Task 5. (Bonus).
This task is a continuation of Task2.
Solve Task 2 again, but this time multiply the matrix and the vector in the following way: the array **c** should be computed as the first column of **A** the first entry in **b**, to which you add the second column of **A** scaled by the second entry of **b**, to which you add the third column of **A** scaled by the third entry of **b**, etc. Make sure your code produces the same outcome as for Task 2.

Additionally, answer the following in writing: Assume that the amount of time required to solve the problem in Task 2 is T2, while the amount of time required to solve the problem in Task 5 is T5.

- How do T2 and T5 compare? When answering this question, make sure you run the codes several times to get a sample of T2 values and T5 values – just to avoid a situation where you get some sort of outlier results.
- Do you have any insight into why the values might be different?

Notes:

- The norm 2 of **c** should be the same, since the values in **c** should be like the ones you calculated in Task 2.
- You should also use the same helper functions, `randomT2` and `outputT2`, for this task.
- We will compile your program with the following command:
  `gcc task5.c output.c -o task5 -lm`
- We will run your program with the following command:
  `./task5`

1. cd /srv/home/cmiao/ME459Upstream/HW06
2. touch task5.c
3. nano task5.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++)
            A[j*1000+i]=A[j*1000+i]*b[i];


    for(i=0;i<1000;i++){
        for(j=0;j<1000;j++){
            sum=A[i*1000+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 task5.c output.c -o task5 -lm -std=c99

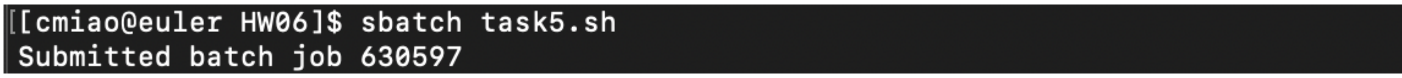
6. touch task5.sh

7. nano task5.sh

8. input:

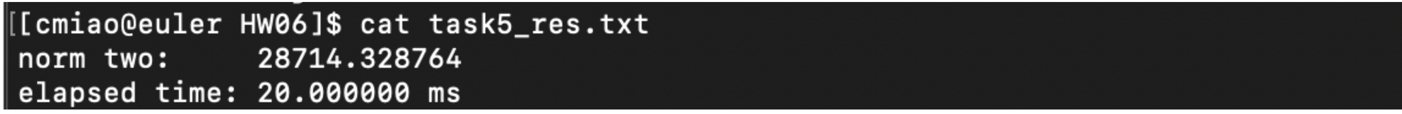
```
#!/usr/bin/env bash
#SBATCH --job-name=Task5
#SBATCH -p shortgpu
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0-00:00:10
#SBATCH --output=task5_res.txt

cd $SLURM_SUBMIT_DIR

./task5
```

9. sbatch task5.sh



```
[[cmiao@euler HW06]$ sbatch task5.sh
 Submitted batch job 630597
```

10. cat task5_res.txt



```
[[cmiao@euler HW06]$ cat task5_res.txt
 norm two:     28714.328764
 elapsed time: 20.000000 ms
```

Time for Task5 is longer, because it lost the locality. It jumps 1000 numbers to read the next one. So it will takes more time.