

# Physics 171.416, Spring 2007

## Problem Sets 1 and 2

### Ranges, precisions, and rates

Using the range and precisions for a 32-bit machine are as follows

	Float	Double	Long Double
Precision	$10^{-7}$	$10^{-15}$	$10^{-34}$
Upper Range	$10^{38}$	$10^{307}$	$10^{4932}$
Lower Range	$10^{-45}$	$10^{-324}$	$10^{-4965}$

The difference between the upper and lower ranges is due to the representation of floating point numbers in a 32-bit word. The IEEE standard for single precision floating points has the following standard

(Refer: <http://www.psc.edu/general/software/packages/ieee/ieee.html> or any other standard text on floating point operations)

```
S EEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFF
  0 1      8 9                      31
```

so that the value represented by a word is determined as

$V = (-1)^S * 2^{(E-127)} * (1.F)$  when  $0 < E < 255$

And  $V = (-1)^S * 2^{(E-126)} * (0.F)$  when  $E=0$  and  $F$  is nonzero

This gives 38 for the upper range and -45 for the lower range in single precision.

And similar argument goes for a 64-bit double precision number

In comparing the rates of various operations, you are looking at the user time from the time function. User times depend on other processes running on the compiler and thus may vary from run to run. Thus you want to get a trend, or you can do several runs and get average values. In general, time taken for double precision numbers is larger than for single precision. Addition takes less time than multiplication/division. However, the compiler for eta does multiplications fastest. It seems to be optimized for multiplication.

Every compiler has options for run time optimization. Using the "run time optimization" (with `-O` option when using gcc or any other runtime command ) cuts down the time by almost half to a third of the time taken otherwise for all the operations, specially multiplication.

1. (a) **Code:**

Your code should typically print out a some quantities that help check the accuracy of any results you obtain. Two such quantities are the determinant of the matrix  $A$  and the deviation of  $A\mathbf{x}$  from  $\mathbf{b}$ . This deviation can be quantified, for example, in the form of  $\chi^2$ :

$$\chi^2 = \sum_{i=1}^N ((A\mathbf{x})_i - b_i)^2$$

The matrix elements  $A_{ij}$  are

$$A_{ij} = i + j + 10^n \sin e^{3(i+j)/N}$$

Compare this to the matrix with elements

$$B_{ij} = i + j,$$

which becomes singular for  $N > 2$ . This means that whenever the term involving both  $n$  and  $N$  in the expression for  $A_{ij}$  becomes small, it becomes difficult to obtain an accurate solution for  $\mathbf{x}$ .

The table on the following page shows a list of output parameters obtained from the code including  $\sqrt{(\chi^2)}$  values obtained on execution of different subroutines:  $\chi_{LUD}$  is obtained from LU Decomposition,  $\chi_{IMP}$  is obtained from linear improvement of the LU Decomposition solution,  $\chi_{SVD}$  is obtained from Singular Value Decomposition and finally  $\chi_{SVDZ}$  is the result of Singular Value Decomposition with zeroing - any element of the  $W$ -matrix smaller than  $CN = 10^{-6}$  times the maximum element was taken to be zero.

A sample printout for  $N = 4$  and  $n = -3$  is also shown below:

```

-----
N=4 M=-3
A(N,M)
1.999027 2.999937 4.000945 4.999006
2.999937 4.000945 4.999006 6.000886
4.000945 4.999006 6.000886 7.000878
4.999006 6.000886 7.000878 8.000965
Det(A)=8.008620e-05
After SVD1:  wmin=1.875343e-03  wmax=2.095567e+01  wmin/wmax=8.949096e-05
chilud=1.364788e-04  chilud2=1.364788e-04  chisvd=1.831055e-04  chisvd2=1.831055e-04
xlud axlud xlud2 axlud2 xsvd axsvd xsvd2 axsvd2 w1 w2
-9.636284e+01 1.000061e+00 -9.636284e+01 1.000061e+00 -9.633376e+01 1.000000e+00 -9.633376e+01 1.000000e+00 2.095567e+01 2.095567e+01
3.600128e+01 0.000000e+00 3.600128e+01 0.000000e+00 3.596771e+01 -6.103516e-05 3.596771e+01 -6.103516e-05 9.541042e-01 9.541042e-01
2.153391e+02 1.220703e-04 2.153391e+02 1.220703e-04 2.153187e+02 -1.220703e-04 2.153187e+02 -1.220703e-04 2.136050e-03 2.136050e-03
-1.552168e+02 0.000000e+00 -1.552168e+02 0.000000e+00 -1.551919e+02 -1.220703e-04 -1.551919e+02 -1.220703e-04 1.875343e-03 1.875343e-03
-----

```

$N$	$n$	$\text{Det}(A)$	$W_{min}/W_{max}$	$\sqrt{\chi_{LUD}^2}$	$\sqrt{\chi_{IMP}^2}$	$\sqrt{\chi_{SVD}^2}$	$\sqrt{\chi_{SVDZ}^2}$
4	3	2.346522e+11	1.303036e-02	6.743496e-07	8.481136e-05	3.198721e-06	3.198721e-06
4	0	4.214004e+01	1.893296e-02	5.331201e-07	1.168008e-06	2.384186e-07	2.384186e-07
4	-3	8.008620e-05	8.949096e-05	1.364788e-04	1.364788e-04	1.831055e-04	1.831055e-04
4	-6	8.217430e-11	1.003048e-07	8.838835e-02	1.397542e-01	5.796012e-01	5.477222e-01
4	-9	1.918465e-13	2.857179e-10	4.582576e+00	6.082763e+00	1.688194e+01	5.477224e-01
8	3	9.238336e+25	1.041944e-01	9.884312e-08	6.837790e-05	7.682412e-07	7.682412e-07
8	0	1.868856e+04	1.503530e-02	4.298152e-07	2.480585e-06	3.392756e-06	3.392756e-06
8	-3	3.573042e-14	1.866399e-05	1.398491e-04	2.221713e-04	1.090123e-03	1.090123e-03
8	-6	3.034619e-32	1.917102e-08	3.792082e-01	4.074502e-01	1.253121e+00	7.637625e-01
8	-9	-2.892816e-39	2.817424e-09	1.272517e+02	1.272517e+02	2.281036e+00	7.637627e-01
12	3	3.742269e+37	1.338243e-03	2.827271e-05	3.891459e-04	9.266375e-05	9.266375e-05
12	0	4.303741e+05	6.443983e-04	1.283038e-05	1.194950e-05	2.486878e-05	2.486878e-05
12	-3	4.256572e-25	7.190277e-07	1.239119e-02	1.239119e-02	3.754372e-02	5.850741e-01
12	-6	2.723785e-55	5.230687e-09	7.582875e+00	3.741657e+00	3.629308e+00	8.397192e-01
12	-9	-7.009596e-76	5.444041e-10	1.759211e+13	3.406698e+13	5.024938e+00	8.397191e-01
16	3	1.490841e+54	6.966060e-02	7.673736e-07	5.484762e-04	1.280807e-06	1.280807e-06
16	0	1.018748e+09	1.340901e-03	3.090258e-06	3.162980e-06	6.877043e-06	6.877043e-06
16	-3	1.190522e-33	1.338861e-06	3.009968e-03	2.290242e-03	9.753410e-03	9.753410e-03
16	-6	7.047999e-76	5.933335e-09	1.924351e+00	2.439903e+00	4.401732e+00	8.786688e-01
16	-9	-5.513885e-85	2.480770e-10	4.078296e+01	2.559785e+01	7.178527e+00	8.786688e-01
20	3	1.069619e+68	2.967742e-02	1.457425e-06	2.808190e-04	2.849041e-06	2.849041e-06
20	0	5.662532e+11	6.079103e-04	5.197049e-06	6.050918e-06	3.767271e-05	3.767271e-05
20	-3	6.386405e-43	6.481382e-07	7.338637e-03	9.663429e-03	4.051505e-02	3.059420e-01
20	-6	4.063815e-96	2.228533e-09	4.330127e+00	4.062019e+00	1.112781e+00	9.023778e-01
20	-9	-3.050931e-109	1.670169e-10	3.387994e+02	6.942744e+02	7.332505e+00	9.023778e-01

A plot of  $\log|\text{Det } A|$  and a plot of  $W_{min}/W_{max}$  for different values of  $N$  and  $n$  is included.

(b) **Questions:**

- i. Is the LU method satisfactory for all values of  $n$  and  $N$ ?

The dependence on  $n$  is a lot stronger than the dependence on  $N$ , as can be seen from the expression for  $A_{ij}$ . This means that the solution obtained from either method should be more affected by changes in  $n$  than in  $N$ . For negative values of  $n$ , the elements start to approach  $i + j$  and the matrix approaches singularity. The LU method is thus unsatisfactory for negative values of  $n$ , especially if  $N$  is large. See Figure 1 and the table above.

- ii. What happens to the character and accuracy of the solution as  $n$  decreases? Why?

The solution becomes highly inaccurate as  $n$  decreases. This is because as  $n$  decreases, the elements  $A_{ij}$  start to approach  $B_{ij} = i + j$  which is a singular matrix for  $N > 2$ .

iii. What is the effect of increasing  $N$ ?

If we keep  $n$  constant, we see from the table above and Figure 1 that  $\text{Det}(A)$  approaches zero rapidly as  $N$  increases, so for higher  $N$ ,  $A$  becomes singular and we expect the solutions from LUD to become less accurate - and this can be verified from the table above. Increasing  $N$  affects the roundoff error thus bringing down the accuracy of the solution.

iv. Does linear improvement affect the result?

On applying linear improvement once, it might look like the result has a lower or

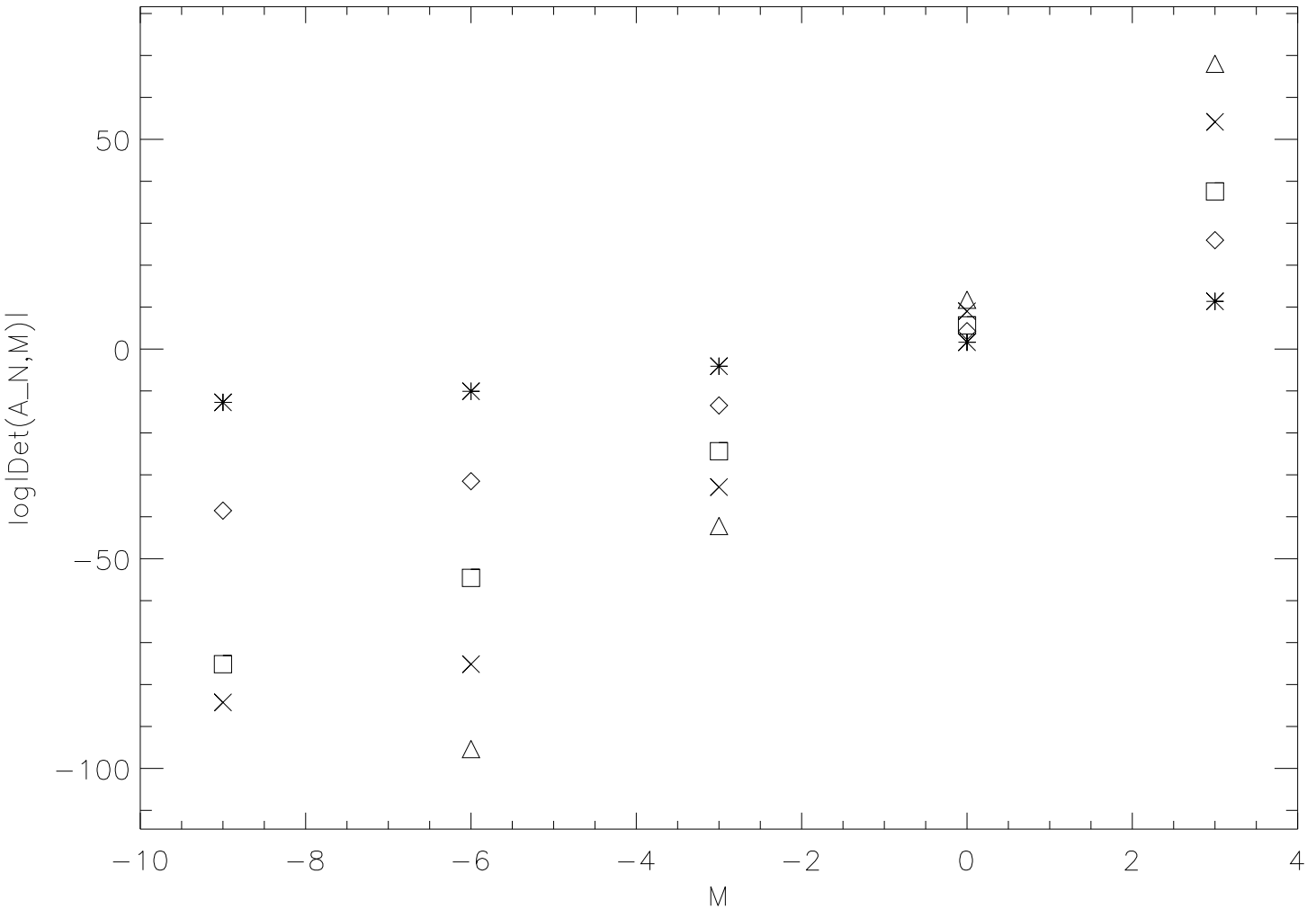


Figure 1: The logarithm of the absolute value of the determinant of matrix  $A$  versus  $n$  ( $M$  in this plot) for  $N = 4$  (stars),  $N = 8$  (diamonds),  $N = 12$  (squares),  $N = 16$  (X's) and  $N = 20$  (triangles)

comparable  $\chi^2$  (see table above), but in reality subsequent calls to the improvement routine only result in an oscillating  $\chi^2$ , so the effect of linear improvement can not easily be determined for this problem. The oscillating  $\chi_{IMP}^2$  is demonstrated in the table below for  $N = 8$  and  $n = -6$  with 1 to 10 successive calls to `mprove`.

$\sqrt{\chi_{LUD}^2}$	$\sqrt{\chi_{IMP}^2}$	$\sqrt{\chi_{SVD}^2}$	$\sqrt{\chi_{SVDZ}^2}$
3.792082e-01	4.074502e-01	1.253121e+00	7.637625e-01
3.792082e-01	5.728220e-01	1.253121e+00	7.637625e-01
3.792082e-01	5.359398e-01	1.253121e+00	7.637625e-01
3.792082e-01	4.881406e-01	1.253121e+00	7.637625e-01
3.792082e-01	6.555055e-01	1.253121e+00	7.637625e-01
3.792082e-01	3.814549e-01	1.253121e+00	7.637625e-01
3.792082e-01	5.674694e-01	1.253121e+00	7.637625e-01
3.792082e-01	6.434769e-01	1.253121e+00	7.637625e-01
3.792082e-01	5.557319e-01	1.253121e+00	7.637625e-01
3.792082e-01	7.654655e-01	1.253121e+00	7.637625e-01

- v. (Singular Value Decomposition) What does the list of diagonal elements in the “ $W$ ” matrix tell you about what’s happening?

As the matrix approaches singularity, the ratio  $CN = W_{min}/W_{max}$ , the ratio of the smallest diagonal element in the  $W$  matrix to its largest element, decreases. For float arithmetic, when this value dips below  $\sim 10^{-6}$ , the matrix is considered to be singular. In such a case, singular value decomposition will not work unless all matrix elements below this value are zeroed out. Figure 2 shows a plot of the condition number  $CN$  versus  $n$  for different values of  $N$ .

- vi. Is the solution you obtain using SVD more meaningful than the one yielded by the LU decomposition method?

The solution using SVD, while not necessarily more accurate (as is seen from  $\chi^2$ ), is definitely more meaningful because the  $W$  matrix elements are a measure of the singularity of  $A$ .

- vii. Try zeroing small  $w_j$ ’s prior to the inversion procedure

Once this is done, it is seen that as  $n$  increases, the SVD solution becomes more accurate than that obtained from the LUD solution.

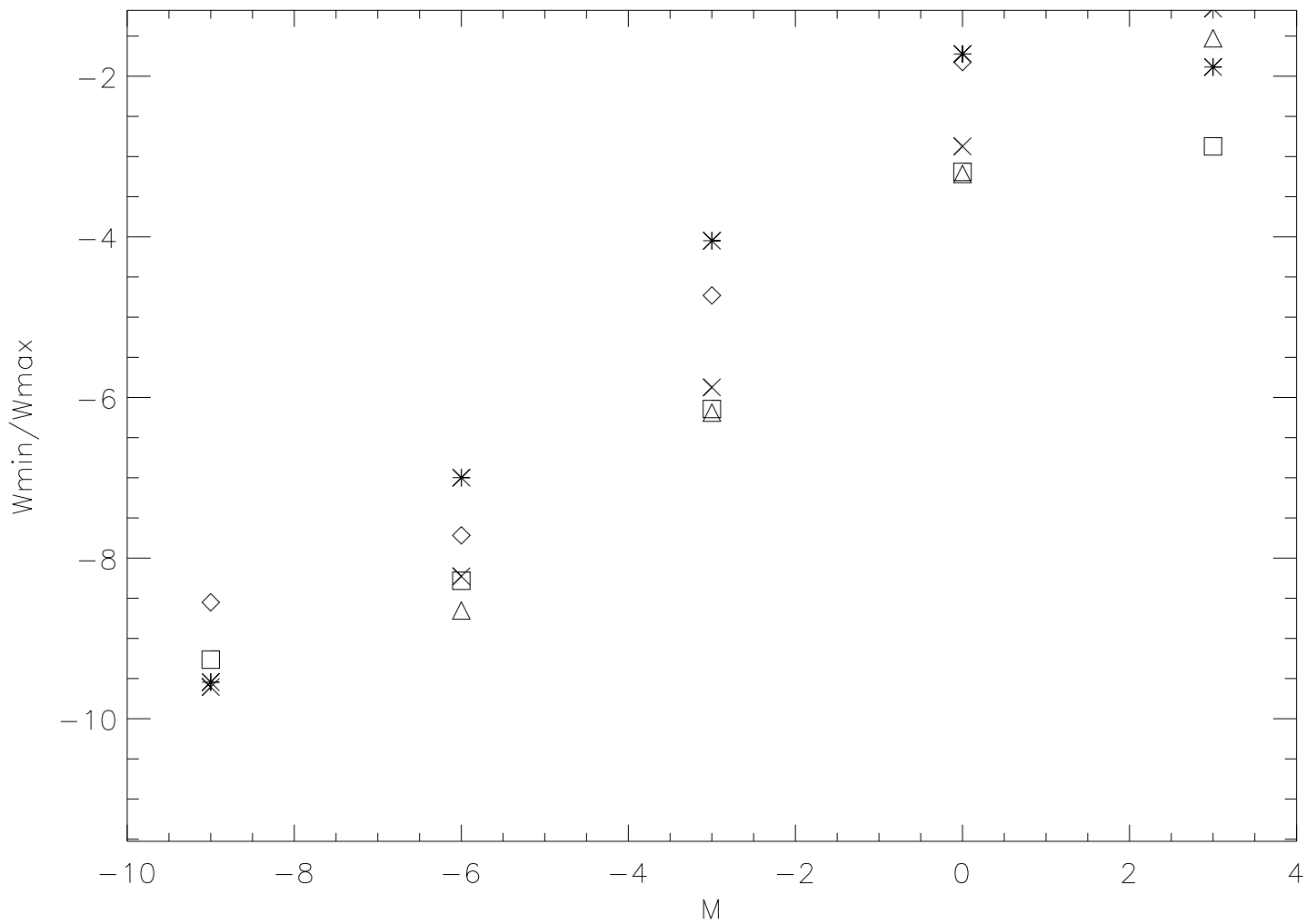


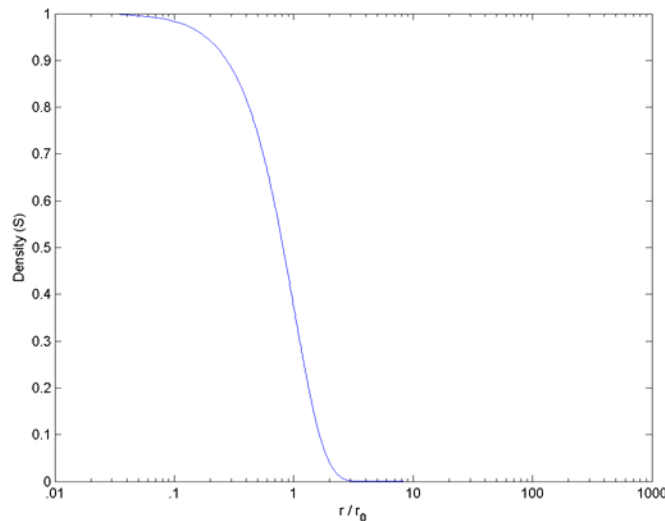
Figure 2: The logarithm of the condition number  $CN = W_{min}/W_{max}$  versus  $n$  ( $M$  in this plot) for  $N = 4$  (stars),  $N = 8$  (diamonds),  $N = 12$  (squares),  $N = 16$  (X's) and  $N = 20$  (triangles). When this ratio dips below  $10^{-6}$ , the corresponding elements have to be zeroed in order for the SVD routine to work.

## **Problem 2-2 Solution**

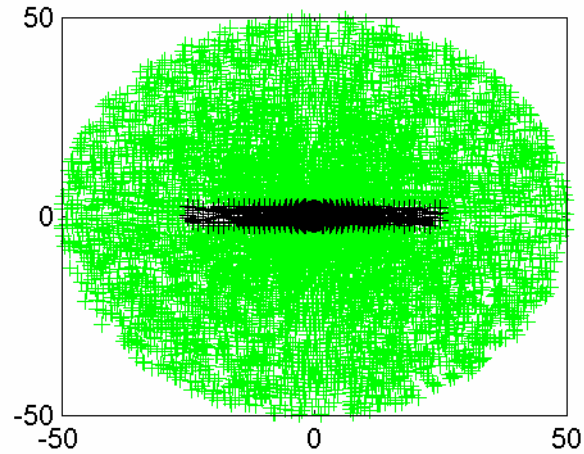
This problem can be solved using a Monte Carlo technique – by randomly generating photons and determining if they pass through the slit or not. The photons are selected so that they fit the distribution:

$$S \propto \exp\left(-\frac{r}{r_0}\right)^{1.7}$$

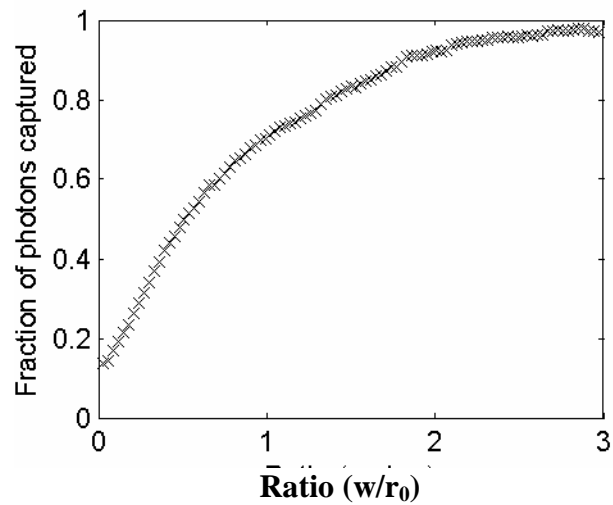
A picture that illustrates the density of the photons that are generated is shown below. To generate the proper density, points can be randomly drawn in the image and selected if they are within the region to the left of the line.



A random angle is also assigned to the photons so that they form the proper 2D distribution. We then determine if the 2D distribution lies within the boundary of the slit. An image which illustrates that is given in the next figure. Photons that fall within the slit are black, those that fall outside are green. Units on the x and y axes correspond to Cartesian coordinates of the generated photons ( $r_0$  was set to 100).



The ratio of the number of photons inside the slit to those outside the slit is then the computed quantity. By repeating this procedure for different ratios of  $w/r_0$  we get the desired output: Fraction captured vs.  $w/r_0$ . It is plotted below:



The fraction capture tends towards 1 as the slit becomes larger relative to  $r_0$ . Code used to generate the data follows:

```

pi = 3.14159;
r0 = 100;
S0 = 1;

for j=1:10;
numphotons = 10000;
n = 10*r0;
ratio(j) = (.005)*(j);
w = ratio(j)*r0;
slitwidth = w*5;
slitheight = w/2;

for i=1:round(w*100)
    r(i) = (i-1);
    S(i) = S0*exp(-(r(i)/r0)^1.7);
end

insidenum = 0;
pnum = 0;
px = 0;
py = 0;
insidex = 0;
insidey = 0;
for i=1:numphotons
    y(i) = rand;
    x(i) = round(rand*w*10);
    angle(i) = rand*2*pi;
    if x(i) == 0;
        x(i) = 1;
    end
    if S(x(i)) > y(i)
        pnum = pnum + 1;
        theta(pnum) = angle(i);
        px(pnum) = cos(theta(pnum))*x(i);
        py(pnum) = sin(theta(pnum))*x(i);

        if abs(px(pnum)) < slitwidth
            if abs(py(pnum)) < slitheight
                insidenum = insidenum + 1;
                insidex(insidenum) = px(pnum);
                insidey(insidenum) = py(pnum);
            end
        end
    end
end
end
end

fraction(j) = insidenum/pnum;

end

```