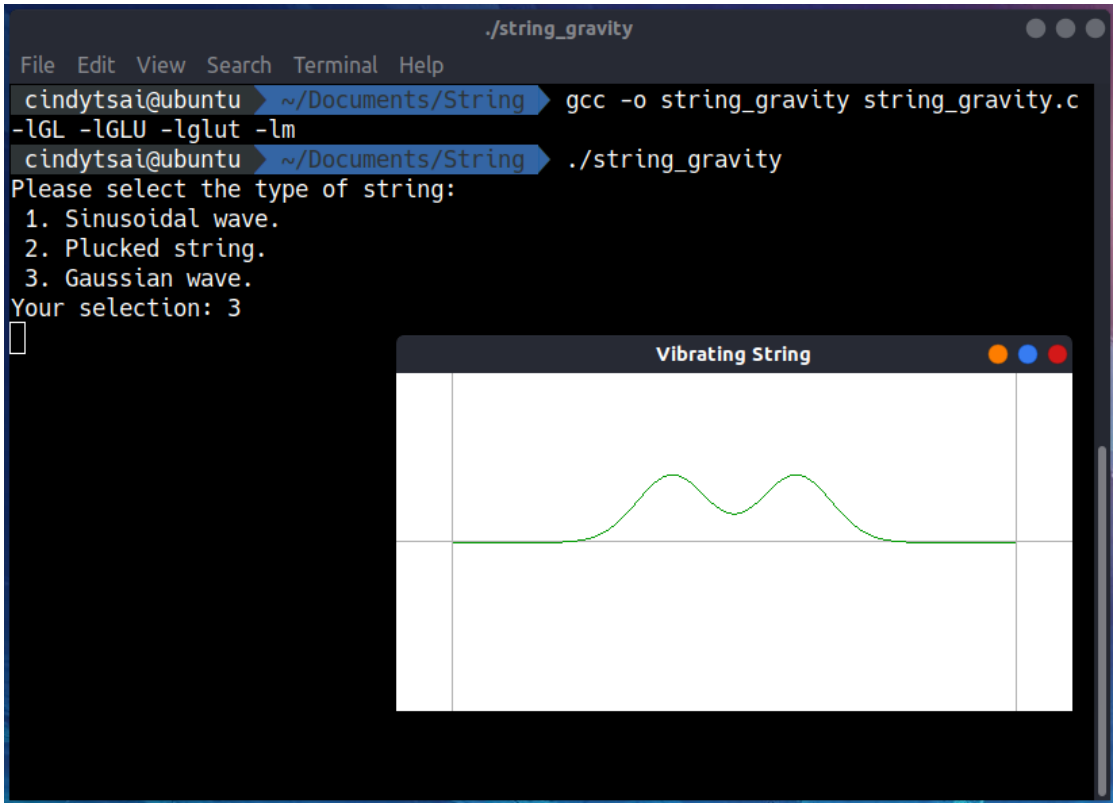
Problem Set 8 **Shin-Rong, Tsai R08244002**

- **Vibrating String Under Gravitational Acceleration**

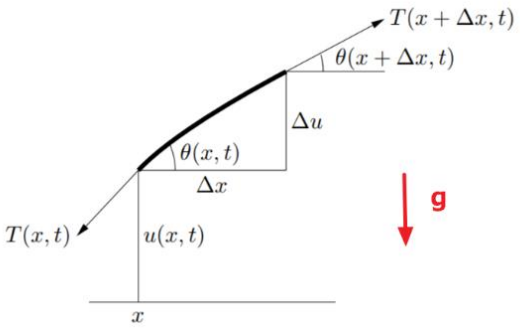
1. Result

Slightly change the code by professor in *string.c* in the function PDE, and save it as *string_gravity.c*. And also, I change the tensor of the string to 100.



2. Discussion

A. Derivation of the equation of motion of vibrating string with gravity.



Consider vertical force,

$$\rho(x)\sqrt{\Delta x^2+\Delta u^2}\frac{\partial^2 u(x,t)}{\partial t^2}$$
$$= T(x+\Delta x,t)\cdot sin\big(\theta(x+\Delta x,t)\big) - T(x,t)\cdot sin\big(\theta(x,t)\big) - g$$
$$\cdot \rho(x)\sqrt{\Delta x^2+\Delta u^2}$$

Take $\Delta x \to \infty$, we get

$$\rho(x)\sqrt{1+\left(\frac{\partial u}{\partial x}\right)^2}\frac{\partial^2 u(x,t)}{\partial t^2} = \frac{\partial}{\partial x}\left[T(x,t)\cdot sin\big(\theta(x,t)\big)\right] - g\cdot\rho(x)$$

$$= \frac{\partial T}{\partial x}(x,t)\cdot sin\big(\theta(x,t)\big) + T(x,t)\cdot cos\big(\theta(x,t)\big)\cdot\frac{\partial \theta}{\partial x}(x,t) - g\cdot\rho(x)$$

Using trigonometric functions approximation, we get

$$\rho(x)\frac{\partial^2 u(x,t)}{\partial t^2} = \frac{\partial T}{\partial x}(x,t)\frac{\partial u}{\partial x}(x,t) + T(x,t)\frac{\partial^2 u}{\partial x^2}(x,t) - g\cdot\rho(x)$$

And since the string we are dealing with is assumed to have the same tension and same mass density everywhere, which gives

$$\frac{\partial^2 u(x,t)}{\partial t^2} = \frac{T}{\rho}\frac{\partial^2 u}{\partial x^2}(x,t) - g \quad let\ \ c \equiv \sqrt{\frac{T}{\rho}}$$

Discretize this equation, and rearrange them, we get

$$\frac{u_i^{j+1} - 2u_i^j + u_i^{j-1}}{\delta^2} - c^2\frac{u_{i+1}^j - 2u_i^j + u_{i-1}^j}{h^2} + g = 0$$

$$\Rightarrow u_i^{j+1} = \frac{\delta^2 c^2}{h^2}\left(u_{i+1}^j + u_{i-1}^j\right) + 2\left(1 - \frac{\delta^2 c^2}{h^2}\right)u_i^j - u_i^{j-1} - g\delta^2 \cdots (*)$$

To initialize this, from

$$\left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0} = \frac{u_i^1 - u_i^{-1}}{2\delta} \Rightarrow u_i^{-1} = u_i^1 - 2\delta\left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0}$$

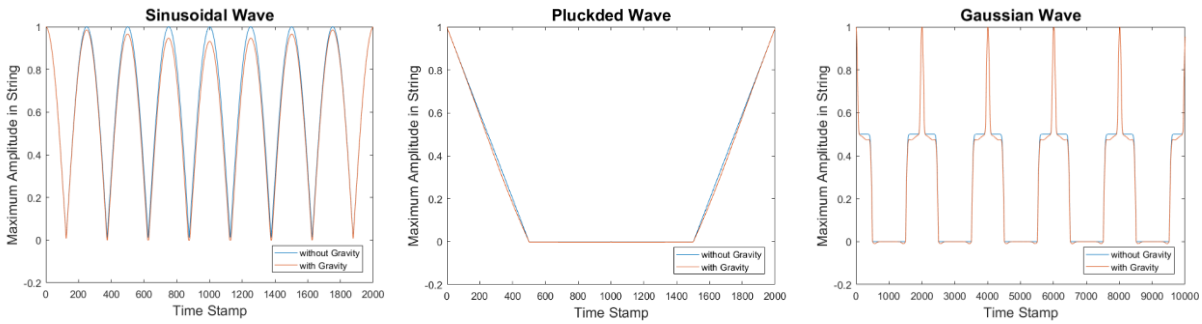
Bring this equation into the equation $(*)$, and rearrange them, we have

$$u_i^1 = \frac{\delta^2 c^2}{2h^2}(u_{i+1}^0 + u_{i-1}^0) + \left(1 - \frac{\delta^2 c^2}{h^2}\right)u_i^0 + \delta\left.\frac{\partial u(x,t)}{\partial t}\right|_{t=0} - \frac{1}{2}g\delta^2 \cdots (\blacksquare)$$

So by $(*)$ and $(\blacksquare)$, we can have a time series of u_i^j.

B. Compare maximum amplitude in the string with and without gravity.

Since the gravity acts like the drag down term in the equation of motion, it is obvious that the maximum amplitude in the vibrating string with gravity will be lower than the one without gravity. The shape of their trajectory are the same. The Gaussian wave figure may seem to have an orange color needle-like bulge,



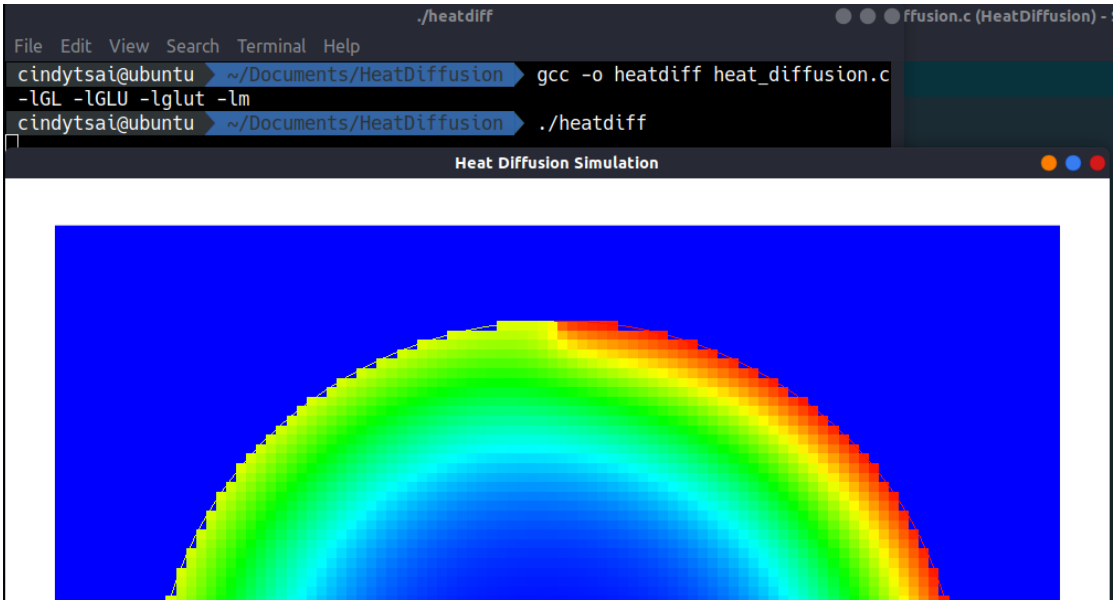
but in fact, it's just the blue one and the orange one are highly overlapping. So if we zoom in, we will also see that the orange line (with gravity) is slightly below the blue line (without gravity).

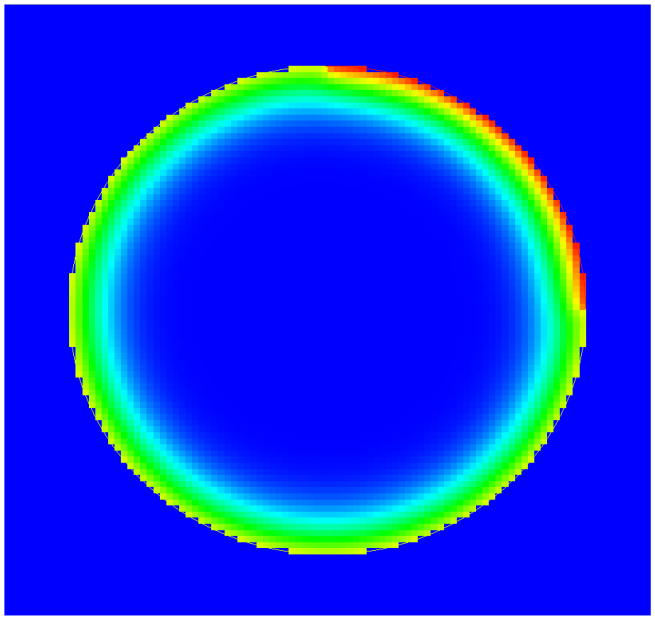
- ## Heat Diffusion

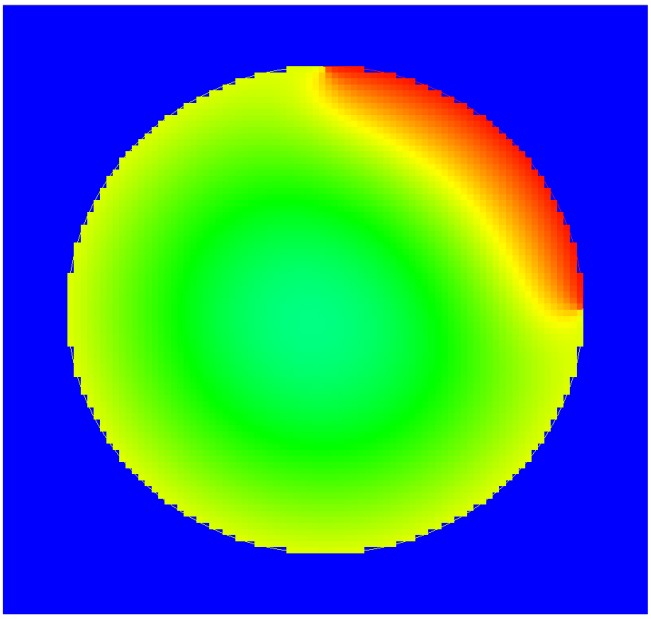
1. Result

Compile the code *heat_diffusion.c* in Linux, we will see how it diffuse into the plate. It takes a bit long, so I did not wait till the end.

I set the color bar range at 0 ~ 380, so that we can easily distinguish them.







2. Discussion

A. The heat diffusion formula with irregular boundary value.

From the lecture we know what $\frac{d^2u}{dx^2}$ and $\frac{d^2u}{dy^2}$ are when it is near the boundary, and we can also write these form back to the original one, when all neighbors are inside the boundary.

So when we discretize the heat diffusion formula, we can have

$$\nabla^2 u(x,y,t) = \frac{1}{c^2}\frac{\partial u}{\partial t} \Longrightarrow \nabla^2 u = \frac{d^2u}{dx^2} + \frac{d^2u}{dy^2} = \frac{1}{c^2}\frac{u_{ij}^{k+1} - u_{ij}^{k}}{\delta t}$$

$$\Longrightarrow u_{ij}^{k+1} = c^2 \cdot \delta t \cdot \nabla^2 u + u_{ij}^{k}$$

B. Building up the short-rainbow color map.

I think it is really interesting!! First invert the scale, since we start from red to blue. Then divide into 4 groups. And match their red, green, blue respectively.

	R	G	B
Group 1	255	0 ⇢ 255	0
Group 2	255 ⇢ 0	255	0
Group 3	0	255	0 ⇢ 255
Group 4	0	255 ⇢ 0	255

C. Some other observations

The more nearer to the ring, the faster the temperature changes and that place and every individual spot will affect its neighbors.

And we verify that the heat flows from the higher temperature to the lower one.

- ## Poisson Equation

1. Result

I did not split the algorithm in different file, so it is large.

Compile and run the file *solve_poisson_eq.c*.

```
cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_8
$ gcc -o a.out solve_poisson_eq.c

cindytsai@TURQUOISEA /cygdrive/d/GitHub/Computational_Physics/Assignment/ProblemSet_8
$ ./a.out
Enter the size L of the cube L x L x L:
200
200
Choose the algorithm:
0: Jacobi method
1: Gauss Seidel method
2: Successive Over-Relaxation method
3: CG Algorithm
4: CG Algorithm with restart
4
4
0: Double Precision (double) with Restart
1: Mixed Precision (single) with Restart
1
1
Input photon mass :
1e-4
1.00000e-04
```

2. Discussion

A. Algorithm

We are dealing Laplace in 3D, when we discretize it

$$\nabla^2\phi(x,y,z) = -\rho$$

$$\rightarrow \frac{1}{\triangle^2}[\phi(x+1,y,z) + \phi(x-1,y,z) + \phi(x,y+1,z) + \phi(x,y-1,z) + \phi(x,y,z+1) + \phi(x,y,z-1) - 6\phi(x,y,z)] = -q\delta(x)\delta(y)\delta(z)$$

Let discretized $\nabla^2\phi(x,y,z) = -A$

$\Rightarrow -A\phi = -\rho \qquad \therefore A\phi = \rho$

If we decompose *A* into *L*, *U*, *D*, we get

D: Matrix with diagonal equals 6, and off-diagonal equals 0.

U: Calculating $U\phi$ is the same as getting the {right, forward, top} neighbor.

L: Calculating $L\phi$ is the same as getting the {left, backward, down} neighbor.

Thus we can easily implement this instead of doing a real matrices multiplication.

B. Exact Solution

Use $\phi(r=1)$ as base and let $q = 1$.

$$\vec{E} = -\nabla\phi = \frac{q}{4\pi r^2}$$

$$\Rightarrow \phi(r) - \phi(1) = \int_r^1 \vec{E}\cdot d\vec{r} = \int_r^1 \frac{q}{4\pi r^2}\cdot d\vec{r} = -\frac{q}{4\pi r}\Big|_r^1 = \frac{1}{4\pi}\left(\frac{1}{r} - 1\right)$$

$$\Rightarrow \phi(r) = \frac{1}{4\pi r} - \frac{1}{4\pi} + \phi(1)$$

C. Efficiency

<u>General settings:</u>

box length $\mathrm{L} = 200$ $\Rightarrow$ a 8000000×8000000 matrix

target error $= 1 \times 10^{-7}$, it seems like it take too long.

maximum iteration $= 6001$

<u>For CG with restart:</u>

inner loop error $= 0.1$

photon mass $= 0.0001$

	Time Used (sec)	Error	Iteration
Jacobi	2720.891	7.23×10^{-4}	6001
Gauss-Seidel	3113.750	3.76×10^{-4}	6001
SOR	3241.125	3.56×10^{-4}	6001

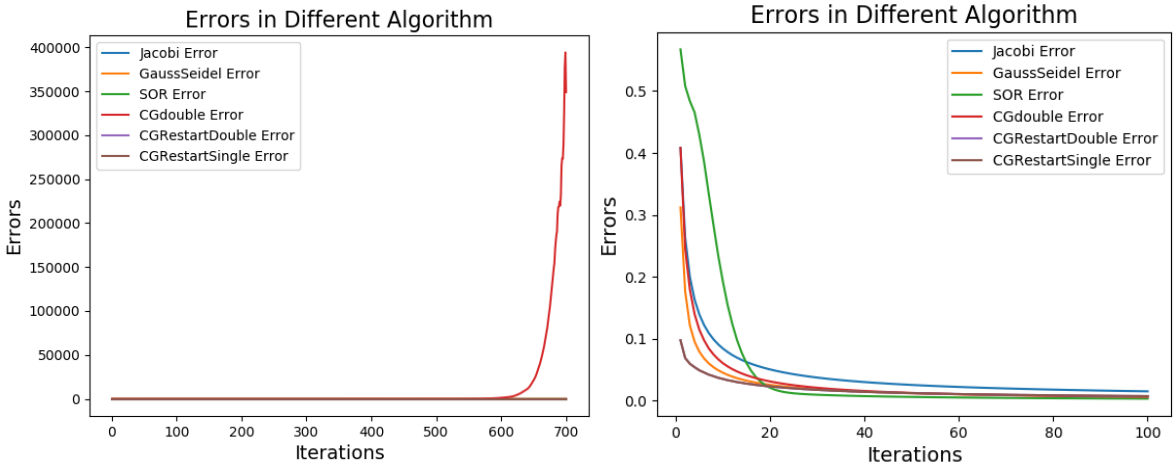
CG		159.812	7.76×10^{-3}	174
CG Restart	Double	2530.750	1.00×10^{-3}	1543
	Mixed	2948.938	1.00×10^{-3}	1543

- The result in CG Algorithm (without restart) actually did not converge, when it reaches the minimum error, it bumps back and starts to diverge. Since the matrix we are dealing with is eight million times eight million, which is really large. So the orthogonal properties might not hold due to finite precision arithmetic. We should avoid CG Algorithm when dealing with huge square matrix.
- We can see that the order of time and iteration used in both CG Restart are the same. For CG restart with mixed precision, it takes less memory, since the inner loop's data type is float. So CG Restart with mixed precision is better than with double precision. Though it might take some time to do type-casting between "double" and "float".
- Jacobi/Gauss-Seidel/SOR/CG Restart took the same order amount of time ($2500 \sim 3000$), to reach a solution with error around $10^{-4} \sim 10^{-3}$. Though Jacobi/Gauss-Seidel/SOR have errors less than CG Restart, it takes more iterations.
- If we let CG Restart Mixed run the same iteration as Jacobi/Gauss-Seidel/SOR, CG Restart Mixed precision doesn't seem to be a good choice. Though it has less error, it used up too much time.

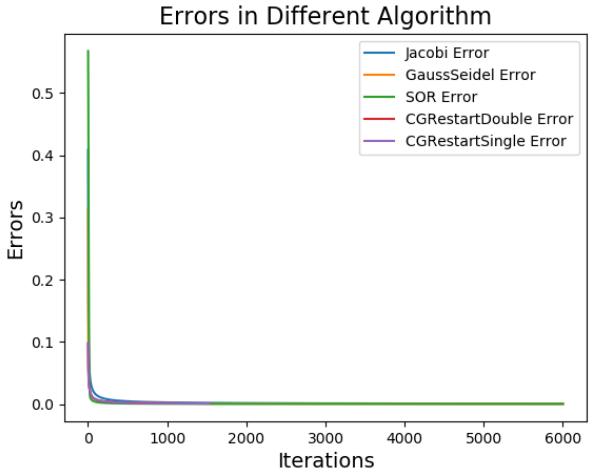
	Time Used (sec)	Error	Iteration
Jacobi	2720.891	7.23×10^{-4}	6001
Gauss-Seidel	3113.750	3.76×10^{-4}	6001
SOR	3241.125	3.56×10^{-4}	6001
CG Restart Mixed	11120.828	7.05×10^{-4}	6001

D. Errors

We can see that CG Algorithm without restart will diverge when it reaches a minimum, as others will slowly converge and the errors are getting smaller. But if we focus on the region before the point where CG Double (without restart) starts to diverge, it acts just like others, so it isn't that bad if we did not demand the error to be too small. (In file *Q3-Error_plot.py*)



The errors in the other methods will saturate, so it takes much more efforts to achieve a smaller error in such a large matrix.



E. Compare to the exact solution

There is a gap between exact solution and simulation. I think it is because we set $\triangle = dx = 1$, which is too large for 3D Laplace. (In file *Q3_exactSol.py*)