Numerical Implementation of Fourier Galerkin Method

Quanhui Zhu

November 30, 2018

1 Introduction

The Fourier-Galerkin Method is to solve the differential equations with periodic boundary conditions. We project the solutions from infinite dimensional space into finite dimensional space $\hat{B_N} = span\{e^{inx}|n \leq N/2\}$.

The numerical solution $u_h = \sum\limits_{n=-N/2}^{N/2} \hat{u}_n e^{inx_h}$, where $x_j = \frac{2\pi j}{N}, j = 0, 1, \cdots, N-1$ are the nodes of the domain and $\hat{u}_n$ are the Fourier coefficients given by

$$\hat{u}_n = \frac{1}{2\pi}\int_0^{2\pi} u(x)e^{-inx}dx \tag{1}$$

In discrete cases, it can be rewritten by

$$\hat{u}_n = \frac{1}{N}\sum_{j=0}^{N-1} u(x_j)e^{-inx_j} \tag{2}$$

For the differential equations

$$u_t = \mathcal{L}u + \mathcal{N}u \tag{3}$$

the numerical problem using semi-implicit scheme can be rewritten by

$$\frac{u^{n+1} - u^n}{\delta t} = \mathcal{L}u^{n+1} + \mathcal{N}u^n. \tag{4}$$

Then we calculate the problem in spectral space

$$\frac{\hat{u}_k^{n+1} - \hat{u}_k^n}{\delta t} = f(k)\hat{u}_k^{n+1} + (\hat{\mathcal{N}u^n})_k. \tag{5}$$

The matrix is diagonal.

2 Matching with Matlab FFT

2.1 1-D Match

Assume the number of the nodes is $2N+1$(odd), the even nodes have a little difference.

In matlab the ifft function is

$$X(k) = \frac{1}{2N+1}\sum_{j=1}^{2N+1} x(j)e^{\frac{2\pi i(j-1)(k-1)}{2N+1}}, \quad k = 1 : 2N+1. \tag{6}$$

For matching

$$\hat{u}_n = \frac{1}{2N+1}\sum_{j=0}^{2N} u_j e^{\frac{2\pi i j n}{2N+1}}, \quad n = -N : N, \tag{7}$$

we shift $k = n + N + 1$, thus

$$\hat{u}_n = \frac{1}{2N+1}\sum_{j=1}^{2N+1} u_{j-1} e^{\frac{2\pi i (j-1)(k-N-1)}{2N+1}} = \frac{1}{2N+1}\sum_{j=1}^{2N+1} u_{j-1} e^{\frac{-2N\pi i (j-1)}{2N+1}} e^{\frac{2\pi i (j-1)(k-1)}{2N+1}} \tag{8}$$

So we can obtain $\hat{u} = ifft(\{u_{j-1}e^{\frac{-2N\pi i(j-1)}{2N+1}}\}_j)$.

Similarly, we can obtain $u = \{e^{\frac{-2N\pi i(j-1)}{2N+1}}\}_j .* fft(\hat{u})$.

2.2 Arbitrary Period

When the domain changes $x \in [a, b]$, we have $x_j = \frac{b-a}{2N+1} j + a, \quad j = 0, \cdots, 2N$. It just need a shift to the domain $[0, 2\pi]$,

$$y = \frac{2\pi}{b-a}(x - a) \tag{9}$$

So we have the spectral

$$\hat{u}_n = \frac{1}{2N+1}\sum_{j=1}^{2N+1} u_{j-1} e^{in\frac{2\pi}{b-a}(x_{j-1}-a)} = \frac{1}{2N+1}\sum_{j=1}^{2N+1} u_{j-1} e^{\frac{2\pi i n (j-1)}{2N+1}} \tag{10}$$

Then the transform is the same.

2.3 2-D Match

In the domain $[0, 2\pi] \times [0, 2\pi]$, assume the number of the nodes is $(2N + 1) \cdot (2N + 1)$.

In matlab the ifft2 is the 2-D Inverse Fast Fourier Transform which has the equation that

$$Y_{p,q} = \frac{1}{mn}\sum_{j=1}^{m}\sum_{k=1}^{n} X_{j,k}\omega_m^{(j-1)(p-1)}\omega_n^{(k-1)(q-1)} \tag{11}$$

where $\omega_N = e^{\frac{2\pi i}{N}}$.

It's the same with 1-D's transform. For matching with the textbook, we just need to shift both in x and y direction. We have

$$\hat{u}_{p,q} = \frac{1}{(2N+1)^2}\sum_{j=1}^{2N+1}\sum_{j=1}^{2N+1} u_{j-1,k-1} \cdot e^{\frac{-2N\pi i (j-1)}{2N+1}} \cdot e^{\frac{-2N\pi i (k-1)}{2N+1}} \omega_m^{(j-1)(p-1)}\omega_n^{(k-1)(q-1)} \tag{12}$$

Assume

$$y_{j,k} = u_{j,k} \cdot e^{\frac{-2N\pi i j}{2N+1}} \cdot e^{\frac{-2N\pi i k}{2N+1}}, \quad j = 0 : 2N, k = 0 : 2N \tag{13}$$

Thus we have $\hat{u} = ifft2(y)$.

Similarly, assume $z = fft2(\hat{u})$, we have

$$u_{j,k} = z_{j,k} \cdot e^{\frac{2N\pi i j}{2N+1}} \cdot e^{\frac{2N\pi i k}{2N+1}}, j = 0 : 2N, k = 0 : 2N \tag{14}$$

3 Numerical Examples

3.1 Diffusion Equation

The equation is

$$-u_{xx} = f. \tag{15}$$

Using Fourier galerkin method with periodic boundary condition, the equation can rewritten as

$$\begin{aligned} -\sum_{n=-\frac{N}{2}}^{\frac{N}{2}} \hat{u}_n (e^{inx})_{xx} &= \sum_{n=-\frac{N}{2}}^{\frac{N}{2}} \hat{f}_n e^{inx}; \\ i.e. \qquad n^2 \hat{u}_n &= \hat{f}_n, \qquad n = -\frac{N}{2} : \frac{N}{2}. \end{aligned} \tag{16}$$

Notice that when $n = 0$, we have $\hat{f}_0 = \frac{1}{N}\sum_j f(x_j) = 0$ which is called compatibility condition. And the numerical solution is $u_h = \sum_{n=-\frac{N}{2}, n\neq 0}^{\frac{N}{2}} \hat{u}_n e^{inx} + \hat{u}_0$, where $\hat{u}_0 = \frac{1}{N}\sum_j u(x_j)$ called stability condition should be given.

Firstly we calculate $u = sin2x$ to test. $u = sin2x$ shall be exact for any N since it is a base of B_N. When $N = 10$, the residual is less than $1e - 15$. This test can show whether your codes have grammar error.

Secondly we calculate $u = \frac{3}{5-4cosx}$ and observe how the residual changes with N changing.

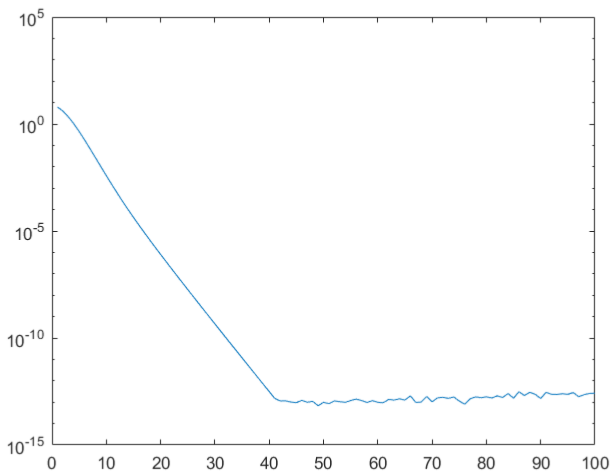

Figure 1: The x-axis is $\frac{N-1}{2}$ where N is the number of nodes and the y-axis is $ln(error)$ where the error is $||u - u_h||_{L^\infty}$.

In Figure 1 it follows $||u - u_h|| = e^{-N}$ until the error attains machine epsilon.

After these two tests finished, the program is much possible to be right and we shall use the method to solve typical problem now.

3.2 1-D Allen Cahn Equation

The equation is

$$u_t = \epsilon u_{xx} + u - u^3. \tag{17}$$

Using Fourier galerkin method and semi-implicit scheme, the iteration takes the form as

$$\frac{\hat{u}_k^{n+1} - \hat{u}_k^n}{\delta t} = -\epsilon k^2 \hat{u}_k^{n+1} + (u^n - \hat{(}u^n)^3)_k, \tag{18}$$

where δt depends on ϵ and h.

Here is a numerical experiment. Set $\Omega = [0, 2\pi], N = 40, \delta = \frac{1}{N^2}, T = 1, \epsilon = 0.01$ and $u(x, 0) = sin(x)$. Figure 2 shows the numerical solution.

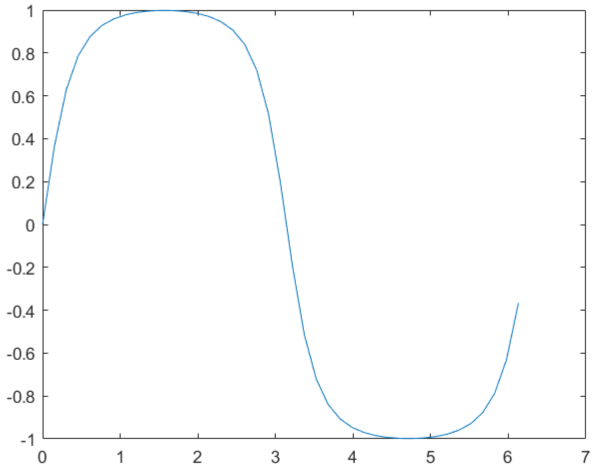

Figure 2: The Allen-Cahn equation's solution when $T = 1$ and $\Omega = [0, 2\pi], N = 40, \delta t = \frac{1}{N^2}, \epsilon = 0.01, u(x, 0) = sin(x)$.

3.3 2-D Allen Cahn Equation

The equation is the same

$$u_t = \epsilon \Delta u + u - u^3. \tag{19}$$

Using Fourier galerkin method and semi-implicit scheme, similarly the iteration takes the form as

$$\frac{\hat{u}_{p,q}^{n+1} - \hat{u}_{p,q}^n}{\delta t} = -\epsilon(p^2 + q^2)\hat{u}_{p,q}^{n+1} + (u^n - \hat{(}u^n)^3)_{p,q}. \tag{20}$$

Figure 3 and Figure 4 are two numerical examples.

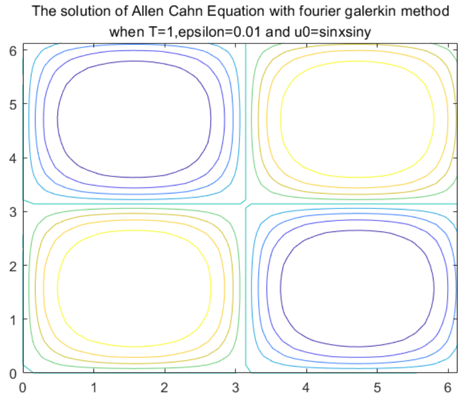

Figure 3: The 2D Allen-Cahn equation's solution when $T = 1$ and $\Omega = [0, 2\pi] \times [0, 2\pi], N = 20, \delta = 0.001, \epsilon = 0.01, u(x, 0) = sin(x)sin(y)$. It's similar with 1D's.

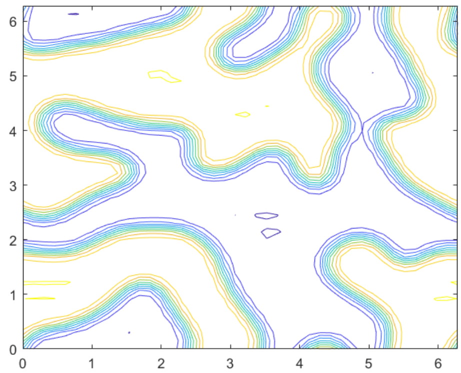

Figure 4: The 2D Allen-Cahn equation's solution when $T = 10$ and $\Omega = [0, 2\pi] \times [0, 2\pi], N = 20, \delta t = 0.001, \epsilon = 0.01, u(x, 0)$ are random numbers in [-1,1]. The figure shows most of u get to -1 or 1 and the interface changes quickly.

3.4 Cahn-Hailliard

Consider the Cahn-Hailliard equation

$$\begin{array}{rcll} \frac{\partial u}{\partial t} & = & \Delta(-\epsilon^2 \Delta u + f(u)), & x \in \Omega, t \in [0, T], \\ u(x, 0) & = & u_0(x), & x \in \Omega, \end{array} \tag{21}$$

where the $f(u) = u^3 - u$. By Fourier Spectral Transform, the equation can be rewritten with the form of

$$\frac{\hat{u}^{n+1}_{p,q} - \hat{u}^n_{p,q}}{\delta t} = -\epsilon^2 (p^2 + q^2)^2 \hat{u}^{n+1}_{p,q} + (p^2 + q^2) \hat{f(u)}_{p,q}. \tag{22}$$

Here are three numerical solutions of Cahn-Hailliard equation.

In Figure 6, we choose the soltion when $T = 2$ and the initial function

$$u_0(x, y) = sinxsiny \tag{23}$$

We observe the difference between different dt and N. Find that when dt is not small enough, the Fourier Galerkin does not converges. And with N growing, the solution can be more exactly but can not influence the convergence.

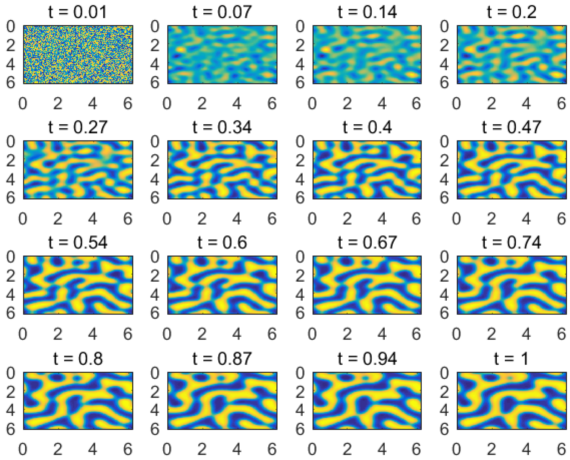

Figure 5: The time procedure of Cahn-Hailliard equation where $dt = 0.001, \epsilon^2 = 0.01, N = 100, u0 = rand(-1, 1)$

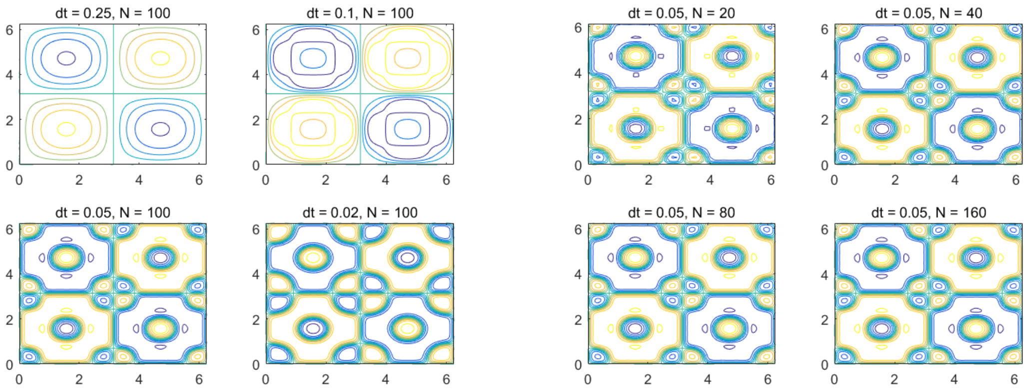

Figure 6: The numerical solutions of Cahn-Hailliard equation when time step dt and nodes N changs.

4 Matlab Code

```
%Allen-Cahn equation
%parameter initialization
%2N + 1 is the number of the points, [a,b] is the domain, dt is the time step, eps is ...
    the parameter, u0=u(x,0)
%t is the beginning time, T is the ending time
N = 20;
a = 0; b = 2 * pi ; dt = 0.001; h = (b - a) / (2 * N + 1);
x = h * (0 : 2 * N)'; y = h * (0 : 2 * N)';
eps = 0.01; u0 = sin(x) * sin(y');
%u0 = 2 * rand(2 * N + 1) - 1;
t = 0; T = 1;

%initialization
%w is the nonlinear term, hatu is the spectral of u, A is the iteration matrix
%W1,W2 is the translation matrix for fft
wp = u0 - u0.^3;
hatup = spectral_fft2(u0);
j = -N : N;
j = j .^ 2;
```

```
A = zeros(2 * N + 1, 2 * N + 1);
for k = 1 : 2 * N + 1
    A(1 : 2 * N + 1, k) = 1 ./ (1 + dt * eps * (j(1 : 2 * N + 1) + j(k)));
end
up = u0;
j = 0 : 2 * N;
W1 = (exp(-2 * pi * N / (2 * N + 1) * j' * 1i) * exp(-2 * pi * N / (2 * N + 1) * j * 1i));
W2 = (exp(2 * pi * N / (2 * N + 1) * j' * 1i) * exp(2 * pi * N / (2 * N + 1) * j * 1i));

%Iteration, *p shows the value of time t, *n shows the value of time t+dt
while t < T
    %calculate the spectral of w
    hatwp = ifft2(wp .* W1);

    %update the spectral of u
    hatun = (hatup  + dt .* hatwp ).*A;

    %transform the spectral to physical space, update u and w
    un = real(fft2(hatup) .* W2);
    wn = un - un.^3;

    %record residual
    res = max(abs(un - up));

    %time developing
    t = t + dt; up = un; wp = wn; hatup = hatun;

    %plot images
    %contour(X,Y,U);
    %pause(0.001);
end

%take in the end point
X=[x;b];
Y=[y;b];
U=[up up(1:2*N+1,1);up(1,1:2*N+1) up(1,1)];

%plot contour line
contour(X,Y,U);
```