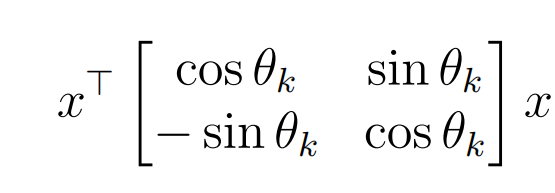
第四题解答：

Using the following codes from matlab to visualize the relative error (“er” for short) of :

k=2.^10;

x=[rand();rand()];%随机生成向量x

y=x';%转置

%对绝对误差和相对误差e与er（都是k+1维数组）进行初始化

e=zeros(k+1,1);

er= zeros(k+1,1);

for n =0:k;

theta=2\*pi\*(n/k);

A=[cos(theta),sin(theta);-sin(theta),cos(theta)];

tr=(y\*x)\*cos(theta); %真实值（或者说尽可能接近真实的值），按照向量的投影后的几何关系计算

fl=single(y)\*single(A)\*single(x); %机器按照矩阵乘法计算后所得浮点数

e(n+1,1)=abs(fl-tr);%绝对误差

er(n+1,1)=e(n+1,1)./(abs(tr)) %相对误差，此处为了监控计算过程，让er输出，没有打分号

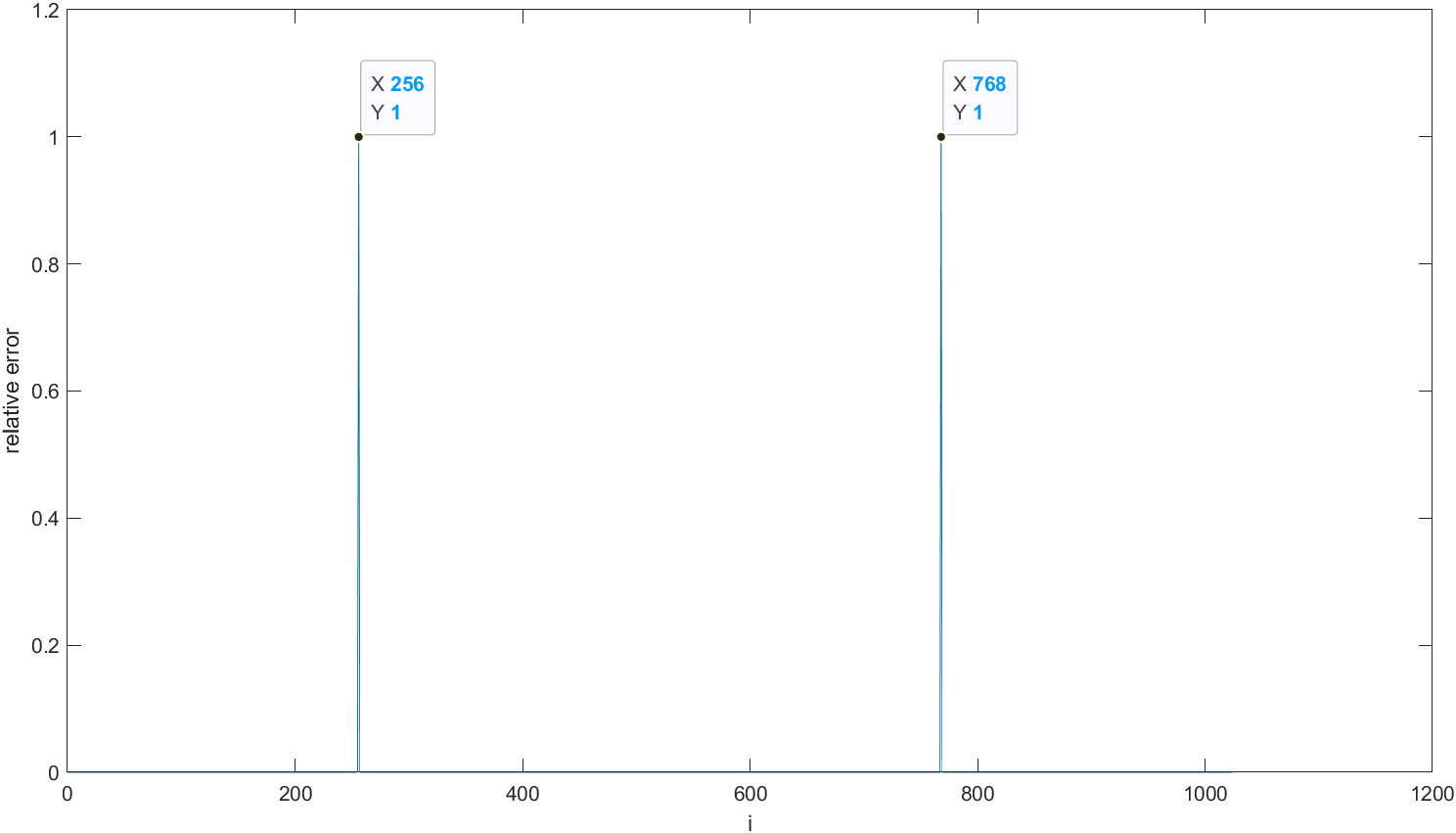
end

plot(0:k,er)%绘制图表，观察er随着i的变化情况

xlabel("n")

ylabel("relative error")

The figure 1 shows itself as:



**Figure 1**

We can only see two large peaks in Figure 1, where n is 256 and 768.

I guess that when the x and xT are nearly perpendicular, the er increases so dramatically that the figure above hardly indicates how the er changes as n increases. Therefore, I implement a trick that the er will be defined 0 if it is over 0.95. The corresponding codes are as follows.

k=2.^10;

x=[rand();rand()];%随机生成向量x

y=x';%转置

%对绝对误差和相对误差e与er（都是k+1维数组）进行初始化

e=zeros(k+1,1);

er= zeros(k+1,1);

for n =0:k;

theta=2\*pi\*(n/k);

A=[cos(theta),sin(theta);-sin(theta),cos(theta)];

tr=(y\*x)\*cos(theta); %真实值（或者说尽可能接近真实的值），按照向量的投影后的几何关系计算

fl=single(y)\*single(A)\*single(x); %机器按照矩阵乘法计算后所得浮点数

e(n+1,1)=abs(fl-tr);%绝对误差

er(n+1,1)=e(n+1,1)./(abs(tr)) %相对误差，此处为了监控计算过程，让er输出，没有打分号

if er(n+1,1)>0.95;

er(n+1,1)=0;

%当x和xT接近垂直的时候，er会变得非常大。

%为了看清er在其他情况下的变化情况，此处将er太大的都记作0，方便之后绘制统计图。

end

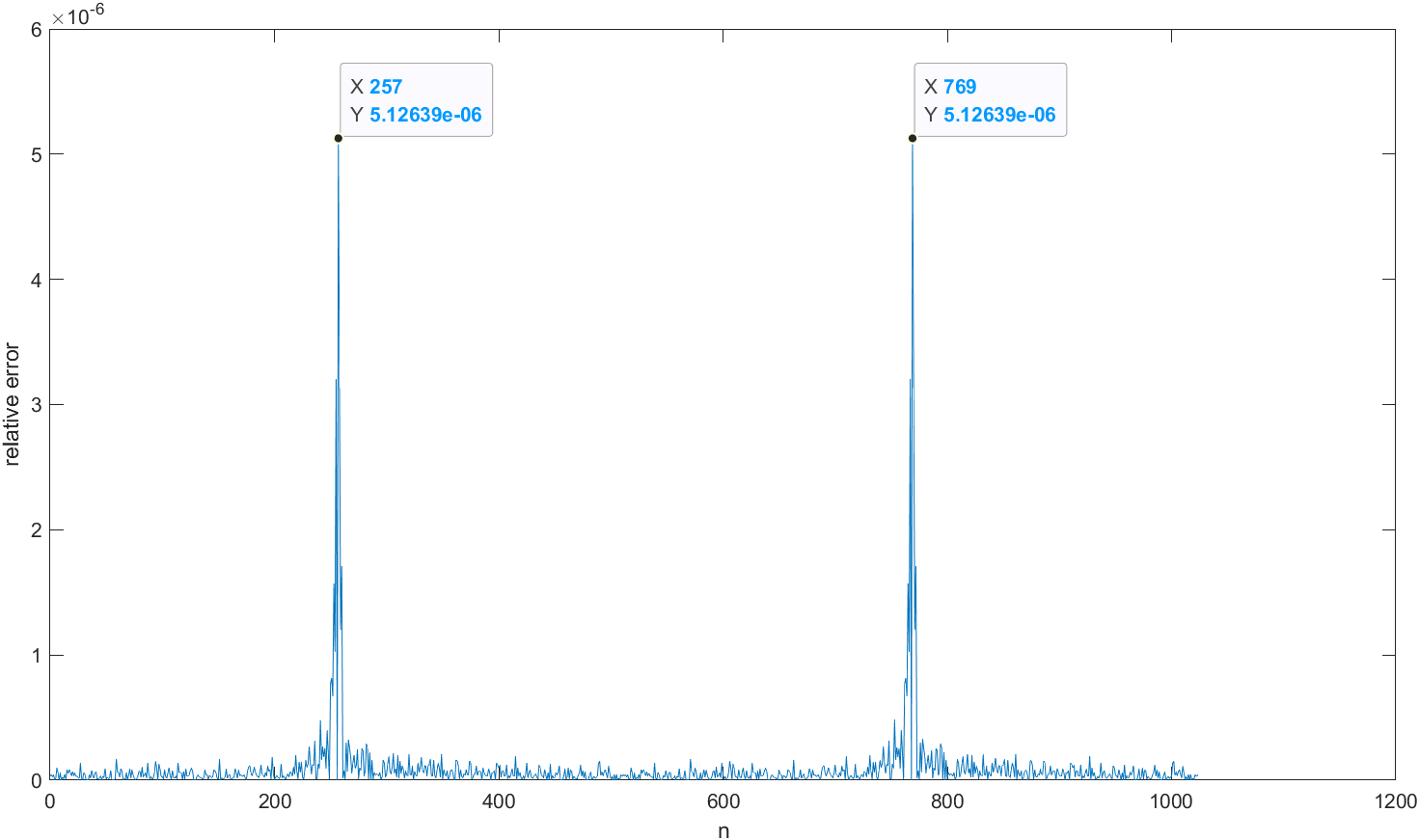
end

plot(0:k,er)%绘制图表，观察er随着i的变化情况

xlabel("n")

ylabel("relative\_error")

Then the figure 2 shows:



**Figure 2 (Er is neglected if it is over 0.95)**

The figure 2 indicates two peaks where n is around 256 and around 768, which is consistent with Figure 1. In the codes above, n increases from 0 to 1024. Hence, when n is close to 256 and 768, theta is approximately 1/4\*pi and 3/4\*pi. I can draw the **conclusion** here: that the more perpendicular x and Ax are, the more enormous er will be.

第五题解答：

Using the following codes from matlab to evaluate the infinite series by single precision float-point arithmetic:

k=10.^8;

Sum1=0;

Sum2=0;

for n=1:k;

a=single(1/(n));

Sum1=Sum1 + a;

if Sum1==Sum2;

End\_of\_n=n-1;%记录最后一个可以表示的a的下标n-1

break %如果Sum1==Sum2，即1/n在单精度计算下，发生了下溢，变成了0，就停止循环

end

Sum2=Sum1;

end

Consequently, End\_of\_n is 2097151 and Sum1 is 15.4036827.

The result means that under single precision float-point arithmetic, the smallest 1/n is 1/2097151 and underflow of “1/n” occurs when n is bigger than 2097151. The infinite series go to infinity theoretically but the matlab fails to substantiate its infinity because the smallest 1/n always exists for a computer.