$$\hat{AE}_1 = \sum_{m+n=p\pm N} \hat{u}_n \hat{v}_m \quad , \quad \hat{AE}_2 = \sum_{m+n=p\pm N} \hat{u}_n^* \hat{v}_m^*$$

4. If Aliasing error induced, Burger eqn in fourier space can be written as:

$$\frac{d\hat{u}_n}{dt} = -(\hat{H}_n + \hat{AE}) + L\hat{u}_n$$

First step $\hat{u}_n^* = \hat{u}_n + \alpha_1 \Delta t L\hat{u}_n + \beta_1 \Delta t L\hat{u}_n^* + \gamma_1 \Delta t (\hat{H}_n + \hat{AE})$

$$(1-\beta_1 \Delta t L)\hat{u}_n^* = \hat{u}_n + \alpha_1 \Delta t L\hat{u}_n + \gamma_1 \Delta t(\hat{H}_n + \hat{AE})$$

$$\Rightarrow \quad \hat{u}_n^* = (1-\beta_1 \Delta t L)^{-1}\left[\hat{u}_n + \alpha_1 \Delta t L\hat{u}_n + \gamma_1 \Delta t(\hat{H}_n + \hat{AE})\right]$$

Expand $(1-\beta_1 \Delta t L)^{-1}$ By Taylor series

$$(1-\beta_1 \Delta t L)^{-1} = 1 + \beta_1 \Delta t L + \beta_1^2 \Delta t^2 L^2 + \dots$$

Then $\hat{u}_n^* = (1+\beta_1 \Delta t L + \beta_1^2 \Delta t^2 L^2 + \dots)\left[\hat{u}_n + \alpha_1 \Delta t L\hat{u}_n + \gamma_1 \Delta t(\hat{H}_n + \hat{AE})\right]$

$$\hat{u}_n^* = \left[\hat{u}_n + \alpha_1 \Delta t L\hat{u}_n + \gamma_1 \Delta t(\hat{H}_n + \hat{AE})\right] + \left[\beta_1 \Delta t L\hat{u}_n + \beta_1 \alpha_1 \Delta t^2 L\hat{u}_n + \beta_1 \gamma_1 \Delta t^2 L^2(\hat{H}_n + \hat{AE})\right] + \left[\beta_1^2 \Delta t^2 L^2 \hat{u}_n + HOT \dots\right] + \dots$$

Step 2. $\hat{u}_{n+1} = \hat{u}_n^* + \alpha_2 \Delta t L\hat{u}_n^* + \beta_2 \Delta t L\hat{u}_{n+1} + \Delta t \gamma_2 H(\hat{u}_n^*) + \Delta t \zeta_1 H(\hat{u}_n)$

From Aliasing and phase shifting: $H(\hat{u}_n^*) = \hat{H}_n^* - \hat{AE}_2$

$$\begin{cases} \hat{AE}_1 = \sum_{m+n=p\pm N} \hat{u}_n \hat{v}_m \\ \hat{AE}_2 = \sum_{m+n=p\pm N} \hat{u}_n^* \hat{v}_m^* \end{cases}$$

Taylor expand $\hat{AE}_2$ & $\hat{H}_n^*$

4(a) $\rightarrow$ $\hat{AE}_2 = \hat{AE}_1 + \frac{dAE}{du}(\hat{u}_n^* - \hat{u}_n) + \frac{dAE}{du}\frac{(\hat{u}_n^* - \hat{u}_n)^2}{2} + HOT$

$$\hat{H}_n^* = \hat{H}_n + \frac{dH}{du}(\hat{u}_n^* - \hat{u}_n) + \frac{1}{2}(\hat{u}_n^* - \hat{u}_n)^2 \frac{d^2 H}{du^2} + HOT$$

Step 2: $(1-\beta_2\Delta t L)\hat{u}_{n+1} = \hat{u}_n^* + \alpha_2\Delta t L\hat{u}_n^* + \Delta t\gamma_2(\hat{H}_n^* - \hat{AE}_2)$

$$+\Delta t\zeta_1(\hat{H}_n + \hat{AE}_1)$$

$$\Rightarrow \hat{u}_{n+1} = (1-\beta_2\Delta t L)^{-1}\Big[\hat{u}_n^* + \alpha_2\Delta t L\hat{u}_n^* + \Delta t\gamma_2\Big(\hat{H}_n + (\hat{u}_n^* - \hat{u}_n)\frac{\partial H}{\partial u}$$

$$\underbrace{+\frac{1}{2}(\hat{u}_n^* - \hat{u}_n)^2\frac{\partial^2 H}{\partial u^2} + HOT}_{\text{HOT in Taylor Expansion, neglect}} - \hat{AE}_1 - (\hat{u}_n^* - \hat{u}_n)\frac{\partial\hat{AE}}{\partial u} \underbrace{- \frac{1}{2}(\hat{u}_n^* - \hat{u}_n)^2\frac{\partial^2\hat{AE}}{\partial u^2}}_{\text{High order in } \hat{AE}_1\text{, neglect}}$$

$$+HOT\Big) + \Delta t\zeta_1(\hat{H}_n + \hat{AE}_1)\Big]$$

$$\Rightarrow \hat{u}_{n+1} = (1+\beta_2\Delta t L + \beta_2^2\Delta t^2 L^2)\Big[\hat{u}_n^* + \alpha_2\Delta t L\hat{u}_n^* + \Delta t\gamma_2\Big(\hat{H}_n + \frac{\partial H}{\partial u}\big(\alpha_1\Delta t L\hat{u}_n$$

$$-\gamma_1\Delta t(\hat{H}_n + \hat{AE}_1) + \beta_2\Delta t L\hat{u}_n + \beta_1\alpha_1\Delta t^2 L\hat{u}_n - \gamma_1\beta_1^2\Delta t^2 L^2(\hat{H}_n + \hat{AE}_1)$$

$$-\beta_1^2\Delta t L^2\hat{u}_n\big) - \hat{AE}_1 - \big(\alpha_1\Delta t L\hat{u}_n - \gamma_1\Delta t(\hat{H}_n + \hat{AE}_1) + \beta_2\Delta t L\hat{u}_n + \beta_1\alpha_1\Delta t^2 L\hat{u}_n$$

$$-\gamma_1\beta_1^2\Delta t^2 L^2(\hat{H}_n + \hat{AE}_1) + \beta_1^2\Delta t^2 L^2\hat{u}_n\big)\frac{\partial\hat{AE}}{\partial u}\Big) + \Delta t\zeta_1(\hat{H}_n + \hat{AE}_1)\Big]$$

$O(\Delta t)$ $\hat{AE}_1$ term: $\underbrace{\gamma_1\Delta t\hat{AE}_1}_{\text{from } \hat{u}_n^*} - \gamma_2\Delta t\hat{AE}_1 + \Delta t\zeta_1\hat{AE}_1$

$$\Rightarrow \boxed{\gamma_1 - \gamma_2 + \zeta_1 = 0}$$

This is the condition for $O(\Delta t)$ Aliasing Error to be canceled out

Conditions for the scheme to be 2nd order Accurate which given by Homework 1 are:

1. $\gamma_1 + \gamma_2 + \zeta_1 = 1$
2. $\alpha_1 + \alpha_2 + \beta_1 + \beta_2 = 1$
3. $\beta_1(\alpha_1 + \beta_1 + \beta_2 + \alpha_2) + \beta_2(\alpha_1 + \alpha_2 + \beta_2) + \alpha_1\alpha_2 = \frac{1}{2}$
4. $\beta_2(\gamma_1 + \gamma_2 + \zeta_1) + \gamma_1(\beta_1 + \alpha_2) = \frac{1}{2}$
5. $\alpha_1\gamma_2 + \beta_1\gamma_2 = \frac{1}{2}$
6. $\gamma_1\gamma_2 = \frac{1}{2}$

Also, with additional condition of $\beta_1 = \beta_2$, we can solve these 8 Eqns to get the coefficients

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \\ \delta_1 \\ \delta_2 \\ z_1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ -\frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \\ \frac{1}{2} \\ -\frac{1}{2} \end{bmatrix}$$

5. I Raw Burger

step1:
$$\begin{tikzcd} u \arrow[r, "fft"] & \hat{u} \arrow[r, "ifft"] \arrow[d, "N\ f.p.\ \cdot ik"'] & u \\ & \hat{v} \arrow[r, "ifft"] & v \end{tikzcd} \Bigg\} \xrightarrow{N\ f.p.} uv \xrightarrow{fft} \widehat{uv}$$

(where $fft \sim N\log_2 N$)

step2:
$$\begin{tikzcd} \hat{u}^* \arrow[r, "ifft"] \arrow[d, "N\ f.p.\ \cdot ik"'] & u^* \\ \hat{v}^* \arrow[r, "ifft"] & v^* \end{tikzcd} \Bigg\} \xrightarrow{N\ f.p.} (uv)^* \xrightarrow{fft} \widehat{uv}$$

Forget About the Additional ifft which transfer our solution back to physical domain because All four methods need it. we only care About the difference during Integration

thus two step need

$3\ fft + 4\ ifft + 4N\ f.p.$

II. zero padding ($\frac{M}{N} = \frac{3}{2}$)

Same procedure As before, $3\ fft + 4\ ifft + 4N\ f.p.$

However, since more grid points are used, cost of fft and ifft are larger ($\sim M\log_2 M$); Also more memory Are required to store the fourier coefficients, by factor of $3/2$ compare with I

III phase shifting

Step 1

Route 1:

$$\left.\begin{array}{ccccc} u & \xrightarrow{fft} & \hat{u} & \xrightarrow{ifft} & u \\ & & \Big\downarrow{\scriptstyle N\,flp,\ \cdot ik\cdot \hat{u}} & & \\ & & \hat{v} & \xrightarrow{ifft} & v \end{array}\right\} \xrightarrow{N\text{-}flp} uv \xrightarrow{fft} \widehat{uv}$$

Route 2:

$$\left.\begin{array}{ccccc} \hat{u} & \xrightarrow[N\,flp]{\cdot e^{ik\Delta}} & \hat{u}_\Delta & \xrightarrow{ifft} & u_\Delta \\ & & \Big\downarrow{\scriptstyle N\,flp,\ \cdot ik} & & \\ & & \hat{v}_\Delta & \xrightarrow{fft} & v_\Delta \end{array}\right\} \xrightarrow{N\cdot flp} (uv)_\Delta \xrightarrow{fft} (\widehat{uv})_\Delta$$

Step 2

Route 1:

$$\left.\begin{array}{ccc} \hat{u}^* & \xrightarrow{ifft} & u^* \\ \Big\downarrow{\scriptstyle N\,flp,\ \cdot ik} & & \\ \hat{v}^* & \xrightarrow{ifft} & v^* \end{array}\right\} \xrightarrow{N\,flp} (uv)^* \xrightarrow{fft} \widehat{uv}$$

Route 2:

$$\left.\begin{array}{ccccc} \hat{u}^* & \xrightarrow[N\,flp]{\cdot e^{ik\Delta}} & \hat{u}^*_\Delta & \xrightarrow{ifft} & u_\Delta \\ & & \Big\downarrow{\scriptstyle N\,flp,\ \cdot ik} & & \\ & & \hat{v}^*_\Delta & \xrightarrow{ifft} & v_\Delta \end{array}\right\} \xrightarrow{N\,flp} (uv)_\Delta \xrightarrow{fft} (\widehat{uv})_\Delta$$

Two step requires: 5 fft + 8 ifft + 8N flp.

For memory usage. since $\hat{u}$ and $\hat{u}_\Delta$, $\hat{v}$ and $\hat{v}_\Delta$, $\widehat{uv}$ and $\widehat{uv}_\Delta$ are essentially able to share the same register. Then the memory usage should be the same As I.

IV. Cheap Flipped Burger

same As I but with Additional 2N f.p. coming from phase shifting.

Then two step Require

$3\,\mathrm{fft} + 4\,\mathrm{ifft} + 6N$ f.p.

For Quality of Result

I. Raw Burger.

Aliasing Error will pollute the solution, the solution will become unstable and blow up #

II. zero padding

Good quality since the leading Aliasing Error was Cancelled out. However, for $M = 1e2$ points, instead of having $1e2$ modes, we only have $N = 128$ modes to represent the function.

III phase shifting

Good quality, since we cancelled out All leading AE

IV Cheap Flipped burger

Good quality with cheaper cost compare with II III.

Problem1. Raw Burger;

```
close all;clear;clc;
% Raw Burger
nu = 0.001; %Kinematic Viscosity;
CFL = 0.1; %CFL Number;
N = 128; %Total number of Grid Points;
dx = 2*pi/N; %Spatial Resolution;

x = 0:dx:(2*pi-dx); %Spatial Domain [0,2*pi);
x = x'; %Transfer it into Column Vector;

u = sin(x); %Initial Condition;
u_hat(:,1) = fft(u);%Fourier Coefficients of Initial
Condition;

%Paramters of RK-Theta;
alpha1 = sqrt(2)-1; alpha2 = 0;  beta1 = 1-sqrt(2)/2; beta2
= beta1;
gamma1 = sqrt(2)/2; gamma2 = gamma1; zeta1 = 1-sqrt(2);

%Fourier Modes we are working with;
n = linspace(-N/2,N/2-1,N);
kn = reorder(n'); %Only For Function with Period of 2*pi,
and change it into Column Vector;

figure;
i = 1; t = 0; dt = 100;

while t < 2
    %Adaptive Time Stepping
    dt = (CFL*dx)/abs((max(u(:,i))));

    %Apply Pseduospectral Method to deal with Non-Linear
Terms;
    uv_hat = PseduoSpectral(u_hat,kn);

    %Application of RK-theta to advance fourier coefficient
forward;
    u_hat = (u_hat - dt*alpha1*nu*(kn.^2).*u_hat -
dt*gamma1*uv_hat)./...
        (1+dt*beta1*nu*kn.^2); %u_hat here is essentially
u_hat_star;

    uv_star_hat = PseduoSpectral(u_hat,kn);

    u_hat = (u_hat - dt*alpha2*nu*(kn.^2).*u_hat -
dt*gamma2*uv_star_hat - dt*zeta1*uv_hat)./...
```

```
        (1 + dt*nu*beta2*kn.^2);

    %Inverse Fourier Transform to back to physical domain;
    u(:,i+1) = ifft(u_hat);

    plot(x,real(u(:,i+1))); title(num2str(t)); ylim([-
1.5,1.5])
    pause(0.0001);

    t = t + dt;
    i = i + 1;
end


% Exchange the Order of Our Modes with that Given by Matlab
fft
function u_fft = reorder(u_fft)
N = max(size(u_fft));
u_fft = -flip(u_fft);
%Store First N/2-1 elements
u_fft_inter = u_fft(1:(N/2-1));

u_fft(1:(N/2+1)) = u_fft((N/2):N);
u_fft((N/2+2):N) = u_fft_inter;
end

function uv_hat = PseduoSpectral(u_hat,kn)

v_hat = 1i*kn.*u_hat;
uv_phy = ifft(u_hat).*ifft(v_hat);
uv_hat = fft(uv_phy);
end
```

Problem2. Well Done Padded Burger

```
close all;clear;clc;

%Well Done Padded Burgers
nu = 0.001; %Kinematic Viscosity;
CFL = 0.1; %CFL Number;
N = 128; %Original Grid
M = 192; %Extended Grid Points
dx = 2*pi/M; %Spatial Resolution;
x = 0:dx:(2*pi-dx); %Spatial Domain [0,2*pi);
x = x'; %Transfer it into Column Vector;


u = sin(x); %Initial Condition;
u_hat(:,1) = fft(u);%Fourier Coefficients of Initial
Condition;

%Paramters of RK-Theta;
alpha1 = sqrt(2)-1; alpha2 = 0;  beta1 = 1-sqrt(2)/2; beta2
= beta1;
gamma1 = sqrt(2)/2; gamma2 = gamma1; zeta1 = 1-sqrt(2);

%Fourier Modes we are working with;
n = linspace(-M/2,M/2-1,M);
kn = reorder(n'); %Only For Function with Period of 2*pi,
and change it into Column Vector;

figure;
i = 1; t = 0; dt = 0 ;
while t < 2
    %Adaptive Time Stepping
    dt = (CFL*dx)/abs((max(u(:,i))));

    %Zero Padding
    uv_hat = zeropadding(u_hat,N,M);

    %Application of RK-theta to advance fourier coefficient
forward;
    u_hat = (u_hat - dt*alpha1*nu*(kn.^2).*u_hat -
dt*gamma1*uv_hat)./...
        (1+dt*beta1*nu*kn.^2); %Here u_hat is essentially
u_hat_star

    %Zero Padding
    u_star_hat = u_hat;
    uv_star_hat = zeropadding(u_star_hat,N,M);
```

```
    u_hat = (u_hat - dt*alpha2*nu*(kn.^2).*u_hat -
dt*gamma2*uv_star_hat - dt*zeta1*uv_hat)./...
        (1 + dt*nu*beta2*kn.^2);

    %Inverse Fourier Transform to to back to physical
domain;
    u(:,i+1) = ifft(u_hat);

    t = t + dt;
    plot(x,real(u(:,i+1))); title(num2str(t)); ylim([-1.5
1.5]);
    pause(0.0001);
    i = i + 1;

end


% Exchange the Order of Our Modes with that Given by Matlab
fft
function u_fft = reorder(u_fft)
N = max(size(u_fft));
u_fft = -flip(u_fft);
%Store First N/2-1 elements
u_fft_inter = u_fft(1:(N/2-1));

u_fft(1:(N/2+1)) = u_fft((N/2):N);
u_fft((N/2+2):N) = u_fft_inter;
end
function uv_hat_0 = zeropadding(u_hat,N,M)
n = linspace(-M/2,M/2-1,M);
kn = reorder(n)';
u_hat_pad = u_hat ; u_hat_pad(N/2+2:M+1-N/2) = 0 ;
v_hat_pad = 1i*kn.*u_hat ; v_hat_pad(N/2+2:M+1-N/2) = 0;
uv_phy = ifft(u_hat_pad).*ifft(v_hat_pad);
uv_hat_0 = fft(uv_phy); uv_hat_0(N/2+2:M+1-N/2) = 0 ;
end
```

Problem3. Rare Cheap Flipped Burger.

```
close all; clear;clc;
% Raw Burger
nu = 0.001; %Kinematic Viscosity;
CFL = 0.1; %CFL Number;
N = 128; %Total number of Grid Points;
dx = 2*pi/N; %Spatial Resolution;

x = 0:dx:(2*pi-dx); %Spatial Domain [0,2*pi);
x = x'; %Transfer it into Column Vector;

u = sin(x); %Initial Condition;
u_hat(:,1) = fft(u);%Fourier Coefficients of Initial
Condition;

%Paramters of RK-Theta;
alpha1 = sqrt(2)-1; alpha2 = 0;  beta1 = 1-sqrt(2)/2; beta2
= beta1;
gamma1 = sqrt(2)/2; gamma2 = gamma1; zeta1 = 1-sqrt(2);

%Fourier Modes we are working with;
n = linspace(-N/2,N/2-1,N);
kn = reorder(n'); %Only For Function with Period of 2*pi,
and change it into Column Vector;

%Phase Shifting Configuration;
delta = dx/2;

figure;
i = 1; t = 0; dt = 0.001 ;

while t < 2
    %Adaptive Time Stepping
    dt = (CFL*dx)/abs((max(u(:,i))));

    uv_ps_hat = phaseshifting(u_hat,kn,delta);

    %Application of RK-theta to advance fourier coefficient
forward;
    u_hat = (u_hat - dt*alpha1*nu*(kn.^2).*u_hat -
dt*gamma1*uv_ps_hat)./...
        (1+dt*beta1*nu*kn.^2); %u_hat = u_hat_star

    uv_star_ps_hat = phaseshifting(u_hat,kn,delta);
```

```
    u_hat = (u_hat - dt*alpha2*nu*(kn.^2).*u_hat -
dt*gamma2*uv_star_ps_hat - dt*zeta1*uv_ps_hat)./...
        (1 + dt*nu*beta2*kn.^2);

    %Inverse Fourier Transform to to back to physical
domain;
    u(:,i+1) = ifft(u_hat);

    plot(x,real(u(:,i+1))); title(num2str(t)); ylim([-1.5
1.5]);
    pause(0.001);

    t = t + dt;
    i = i + 1;
end

% Exchange the Order of Our Modes with that Given by Matlab
fft
function u_fft = reorder(u_fft)
N = max(size(u_fft));
u_fft = -flip(u_fft);
%Store First N/2-1 elements
u_fft_inter = u_fft(1:(N/2-1));

u_fft(1:(N/2+1)) = u_fft((N/2):N);
u_fft((N/2+2):N) = u_fft_inter;
end

function uv_ps_hat = phaseshifting(u_hat,kn,delta)
v_hat = 1i*kn.*u_hat;
%Apply Shifting
u_delta_hat = u_hat.*exp(1i*kn*delta);
v_delta_hat = 1i*kn.*u_delta_hat;

uv_phy = ifft(u_hat).*ifft(v_hat);
uv_delta_phy = ifft(u_delta_hat).*ifft(v_delta_hat);

uv_hat = fft(uv_phy);
uv_delta_hat = fft(uv_delta_phy);

%Shift Back
uv_ps_hat = (1/2)*(uv_hat + uv_delta_hat.*exp(-
1i*kn*delta));
end
```

Problem4. Coefficient

```
close all;clear;clc
syms a1 a2 b1 b2 g1 g2 z1;
eq1 = g1 + g2 + z1 -1;
eq2 = a1 + a2 + b1 + b2 - 1;
eq3 = b1*(a1 + a2 + b1 + b2) + b2*(a1 + a2 + b2) + a1*a2 -
1/2;
eq4 = b2*(g1 + g2 + z1) + g1*(b1 + a2) - 1/2;
eq5 = a1*g2 + b1*g2 -1/2;
eq6 = g1*g2 - 1/2
eq7 = b1 - b2;
eq8 = g1 - g2 + z1;
eqns = [eq1 eq2 eq3 eq4 eq5 eq6 eq7 eq8];
solu = solve(eqns, [a1 a2 b1 b2 g1 g2 z1])

a1_s = solu.a1
a2_s = solu.a2
b1_s = solu.b1
b2_s = solu.b2
g1_s = solu.g1
g2_s = solu.g2
z1_s = solu.z1
```

Problem4. Rare Cheap Burger

```
clear;clc;
% Raw Burger
nu = 0.001; %Kinematic Viscosity;
CFL = 0.1; %CFL Number;
N = 128; %Total number of Grid Points;
dx = 2*pi/N; %Spatial Resolution;

x = 0:dx:(2*pi-dx); %Spatial Domain [0,2*pi);
x = x'; %Transfer it into Column Vector;

u = sin(x); %Initial Condition;
u_hat(:,1) = fft(u);%Fourier Coefficients of Initial
Condition;

%Paramters of RK-Theta;
alpha1 = 1/2; alpha2 = -1/2;  beta1 = 1/2; beta2 = 1/2;
gamma1 = 1; gamma2 = 1/2; zeta1 = -1/2;

%Fourier Modes we are working with;
n = linspace(-N/2,N/2-1,N);
kn = reorder(n'); %Only For Function with Period of 2*pi,
and change it into Column Vector;

%Phase Shifting Configuration;
delta = dx/2;

figure;
i = 1; t = 0; dt = 100;

while t < 2
    %Adaptive Time Stepping
    dt = (CFL*dx)/abs((max(u(:,i))));

    %Apply Pseduospectral Method to deal with Non-Linear
Terms;
    uv_hat = PseduoSpectral(u_hat,kn);

    %Application of RK-theta to advance fourier coefficient
forward;
    u_hat = (u_hat - dt*alpha1*nu*(kn.^2).*u_hat -
dt*gamma1*uv_hat)./...
        (1+dt*beta1*nu*kn.^2); %u_hat here is essentially
u_hat_star;

    uv_star_ps_hat = phaseshifting2(u_hat,kn,delta);
```

```
    u_hat = (u_hat - dt*alpha2*nu*(kn.^2).*u_hat -
dt*gamma2*uv_star_ps_hat - dt*zeta1*uv_hat)./...
        (1 + dt*nu*beta2*kn.^2);

    %Inverse Fourier Transform to back to physical domain;
    u(:,i+1) = ifft(u_hat);

    plot(x,real(u(:,i+1))); title(num2str(t)); ylim([-
1.5,1.5])
    pause(0.0001);

    t = t + dt;
    i = i + 1;
end


% Exchange the Order of Our Modes with that Given by Matlab
fft
function u_fft = reorder(u_fft)
N = max(size(u_fft));
u_fft = -flip(u_fft);
%Store First N/2-1 elements
u_fft_inter = u_fft(1:(N/2-1));

u_fft(1:(N/2+1)) = u_fft((N/2):N);
u_fft((N/2+2):N) = u_fft_inter;
end

function uv_hat = PseduoSpectral(u_hat,kn)

v_hat = 1i*kn.*u_hat;
uv_phy = ifft(u_hat).*ifft(v_hat);
uv_hat = fft(uv_phy);
end

function uv_ps_hat = phaseshifting2(u_hat,kn,delta)

%Apply Shifting
u_delta_hat = u_hat.*exp(1i*kn*delta);
v_delta_hat = 1i*kn.*u_delta_hat;

uv_delta_phy = ifft(u_delta_hat).*ifft(v_delta_hat);
uv_delta_hat = fft(uv_delta_phy);

%Shift Back
uv_ps_hat = uv_delta_hat.*exp(-1i*kn*delta);
end
```