Fourier Spectral Method (II)

Application

Fast Fourier Transformation

- A fast Fourier transformation (FFT) is an algorithm to compute the discrete Fourier transformation (DFT) and its inverse.
- There are many different FFT algorithm, and most commonly used one is Cooley-Tukey algorithm.
- Computing the DFT of N points in the naive way, using the definition, takes $O(N^2)$ arithmetical operations.

Fast Fourier Transformation

- By Cooley-Tukey algorithm, we can compute the same result with only O(N log N).

- It is based on a divide and conquer that recursively breaks down a DFT of any composite size $N = N_1N_2$ into many smaller DFTs of size N_1 and N_2.

- We also use FFT in multidimensional domain and order of computation is still O(N log N).

Heat Equation

- For an example of application, we consider the heat equation in [0, L] :

$$u_t = u_{xx}$$

- Usually, we use the periodic boundary condition (BC) for the Fourier transformation. However, we can also use the Dirichlet or Neumann BC using sine or cosine Fourier transformation, respectively.

- Here, we use periodic BC which is a general case.

Heat Equation

- The DFT and its inverse (iDFT) of u is given by

$$\hat{u}_p^n = \sum_{m=1}^{M} u_m^n e^{-ix_m \xi_p}$$

$$u_m^n = \frac{1}{M} \sum_{p=1-M/2}^{M/2} \hat{u}_p^n e^{ix_m \xi_p}$$

where M is the number of grid points and $\xi_p = \frac{2\pi(p-1)}{L}$

- We plug this into the heat equation.

Heat Equation

- By using forward difference in time, the heat equation is written as

$$\frac{u^{n+1} - u^n}{\Delta t} = \frac{\partial^2}{\partial x^2} u^n$$

and applying the Fourier transformation in both side,

$$\frac{1}{M} \sum_p \frac{\hat{u}_p^{n+1} - \hat{u}_p^n}{\Delta t} e^{ix_m \xi_p} = \frac{1}{M} \sum_p (i\xi_p)^2 \hat{u}_p^n e^{ix_m \xi_p}$$

$$= -\frac{1}{M} \sum_p \xi_p^2 \hat{u}_p^n e^{ix_m \xi_p}$$

Heat Equation

- By orthogonality, we can cancel out the same terms and summation :

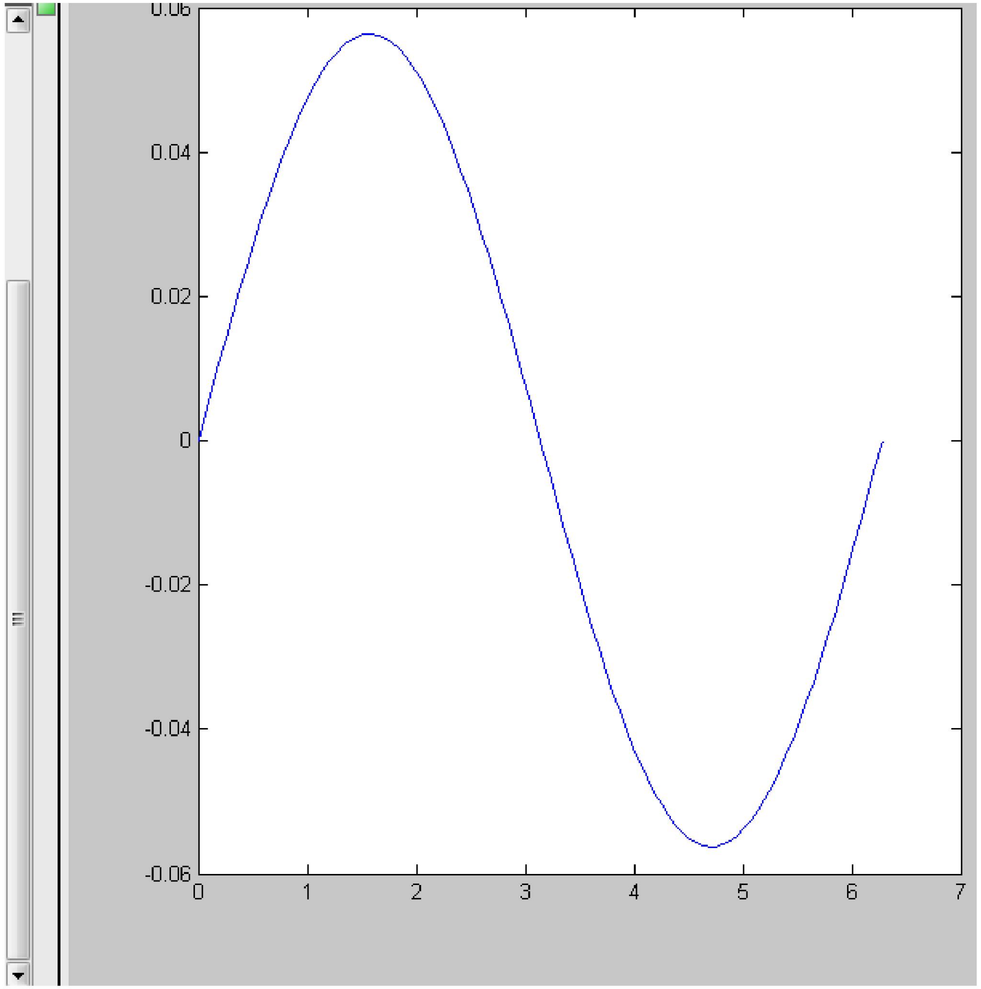
$$\frac{\hat{u}_p^{n+1} - \hat{u}_p^n}{\Delta t} = -\xi_p^2 \hat{u}_p^n$$

$$\hat{u}_p^{n+1} = \hat{u}_p^n \left(1 - \Delta t \xi_p^2\right)$$

- And by the inverse transformation, we can reconstruct u^{n+1} from u^n.

Code

```
clc; clf; clear all;

M = 100; x = linspace(0, 2*pi, M); dx = x(2) - x(1); L = x(end) - x(1);
u = sin(x); T = 0.05; dt = 0.5*dx*dx; nt = round(T/dt);

for iter = 1:nt
    uh = 0*u;
    for ii = 1:100
        for jj = 1:100
            uh(ii) = uh(ii) + u(jj)*exp(-1i*x(jj)*2*pi*(ii-1)/L);
        end
    end

    nuh = uh*0;
    for ii = 1:100
        nuh(ii) = uh(ii)*(1-dt*2*pi*(ii-1)/L);
    end

    nu = 0*u;
    for ii = 1:100
        for jj = 1:100
            nu(jj) = nu(jj) + nuh(ii)*exp(1i*x(jj)*2*pi*(ii-1)/L);
        end
    end
    nu = real(nu)/M;

    plot(x, nu)
    pause(0.01);

    u = nu;
end
```



Code

```
clc; clf; clear all;

M = 100; x = linspace(0, 2*pi, M); dx = x(2) - x(1); L = x(end) - x(1);
u = sin(x); T = 0.05; dt = 0.5*dx*dx; nt = round(T/dt);

for iter = 1:nt
    uh = 0*u;
    for ii = 1:100
        for jj = 1:100
            uh(ii) = uh(ii) + u(jj)*exp(-1i*x(jj)*2*pi*(ii-1)/L);
        end
    end

    nuh = uh*0;
    for ii = 1:100
        nuh(ii) = uh(ii)*(1-dt*2*pi*(ii-1)/L);
    end

    nu = 0*u;
    for ii = 1:100
        for jj = 1:100
            nu(jj) = nu(jj) + nuh(ii)*exp(1i*x(jj)*2*pi*(ii-1)/L);
        end
    end
    nu = real(nu)/M;

    plot(x, nu)
    pause(0.01);

    u = nu;
end
```

- At first three lines, there are basic settings such as the number of grid points, domain size, initial condition and total time.

- The first loop in the loop-by-'iter' is the Fourier transformation.

Code

```
clc; clf; clear all;

M = 100; x = linspace(0, 2*pi, M); dx = x(2) - x(1); L = x(end) - x(1);
u = sin(x); T = 0.05; dt = 0.5*dx*dx; nt = round(T/dt);

for iter = 1:nt
    uh = 0*u;
    for ii = 1:100
        for jj = 1:100
            uh(ii) = uh(ii) + u(jj)*exp(-1i*x(jj)*2*pi*(ii-1)/L);
        end
    end

    nuh = uh*0;
    for ii = 1:100
        nuh(ii) = uh(ii)*(1-dt*2*pi*(ii-1)/L);
    end

    nu = 0*u;
    for ii = 1:100
        for jj = 1:100
            nu(jj) = nu(jj) + nuh(ii)*exp(1i*x(jj)*2*pi*(ii-1)/L);
        end
    end
    nu = real(nu)/M;

    plot(x, nu)
    pause(0.01);

    u = nu;
end
```

- The second loop is to update $\hat{u}^{n+1}$.
- The third loop is the inverse Fourier transformation.
- Last three lines are for plot the figure and updating for next time step.