 3.65366793 0.10834657 0.00000046
-0.00000007
-99.51556396 -4.83382273 -19.05870438 4.11803722 0.30772084 0.01250987 -0.00000125
0.00000001
-128.59197998 -6.15910578 -24.73928070 5.53083372 -3.08790541 -0.08459044 0.00000034
-0.00000005
-524.57934570 -18.14218712 30.20441246 0.99104917 0.22416836 -0.00031044 0.00000001
0.00000000
-231.43846130 90.10669708 -17.96529961 -4.50125551 -0.09405573 -0.00024709 -0.00000001
-0.00000000
-79.77407074 -50.68514252 -15.16551304 -7.52844763 0.65938056 -0.07855944 -0.00000011
-0.00000005
-95.32144928 -36.49827576 -18.15265846 -3.44808912 -0.52585292 0.10852947 -0.00000012
0.00000002
-113.49942780 -19.88470650 -21.34436226 0.89088959 -1.76197112 0.01323600 0.00000056
0.00000009
Call mul u * ds * vt
USDVT row = 9 col = 9
0.69772142 2.00000381 1.00000060 22.00000572 11.00001431 22.00000381 11.00000095
11.00000095 0.00000000
1.30220616 4.00000572 1.99999976 40.99999619 20.99999428 40.99999237 20.99999428
11.00000000 0.00000000
2.00916553 6.00000811 3.00000024 62.99999619 32.99999237 63.00001144 33.00000000
10.99999905 0.00000000
2.59620261 8.00001335 3.99999905 82.00000000 42.00000000 81.99999237 41.99999237
10.99999714 0.00000000
10.59097385 8.00004959 3.99999619 281.99996948 241.99998474 282.00000000 241.99996948
10.99999905 0.00000000
4.67261744 8.00002956 4.00000429 182.00000000 142.00001526 82.00000000 41.99999237
10.99999905 0.00000000
1.61059546 2.00000882 0.99999887 21.99999428 10.99999428 81.99999237 42.00000000
10.99999905 0.00000000
1.92448866 4.00000858 1.99999881 40.99999619 20.99999809 82.00000000 41.99999619
11.00000000 0.00000000
2.29149222 6.00001287 2.99999928 63.00000381 33.00000381 82.00001526 42.00000000
11.00000095 0.00000000

Main Lazarus Window

From the Main Lazarus Window Run select
Clean up and Build



Depress the button Clean up and build which will create the kernel7.img
Source Window

Message Window When the bar turns green the kernel7.img is ready.

```
Compile Project, OS: ultibo, Target: svd_FS_RPi2: Success, Hints: 4
  svd_FS_RPi2.lpr(37,2) Note: Local variable "MyPLoggingDevice" not used
  svd_FS_RPi2.lpr(39,2) Note: Local variable "Handle1" not used
  svd_FS_RPi2.lpr(40,2) Note: Local variable "Handle3" not used
  svd_FS_RPi2.lpr(42,2) Note: Local variable "Window" not used
```

The kernel7.img can be transferred to the Bare Metal Ultibo system with the ***"tftp 192.168.1.202 < cmdstftp" Following the transfer the Bare Metal Ultibo system in 15.85 perform the SVD***