Task 2.
The purpose of this task is to learn how to time how long it takes the CPU to execute a portion of your code. The timing should be done as explained in the GNU documentation. What you will get reported is CPU time; i.e., the time the CPU took to solve your problem. Keep in mind that the elapsed time can be larger since the operating system engages in time slicing and will execute other programs while your program is in flight.
Given a matrix $\mathbf{A}$ of dimension 1000 by 1000, containing all random numbers stored in double precision, you will have to multiply this matrix by a vector $\mathbf{b}$ of size 1000 whose entries are defined as follows: the first entry will be 0.5, the next one will be -0.5, the next one 0.5, the next one -0.5; i.e., the entries alternate between 0.5 and -0.5.

For program output, you will need to pass two values to the function outputT2:

- The first value is the norm two of the vector $\mathbf{c} = \mathbf{A} \cdot \mathbf{b}$
- The second value is the amount of time, in milliseconds, that the CPU took to multiply carry out $\mathbf{c} = \mathbf{A} \cdot \mathbf{b}$ and then evaluate the norm-2 of $\mathbf{c}$.

Notes:

- The way the matrix $\mathbf{A}$ is stored is as follows: in one array of doubles of size 1,000,000, you will store the first row of A in entries 0 through 999, the second row of $\mathbf{A}$ in entries 1000 through 1999 of the big array, etc. This is called "row-wise storage of $\mathbf{A}$". This array should be *dynamically-allocated*.
- You should not generate the random values yourself, instead you will pass your dynamically-allocated array to the function `randomT2` which will fill it with random numbers.
- There are several ways to multiply a matrix by a vector. The strategy that you must implement is as follows: to get the first entry in $\mathbf{c}$ you will multiply the first row of $\mathbf{A}$ with the $\mathbf{b}$ vector; to get the second entry in $\mathbf{c}$ you will multiply the second row of $\mathbf{A}$ with the $\mathbf{b}$ vector; to get the third entry in $\mathbf{c}$ you will multiply the third row of $\mathbf{A}$ with the $\mathbf{b}$ vector; etc.
- We will compile your program with the following command:
```
gcc task2.c output.c -o task2 -lm
```
- We will run your program with the following command:
```
./task2
```

1. cd /srv/home/cmiao/ME459Upstream/HW06
2. touch task2.c
3. nano task2.c
4. input:

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#include "output.h"




int main(int argc, char *argv[]){
```

```
int i,j;
double sum=0,norm2;
double *A=(double*)malloc(sizeof(double)*1000000);
double *b=(double*)malloc(sizeof(double)*1000);
double *c=(double*)malloc(sizeof(double)*1000);
clock_t start, end;
double cpu_time_used;

b[0]=0.5;
for(i=0;i<999;i++){
   b[i+1]=b[i]*(-1);
}
randomT2(A);

start = clock();

for(i=0;i<1000;i++){
   for(j=0;j<1000;j++){
      sum=A[i*1000+j]*b[j]+sum;
   }
   c[i]=sum;
   sum=0;
}

for(i=0;i<1000;i++){
   sum=c[i]*c[i]+sum;
}
norm2=sqrt(sum);

end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
```

```
    cpu_time_used = cpu_time_used*1000;

    outputT2(norm2,cpu_time_used);

    free(A);

    free(b);

    free(c);

}
```

5. gcc task2.c output.c -o task2 -lm -std=c99

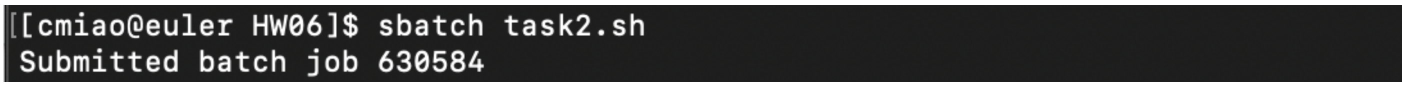
6. touch task2.sh

7. nano task2.sh

8. input:

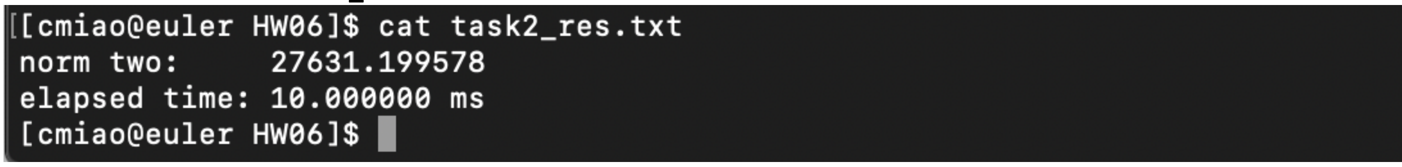
```
#!/usr/bin/env bash
#SBATCH --job-name=Task2
#SBATCH -p shortgpu
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0-00:00:10
#SBATCH --output=task2_res.txt

cd $SLURM_SUBMIT_DIR

./task2
```

9. sbatch task2.sh



```
[[cmiao@euler HW06]$ sbatch task2.sh
 Submitted batch job 630584
```

10. cat task2_res.txt



```
[[cmiao@euler HW06]$ cat task2_res.txt
 norm two:     27631.199578
 elapsed time: 10.000000 ms
 [cmiao@euler HW06]$
```